The bnumexpr package

Jean-François Burnol
jfbu (at) free (dot) fr

Package version: 1.4a (2021/05/13); documentation date: 2021/05/13.
From source file bnumexpr.dtx. Time-stamp: <13-05-2021 12:37:08 CEST>.

Contents

1 \bnumeval (\thebnumexpr), \evaltohex

LaTeX Package bnumexpr provides \thebnumexpr⟨*expression*⟩\relax: it 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 is ignored and can thus be used to separate in the source blocks of digits for better readability of long numbers,
- also the underscore _ may be used as visual digit separator,
- comma separated expressions are allowed.

There is also a more core-level \bnumexpr ...\relax construct[1], which expands to a self-contained unit, rather than to explicit digit tokens (and commas). See section 3 for some related information.

There is also the alternative interface \bnumeval{⟨*expression*⟩}, where the expression is fetched as braced argument.

And there is \evaltohex{⟨*expression*⟩} which does the same as \bnumeval but with a conversion to hexadecimal notation of the (possibly comma separated) output. Hexadecimal input uses the " prefix.

This package parser is a scaled-down variant of \xintiiexpr from package xintexpr, dropping support for nested structures, functions, variables, booleans, etc..., but incorporating by default support for hexadecimal input as xintbinhex will be automatically loaded.

The ε-TEX extensions are required, this is the default on all modern installations for latex|pdflatex and also for xelatex|lualatex.

Further, at 1.4 (2021/05/12) the \expanded primitive is required. It is available in all engines since TEXLive 2019.

By default the arithmetic operations are executed via the xintcore macros, but via option custom and usage of \bnumexprsetup it is possible to replace them by expandable macros of a custom origin.

2 Examples

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

```
1113492904235346927
```

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

```
-2787514672889976289932
```

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

```
-4006240736596543944035189
```

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

```
30
```

```
\bnumeval {13^50//12^50, 13^50/:12^50}
```

```
54, 650556287901099025745221048683760161794567947140168553
```

```
\bnumeval {13^50/12^50, 12^50}
```

```
55, 91004381500021497733275852753425663249271526032565862 4
```

[1] Since 1.4, one can use \bnumexpr ...\relax directly in typesetting context, it is not mandatory to prefix it with \bnethe or to use \thebnumexpr.

```
\bnumeval {(1^10+2^10+3^10+4^10+5^10+6^10+7^10+8^10+9^10)^3}
                    11868507546269898170062082812 5
\bnumeval {100!/36^100}
                                    219
\bnumeval {"10*"100*"1000*"A0000, 16^(1+2+3+4)*10}
                    10995116277760,10995116277760
\evaltohex {"7FFFFFFF+1, "400^3, "ABCDEF*"FEDCBA}
                    80000000,40000000,AB0A74EF03A6
```

3 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. contrarily to \numexpr, the \bnumexpr parser stops only after having found (and swallowed) a mandatory ending \relax token (it can arise from expansion),
2. in particular note that 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

   \thebnumexpr 3 5+79\relax expands (in two steps) to 114
3. with \edef \myVar {\thebnumexpr 1+2\relax }, the computation is of course done at time of the \edef. But one is also allowed to do \edef \myVar {\bnumexpr 1+2\relax } which prepares \myVar as a macro which can be inserted in other bnumexpr expressions and behave there as a self-contained pre-computed unit triggering tacit multiplication, or be typeset directly if inserted in the typesetting stream.[2] There is no analog with \numexpr as \edef \myVar {\numexpr 1+2\relax } does not pre-compute anything and furthermore \the \numexpr 2\myVar \relax in typesetting flow then triggers the You can't use `\numexpr ' in horizontal mode error.
4. expressions may be comma separated. On input, spaces are ignored, and on output the values are comma separated with a space after each comma,
5. \bnumexpr -(1+1)\relax is legal contrarily to \numexpr -(1+1)\relax which raises an error,

[2] Prior to 1.4, one would have had to use \bnethe \myVar for typesetting, or \bnumeval {\myVar }.

6. \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,

7. more generally, tacit multiplication applies in front of parenthesized sub-expressions, or sub \bnumexpr ...\relax (or \numexpr ...\relax), or also after parentheses in front of numbers,

8. the underscore _ is accepted within the digits composing a number and is silently ignored by \bnumexpr.

As hinted above \bnumexpr ...\relax differs from \thebnumexpr ...\relax as the latter expands to explicit digit tokens, but the former expands to a private self-contained format which can serve as sub-unit in other expressions, or be used inside \edef. Since 1.4 the former idiom can also be inserted directly inside the typesetting stream, or be written out to an external file where it will expand to some control sequences, braces, and character tokens, all with their standard catcodes.

