

The `bnumexpr` package

JEAN-FRANÇOIS BURNOL

jfbu (at) free (dot) fr

Package version: 1.2d (2019/01/07); documentation date: 2019/01/07.

From source file bnumexpr.dtx. Time-stamp: <07-01-2019 at 11:53:27 CET>.

Contents

1	Examples	1
2	Differences from <code>\numexpr</code>	2
3	Printing big numbers	4
4	Expression syntax	5
5	Options	6
6	<code>\bnumexprsetup</code>	6
7	Readme	7
8	Changes	9
9	Package <code>bnumexpr</code> implementation	11

1 Examples

Package `bnumexpr` provides `\thebnumexpr{<expression>}\relax` which is analogous to `\the \numexpr{<expression>}\relax`, with these extensions:

- it allows arbitrarily big integers,
- it computes powers (with either `**` or `^` as infix operator),
- it computes factorials (with `!` as postfix operator),
- it has an operator `//` for floored division and `/:` for the associated modulo,
- the space character can be used to separate in the source blocks of digits for better readability of long numbers,
- since 1.2b the underscore `_` may also be used as visual digit separator,
- comma separated expressions are allowed.

It expands completely, but in two steps (whereas `\the \numexpr{<expression>}\relax` expands in one step only, and `\numexpr` by itself is not expandable).

1.2d adds a `\bnumeval{<expression>}` interface. The `\bnumeval` macro also expands completely in two steps.

2 Differences from `\numexpr`

Examples:

```
\thebnumexpr 1 208 637 867 * (2 187 917 891 + 3 109 197 072)\relax
6402293730134103921

\bnumeval {(13_8089_1090-300_1890_2902)*(1083_1908_3901-109_8290_3890)}
-2787514672889976289932

\thebnumexpr (92_874_927_979**5-31_9792_7979**6)/30!\relax
-4006240736596543944035189

\bnumeval {30!/20!/21/22/23/24/25/(26*27*28*29)}
30

\thebnumexpr 13^50//12^50, 13^50/:12^50\relax
54, 650556287901099025745221048683760161794567947140168553

\bnumeval {13^50/12^50, 12^50}
55, 910043815000214977332758527534256632492715260325658624

\thebnumexpr (1^10+2^10+3^10+4^10+5^10+6^10+7^10+8^10+9^10)^3\relax
118685075462698981700620828125

\bnumeval {100!/36^100}
219
```

2 Differences from `\numexpr`

Apart from the extension to big integers (i.e. exceeding the \TeX limit at 2147483647), and the added operators, there are a number of important differences between `\bnumexpr` and `\numexpr`:

1. one must use either `\thebnumexpr` or `\bnethe \bnumexpr` to get a printable result, as `\bnumexpr ... \relax` expands to a private format (using `\number` as prefix to `\bnumexpr` would only serve to trigger the expansion to the `bnumexpr` private format, hence will raise a \TeX error),
2. one may embed directly (without `\bnethe`) a `\bnumexpr ... \relax` in another one (or in a `\xintexpr ... \relax`), but not in a `\numexpr ... \relax`; on the other hand a `\numexpr ... \relax` does not need to be prefixed by `\the` or `\number` inside `\bnethe \bnumexpr` or `\thebnumexpr`,
3. contrarily to `\numexpr`, the `\bnumexpr` parser stops only after having found (and swallowed) a mandatory ending `\relax` token,

2 Differences from `\numexpr`

4. in particular spaces between digits do not stop `\bnumexpr`, in contrast with `\numexpr`:
`\the \numexpr 3 5+79\relax` expands (in one step) to `35+79\relax`
`\the\bnumexpr 3 5+79\relax` expands (in two steps) to `114`
5. one may do `\edef \tmp {\bnumexpr 1+2\relax }`, and then either use `\tmp` in another `\bnumexpr ... \relax`, or print it via `\bnethe \tmp`. The computation is done at the time of the `\edef` (and two expansion steps suffice). This is again in contrast with `\numexpr ... \relax` which, without `\the` (or `\number` or `\romannumeral`) as prefix would not expand inside an `\edef`,
6. expressions may be comma separated. On input, spaces are ignored, naturally, and on output the values are comma separated with a space after each comma,
7. `\bnumexpr -(1+1)\relax` is legal contrarily to `\numexpr -(1+1)\relax` which raises an error,
8. `\numexpr 2\cnta \relax` is illegal (with `\cnta` a `\count`-variable.) But `\bnumexpr 2\cnta \relax` is perfectly legal and will do the tacit multiplication,
9. more generally, tacit multiplication applies in front of parenthesized sub-expressions, or sub `\bnumexpr ... \relax` (or `\numexpr ... \relax`),
10. the underscore `_` is accepted within the digits composing a number and is silently ignored by `\bnumexpr`,
11. `\numexpr` accepts \TeX syntax for hexadecimal input `"EF` (or octal input `'377`), but currently `\bnumexpr` does not. This could possibly be added in future¹ (hexadecimal prefix `"` is part of the recognized `\xintiexpr` syntax from `xintexpr` if `xintbinhex` is loaded.) In the meantime, one may encapsulate such inputs (obeying \TeX 's bound on numbers) into a sub `\numexpr ... \relax`.

An important thing to keep in mind is that if one has a calculation whose result is a small integer, acceptable by \TeX in `\ifnum` or count assignments, this integer produced by `\the\bnumexpr` is not self-delimiting, contrarily to a `\numexpr ... \relax` construct: the situation is exactly as with a `\the \numexpr ... \relax`, thus one may need to terminate the number to avoid premature expansion of following tokens; for example with the `\space` token.

