The xint source code

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

Package version: 1.3c (2018/06/17); documentation date: 2018/06/17.
From source file xint.dtx. Time-stamp: <17-06-2018 at 20:00:43 CEST>.

Contents

1 Introduction to the implementation (recent changes)

This is 1.3c of 2018/06/17.

- Release 1.3c of 2018/06/17:
  - \xintglobaldefstrue to give global scope to definitions done by \xintdefvar, \xintdeffunc, \xintNewExpr, et. al.
  - qraw() to allow bypassing the normal mode of functioning and reduce memory footprint from parsing (very many) comma separated numerical items.
  - the colon in the := part of the syntax for \xintdefvar and variants is now optional; and if present it may be an active character or have any (reasonable) catcode.
  - \xintdefvar, \xintdeffunc and their variants try to set the catcode of the semi-colon which delimits their arguments; of course this will not work if that catcode is already frozen.
  - \xintexprSafeCatcodes sets a toggle to allow some nested usage.
  - \xintUniformDeviate is better documented and sourcexint.pdf is better hyperlinked and includes indices for the macros defined by each package.
  - since 1.3 release, xintfrac (hence xintexpr) loaded xintgcd in contradiction to what documentation says. Removed for now.
- Release 1.3b of 2018/05/18:
  - \xintUniformDeviate, random(), qrand(), randrange(),
  - and their support macros \XINTinRandomFloatSdigits, \XINTinRandomFloatSixteen, \xintRandomDigits, \xintiiRandRange, \xintiiRandRangeAtoB, also \xintXRandomDigits,
  - functions defined via \xintdeffunc (et al.) to be without argument can be used as foo(): foo(nil) syntax not anymore required.
- Release 1.3a of 2018/03/07:
  - removes from xintcore, xint and xintfrac the whole deprecation mechanism, as there are no more currently any deprecated macro,
  - adds ifone() and ifint() conditionals to the expression parsers,
  - has a completely redone \XINT_factortens, in the style of 1.2 release (but about 100 digits at least are needed for noticeable speed gain),
  - and last but not least fixes via addition of \xintExpandArgs the meaning of user defined functions, which in case of recursivity (as made possible at 1.3) were badly inefficient for lack of expansion of their arguments.
- Release 1.3 of 2018/03/01:
  - removed all macros previously deprecated at 1.2o,
  - modified addition, subtraction and modulo operations to use a least common multiple for the denominator of the result,
  - added \xintPIrr, \xintDecToString and preduce(),
  - and last but not least refactored extensively the \xintNewExpr/\xintdeffunc mechanism. It got both leaner and stronger and makes possible recursive function definitions.
- Release 1.2q of 2018/02/06 was a bugfix release with these changes:
  - fix to an 1.2l xintcore subtraction bug causing a breakage under some rare circumstances :-(. It was caused by a refactoring left-over (extra ! in XINT_sub_l_Ida replacement text), and should have been detected by the test suite, but manual testing from early days was not yet entirely included in our automated procedure.
  - new feature: tacit multiplication in front of digits, for example 10!20!30! or (10+10)10.
- Release 1.2p of 2017/12/05 had some breaking changes:
  - new output for the \xintBezout macro (xintgcd),

- the xintexpr operators /: (aka 'mod') and //, and the supporting macros from xintcore and xintfrac, are now associated with the *floored* division. Formerly it was the *truncated* division. This is breaking change for operands of opposite signs.

Improvements and new features:

- xinttools macro \xintListWithSep is faster (first update since 1.04-2013/04/25...).
- divmod() function added to the xintexpr parsers,
- \xintdefvar, \xintdeffloatvar, \xintdefiivar extended to allow multiple simultaneous assignments.

- Release 1.2o of 2017/08/29 deprecated those macros from xintcore and xint which filtered their arguments via \xintNum. Currently these macros execute as formerly but raise an error message. This deprecation is overruled for most if xintfrac is loaded as it provides their proper definitions. Some however (like \xintiAdd) remain deprecated even if loading xintfrac. All these deprecated macros are destined to be removed at some future release.

  A few macros got renamed (e.g. \xintNot became \xintNOT.) Former names emit a deprecation error and will get removed at some future release.

- Release 1.2n of 2017/08/06 removed the xintbinhex dependencies upon xintcore; the package now depends upon, and loads, only xintkernel. The allowed maximal size for the inputs of the base conversion macros got increased. The speed got slightly improved.

- Release 1.2m of 2017/07/31 rewrote entirely the xintbinhex module in the style of the techniques from 1.2 xintcore. The new macros expand faster but their inputs are now limited to a few thousand characters (the earlier routines, which dated back to 1.08 could handle (slowly) tens of thousands of digits.)