One can use \numexpr ...\relax as a sub-unit in \bnumexpr ...\relax but the reverse does not apply: it would either cause an error or an anticipated end to the \numexpr which will think having hit a \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 \thebnumexpr 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 control sequence. When using \bnumeval {...} syntax as in

```
\ifnum\bnumeval{...}
...
\fi
```

the end of line will insert a terminating space token. Again, here \bnumeval {...} must produce an integer acceptable to TEX, i.e. at most 2147483647 in absolute value.

4 Printing big numbers

LATEX 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
```

0586316668729948085589013238296699445909974245040870737599188236277271887
3251977950595099527612087497546249704360141827809464649629105639388743788
6487337119181045825783647849977012476632889835955735432513185323958463075
5574091142624174743493475534286465766116677973966688202912073791438537195
8824980812686783837455973174613608537953452422158659320192809087829730843
1392844403281231558611036976801357304216168747609675871348312025478589320
7671691324484262361314125087802080002616831510273418279777047846358681701
6436502415369139828126481021309276124489635992870511496497541990934222156
6832572080821333186116811553615836546984046708975602900950537616475847728
4218896796462449451607653534081989013854424879849599533191017233555660213
9450399736280750137837615307127761926849034352625200015888535147331611702
1039681759215109077880193931781141945452572238655414610628921879602238389
7147608850627686296714667469756291123408243920816015378088989396451826324
3671616762179168909779911903754031274622289988005195444414282012187361745
9926429565817466283029555702990243241531816172104658320367869061172601587
8352075151628422554026517048330422614397428693306169089796848259012545832
7168226458066526769958652682272807075781391858178889652208164348344825993
2660433676601769996128318607883861502794659551311565520360939881806121385
5860030143569452722420634463179746059468257310379008402443243846565724501
4402821885252470935190620929023136493273497565513958720559654228749774011
4133469627154228458623773875382304838656889764619273838149001407673104466
4025989949022222176590433990188601856652648506179970235619389701786004081
1889729918311021171229845901641921068884387121855646124960798722908519296
8193723886426148396573822911231250241866493531439701374285319266498753372
1894069428143411852015801412334482801505139969429015348307764456909907315
2433278288269864602789864321139083506217095002597389863554277196742822248
7575867657523442202075736305694988250879689281627538488633969099598262809
5612145099487170124451646126037902930912088908694202851064018215439945715
6805941872748998094254742173582401063677404595741785160829230135358081840
0969963725242305608559037006242712434169090041536901059339838357779394109
7002775347200000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000
000

5 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). One can input hexadecimal numbers as in TEX syntax for number assignments, i.e. using a `"` prefix and only uppercase letters `ABCDEF`.

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. Tacit multiplication applies in front of parentheses, and after them, and it has an elevated precedence compared to multiplication explicitly induced by *.

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,[3] hence:

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

4096, 4096, 241785163922925834941235 2

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

6 Options

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

7 \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) 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,
               opp=\xintiiOpp}%
```

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.

The macros serving as custom user replacements must be *f*-expandable.[4] (except for the computation of factorials, which only has to be *x*-expandable).

[3] But it has been announced at xintexpr 1.4 that probably in future power operator will become right-associative, and if this is done, this will be transferred to bnumexpr also.

[4] Prior to 1.4, only *x*-expandability was required. The author could relax again the constraint, if asked to do so. Or perhaps simply add an option for it.

They will by default receive arguments composed of explicit digit tokens, with no leading zeros, with at most one leading minus sign and no plus sign.

The format of these arguments may depend on what the opp=\foo replacement macro does. If the custom \foo inserts a + when taking the opposite of a negative number, then the custom macros for arithmetic (and the \foo macro itself) must be able to handle arguments starting optionally with such a +.

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.

At 1.4 release there is no option to customize how the hexadecimal input and output is handled, it goes necessarily via the xintbinhex macros.

8 Readme

```
| Source:  bnumexpr.dtx
| Version: v1.4a, 2021/05/13 (doc: 2021/05/13)
| Author:  Jean-Francois Burnol
| Info:    Expressions with big integers
| License: LPPL 1.3c

bnumexpr usage
==============

The LaTeX package `bnumexpr` allows expandable computations with
integers and the four infix operators `+`, `-`, `*`, `/` using the
expression syntax familiar from the `\numexpr` e-TeX parser, with
these extensions:

- arbitrarily big integers,
- floored division `//`,
- associated modulo `/:`,
- power operators `^` and `**`,
- factorial post-fix operator `!`,
- comma separated expressions,
- the space character as well as the underscore may serve
  to separate groups of digits,
- optional conversion of output to hexadecimal.

The expression parser is a scaled-down variant from the
`\xintiiexpr...\relax`
parser from package [xintexpr](http://ctan.org/pkg/xintexpr).

To support hexadecimal input and output, the package
[xintbinhex](http://ctan.org/pkg/xint) is loaded automatically.

The package loads by default [xintcore](http://ctan.org/pkg/xint)
but the option _custom_ together with macro `\bnumexprsetup` allow to map
the syntax elements to macros from an alternative big integer
expandable engine of the user own choosing,
and then [xintcore](http://ctan.org/pkg/xint) is not loaded.

Installation
============

Use your installation manager to install or update `bnumexpr`.

Else, obtain `bnumexpr.dtx`, from CTAN:
```

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

Run `"etex bnumexpr.dtx"` to extract these files:

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

`README.md`

`bnumexprchanges.tex`
  : change history.