The parser `\bnumexpr` is a scaled-down version of parser `\xintiexpr` from package `xintexpr`. It lacks in particular boolean operators, square roots and other functions, variables, hexadecimal inputs, etc... it may be slightly

¹It is only a matter of copying pasting relevant code from `xintexpr`, so depends upon user demands addressed to the author ;-).

3 Printing big numbers

faster when handling complicated expressions as it does not have to check so many things.

The documentation of `xintexpr` explains that there is an impact on the memory of \TeX (the string pool, the hash table) as each intermediate number is stored as a dummy control sequence name during processing. After thousands of evaluations with numbers having hundreds of digits, parts of the \TeX memory can become saturated and end the `latex|pdflatex` run, but the problem can be avoided via enlarged memory parameters for `pdfTeX`, as made possible by modern \TeX installations. Anyhow, computations with thousands of digits take time, and this is probably a more stringent constraint.

If the same operations need to be repeated again and again tens of thousands of times on varying (big) numbers, the memory problem mentioned above may be avoided by using nested macros rather than `\bnumexpr` or `\xintexpr` expressions. Utility `\xintNewIIExpr` from package `xintexpr` can be used to construct the possibly very complicated nested macro from a given expression with the needed operators and usual `#1`, `#2`, `#3`, ... placeholders.

The ε - \TeX extensions are required (this is the default on all modern installations for `latex|pdflatex` and also for `xelatex|lualatex`).

3 Printing big numbers

\TeX will not split long numbers at the end of lines. I personally often use helper macros (not in the package) of the following type:

```
\def\allowsplits #1{\ifx #1\relax \else #1\hskip 0pt plus 1pt\relax
\expandafter\allowsplits\fi}%
\def\printnumber #1{\expandafter\allowsplits \romannumeral-`0#1\relax }%
% \printnumber thus first ``fully'' expands its argument.
```

```
\thebnumexpr 1000!\relax = 402387260077093773543702433923003985719374864
210714632543799910429938512398629020592044208486969404800479988610197196
058631666872994808558901323829669944590997424504087073759918823627727188
732519779505950995276120874975462497043601418278094646496291056393887437
886487337119181045825783647849977012476632889835955735432513185323958463
075557409114262417474349347553428646576611667797396668820291207379143853
719588249808126867838374559731746136085379534524221586593201928090878297
308431392844403281231558611036976801357304216168747609675871348312025478
58932076716913244842623613141250878020800026168315102734182797704784635
868170164365024153691398281264810213092761244896359928705114964975419909
342221566832572080821333186116811553615836546984046708975602900950537616
475847728421889679646244945160765353408198901385442487984959953319101723
355556602139450399736280750137837615307127761926849034352625200015888535
147331611702103968175921510907788019393178114194545257223865541461062892
187960223838971476088506276862967146674697562911234082439208160153780889
893964518263243671616762179168909779911903754031274622289988005195444414
282012187361745992642956581746628302955570299024324153181617210465832036
786906117260158783520751516284225540265170483304226143974286933061690897
```

4 Expression syntax

```
968482590125458327168226458066526769958652682272807075781391858178889652
208164348344825993266043367660176999612831860788386150279465955131156552
036093988180612138558600301435694527224206344631797460594682573103790084
024432438465657245014402821885252470935190620929023136493273497565513958
720559654228749774011413346962715422845862377387538230483865688976461927
38381490014076731044664025989949022221765904339901886018566526485061799
702356193897017860040811889729918311021171229845901641921068884387121855
646124960798722908519296819372388642614839657382291123125024186649353143
970137428531926649875337218940694281434118520158014123344828015051399694
290153483077644569099073152433278288269864602789864321139083506217095002
597389863554277196742822248757586765752344220207573630569498825087968928
162753848863396909959826280956121450994871701244516461260379029309120889
086942028510640182154399457156805941872748998094254742173582401063677404
595741785160829230135358081840096996372524230560855903700624271243416909
004153690105933983835777939410970027753472000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000
000
```

4 Expression syntax

It is the expected one with infix operators and parentheses, the recognized operators being `+`, `-`, `*`, `/` (rounded division), `^` (power), `**` (power), `//` (by default floored division), `/:` (the associated modulo) and `!` (factorial).²

The modulo `/:` is by default associated with the floored division `//`, but using `\bnumexprsetup {mod=...}` it can, like the other operators, be remapped to any macro of one's choice.

Different computations may be separated by commas. The whole expression is handled token by token, any component (digit, operator, parenthesis... even the ending `\relax`) may arise on the spot from macro expansions.

The precedence rules are the expected ones.

There is currently no user interface to change precedence levels. The three operators `/`, `//`, `/:` are at the same level of precedence as the multiplication `*`. The factorial postfix `!` has highest precedence. The minus signs inherit the precedence level of the previously encountered infix operators.

In case of equal precedence the operations are left-associative, hence:

```
\bnumeval {2^3^4, (2^3)^4, 2^(3^4)}
```

```
4096, 4096, 2417851639229258349412352
```

The underscore `_` can be used to separate digits in long numbers, for readability of the input.

²Releases 1.2b and earlier associated in the default configuration truncated division to `//`. This was changed at 1.2c to stay in sync with `xintcore 1.2p`. For backwards compatibility, one may add to existing document `\bnumexprsetup{div=\xintiiDivTrunc, mod=\xintiiModTrunc}`.

5 Options

The sole package option is `custom`: it tells `bnumexpr` not to load package `xintcore`.