- Release 1.2l of 2017/07/26 refactored the subtraction and also \xintiiCmp got a rewrite. It should certainly use \pdfstrcmp for dramatic speed-up but I do not want to have to worry about multi-engine usage.

  Some utility routines in xintcore manipulating blocks of eight digits and still in O(N^2) style got rewritten analogously to the 1.2i version of macros such as \xintInc. Also \xintiNum was revisited.

  The arithmetic macros of xintcore and all macros of xintfrac using \XINT_infrac were made compatible with arguments using non-delimited \the\numexpr or \the\mathcode etc... But \xintiiAbs and \xintiiOpp were not modified (to avoid some overhead) as well as routines such as \xintInc which are primarily for internal usage.

- Release 1.2i of 2016/12/13 rewrote some legacy macros like \xintDSR or \xintDecSplit in the style of the techniques of 1.2. But this means also that they got limited to about 22480 digits for the former and 19970 digits for the latter (this is with the input stack size at 5000 and the maximal expansion depth at 10000.) This is not really an issue from the point of view of calling macros (such as \xintTrunc, \xintRound), because they usually had since 1.2 their own limitation at about 19950 digits from other code parts (such as division.) The macro \xintXTrunc (which is not f-expandable however) can produce tens of thousands of digits and it escapes these limitations. Old macros such as \xintLength are not limited either (incidentally it got a lifting in 1.2i.) The macros from xinttools (\xintKeep, \xintTrim, \xintNthElt) also are not limited (but slower.)

- Release 1.2 of 2015/10/10 entirely rewrote the core arithmetic routines located in xintcore. The parser of xintexpr got faster and the limitation at 5000 digits per number was removed.

- Extensive changes in release 1.1 of 2014/10/28 were located in xintexpr. Also with that release, packages xintkernel and xintcore were extracted from xinttools and xint, and \xintAdd was modified to not multiply denominators blindly.

Some parts of the code still date back to the initial release, and at that time I was learning my trade in expandable TeX macro programming. At some point in the future, I will have to re-examine the older parts of the code.

Warning: pay attention when looking at the code to the catcode configuration as found in `\XINT_setcatcodes`. Additional temporary configuration is used at some locations. For example `!` is of catcode letter in `xintexpr` and there are locations with funny catcodes e.g. using some letters with the math shift catcode.

2 Package xintkernel implementation

This package provides the common minimal code base for loading management and catcode control and also a few programming utilities. With 1.2 a few more helper macros and all \chardef's have been moved here. The package is loaded by both xintcore.sty and xinttools.sty hence by all other packages.

1.1. separated package.

1.2i. \xintreplicate, \xintgobble, \xintLengthUpTo and \xintLastItem, and faster \xintLength.

1.3b. \xintUniformDeviate.

2.1 Catcodes, ε-TEX and reload detection

The code for reload detection was initially copied from Heiko Oberdiek's packages, then modified.

The method for catcodes was also initially directly inspired by these packages.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode35=6    % #
  \catcode44=12   % ,
  \catcode45=12   % -
  \catcode46=12   % .
  \catcode58=12   % :
  \catcode95=11   % _
  \expandafter
    \ifx\csname PackageInfo\endcsname\relax
      \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
    \else
      \def\y#1#2{\PackageInfo{#1}{#2}}%
    \fi
  \let\z\relax
  \expandafter
    \ifx\csname numexpr\endcsname\relax
     \y{xintkernel}{\numexpr not available, aborting input}%
     \def\z{\endgroup\endinput}%
    \else
    \expandafter
     \ifx\csname XINTsetupcatcodes\endcsname\relax
      \else
       \y{xintkernel}{I was already loaded, aborting input}%
       \def\z{\endgroup\endinput}%
      \fi
    \fi
  \ifx\z\relax\else\expandafter\z\fi%
```

2.1.1 \XINT_setcatcodes, \XINT_storecatcodes, \XINT_restorecatcodes_endinput