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

  :  - with latex+dvipdfmx: `"latex bnumexpr.tex"` (thrice) then
       `"dvipdfmx bnumexpr.dvi"`.

  :  - with pdflatex: `"pdflatex bnumexpr.tex"` (thrice).

  : In both cases files `README.md` and `bnumexprchanges.tex` must
    be located in the same repertory as `bnumexpr.tex` and `bnumexpr.dtx`.

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.pdf   --> TDS:doc/latex/bnumexpr/
               README.me   --> TDS:doc/latex/bnumexpr/

License
=======

Copyright (C) 2014-2021 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.tex, bnumexprchanges.tex,
    and README.md
```

9 Changes

1.4a (2021/05/13)
- fix undefined control sequences errors encountered by the parser in case of either extra or missing closing parenthesis (due to a problem in technology transfer at 1.4 from upstream xintexpr).
- fix \BNE_Op_opp must now be f-expandable (also caused as a collateral to the technology transfer).
- fix user documentation regarding the constraints applying to the user replacement macros for the core algebra, as they have changed at 1.4.

1.4 (2021/05/12)
- technology transfer from xintexpr 1.4 of 2020/01/31. The \expanded primitive is now required (TeXLive 2019).
- addition to the syntax of the " prefix for hexadecimal input.
- addition of \evaltohex which is like \bnumeval with an extra conversion step to hexadecimal notation.

1.2e (2019/01/08) Fixes a documentation glitch (extra braces when mentioning \the \numexpr or \thebnumexpr).

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 1.2p.
- 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 \bnumexprusesbigintcalc.
- 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 LaTeX3 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).

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 LaTeX3. 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.

10 Package bnumexpr implementation

Contents

Comments are sparse. Actually at 1.4, there are simply no comments. I transferred from xintexpr its \expanded based infra-structure from its own 1.4 release of January 2020.

Error handling by the parser is kept to a minimum; if something goes wrong, the offensive token gets discarded, and it is not even always the case that some expandable error message is issued.

A few comments at 1.4a:

- It looked a bit costly and probably would have been mostly useless to end users to integrate in bnumexpr support for nested structures via square brackets [,], which is in xintexpr since its January 2020 1.4 release. But some underlying related architecture remains here; we could make some micro-gains probably but diverging from upstream code would make maintenance a nightmare.
- The \bnumexpr \relax syntax creating an empty ople is by itself now legal, and can be injected (comma separated) in an expression, keeping it invariant, however \bnumeval {} ends in a File ended while scanning use of \BNE_print_c error because \BNEprint makes the tacit requirement that the 1D ople to output has at least one item.
- Formerly, the \csname ...\endcsname encapsulation technique had the after-effect to allow the macros supporting the infix operators to be only x-expandable. At 1.4, I could have still allowed only x-expandable macros, but, keeping in sync with upstream, I have used only a \romannumeral trigger and did not insert an \expanded, so now the support macros must be f-expandable. The 1.4a release fixes the related user documentation of \bnumexprsetup which was not updated at 1.4. The support macro for the factorial however needs only be x-expandable.

- Also, I simply do not understand why the legacy user documentation said that the support macros were supposed to f-expand their arguments, as they are used only with arguments being explicit digit tokens (and optional minus sign).
- I hesitated a bit about making the decimal to hexadecimal and hexadecimal to decimal support macros customizable, but dropped the idea. Loading xintbinhex unconditionally has also the advantage to not have to define some xintkernel provided helper macros; and it still does not load xintcore, so is keeping dependencies somewhat low and the custom option significant.
- I had fleetingly consider a parser \hexexpr where input is hexadecimal with no " prefix, but implementing this basically means duplicating with new names a large chunk of the parser code to have parser specific "getnext" macros for example. Not worth it.

10.1 Package identification and catcode setup

```
\NeedsTeXFormat{LaTeX2e}%
\ProvidesPackage{bnumexpr}[2021/05/13 v1.4a Expressions with big integers (JFB)]%
```

10.2 Load unconditionally xintbinhex

Newly done at 1.4. Formerly, bnumexpr had no dependency if loaded with option custom. But for 1.4 release I have decided to add unconditional support for hexadecimal notation.

Let's require the most recent xint date at time of writing. We should check for availability of \expanded but well.