6 `\bnumexprsetup`

Package `bnumexpr` needs that some big integer engine provides the macros doing the actual computations. By default, it loads package `xintcore` (a subset of `xint`; version 1.3d is required) and uses `\bnumexprsetup` in the following way:

```
\usepackage{xintcore}
\bnumexprsetup{add=\xintiiAdd, sub=\xintiiSub, mul=\xintiiMul,
               divround=\xintiiDivRound, div=\xintiiDivFloor,
               mod=\xintiiMod, pow=\xintiiPow, fac=\xintiiFac}
```

If using `\bnumexprsetup`, it is not necessary to specify all keys, for example one can do `\bnumexprsetup {mul=\MyFasterMul }`, and only multiplication will be changed.

Naturally it is up to the user to load the appropriate package for the alternative macros.

As per the macros which are the key values, they must have the following properties:

1. they must be completely expandable (in the sense of an `\edef` or a `\csname ... \endcsname`.)
2. they must fully expand their arguments first (in the sense of `\romannumeral -`0`.)
3. they must output a number with no leading zeros, at most one minus sign and no plus sign.

The first two items are truly mandatory, the last one may be not obeyed if the extra key `opp` is used with `\bnumexprsetup` to specify a suitable macro for the opposite of a number. This macro will be presented not with a braced argument but directly with a sequence of digits (either as gathered by the parser which skips leading zeros, or as produced by the other arithmetic macros and then there could be a minus, or even a plus if macros others than the ones from `xintcore` have been used). Thus, `opp` could identify a plus sign `+` upfront and then act adequately.³

Macro `\bnumexprsetup` can be used multiple times in the same document, thus allowing to switch math engines or to remap operators to some other arithmetic macros of the same math engine. Its effect obeys the local scope.

³see `\BNE_Op_opp` in the code for the default.

7 Readme

```
| Source:  bnumexpr.dtx
| Version: v1.2d, 2019/01/07 (doc: 2019/01/07)
| Author:  Jean-Francois Burnol
| Info:    Expressions with big integers
| License: LPPL 1.3c
```

README: [Usage], [Installation], [License]

Usage

The package `\bnumexpr` allows `_expandable_` computations with big integers and the four infix operators `+`, `-`, `*`, `/` familiar from the `\numexpr` e-TeX parser.

Besides extending the scope to arbitrarily big numbers (and having a more complete syntax, for example `-(1)` is legal input), it adds the (by default) floored division operator `//`, and its associated modulo `/:`, the power operator `^` (or equivalently `**`), and the factorial post-fix operator `!`. The space character as well as the underscore character `_` both may serve to optionally separate digits in long numbers, for better readability of the input.

For example:

```
\bnumeval{( 92_874_927_979^5 - 31_9792_7979^6 ) / 30!}
```

The above expands (in two steps) to `-4006240736596543944035189` (the `/:` does rounded division to match the `\numexpr` behaviour).

The expression parser is scaled-down from the `\xinttheiiexpr...\relax` parser as provided by package `xintexpr`^[1]: it does not handle hexadecimal input, boolean operators, dummy or user defined variables, functions, etc...

By default, the package loads `xintcore`^[1] (release 1.3d is then required) but it is possible via option `_custom_` and macro `\bnumexprsetup` to map the operators to macros of one's own choice. It is the responsibility of the user to load the packages providing these custom macros.

Notice that the possibility not to use the `xintcore` macros might be removed in the future: perhaps a future release will maintain during computations a private internal representation (especially tailored either for the `xintcore` macros or new ones which would be included within `\bnumexpr.sty` itself) and the constraints this implies may render optional use of other macros impossible.

[1]: <<http://www.ctan.org/pkg/xint>>

Installation

Obtain `\bnumexpr.dtx` (and possibly, `\bnumexpr.ins` and the `\README`) from CTAN:

```
> <http://www.ctan.org/pkg/bnumexpr>
```

Both `"tex bnumexpr.ins"` and `"tex bnumexpr.dtx"` extract from

7 Readme

`bnumexpr.dtx` the following files:

`bnumexpr.sty`
: this is the style file.

`README.md`
: reconstitutes this README.

`bnumexprchanges.tex`
: lists changes from the initial version.

`bnumexpr.tex`
: can be used to generate the documentation:

- : - with latex+dvipdfmx: ``latex bnumexpr.tex`` (thrice) then
 ``dvipdfmx bnumexpr.dvi``.
- : Ignore dvipdfmx warnings, but if the pdf file has problems with
 fonts (possibly from an old dvipdfmx), use then rather pdflatex.
- : - with pdflatex: ``pdflatex bnumexpr.tex`` (thrice).
- : In both cases files `README.md` and `bnumexprchanges.tex` must
 be present in the same repertory.

without `bnumexpr.tex`:
: ``pdflatex bnumexpr.dtx`` (thrice) extracts all files and
 simultaneously generates the pdf documentation.

Finishing the installation:

```
bnumexpr.sty  --> TDS:tex/latex/bnumexpr/
bnumexpr.dtx  --> TDS:source/latex/bnumexpr/
bnumexpr.ins  --> TDS:source/latex/bnumexpr/
bnumexpr.pdf  --> TDS:doc/latex/bnumexpr/
README        --> TDS:doc/latex/bnumexpr/
```

Files `bnumexpr.tex`, `bnumexprchanges.tex`, `README.md` may be
discarded.

License

Copyright (C) 2014-2019 by Jean-Francois Burnol

| This Work may be distributed and/or modified under the
| conditions of the LaTeX Project Public License 1.3c.
| This version of this license is in

> <<http://www.latex-project.org/lppl/lppl-1-3c.txt>>

| and version 1.3 or later is part of all distributions of
| LaTeX version 2005/12/01 or later.

This Work has the LPPL maintenance status "author-maintained".

The Author and Maintainer of this Work is Jean-Francois Burnol.