```
\def\PrepareCatcodes
{%
    \endgroup
    \def\XINT_restorecatcodes
    {% takes care of all, to allow more economical code in modules
         \catcode0=\the\catcode0 %
         \catcode59=\the\catcode59   % ; xintexpr
         \catcode126=\the\catcode126 % ~ xintexpr
         \catcode39=\the\catcode39   % ' xintexpr
         \catcode34=\the\catcode34   % " xintbinhex, and xintexpr
         \catcode63=\the\catcode63   % ? xintexpr
         \catcode124=\the\catcode124 % | xintexpr
         \catcode38=\the\catcode38   % & xintexpr
         \catcode64=\the\catcode64   % @ xintexpr
         \catcode33=\the\catcode33   % ! xintexpr
         \catcode93=\the\catcode93   % ] -, xintfrac, xintseries, xintcfrac
         \catcode91=\the\catcode91   % [ -, xintfrac, xintseries, xintcfrac
         \catcode36=\the\catcode36   % $ xintgcd only
         \catcode94=\the\catcode94   % ^
         \catcode96=\the\catcode96   % `
         \catcode47=\the\catcode47   % /
         \catcode41=\the\catcode41   % )
         \catcode40=\the\catcode40   % (
         \catcode42=\the\catcode42   % *
         \catcode43=\the\catcode43   % +
         \catcode62=\the\catcode62   % >
         \catcode60=\the\catcode60   % <
         \catcode58=\the\catcode58   % :
         \catcode46=\the\catcode46   % .
         \catcode45=\the\catcode45   % -
         \catcode44=\the\catcode44   % ,
         \catcode35=\the\catcode35   % #
         \catcode95=\the\catcode95   % _
         \catcode125=\the\catcode125 % }
         \catcode123=\the\catcode123 % {
         \endlinechar=\the\endlinechar
         \catcode13=\the\catcode13   % ^^M
         \catcode32=\the\catcode32   %
         \catcode61=\the\catcode61\relax   % =
    }%
    \edef\XINT_restorecatcodes_endinput
    {%
         \XINT_restorecatcodes\noexpand\endinput %
```

```
      }%
      \def\XINT_setcatcodes
      {%
        \catcode61=12   % =
        \catcode32=10   % space
        \catcode13=5    % ^^M
        \endlinechar=13 %
        \catcode123=1   % {
        \catcode125=2   % }
        \catcode95=11   % _ LETTER
        \catcode35=6    % #
        \catcode44=12   % ,
        \catcode45=12   % -
        \catcode46=12   % .
        \catcode58=11   % : LETTER
        \catcode60=12   % <
        \catcode62=12   % >
        \catcode43=12   % +
        \catcode42=12   % *
        \catcode40=12   % (
        \catcode41=12   % )
        \catcode47=12   % /
        \catcode96=12   % `
        \catcode94=11   % ^ LETTER
        \catcode36=3    % $
        \catcode91=12   % [
        \catcode93=12   % ]
        \catcode33=12   % ! (xintexpr.sty will use catcode 11)
        \catcode64=11   % @ LETTER
        \catcode38=7    % & for \romannumeral`&&@ trick.
        \catcode124=12  % |
        \catcode63=11   % ? LETTER
        \catcode34=12   % "
        \catcode39=12   % '
        \catcode126=3   % ~ MATH
        \catcode59=12   % ;
        \catcode0=12    % for \romannumeral`&&@ trick
      }%
      \XINT_setcatcodes
  }%
\PrepareCatcodes
```

Other modules could possibly be loaded under a different catcode regime.

```
\def\XINTsetupcatcodes {% for use by other modules
      \edef\XINT_restorecatcodes_endinput
      {%
           \XINT_restorecatcodes\noexpand\endinput %
      }%
      \XINT_setcatcodes
}%
```

2.2 Package identification

Inspired from Heiko Oberdiek's packages. Modified in 1.09b to allow re-use in the other modules. Also I assume now that if \ProvidesPackage exists it then does define \ver@<pkgname>.sty, code of HO for some reason escaping me (compatibility with LaTeX 2.09 or other things ??) seems to set extra precautions.

1.09c uses e-TeX \ifdefined.

```
\ifdefined\ProvidesPackage
  \let\XINT_providespackage\relax
\else
  \def\XINT_providespackage #1#2[#3]%
              {\immediate\write-1{Package: #2 #3}%
               \expandafter\xdef\csname ver@#2.sty\endcsname{#3}}%
\fi
\XINT_providespackage
\ProvidesPackage {xintkernel}%
  [2018/06/17 1.3c Paraphernalia for the xint packages (JFB)]%
```

2.3 Constants

```
\chardef\xint_c_     0
\chardef\xint_c_i    1
\chardef\xint_c_ii   2
\chardef\xint_c_iii  3
\chardef\xint_c_iv   4
\chardef\xint_c_v    5
\chardef\xint_c_vi   6
\chardef\xint_c_vii  7
\chardef\xint_c_viii 8
\chardef\xint_c_ix     9
\chardef\xint_c_x     10
\chardef\xint_c_xii   12
\chardef\xint_c_xiv   14
\chardef\xint_c_xvi   16
\chardef\xint_c_xviii 18
\chardef\xint_c_xxii  22
\chardef\xint_c_ii^v  32
\chardef\xint_c_ii^vi 64
\chardef\xint_c_ii^vii       128
\mathchardef\xint_c_ii^viii   256
\mathchardef\xint_c_ii^xii   4096
\mathchardef\xint_c_x^iv    10000
```

2.4 (WIP) `\xint_texuniformdeviate` and needed counts