```
\RequirePackage{xintbinhex}[2021/05/10]%
```

10.3 \bnumexprsetup

```
{\catcode`! 3 \catcode`_ 11 %
  \gdef\bnumexprsetup #1{\BNE_parsekeys #1,=!,}%
  \gdef\BNE_parsekeys #1=#2#3,%
  {%
    \ifx!#2\expandafter\BNE_parsedone\fi
    \expandafter
    \let\csname BNE_Op_\xint_zapspaces #1 \xint_gobble_i\endcsname%
    =#2\BNE_parsekeys
  }%
  \gdef\BNE_parsedone #1\BNE_parsekeys {}%
}%
```

10.4 Package options

The keys should have been Add, Sub, . . . , not add, sub, . . . , so internally macros \BNE_Op_Add etc. . . macro names would be used, but well, let's simply leave with this.

```
\def\BNEtmpa {0}%
\DeclareOption {custom}{\def\BNEtmpa {1}}%
\ProcessOptions\relax
\edef\BNErestorecatcodes{\XINTrestorecatcodes}%
```

```
\XINTsetcatcodes%
\if0\BNEtmpa\expandafter\xint_secondoftwo\fi
\xint_gobble_i{%
    \RequirePackage{xintcore}[2021/05/10]%
    \bnumexprsetup{add=\xintiiAdd, sub=\xintiiSub, mul=\xintiiMul,
                   divround=\xintiiDivRound, div=\xintiiDivFloor,
                   mod=\xintiiMod, pow=\xintiiPow, fac=\xintiiFac,
                   opp=\xintiiOpp}%
}%
```

Strangely those three are not defined in xintkernel.sty, but only in xint.sty

```
\long\def\xint_firstofthree  #1#2#3{#1}%
\long\def\xint_secondofthree #1#2#3{#2}%
\long\def\xint_thirdofthree  #1#2#3{#3}%
```

10.5 \bnumexpr, \thebnumexpr, \bnethe, \bnumeval

```
\def\XINTfstop {\noexpand\XINTfstop}%
\def\bnumexpr  {\romannumeral0\bnumexpro}%
\def\bnumexpro {\expandafter\BNE_wrap\romannumeral0\bnebareeval }%
\def\BNE_wrap  {\XINTfstop\BNEprint.}%
\def\bnumeval #1%
   {\expanded\expandafter\BNEprint\expandafter.\romannumeral0\bnebareeval#1\relax}%
\def\evaltohex #1%
   {\expanded\expandafter\BNEprinthex\expandafter.\romannumeral0\bnebareeval#1\relax}%
\def\thebnumexpr
   {\expanded\expandafter\BNEprint\expandafter.\romannumeral0\bnebareeval}%
\def\bnebareeval{\BNE_start}%
\def\bnethe#1{\expanded\expandafter\xint_gobble_i\romannumeral`&&@#1}%
\protected\def\BNEprint.#1{{\BNE_print#1.}}%
\def\BNE_print#1{#1\expandafter\BNE_print_a\string}%
\def\BNE_print_a#1{\unless\if#1.\expandafter\BNE_print_b\fi}%
\def\BNE_print_b
   {\expandafter\BNE_print_c\expandafter{\expandafter\xint_gobble_i\string}}%
\def\BNE_print_c#1{, #1\expandafter\BNE_print_a\string}%
\protected\def\BNEprinthex.#1{{\BNE_printhex#1.}}%
\def\BNE_printhex#1{\xintDecToHex{#1}\expandafter\BNE_printhex_a\string}%
\def\BNE_printhex_a#1{\unless\if#1.\expandafter\BNE_printhex_b\fi}%
\def\BNE_printhex_b
   {\expandafter\BNE_printhex_c\expandafter{\expandafter\xint_gobble_i\string}}%
\def\BNE_printhex_c#1{, \xintDecToHex{#1}\expandafter\BNE_printhex_a\string}%
```

10.6 \BNE_getnext

```
\def\BNE_getnext #1%
{%
    \expandafter\BNE_put_op_first\romannumeral`&&@%
    \expandafter\BNE_getnext_a\romannumeral`&&@#1%
}%
\def\BNE_put_op_first #1#2#3{\expandafter#2\expandafter#3\expandafter{#1}}%
\def\BNE_getnext_a #1%
{%
    \ifx\relax #1\xint_dothis\BNE_foundprematureend\fi
    \ifx\XINTfstop#1\xint_dothis\BNE_subexpr\fi
```