This Work consists of the main source file `bnumexpr.dtx`
and the derived files

bnumexpr.sty, bnumexpr.pdf, bnumexpr.ins, bnumexpr.tex,
bnumexprchanges.tex, README.md

8 Changes

- 1.2d (2019/01/07)
 - requires [xintcore 1.3d](#) or later (if not using option [custom](#)).
 - adds `\bnumeval{<expression>}` user interface.
- 1.2c (2017/12/05) **Breaking changes:**
 - requires [xintcore 1.2p](#) or later (if not using option [custom](#)).
 - `divtrunc` key of `\bnumexprsetup` is renamed to `div`.
 - the `//` and `/:` operators are now by default associated to the *floored* division. This is to keep in sync with the change of [xintcore](#) at [1.2 2p](#).
 - for backwards compatibility, one may add to existing document:
`\bnumexprsetup{div=\xintiiDivTrunc, mod=\xintiiModTrunc}`
- 1.2b (2017/07/09)
 - the `_` may be used to separate visually blocks of digits in long numbers.
- 1.2a (2015/10/14)
 - requires [xintcore 1.2](#) or later (if not using option [custom](#)).
 - additions to the syntax: factorial `!`, truncated division `//`, its associated modulo `/:` and `**` as alternative to `^`.
 - all options removed except [custom](#).
 - new command `\bnumexprsetup` which replaces the commands such as `\bn2umexprusesbigintcalc`.
 - the parser is no more limited to numbers with at most 5000 digits.
- 1.1b (2014/10/28)
 - README converted to [markdown/pandoc](#) syntax,
 - the package now loads only [xintcore](#), which belongs to [xint](#) bundle version [1.1](#) and extracts from the earlier [xint](#) package the core arithmetic operations as used by [bnumexpr](#).
- 1.1a (2014/09/22)
 - added [l3bigint](#) option to use experimental ~~TeX~~^{LaTeX} package of the same name,
 - added Changes and Readme sections to the documentation,
 - better `\BNE_protect` mechanism for use of `\bnumexpr ... \relax` inside an `\edef` (without `\bnethe`). Previous one, inherited from [xintexpr.sty 1.09n](#), assumed that the `\.=<digits>` dummy control sequence encapsulating the computation result had `\relax` meaning. But removing this assumption was only a matter of letting `\BNE_protect` protect two, not one, tokens. This will be backported to next version of [xintexpr](#), naturally (done with [xintexpr.sty 1.1](#)).

8 Changes

- 1.1 (2014/09/21) First release. This is down-scaled from the (development version of) `xintexpr`. Motivation came the previous day from a chat with JOSEPH WRIGHT over big int status in ~~TeX~~^{LaTeX}3. The `\bnumexpr ... \relax` parser can be used on top of big int macros of one's choice. Functionalities limited to the basic operations. I leave the power operator `^` as an option.

9 Package `bnumexpr` implementation

Contents

9.1 Package identification and catcode setup	11
9.2 Some helper macros and constants from xint	11
9.3 <code>\bnumexprsetup</code>	12
9.4 Package options	12
9.5 <code>\bnumexpr</code> , <code>\thebnumexpr</code> , <code>\bnethe</code> , <code>\bnumeval</code>	13
9.6 <code>\BNE_getnext</code>	13
9.7 Parsing an integer	14
9.8 <code>\BNE_getop</code>	16
9.9 Until macros for global expression and parenthesized sub-ones	17
9.10 The arithmetic operators.	17
9.11 <code>!</code> as postfix factorial operator	18
9.12 The minus as prefix operator of variable precedence level	19
9.13 The comma may separate expressions.	19
9.14 Cleanup	20

Comments are sparse. Error handling by the parser is kept to a minimum; if something goes wrong, the offensive token gets discarded, and some undefined control sequence attempts to trigger writing to the log of some sort of informative message. It is recommended to set `\errorcontextlines` to at least 2 for more meaningful context.

9.1 Package identification and catcode setup

`v1.2c` forgot to identify itself as such `:(`. Fixed (of course) at `v1.2d`.

```

1 \NeedsTeXFormat{LaTeX2e}%
2 \ProvidesPackage{bnumexpr}[2019/01/07 v1.2d Expressions with big integers (JFB)]%
3 \edef\BNErestorecatcodes {\catcode`\noexpand!\the\catcode`\!
4           \catcode`\noexpand?\the\catcode`\?
5           \catcode`\noexpand_\the\catcode`\_
6           \catcode`\noexpand:\the\catcode`\:
7           \catcode`\noexpand(\the\catcode`\(
8           \catcode`\noexpand)\the\catcode`\)
9           \catcode`\noexpand*\the\catcode`\*
10          \catcode`\noexpand\,\the\catcode`\,\relax}%
11 \catcode`\! 11
12 \catcode`\? 11
13 \catcode`\_ 11
14 \catcode`\: 11
15 \catcode`\, 12
16 \catcode`\* 12
17 \catcode`\( 12

```

9.2 Some helper macros and constants from xint

These macros from xint should not change, hence overwriting them here should not be cause for alarm. I opted against renaming everything with `\BNE_` prefix rather than `\xi`