```
\ifdefined\pdfuniformdeviate  \let\xint_texuniformdeviate\pdfuniformdeviate\fi
\ifdefined\uniformdeviate     \let\xint_texuniformdeviate\uniformdeviate   \fi
\ifx\xint_texuniformdeviate\relax\let\xint_texuniformdeviate\xint_undefined\fi
\ifdefined\xint_texuniformdeviate
  \csname newcount\endcsname\xint_c_ii^xiv
  \xint_c_ii^xiv   16384 % "4000, 2**14
  \csname newcount\endcsname\xint_c_ii^xxi
  \xint_c_ii^xxi 2097152 % "200000, 2**21
\fi
```

2.5 Token management utilities

1.3b. \xint_gobandstop_... macros because this is handy for \xintRandomDigits.

```
\def\XINT_tmpa { }%
\ifx\XINT_tmpa\space\else
   \immediate\write-1{Package xintkernel Warning: ATTENTION!}%
   \immediate\write-1{\string\space\XINT_tmpa macro does not have its normal
     meaning.}%
   \immediate\write-1{\XINT_tmpa\XINT_tmpa\XINT_tmpa\XINT_tmpa
                      All kinds of catastrophes will ensue!!!!}%
\fi
\def\XINT_tmpb {}%
\ifx\XINT_tmpb\empty\else
    \immediate\write-1{Package xintkernel Warning: ATTENTION!}%
    \immediate\write-1{\string\empty\XINT_tmpa macro does not have its normal
      meaning.}%
    \immediate\write-1{\XINT_tmpa\XINT_tmpa\XINT_tmpa\XINT_tmpa
                       All kinds of catastrophes will ensue!!!!}%
\fi
\let\XINT_tmpa\relax \let\XINT_tmpb\relax
\ifdefined\space\else\def\space { }\fi
\ifdefined\empty\else\def\empty {}\fi
\let\xint_gobble_\empty
\long\def\xint_gobble_i    #1{}%
\long\def\xint_gobble_ii   #1#2{}%
\long\def\xint_gobble_iii  #1#2#3{}%
\long\def\xint_gobble_iv   #1#2#3#4{}%
\long\def\xint_gobble_v    #1#2#3#4#5{}%
\long\def\xint_gobble_vi   #1#2#3#4#5#6{}%
\long\def\xint_gobble_vii  #1#2#3#4#5#6#7{}%
\long\def\xint_gobble_viii #1#2#3#4#5#6#7#8{}%
\let\xint_gob_andstop_\space
\long\def\xint_gob_andstop_i    #1{ }%
\long\def\xint_gob_andstop_ii   #1#2{ }%
\long\def\xint_gob_andstop_iii  #1#2#3{ }%
\long\def\xint_gob_andstop_iv   #1#2#3#4{ }%
\long\def\xint_gob_andstop_v    #1#2#3#4#5{ }%
\long\def\xint_gob_andstop_vi   #1#2#3#4#5#6{ }%
\long\def\xint_gob_andstop_vii  #1#2#3#4#5#6#7{ }%
\long\def\xint_gob_andstop_viii #1#2#3#4#5#6#7#8{ }%
\long\def\xint_firstofone  #1{#1}%
\long\def\xint_firstoftwo  #1#2{#1}%
\long\def\xint_secondoftwo #1#2{#2}%
\let\xint_gobble_thenstop\xint_gob_andstop_i
\long\def\xint_firstofone_thenstop  #1{ #1}%
\long\def\xint_firstoftwo_thenstop  #1#2{ #1}%
\long\def\xint_secondoftwo_thenstop #1#2{ #2}%
\long\def\xint_exchangetwo_keepbraces    #1#2{{#2}{#1}}%
```

2.6 “gob til” macros and UD style fork

```
\long\def\xint_gob_til_R #1\R {}%
\long\def\xint_gob_til_W #1\W {}%
```

```
\long\def\xint_gob_til_Z #1\Z {}%
\long\def\xint_gob_til_zero #10{}%
\long\def\xint_gob_til_one  #11{}%
\long\def\xint_gob_til_zeros_iii   #1000{}%
\long\def\xint_gob_til_zeros_iv    #10000{}%
\long\def\xint_gob_til_eightzeroes #100000000{}%
\long\def\xint_gob_til_dot    #1.{}%
\long\def\xint_gob_til_G     #1G{}%
\long\def\xint_gob_til_minus #1-{}%
\long\def\xint_UDzerominusfork #10-#2#3\krof {#2}%
\long\def\xint_UDzerofork       #10#2#3\krof {#2}%
\long\def\xint_UDsignfork       #1-#2#3\krof {#2}%
\long\def\xint_UDwfork         #1\W#2#3\krof {#2}%
\long\def\xint_UDXINTWfork     #1\XINT_W#2#3\krof {#2}%
\long\def\xint_UDzerosfork     #100#2#3\krof {#2}%
\long\def\xint_UDonezerofork   #110#2#3\krof {#2}%
\long\def\xint_UDsignsfork     #1--#2#3\krof {#2}%
\let\xint:\char
\long\def\xint_gob_til_xint:#1\xint:{}%
\def\xint_bracedstopper{\xint:}%
\long\def\xint_gob_til_exclam #1!{}%
\long\def\xint_gob_til_sc #1;{}%
```

2.7 \xint_afterfi