```
    \ifcat\relax#1\xint_dothis\BNE_countetc\fi
    \xint_orthat{}\BNE_getnextfork #1%
}%
\def\BNE_foundprematureend\BNE_getnextfork #1{{}\xint_c_\relax}%
\def\BNE_subexpr #1.#2%
{%
    \expanded{\unexpanded{{#2}}\expandafter}\romannumeral`&&@\BNE_getop
}%
\def\BNE_countetc\BNE_getnextfork#1%
{%
    \if0\ifx\count#11\fi
        \ifx\dimen#11\fi
        \ifx\numexpr#11\fi
        \ifx\dimexpr#11\fi
        \ifx\skip#11\fi
        \ifx\glueexpr#11\fi
        \ifx\fontdimen#11\fi
        \ifx\ht#11\fi
        \ifx\dp#11\fi
        \ifx\wd#11\fi
        \ifx\fontcharht#11\fi
        \ifx\fontcharwd#11\fi
        \ifx\fontchardp#11\fi
        \ifx\fontcharic#11\fi 0\expandafter\BNE_fetch_as_number\fi
   \expandafter\BNE_getnext_a\number #1%
}%
\def\BNE_fetch_as_number
    \expandafter\BNE_getnext_a\number #1%
{%
    \expanded{{{\number#1}}\expandafter}\romannumeral`&&@\BNE_getop
}%
```

In the case of hitting a `(`, previous release inserted directly a `\BNE_oparen`. But the expansion architecture imported from upstream `\xintiiexpr` has been refactored, and the `..._oparen` meaning and usage evolved. We stick with `{}\xint_c_ii ^v (` from upstream, which works (I am 15 months away from `xintexpr 1.4`).

```
\def\BNE_getnextfork #1{%
    \if#1+\xint_dothis \BNE_getnext_a \fi
    \if#1-\xint_dothis {{}{}-}\fi
    \if#1(\xint_dothis {{}\xint_c_ii^v (}\fi
    \xint_orthat {\BNE_scan_number #1}%
}%
```

10.7 Parsing an integer in decimal or hexadecimal notation

```
\def\BNE_scan_number #1%
{%
    \if "#1\xint_dothis \BNE_scanhex\fi
    \ifnum \xint_c_ix<1\string#1 \xint_dothis \BNE_startint\fi
    \xint_orthat \BNE_notadigit #1%
}%
\def\BNE_notadigit#1{\BNE_getnext }%
\def\BNE_startint #1%
{%
```

```
    \if #10\expandafter\BNE_gobz_a\else\expandafter\BNE_scanint_a\fi #1%
}%
\def\BNE_scanint_a #1#2%
    {\expanded\bgroup{{\iffalse}}\fi #1%
     \expandafter\BNE_scanint_main\romannumeral`&&@#2}%
\def\BNE_gobz_a #1#2%
    {\expanded\bgroup{{\iffalse}}\fi
     \expandafter\BNE_gobz_scanint_main\romannumeral`&&@#2}%
\def\BNE_scanint_main #1%
{%
    \ifcat \relax #1\expandafter\BNE_scanint_hit_cs \fi
    \ifnum\xint_c_ix<1\string#1 \else\expandafter\BNE_scanint_next\fi
    #1\BNE_scanint_again
}%
\def\BNE_scanint_again #1%
{%
    \expandafter\BNE_scanint_main\romannumeral`&&@#1%
}%
\def\BNE_scanint_hit_cs \ifnum#1\fi#2\BNE_scanint_again
{%
    \iffalse{{{\fi}}\expandafter}\romannumeral`&&@\BNE_getop#2%
}%
\def\BNE_scanint_next #1\BNE_scanint_again
{%
    \if    _#1\xint_dothis\BNE_scanint_again\fi
    \xint_orthat
    {\iffalse{{{\fi}}\expandafter}\romannumeral`&&@\BNE_getop#1}%
}%
\def\BNE_gobz_scanint_main #1%
{%
    \ifcat \relax #1\expandafter\BNE_gobz_scanint_hit_cs\fi
    \ifnum\xint_c_x<1\string#1 \else\expandafter\BNE_gobz_scanint_next\fi
    #1\BNE_scanint_again
}%
\def\BNE_gobz_scanint_again #1%
{%
    \expandafter\BNE_gobz_scanint_main\romannumeral`&&@#1%
}%
\def\BNE_gobz_scanint_hit_cs\ifnum#1\fi#2\BNE_scanint_again
{%
    0\iffalse{{{\fi}}\expandafter}\romannumeral`&&@\BNE_getop#2%
}%
\def\BNE_gobz_scanint_next #1\BNE_scanint_again
{%
    \if    _#1\xint_dothis\BNE_gobz_scanint_again\fi
    \if    0#1\xint_dothis\BNE_gobz_scanint_again\fi
    \xint_orthat
    {0\iffalse{{{\fi}}\expandafter}\romannumeral`&&@\BNE_getop#1}%
}%
\def\BNE_hex_in #1.%
{%
    \expanded{{{\xintHexToDec{#1}}}\expandafter}\romannumeral`&&@\BNE_getop
```