`nt_`. The `\xint_dothis/\xint_orthat` thing is a new style I have adopted for expandably forking. The least probable branches should be specified first, for better efficiency. See examples of uses in the present code.

```

18 \chardef\xint_c_      0
19 \chardef\xint_c_i     1
20 \chardef\xint_c_ii    2
21 \chardef\xint_c_vi    6
22 \chardef\xint_c_vii   7
23 \chardef\xint_c_viii  8
24 \chardef\xint_c_ix    9
25 \chardef\xint_c_x     10
26 \long\def\xint_gobble_i    #1{}%
27 \long\def\xint_gobble_iii  #1#2#3{}%
28 \long\def\xint_firstofone  #1{#1}%
29 \long\def\xint_firstoftwo  #1#2{#1}%
30 \long\def\xint_secondoftwo #1#2{#2}%
31 \long\def\xint_firstofthree #1#2#3{#1}%
32 \long\def\xint_secondofthree #1#2#3{#2}%
33 \long\def\xint_thirdofthree #1#2#3{#3}%
34 \def\xint_gob_til_!        #1!{}% this ! has catcode 11
35 \long\def\xint_UDsignfork  #1-#2#3\krof {#2}%
36 \long\def\xint_afterfi     #1#2\fi {\fi #1}%
37 \long\def\xint_dothis      #1#2\xint_orthat #3{\fi #1}%
38 \let\xint_orthat           \xint_firstofone
39 \long\def\xint_zapspaces   #1 #2{#1#2\xint_zapspaces }%
```

9.3 `\bnumexprsetup`

New with **v1.2a**. Replaces removed `\bnumexprUsesbigintcalc` etc...

```

40 \catcode\! 3
41 \def\bnumexprsetup #1{\BNE_parsekeys #1,=!,}%
42 \def\BNE_parsekeys #1=#2#3,{\ifx!#2\expandafter\BNE_parsedone\fi
43   \expandafter
44 \let\csname BNE_Op_\xint_zapspaces #1 \xint_gobble_i\endcsname=#2\BNE_parsekeys
45 }%
46 \catcode\! 11
47 \def\BNE_parsedone #1\BNE_parsekeys {}%
```

9.4 Package options

v1.2c replaces former key `divtrunc` by `div`. **v1.2d** requires (by default) `xintcore 1.3d`.

```

48 \def\BNE_tmpa {0}%
49 \DeclareOption {custom}{\def\BNE_tmpa {1}}%
50 \ProcessOptions\relax
51 \if0\BNE_tmpa % Default is to load xintcore.sty
52   \RequirePackage{xintcore}[2019/01/06]% xintcore 1.3d
53   \bnumexprsetup{add=\xintiiAdd, sub=\xintiiSub, mul=\xintiiMul,
54     divround=\xintiiDivRound, div=\xintiiDivFloor,
55     mod=\xintiiMod, pow=\xintiiPow, fac=\xintiiFac}%
56 \fi
```

9.5 `\bnumexpr`, `\thebnumexpr`, `\bnethe`, `\bnumeval`

In the full `\xintexpr`, the final unlocking may involve post-treatment of the comma separated values, hence there are `_print` macros to handle the possibly comma separated values. Here we may just identify `_print` with `_unlock`.

With **v1.2a** the gathering of numbers happens directly inside `\csname ... \endcsname`. There is no more a ```locking''` macro.

Attention **v1.2d** gives new meaning to `\bnumeval`, the former meaning is now named `\bnumexpr`. The leading space in `\BNE_wrap` is removed as it served to nothing.

```

57 \def\bnumexpr {\romannumeral0\bnumexpr }%
58 \def\bnumexpr{\expandafter\BNE_wrap\romannumeral0\BNE_eval }%
59 \def\BNE_eval {\expandafter\BNE_until_end_a\romannumeral-`0\BNE_getnext }%
60 \def\BNE_wrap {\!\BNE_usethe\BNE_protect\BNE_unlock }%
61 \protected\def\BNE_usethe\BNE_protect {\BNE:missing_bnethe!}%
62 \def\BNE_protect\BNE_unlock
63   {\noexpand\BNE_protect\noexpand\BNE_unlock\noexpand }%
64 \let\BNE_done\space
65 \def\thebnumexpr
66   {\romannumeral-`0\expandafter\BNE_unlock\romannumeral0\BNE_eval }%
67 \def\bnethe #1{\romannumeral-`0\expandafter\xint_gobble_iii\romannumeral-`0#1}%
68 \def\bnumeval#1%
69   {\romannumeral-`0\expandafter\BNE_unlock\romannumeral0\BNE_eval#1\relax}%
70 \def\BNE_unlock {\expandafter\BNE_unlock_a\string }%
71 \def\BNE_unlock_a #1.={}%
```

9.6 `\BNE_getnext`

The `getnext` scans forward to find a number: after expansion of what comes next, an opening parenthesis signals a parenthesized sub-expression, a `!` with catcode 11 signals there was there a sub `\bnumexpr ... \relax` (now evaluated), a minus sign is treated as a prefix operator inheriting its precedence level from the previous operator, a plus sign is swallowed, a `\count` or `\dimen` will get fetched to `\number` (in case of a count variable, this provides a full locked number but `\count 0 1` for example is like `1231` if `\count 0`'s value is `123`); a digit triggers the number scanner. With **v1.2a** the gathering of digits happens directly inside `\csname . = ... \endcsname`. Leading zeroes are trimmed directly. The flow then proceeds with `\BNE_getop` which looks for the next operator or possibly the end of the expression. Note: `\bnumexpr \relax` is illegal.