```
\long\def\xint_afterfi #1#2\fi {\fi #1}%
```

2.8 \xint_bye, \xint_Bye

1.09. \xint_bye

1.2i. \xint_Bye for \xintDSRr and \xintRound. Also \xint_bye_thenstop.

```
\long\def\xint_bye #1\xint_bye {}%
\long\def\xint_Bye #1\xint_bye {}%
\long\def\xint_bye_thenstop #1\xint_bye { }%
```

2.9 \xintdothis, \xintorthat

1.1.

1.2. names without underscores.

To be used this way:

```
\if..\xint_dothis{..}\fi
\if..\xint_dothis{..}\fi
\if..\xint_dothis{..}\fi
...more such...
\xint_orthat{...}
```

Ancient testing indicated it is more efficient to list first the more improbable clauses.

```
\long\def\xint_dothis #1#2\xint_orthat #3{\fi #1}% 1.1
\let\xint_orthat \xint_firstofone
\long\def\xintdothis #1#2\xintorthat #3{\fi #1}%
\let\xintorthat \xint_firstofone
```

2.10 \xint_zapspaces

1.1.
This little utility zaps leading, intermediate, trailing, spaces in completely expanding context (\edef, \csname...\endcsname).

Usage: \xint_zapspaces foo<space>\xint_gobble_i

Will remove some brace pairs (but not spaces inside them). By the way the \zap@spaces of LaTeX2e handles unexpectedly things such as

\zap@spaces 1 {22} 3 4 \@empty

(spaces are not all removed). This does not happen with \xint_zapspaces.

Explanation: if there are leading spaces, then the first #1 will be empty, and the first #2 being undelimited will be stripped from all the remaining leading spaces, if there was more than one to start with. Of course brace-stripping may occur. And this iterates: each time a #2 is removed, either we then have spaces and next #1 will be empty, or we have no spaces and #1 will end at the first space. Ultimately #2 will be \xint_gobble_i.

This is not really robust as it may switch the expansion order of macros, and the \xint_zapspaces token might end up being fetched up by a macro. But it is enough for our purposes, for example:

\the\numexpr\xint_zapspaces 1 2 \xint_gobble_i\relax

expands to 12, not to 12\relax.

1.2e. \xint_zapspaces_o. Expansion of #1 should not gobble a space!

1.2i. made \long.

```
\long\def\xint_zapspaces #1 #2{#1#2\xint_zapspaces }% 1.1
\long\def\xint_zapspaces_o #1{\expandafter\xint_zapspaces#1 \xint_gobble_i}%
```

2.11 \odef, \oodef, \fdef

May be prefixed with \global. No parameter text.

```
\def\xintodef #1{\expandafter\def\expandafter#1\expandafter }%
\def\xintoodef #1{\expandafter\expandafter\expandafter\def
                  \expandafter\expandafter\expandafter#1%
                  \expandafter\expandafter\expandafter }%
\def\xintfdef #1#2%
    {\expandafter\def\expandafter#1\expandafter{\romannumeral`&&@#2}}%
\ifdefined\odef\else\let\odef\xintodef\fi
\ifdefined\oodef\else\let\oodef\xintoodef\fi
\ifdefined\fdef\else\let\fdef\xintfdef\fi
```

2.12 \xintReverseOrder

1.0. does not expand its argument. The whole of xint codebase now contains only two calls to \XINT_rord_main (in xintgcd).
Attention: removes brace pairs (and swallows spaces).
For digit tokens a faster reverse macro is provided by (1.2) \xintReverseDigits in xint.
For comma separated items, 1.2g has \xintCSVReverse in xinttools.

```
\def\xintReverseOrder {\romannumeral0\xintreverseorder }%
\long\def\xintreverseorder #1%
{%
    \XINT_rord_main {}#1%
      \xint:
        \xint_bye\xint_bye\xint_bye\xint_bye
        \xint_bye\xint_bye\xint_bye\xint_bye
      \xint:
}%
\long\def\XINT_rord_main #1#2#3#4#5#6#7#8#9%
{%
    \xint_bye #9\XINT_rord_cleanup\xint_bye
    \XINT_rord_main {#9#8#7#6#5#4#3#2#1}%
}%
\def\XINT_rord_cleanup #1{%
\long\def\XINT_rord_cleanup\xint_bye\XINT_rord_main ##1##2\xint:
{%
    \expandafter#1\xint_gob_til_xint: ##1%
}}\XINT_rord_cleanup { }%
```

2.13 \xintLength

1.0. does not expand its argument. See \xintNthElt{0} from xinttools which f-expands its argument.

1.2g. added \xintCSVLength to xinttools.

1.2i. rewrote this venerable macro. New code about 40% faster across all lengths.