```
}%
\def\BNE_scanhex #1% #1="
{%
    \expandafter\BNE_hex_in\expanded\bgroup\BNE_scanhex_a
}%
\def\BNE_scanhex_a #1%
{%
    \ifcat #1\relax\xint_dothis{.\iffalse{\fi}#1}\fi
    \xint_orthat {\BNE_scanhex_aa #1}%
}%
\def\BNE_scanhex_aa #1%
{%
    \if\ifnum`#1>`/
       \ifnum`#1>`9
       \ifnum`#1>`@
       \ifnum`#1>`F
       0\else1\fi\else0\fi\else1\fi\else0\fi 1%
       \expandafter\BNE_scanhex_b
    \else
       \if _#1\xint_dothis{\expandafter\BNE_scanhex_bgob}\fi
       \xint_orthat {\xint_afterfi {.\iffalse{\fi}}}%
    \fi
    #1%
}%
\def\BNE_scanhex_b #1#2%
{%
    #1\expandafter\BNE_scanhex_a\romannumeral`&&@#2%
}%
\def\BNE_scanhex_bgob #1#2%
{%
    \expandafter\BNE_scanhex_a\romannumeral`&&@#2%
}%
```

10.8 \BNE_getop

```
\def\BNE_getop #1%
{%
    \expandafter\BNE_getop_a\romannumeral`&&@#1%
}%
\catcode`* 11
\def\BNE_getop_a #1%
{%
    \ifx   \relax #1\xint_dothis\xint_firstofthree\fi
    \ifcat \relax #1\xint_dothis\xint_secondofthree\fi
    \ifnum\xint_c_ix<1\string#1 \xint_dothis\xint_secondofthree\fi
    \if    (#1\xint_dothis      \xint_secondofthree\fi %)
    \xint_orthat \xint_thirdofthree
    {\BNE_foundend}%
    {\BNE_precedence_*** *#1}%
    {\expandafter\BNE_scanop_a \string#1}%
}%
\catcode`* 12
\def\BNE_foundend {\xint_c_ \relax}%
```

```
\def\BNE_scanop_a #1#2%
{%
    \expandafter\BNE_scanop_b\expandafter#1\romannumeral`&&@#2%
}%
\def\BNE_scanop_b #1#2%
{%
    \ifcat#2\relax\xint_dothis{\BNE_foundop_a #1#2}\fi
    \ifcsname BNE_itself_#1#2\endcsname
    \xint_dothis
        {\expandafter\BNE_scanop_c\csname BNE_itself_#1#2\endcsname}\fi
    \xint_orthat {\BNE_foundop_a #1#2}%
}%
\def\BNE_scanop_c #1#2%
{%
    \expandafter\BNE_scanop_d\expandafter#1\romannumeral`&&@#2%
}%
\def\BNE_scanop_d #1#2%
{%
  \ifcat#2\relax \xint_dothis{\BNE_foundop #1#2}\fi
  \ifcsname BNE_itself_#1#2\endcsname
  \xint_dothis
        {\expandafter\BNE_scanop_c\csname BNE_itself_#1#2\endcsname }\fi
  \xint_orthat {\csname BNE_precedence_#1\endcsname #1#2}%
}%
\def\BNE_foundop_a #1%
{%
    \ifcsname BNE_precedence_#1\endcsname
        \csname BNE_precedence_#1\expandafter\endcsname
        \expandafter #1%
    \else
        \xint_afterfi{\BNE_getop\romannumeral0%
        \XINT_expandableerror
        {"#1" is unknown as operator. (I)nsert one:} }%<<deliberate space
    \fi
}%
\def\BNE_foundop #1{\csname BNE_precedence_#1\endcsname #1}%
```

10.9 Expansion spanning; opening and closing parentheses

```
\def\BNE_tmpa #1#2#3#4#5%
{%
    \def#1% start
    {%
        \expandafter#2\romannumeral`&&@\BNE_getnext
    }%
    \def#2##1% check
    {%
        \xint_UDsignfork
          ##1{\expandafter#3\romannumeral`&&@#4}%
            -{#3##1}%
        \krof
    }%
    \def#3##1##2% checkp
```