Extended in **v1.2a** to recognize `\ht`, etc...

```

72 \def\BNE_getnext #1%
73 {%
74   \expandafter\BNE_getnext_a\romannumeral-`0#1%
75 }%
76 \def\BNE_getnext_a #1%
77 {%
78   \xint_gob_til_! #1\BNE_gn_foundexpr !% this ! has catcode 11
79   \ifcat\relax#1% \count or \numexpr etc... token or count, dimen, skip cs
80     \expandafter\BNE_gn_countetc
81   \else
82     \expandafter\expandafter\expandafter\BNE_gn_fork\expandafter\string
83   \fi
```

```

84     #1%
85 }%
86 \def\BNE_gn_foundexpr !#1\fi !{\expandafter\BNE_getop\xint_gobble_iii }%
87 \def\BNE_gn_countetc #1%
88 {%
89     \ifx\count#1\else\ifx\dimen#1\else\ifx\numexpr#1\else\ifx\dimexpr#1\else
90     \ifx\skip#1\else\ifx\glueexpr#1\else\ifx\fontdimen#1\else\ifx\ht#1\else
91     \ifx\dp#1\else\ifx\wd#1\else\ifx\fontcharht#1\else\ifx\fontcharwd#1\else
92     \ifx\fontchardp#1\else\ifx\fontcharic#1\else
93     \BNE_gn_unpackvar
94     \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
95     \expandafter\BNE_getnext\number #1%
96 }%
97 \def\BNE_gn_unpackvar\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
98     \expandafter\BNE_getnext\number #1%
99 {%
100 \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
101 \expandafter\BNE_getop\csname .=\number#1\endcsname
102 }%

```

This is quite simplified here compared to `\xintexpr`, for various reasons: we have dropped the `\xintNewExpr` thing, and we can treat the `(` directly as we don't have to get back to check if we are in an `\xintexpr`, `\xintfloatexpr`, etc..

```

103 \def\BNE_gn_fork #1{%
104     \if#1+\xint_dothis \BNE_getnext\fi
105     \if#1-\xint_dothis -\fi
106     \if#1(\xint_dothis \BNE_oparen \fi
107     \xint_orthat      {\BNE_scan_number #1}%
108 }%

```

9.7 Parsing an integer

We gather a string of digits, plus and minus prefixes have already been swallowed. There might be some leading string of zeros which will have to be removed. In the full `\xintexpr` the situation is more involved as it has to recognize and accept decimal numbers, numbers in scientific notation, also hexadecimal numbers, function names, variable names. . .

```

109 \def\BNE_scan_number #1% this #1 has necessarily here catcode 12
110 {%
111     \ifnum \xint_c_ix<1#1 \else\expandafter\BNE_notadigit\fi
112     \BNE_scan_nbr #1%
113 }%
114 \def\BNE_notadigit\BNE_scan_nbr #1{\BNE:not_a_digit!\BNE_getnext }%

```

Scanning for a number. We gather it directly inside `csname`. earlier version did a chain of `romannumeral`. No limit on number of digits anymore from the maximal expansion depth. We only have to be careful about leading zeros.

If we hit against some catcode eleven `!`, this means there was a sub `\bnumexpr ..\relax`. We then apply tacit multiplication.

```

115 \def\BNE_scan_nbr #1%
116 {% the #1 here is a catcode 12 digit
117     \if#10\expandafter\BNE_scan_nbr_gobzeroes

```

Contents

```

118     \else
119         \expandafter\BNE_scan_nbr_start
120     \fi #1%
121 }%
122 \def\BNE_scan_nbr_start #1#2%
123 {%
124     \expandafter\BNE_getop\csname.=#1%
125     \expandafter\BNE_scanint_b\romannumeral-`0#2%
126 }%
127 \def\BNE_scan_nbr_gobzeroes #1%
128 {%
129     \expandafter\BNE_getop\csname.=#1%
130     \expandafter\BNE_gobz_scanint_b\romannumeral-`0#1%
131 }%
132 \def\BNE_scanint_b #1%
133 {%
134     \ifcat \relax #1\expandafter\BNE_scanint_endbycs\expandafter #1\fi
135     \ifnum\xint_c_ix<1\string#1 \else\expandafter\BNE_scanint_c\fi
136     \string#1\BNE_scanint_d
137 }%
138 \def\BNE_scanint_endbycs#1#2\BNE_scanint_d{\endcsname #1}%
139 \def\BNE_scanint_c\string #1\BNE_scanint_d
140 {%
141     \if _#1\xint_dothis{\expandafter\BNE_scanint_d\xint_gobble_i}\fi
142     \ifcat a#1\xint_dothis{\endcsname*}\fi % tacit multiplication
143     \xint_orthat {\expandafter\endcsname \string}#1%
144 }%
145 \def\BNE_scanint_d #1%
146 {%
147     \expandafter\BNE_scanint_b\romannumeral-`0#1%
148 }%
149 \def\BNE_gobz_scanint_b #1%
150 {%
151     \ifcat \relax #1\expandafter\BNE_gobz_scanint_endbycs\expandafter #1\fi
152     \ifnum\xint_c_x<1\string#1 \else\expandafter\BNE_gobz_scanint_c\fi
153     \string#1\BNE_scanint_d
154 }%
155 \def\BNE_gobz_scanint_endbycs#1#2\BNE_scanint_d{0\endcsname #1}%
156 \def\BNE_gobz_scanint_c\string #1\BNE_scanint_d
157 {%
158     \if _#1\xint_dothis\BNE_gobz_scanint_d\fi
159     \ifcat a#1\xint_dothis{0\endcsname*#1}\fi % tacit multiplication
160     \if 0#1\xint_dothis\BNE_gobz_scanint_d\fi
161     \xint_orthat {0\expandafter\endcsname \string#1}%
162 }%
163 \def\BNE_gobz_scanint_d #1%
164 {%
165     \expandafter\BNE_gobz_scanint_b\romannumeral-`0#1%
166 }%