```
\def\xintLength {\romannumeral0\xintlength }%
\def\xintlength #1{\long\def\xintlength ##1%
{%
    \expandafter#1\the\numexpr\XINT_length_loop
    ##1\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
       \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
       \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
    \relax
}}\xintlength{ }%
\long\def\XINT_length_loop #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_xint: #9\XINT_length_finish_a\xint:
    \xint_c_ix+\XINT_length_loop
}%
\def\XINT_length_finish_a\xint:\xint_c_ix+\XINT_length_loop
    #1#2#3#4#5#6#7#8#9%
{%
    #9\xint_bye
}%
```

2.14 \xintLastItem

1.2i (2016/12/10). Output empty if input empty. One level of braces removed in output. Does not expand its argument.

```
\def\xintLastItem {\romannumeral0\xintlastitem }%
\long\def\xintlastitem #1%
{%
    \XINT_last_loop {}.#1%
    {\xint:\XINT_last_loop_enda}{\xint:\XINT_last_loop_endb}%
    {\xint:\XINT_last_loop_endc}{\xint:\XINT_last_loop_endd}%
    {\xint:\XINT_last_loop_ende}{\xint:\XINT_last_loop_endf}%
    {\xint:\XINT_last_loop_endg}{\xint:\XINT_last_loop_endh}\xint_bye
}%
\long\def\XINT_last_loop #1.#2#3#4#5#6#7#8#9%
```

```
{%
    \xint_gob_til_xint: #9%
        {#8}{#7}{#6}{#5}{#4}{#3}{#2}{#1}\xint:
    \XINT_last_loop {#9}.%
}%
\long\def\XINT_last_loop_enda #1#2\xint_bye{ #1}%
\long\def\XINT_last_loop_endb #1#2#3\xint_bye{ #2}%
\long\def\XINT_last_loop_endc #1#2#3#4\xint_bye{ #3}%
\long\def\XINT_last_loop_endd #1#2#3#4#5\xint_bye{ #4}%
\long\def\XINT_last_loop_ende #1#2#3#4#5#6\xint_bye{ #5}%
\long\def\XINT_last_loop_endf #1#2#3#4#5#6#7\xint_bye{ #6}%
\long\def\XINT_last_loop_endg #1#2#3#4#5#6#7#8\xint_bye{ #7}%
\long\def\XINT_last_loop_endh #1#2#3#4#5#6#7#8#9\xint_bye{ #8}%
```

2.15 `\xintLengthUpTo`

1.2i. for use by \xintKeep and \xintTrim (xinttools). The argument N **must be non-negative**.

\xintLengthUpTo{N}{List} produces -0 if length(List)>N, else it returns N-length(List). Hence subtracting it from N always computes min(N,length(List)).

1.2j. changed ending and interface to core loop.

```
\def\xintLengthUpTo {\romannumeral0\xintlengthupto}%
\long\def\xintlengthupto #1#2%
{%
    \expandafter\XINT_lengthupto_loop
    \the\numexpr#1.#2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
         \xint_c_vii\xint_c_vi\xint_c_v\xint_c_iv
         \xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye.%
}%
\def\XINT_lengthupto_loop_a #1%
{%
    \xint_UDsignfork
      #1\XINT_lengthupto_gt
      -\XINT_lengthupto_loop
    \krof #1%
}%
\long\def\XINT_lengthupto_gt #1\xint_bye.{-0}%
\long\def\XINT_lengthupto_loop #1.#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_xint: #9\XINT_lengthupto_finish_a\xint:%
    \expandafter\XINT_lengthupto_loop_a\the\numexpr #1-\xint_c_viii.%
}%
\def\XINT_lengthupto_finish_a\xint:\expandafter\XINT_lengthupto_loop_a
    \the\numexpr #1-\xint_c_viii.#2#3#4#5#6#7#8#9%
{%
    \expandafter\XINT_lengthupto_finish_b\the\numexpr #1-#9\xint_bye
}%
\def\XINT_lengthupto_finish_b #1#2.%
{%
    \xint_UDsignfork
       #1{-0}%
        -{ #1#2}%
    \krof
}%
```

2.16 \xintreplicate

1.2i.

This is cloned from LaTeX3's \prg_replicate:nn, see Joseph's post at

http://tex.stackexchange.com/questions/16189/repeat-command-n-times

I posted there an alternative not using the chained \csname's but it is a bit less efficient (except perhaps for thousands of repetitions). The code in Joseph's post does abs(#1) replications when input #1 is negative and then activates an error triggering macro; here we simply do nothing when #1 is negative.

Usage: \romannumeral\xintreplicate{N}{stuff}

When N is already explicit digits (even N=0, but non-negative) one can call the macro as

\romannumeral\XINT_rep N\endcsname {foo}

to skip the \numexpr.