```
    {%
        \ifcase ##1%
           \expandafter\BNE_done
        \or\expandafter#5%
        \else
           \expandafter#3\romannumeral`&&@\csname BNE_op_##2\expandafter\endcsname
        \fi
    }%
    \def#5%
    {%
        \XINT_expandableerror
        {An extra ) has been removed. Hit Return, fingers crossed.}%
        \expandafter#2\romannumeral`&&@\expandafter\BNE_put_op_first
        \romannumeral`&&@\BNE_getop_legacy
    }%
}%
\let\BNE_done\space
\def\BNE_getop_legacy #1%
{%
    \expanded{\unexpanded{{#1}}\expandafter}\romannumeral`&&@\BNE_getop
}%
\expandafter\BNE_tmpa
    \csname BNE_start\expandafter\endcsname
    \csname BNE_check\expandafter\endcsname
    \csname BNE_checkp\expandafter\endcsname
    \csname BNE_op_-xii\expandafter\endcsname
    \csname BNE_extra_)\endcsname
\catcode`) 11
\def\BNE_tmpa #1#2#3#4#5#6%
{%
    \def #1##1% op_(
    {%
        \expandafter #4\romannumeral`&&@\BNE_getnext
    }%
    \def #2##1% op_)
    {%
        \expanded{\unexpanded{\BNE_put_op_first{##1}}\expandafter}\romannumeral`&&@\BNE_getop
    }%
    \def #3% oparen
    {%
        \expandafter #4\romannumeral`&&@\BNE_getnext
    }%
    \def #4##1% check-
    {%
        \xint_UDsignfork
            ##1{\expandafter#5\romannumeral`&&@#6}%
              -{#5##1}%
        \krof
    }%
    \def #5##1##2% checkp
    {%
        \ifcase ##1\expandafter\BNE_missing_)
```

```
        \or \csname BNE_op_##2\expandafter\endcsname
        \else
          \expandafter #5\romannumeral`&&@\csname BNE_op_##2\expandafter\endcsname
        \fi
    }%
}%
\expandafter\BNE_tmpa
    \csname BNE_op_(\expandafter\endcsname
    \csname BNE_op_)\expandafter\endcsname
    \csname BNE_oparen\expandafter\endcsname
    \csname BNE_check-_)\expandafter\endcsname
    \csname BNE_checkp_)\expandafter\endcsname
    \csname BNE_op_-xii\endcsname
\let\BNE_precedence_)\xint_c_i
\def\BNE_missing_)
   {\XINT_expandableerror{Sorry to report a missing ) at the end of this journey.}%
    \xint_c_ \BNE_done }%
\catcode`) 12
```

10.10 The comma as binary operator

At 1.4, it is simply a union operator for 1D oples. Inserting directly here a <space><comma> separator (as in earlier releases) in accumulated result would avoid having to do it on output but to the cost of diverging from xintexpr upstream code, and to have to let the \evaltohex output routine handle comma separated values rather than braced values.

```
\def\BNE_tmpa #1#2#3#4#5%
{%
    \def #1##1% \BNE_op_,
    {%
      \expanded{\unexpanded{#2{##1}}\expandafter}%
      \romannumeral`&&@\expandafter#3\romannumeral`&&@\BNE_getnext
    }%
    \def #2##1##2##3##4{##2##3{##1##4}}% \BNE_exec_,
    \def #3##1% \BNE_check-_,
    {%
      \xint_UDsignfork
        ##1{\expandafter#4\romannumeral`&&@#5}%
          -{#4##1}%
      \krof
    }%
    \def #4##1##2% \BNE_checkp_,
    {%
      \ifnum ##1>\xint_c_iii
        \expandafter#4%
           \romannumeral`&&@\csname BNE_op_##2\expandafter\endcsname
      \else
        \expandafter##1\expandafter##2%
      \fi
    }%
}%
\expandafter\BNE_tmpa
    \csname BNE_op_,\expandafter\endcsname
    \csname BNE_exec_,\expandafter\endcsname
```

```
    \csname BNE_check-_,\expandafter\endcsname
    \csname BNE_checkp_,\expandafter\endcsname
    \csname BNE_op_-xii\endcsname
\expandafter\let\csname BNE_precedence_,\endcsname\xint_c_iii
```

10.11 The minus as prefix operator of variable precedence level

This \BNE_Op_opp causes trouble as at 1.4 it must be f-expandable, whereas earlier it expanded inside \csname ...\endcsname context, so I could define it as \if -#1\else \if 0#10\else -#1\fi \fi where #1 was the first token of unbraced argument but this meant at 1.4 an added \xint_firstofone here. Well let's return to sanity at 1.4a and not add the \xint_firstofone and simply default \BNE_Op_opp to \xintiiOpp, which it should have been all along! And on this occasion let's trim user documentation of irrelevant complications.