```

9.8 \BNE_getop

This finds the next infix operator or closing parenthesis or expression end. It then leaves in the token flow <precedence> <operator> <locked number>. The <precedence> stops expansion and ultimately gives back control to a `\BNE_until_<op>` command. The code here is derived from more involved context where the actual macro associated to the operator may vary, depending if we are in `\xintexpr`, `\xintfloatexpr` or `\xintiexpr`. Here things are simpler but I have kept the general scheme, thus the actual macro to be used for the <operator> is not decided immediately.

v1.2a adds a technique for allowing two-letters operators, for `//`, `/:` and `**`.

```

167 \def\BNE_getop #1#2% this #1 is the current locked computed value
168 {%
169   \expandafter\BNE_getop_a\expandafter #1\romannumeral-`0#2%
170}%
171 \catcode`* 11
172 \def\BNE_getop_a #1#2%
173 {% if a control sequence is found, must be \relax, or possibly register or
174 % variable if tacit multiplication is allowed
175   \ifx \relax #2\xint_dothis\xint_firstofthree\fi
176   % tacit multiplications:
177   \ifcat \relax #2\xint_dothis\xint_secondofthree\fi
178   \if (#2\xint_dothis \xint_secondofthree\fi
179   \ifx !#2\xint_dothis \xint_secondofthree\fi
180   \xint_orthat \xint_thirdofthree
181   {\BNE_foundend #1}%
182   {\BNE_precedence_* *#1#2}% tacit multiplication
183   {\BNE_scanop_a #2#1}%
184}%
185 \catcode`* 12
186 \def\BNE_foundend {\xint_c_ \relax}% \relax is only a place-holder here.
187 \def\BNE_scanop_a #1#2#3%
188   {\expandafter\BNE_scanop_b\expandafter #1\expandafter #2\romannumeral-`0#3}%
189 \def\BNE_scanop_b #1#2#3%
190 {%
191   \ifcat#3\relax\xint_dothis{\BNE_foundop #1#2#3}\fi
192   \ifcsname BNE_itself_#1#3\endcsname
193     \xint_dothis
194       {\expandafter\BNE_foundop\csname BNE_itself_#1#3\endcsname #2}\fi
195   \xint_orthat {\BNE_foundop #1#2#3}%
196}%
197 \def\BNE_foundop #1%
198 {%
199   \ifcsname BNE_precedence_#1\endcsname
200     \csname BNE_precedence_#1\endcsname\expandafter\endcsname
201     \expandafter #1%
202   \else
203     \BNE_notanoperator {#1}\expandafter\BNE_getop
204   \fi
205}%
206 \def\BNE_notanoperator #1{\BNE:not_an_operator! \xint_gobble_i {#1}}%
```


9.9 Until macros for global expression and parenthesized sub-ones

The minus sign as prefix is treated here.

```

207 \catcode` ) 11
208 \def\BNE_tmpa #1{%
209   \def\BNE_until_end_a ##1%
210   {%
211     \xint_UDsignfork
212       ##1{\expandafter\BNE_until_end_a\romannumeral-`0#1}%
213       -{\BNE_until_end_b ##1}%
214     \krof
215   }%
216 }\expandafter\BNE_tmpa\csname BNE_op_-vi\endcsname
217 \def\BNE_until_end_b #1#2%
218   {%
219     \ifcase #1\expandafter\BNE_done
220     \or
221     \xint_afterfi{\BNE:extra_)?\expandafter
222                 \BNE_until_end_a\romannumeral-`0\BNE_getop }%
223     \else
224     \xint_afterfi{\expandafter\BNE_until_end_a
225                 \romannumeral-`0\csname BNE_op_#2\endcsname }%
226     \fi
227   }%
228 \catcode`( 11
229 \def\BNE_op_( {\expandafter\BNE_until_)_a\romannumeral-`0\BNE_getnext }%
230 \let\BNE_oparen\BNE_op_(
231 \catcode`( 12
232 \def\BNE_tmpa #1{%
233   \def\BNE_until_)_a ##1{\xint_UDsignfork
234     ##1{\expandafter \BNE_until_)_a\romannumeral-`0#1}%
235     -{\BNE_until_)_b ##1}%
236   \krof }%
237 }\expandafter\BNE_tmpa\csname BNE_op_-vi\endcsname
238 \def \BNE_until_)_b #1#2%
239   {%
240     \ifcase #1\expandafter \BNE_missing_)? % missing ) ?
241     \or\expandafter \BNE_getop % found closing )
242     \else \xint_afterfi
243     {\expandafter \BNE_until_)_a\romannumeral-`0\csname BNE_op_#2\endcsname }%
244     \fi
245   }%
246 \def\BNE_missing_)? {\BNE:missing_)_inserted \xint_c_ \BNE_done }%
247 \let\BNE_precedence_ \xint_c_i
248 \let\BNE_op_ \BNE_getop
249 \catcode`) 12

```

9.10 The arithmetic operators.

This is where the infix operators are mapped to actual macros. These macros must ``f-expand'' their arguments, and know how to handle then big integers having no leading zeros and at most a minus sign.

v1.2a adds `//` for truncated division, `/:` for modulo operations and `**` for powers (synonym to `^`).