```
\def\xintreplicate#1%
   {\expandafter\XINT_replicate\the\numexpr#1\endcsname}%
\def\XINT_replicate #1{\xint_UDsignfork
                         #1\XINT_rep_neg
                          -\XINT_rep
                       \krof #1}%
\long\def\XINT_rep_neg #1\endcsname #2{\xint_c_}%
\def\XINT_rep   #1{\csname XINT_rep_f#1\XINT_rep_a}%
\def\XINT_rep_a #1{\csname XINT_rep_#1\XINT_rep_a}%
\def\XINT_rep_\XINT_rep_a{\endcsname}%
\long\expandafter\def\csname XINT_rep_0\endcsname #1%
    {\endcsname{#1#1#1#1#1#1#1#1#1#1}}%
\long\expandafter\def\csname XINT_rep_1\endcsname #1%
    {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1}%
\long\expandafter\def\csname XINT_rep_2\endcsname #1%
    {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1}%
\long\expandafter\def\csname XINT_rep_3\endcsname #1%
    {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1}%
\long\expandafter\def\csname XINT_rep_4\endcsname #1%
    {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1}%
\long\expandafter\def\csname XINT_rep_5\endcsname #1%
    {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1#1}%
\long\expandafter\def\csname XINT_rep_6\endcsname #1%
    {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1#1#1}%
\long\expandafter\def\csname XINT_rep_7\endcsname #1%
    {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1#1#1#1}%
\long\expandafter\def\csname XINT_rep_8\endcsname #1%
    {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1#1#1#1#1}%
\long\expandafter\def\csname XINT_rep_9\endcsname #1%
    {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1#1#1#1#1#1}%
\long\expandafter\def\csname XINT_rep_f0\endcsname #1%
    {\xint_c_}%
\long\expandafter\def\csname XINT_rep_f1\endcsname #1%
    {\xint_c_ #1}%
\long\expandafter\def\csname XINT_rep_f2\endcsname #1%
    {\xint_c_ #1#1}%
\long\expandafter\def\csname XINT_rep_f3\endcsname #1%
```

```
    {\xint_c_ #1#1#1}%
\long\expandafter\def\csname XINT_rep_f4\endcsname #1%
    {\xint_c_ #1#1#1#1}%
\long\expandafter\def\csname XINT_rep_f5\endcsname #1%
    {\xint_c_ #1#1#1#1#1}%
\long\expandafter\def\csname XINT_rep_f6\endcsname #1%
    {\xint_c_ #1#1#1#1#1#1}%
\long\expandafter\def\csname XINT_rep_f7\endcsname #1%
    {\xint_c_ #1#1#1#1#1#1#1}%
\long\expandafter\def\csname XINT_rep_f8\endcsname #1%
    {\xint_c_ #1#1#1#1#1#1#1#1}%
\long\expandafter\def\csname XINT_rep_f9\endcsname #1%
    {\xint_c_ #1#1#1#1#1#1#1#1#1}%
```

2.17 \xintgobble

1.2i.

I hesitated about allowing as many as 9^6-1=531440 tokens to gobble, but 9^5-1=59058 is too low for playing with long decimal expansions.

Usage: \romannumeral\xintgobble{N}...