```
\def\BNE_tmpb #1#2#3#4#5%
{%
    \def #1% \BNE_op_-<level>
    {%
      \expandafter #2\romannumeral`&&@\expandafter#3%
      \romannumeral`&&@\BNE_getnext
    }%
    \def #2##1##2##3% \BNE_exec_-<level>
    {%
      \expandafter ##1\expandafter ##2\expandafter
       {\expandafter{\romannumeral`&&@\BNE_Op_opp##3}}%
    }%
    \def #3##1% \BNE_check-_-<level>
    {%
      \xint_UDsignfork
        ##1{\expandafter #4\romannumeral`&&@#1}%
          -{#4##1}%
      \krof
    }%
    \def #4##1##2% \BNE_checkp_-<level>
    {%
      \ifnum ##1>#5%
        \expandafter #4%
        \romannumeral`&&@\csname BNE_op_##2\expandafter\endcsname
      \else
        \expandafter ##1\expandafter ##2%
      \fi
    }%
}%
\def\BNE_tmpa #1%
{%
\expandafter\BNE_tmpb
    \csname BNE_op_-#1\expandafter\endcsname
    \csname BNE_exec_-#1\expandafter\endcsname
    \csname BNE_check-_-#1\expandafter\endcsname
    \csname BNE_checkp_-#1\expandafter\endcsname
    \csname xint_c_#1\endcsname
```

```
}%
\BNE_tmpa {xii}%
\BNE_tmpa {xiv}%
\BNE_tmpa {xvi}%
\BNE_tmpa {xviii}%
```

10.12 The infix operators.

I could have at the 1.4 refactoring injected usage of \expanded here, but kept in sync with upstream xintexpr code.

Macro names are somewhat bad and there is much risk of confusion in future maintenance of \BNE_Op_ prefix (used for \BNE_Op_add etc...; besides this should have been \BNE_Op_Add) and \BNE_op_ prefix (used for \BNE_op_+ etc...).

```
\def\BNE_defbin_c #1#2#3#4#5#6#7%
{%
  \def #1##1% \BNE_op_<op>
  {%
    \expanded{\unexpanded{#2{##1}}\expandafter}%
    \romannumeral`&&@\expandafter#3\romannumeral`&&@\BNE_getnext
  }%
  \def #2##1##2##3##4% \BNE_exec_<op>
  {%
    \expandafter##2\expandafter##3\expandafter
      {\expandafter{\romannumeral`&&@#6##1##4}}%
  }%
  \def #3##1% \BNE_check-_<op>
  {%
    \xint_UDsignfork
      ##1{\expandafter#4\romannumeral`&&@#5}%
        -{#4##1}%
    \krof
  }%
  \def #4##1##2% \BNE_checkp_<op>
  {%
    \ifnum ##1>#7%
      \expandafter#4%
      \romannumeral`&&@\csname BNE_op_##2\expandafter\endcsname
    \else
      \expandafter ##1\expandafter ##2%
    \fi
  }%
}%
\def\BNE_defbin_b #1#2#3#4%
{%
  \expandafter\BNE_defbin_c
  \csname BNE_op_#1\expandafter\endcsname
  \csname BNE_exec_#1\expandafter\endcsname
  \csname BNE_check-_#1\expandafter\endcsname
  \csname BNE_checkp_#1\expandafter\endcsname
  \csname BNE_op_-#3\expandafter\endcsname
  \csname #4\expandafter\endcsname
  \csname BNE_precedence_#1\endcsname
```

```
  \expandafter
  \let\csname BNE_precedence_#1\expandafter\endcsname
      \csname xint_c_#2\endcsname
}%
\BNE_defbin_b {//} {xiv}{xiv}{BNE_Op_div}%
\BNE_defbin_b {/:} {xiv}{xiv}{BNE_Op_mod}%
\BNE_defbin_b  +   {xii}{xii}{BNE_Op_add}%
\BNE_defbin_b  -   {xii}{xii}{BNE_Op_sub}%
\BNE_defbin_b  *   {xiv}{xiv}{BNE_Op_mul}%
\BNE_defbin_b  /   {xiv}{xiv}{BNE_Op_divround}%
\BNE_defbin_b  ^   {xviii}{xviii}{BNE_Op_pow}%
\expandafter\def\csname BNE_itself_**\endcsname {^}%
\expandafter\def\csname BNE_itself_//\endcsname {//}%
\expandafter\def\csname BNE_itself_/:\endcsname {/:}%
\expandafter\let\csname BNE_precedence_***\endcsname \xint_c_xvi
```

10.13 ! as postfix factorial operator

This one uses \expanded.

```
\catcode`! 11
\let\BNE_precedence_! \xint_c_xx
\def\BNE_op_! #1%
{%
    \expandafter\BNE_put_op_first
    \expanded{{{\BNE_Op_fac#1}}\expandafter}\romannumeral`&&@\BNE_getop
}%
```

10.14 Cleanup

```
\let\BNEtmpa\relax \let\BNE_tmpa\relax \let\BNE_tmpb\relax \let\BNE_tmpc\relax
\BNErestorecatcodes%
```