v1.2c has `//` and `/:` per default associated to floored division.

```

250 \def\BNE_tmpc #1#2#3#4#5#6#7%
251 {%
252   \def #1##1% \BNE_op_<op>
253   {% keep value, get next number and operator, then do until
254     \expandafter #2\expandafter ##1\romannumeral-`0\expandafter\BNE_getnext }%
255   \def #2##1##2% \BNE_until_<op>_a
256   {\xint_UDsignfork
257     ##2{\expandafter #2\expandafter ##1\romannumeral-`0#4}%
258     -{#3##1##2}%
259   \krof }%
260   \def #3##1##2##3##4% \BNE_until_<op>_b
261   {% either execute next operation now, or first do next (possibly unary)
262     \ifnum ##2>#5%
263       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral-`0%
264         \csname BNE_op_##3\endcsname {##4}}%
265       \else \xint_afterfi {\expandafter ##2\expandafter ##3%
266         \csname .=#6{\BNE_unlock ##1}{\BNE_unlock ##4}\endcsname }%
267       \fi }%
268   \let #7#5%
269 }%
270 \def\BNE_tmpb #1#2#3%
271 {%
272   \expandafter\BNE_tmpc
273   \csname BNE_op_#1\expandafter\endcsname
274   \csname BNE_until_#1_a\expandafter\endcsname
275   \csname BNE_until_#1_b\expandafter\endcsname
276   \csname BNE_op_-#2\expandafter\endcsname
277   \csname xint_c_#2\expandafter\endcsname
278   \csname #3\expandafter\endcsname
279   \csname BNE_precedence_#1\endcsname
280 }%
281 \BNE_tmpb +{vi}{BNE_Op_add}%
282 \BNE_tmpb -{vi}{BNE_Op_sub}%
283 \BNE_tmpb *{vii}{BNE_Op_mul}%
284 \BNE_tmpb /{vii}{BNE_Op_divround}%
285 \BNE_tmpb ^{viii}{BNE_Op_pow}%
286 \expandafter\def\csname BNE_itself_**\endcsname {^}% shortcut for alias
287 \expandafter\def\csname BNE_itself_//\endcsname {//}%
288 \expandafter\def\csname BNE_itself_/:\endcsname {/:}%
289 \BNE_tmpb {//}{vii}{BNE_Op_div}%
290 \BNE_tmpb {/:}{vii}{BNE_Op_mod}%

```

9.11 ! as postfix factorial operator

New with **v1.2a**.

```

291 \let\BNE_precedence_! \xint_c_x
292 \def\BNE_op_! #1%
293   {\expandafter\BNE_getop\csname .=\BNE_Op_fac{\BNE_unlock #1}\endcsname }%

```

9.12 The minus as prefix operator of variable precedence level

It inherits the level of precedence of the previous operator.

```

294 \def\BNE_tmpa #1%
295 {%
296 \expandafter\BNE_tmpp
297   \csname BNE_op_-#1\expandafter\endcsname
298   \csname BNE_until_-#1_a\expandafter\endcsname
299   \csname BNE_until_-#1_b\expandafter\endcsname
300   \csname xint_c_#1\endcsname
301 }%
302 \def\BNE_tmpp #1#2#3#4%
303 {%
304   \def #1% \BNE_op_-<level>
305   {% get next number+operator then switch to _until macro
306     \expandafter #2\romannumeral-`0\BNE_getnext
307   }%
308   \def #2##1% \BNE_until_-<level>_a
309   {\xint_UDsignfork
310     ##1{\expandafter #2\romannumeral-`0#1}%
311     -{##1}%
312   \krof }%
313   \def #3##1##2##3% \BNE_until_-<level>_b
314   {%
315     \ifnum ##1>#4%
316       \xint_afterfi {\expandafter #2\romannumeral-`0%
317         \csname BNE_op_##2\endcsname {##3}}%
318     \else
319       \xint_afterfi {\expandafter ##1\expandafter ##2%
320         \csname .=\expandafter\BNE_Op_opp
321           \romannumeral-`0\BNE_unlock ##3\endcsname }%
322     \fi
323   }%
324 }%
325 \BNE_tmpa {vi}%
326 \BNE_tmpa {vii}%
327 \BNE_tmpa {viii}%
328 \def\BNE_Op_opp #1{\if-#1\else\if0#10\else-#1\fi\fi }%

```

9.13 The comma may separate expressions.

It suffices to treat the comma as a binary operator of precedence [ii](#). We insert a space after the comma. The current code in `\xintexpr` does not do it at this stage, but only later during the final unlocking, as there is anyhow need for some processing for final formatting and was considered to be as well the opportunity to insert the space. Here, let's do it immediately. These spaces are not an issue when `\bnumexpr` is identified as a sub-expression in `\xintexpr`, for example in: `\xinttheiexpr lcm(\bnumexpr 175-12,12+23+34,56*31\relax)\relax` (this example requires package `xintgcd`).

```

329 \catcode`, 11
330 \def\BNE_op_, #1%
331 {%

```

```

332   \expandafter \BNE_until_,_a\expandafter #1\romannumeral-`0\BNE_getnext
333 }%
334 \def\BNE_tmpa #1{% #1 = \BNE_op_-vi
335   \def\BNE_until_,_a ##1##2%
336   {%
337     \xint_UDsignfork
338       ##2{\expandafter \BNE_until_,_a\expandafter ##1\romannumeral-`0#1}%
339       -{\BNE_until_,_b ##1##2}%
340     \krof }%
341 }\expandafter\BNE_tmpa\csname BNE_op_-vi\endcsname
342 \def\BNE_until_,_b #1#2#3#4%
343 {%
344   \ifnum #2>\xint_c_ii
345     \xint_afterfi {\expandafter \BNE_until_,_a
346                   \expandafter #1\romannumeral-`0%
347                   \csname BNE_op_-#3\endcsname {#4}}}%
348   \else
349     \xint_afterfi {\expandafter #2\expandafter #3%
350                   \csname .=\BNE_unlock #1, \BNE_unlock #4\endcsname }%
351   \fi
352 }%
353 \let \BNE_precedence_, \xint_c_ii

```

9.14 Cleanup

```

354 \let\BNE_tmpa\relax \let\BNE_tmpb\relax \let\BNE_tmpc\relax
355 \BNErestorecatcodes

```