```
\def\xintgobble #1%
   {\csname xint_c_\expandafter\XINT_gobble_a\the\numexpr#1.0}%
\def\XINT_gobble #1.{\csname xint_c_\XINT_gobble_a #1.0}%
\def\XINT_gobble_a #1{\xint_gob_til_zero#1\XINT_gobble_d0\XINT_gobble_b#1}%
\def\XINT_gobble_b #1.#2%
  {\expandafter\XINT_gobble_c
      \the\numexpr (#1+\xint_c_v)/\xint_c_ix-\xint_c_i\expandafter.%
      \the\numexpr #2+\xint_c_i.#1.}%
\def\XINT_gobble_c #1.#2.#3.%
  {\csname XINT_g#2\the\numexpr#3-\xint_c_ix*#1\relax\XINT_gobble_a #1.#2}%
\def\XINT_gobble_d0\XINT_gobble_b0.#1{\endcsname}%
\expandafter\let\csname XINT_g10\endcsname\endcsname
\long\expandafter\def\csname XINT_g11\endcsname#1{\endcsname}%
\long\expandafter\def\csname XINT_g12\endcsname#1#2{\endcsname}%
\long\expandafter\def\csname XINT_g13\endcsname#1#2#3{\endcsname}%
\long\expandafter\def\csname XINT_g14\endcsname#1#2#3#4{\endcsname}%
\long\expandafter\def\csname XINT_g15\endcsname#1#2#3#4#5{\endcsname}%
\long\expandafter\def\csname XINT_g16\endcsname#1#2#3#4#5#6{\endcsname}%
\long\expandafter\def\csname XINT_g17\endcsname#1#2#3#4#5#6#7{\endcsname}%
\long\expandafter\def\csname XINT_g18\endcsname#1#2#3#4#5#6#7#8{\endcsname}%
\expandafter\let\csname XINT_g20\endcsname\endcsname
\long\expandafter\def\csname XINT_g21\endcsname #1#2#3#4#5#6#7#8#9%
 {\endcsname}%
\long\expandafter\edef\csname XINT_g22\endcsname #1#2#3#4#5#6#7#8#9%
 {\expandafter\noexpand\csname XINT_g21\endcsname}%
\long\expandafter\edef\csname XINT_g23\endcsname #1#2#3#4#5#6#7#8#9%
 {\expandafter\noexpand\csname XINT_g22\endcsname}%
\long\expandafter\edef\csname XINT_g24\endcsname #1#2#3#4#5#6#7#8#9%
 {\expandafter\noexpand\csname XINT_g23\endcsname}%
\long\expandafter\edef\csname XINT_g25\endcsname #1#2#3#4#5#6#7#8#9%
 {\expandafter\noexpand\csname XINT_g24\endcsname}%
\long\expandafter\edef\csname XINT_g26\endcsname #1#2#3#4#5#6#7#8#9%
```

```
 {\expandafter\noexpand\csname XINT_g25\endcsname}%
\long\expandafter\edef\csname XINT_g27\endcsname #1#2#3#4#5#6#7#8#9%
 {\expandafter\noexpand\csname XINT_g26\endcsname}%
\long\expandafter\edef\csname XINT_g28\endcsname #1#2#3#4#5#6#7#8#9%
 {\expandafter\noexpand\csname XINT_g27\endcsname}%
\expandafter\let\csname XINT_g30\endcsname\endcsname
\long\expandafter\edef\csname XINT_g31\endcsname #1#2#3#4#5#6#7#8#9%
 {\expandafter\noexpand\csname XINT_g28\endcsname}%
\long\expandafter\edef\csname XINT_g32\endcsname #1#2#3#4#5#6#7#8#9%
 {\noexpand\csname XINT_g31\expandafter\noexpand\csname XINT_g28\endcsname}%
\long\expandafter\edef\csname XINT_g33\endcsname #1#2#3#4#5#6#7#8#9%
 {\noexpand\csname XINT_g32\expandafter\noexpand\csname XINT_g28\endcsname}%
\long\expandafter\edef\csname XINT_g34\endcsname #1#2#3#4#5#6#7#8#9%
 {\noexpand\csname XINT_g33\expandafter\noexpand\csname XINT_g28\endcsname}%
\long\expandafter\edef\csname XINT_g35\endcsname #1#2#3#4#5#6#7#8#9%
 {\noexpand\csname XINT_g34\expandafter\noexpand\csname XINT_g28\endcsname}%
\long\expandafter\edef\csname XINT_g36\endcsname #1#2#3#4#5#6#7#8#9%
 {\noexpand\csname XINT_g35\expandafter\noexpand\csname XINT_g28\endcsname}%
\long\expandafter\edef\csname XINT_g37\endcsname #1#2#3#4#5#6#7#8#9%
 {\noexpand\csname XINT_g36\expandafter\noexpand\csname XINT_g28\endcsname}%
\long\expandafter\edef\csname XINT_g38\endcsname #1#2#3#4#5#6#7#8#9%
 {\noexpand\csname XINT_g37\expandafter\noexpand\csname XINT_g28\endcsname}%
\expandafter\let\csname XINT_g40\endcsname\endcsname
\expandafter\edef\csname XINT_g41\endcsname
 {\noexpand\csname XINT_g38\expandafter\noexpand\csname XINT_g31\endcsname}%
\expandafter\edef\csname XINT_g42\endcsname
 {\noexpand\csname XINT_g41\expandafter\noexpand\csname XINT_g41\endcsname}%
\expandafter\edef\csname XINT_g43\endcsname
 {\noexpand\csname XINT_g42\expandafter\noexpand\csname XINT_g41\endcsname}%
\expandafter\edef\csname XINT_g44\endcsname
 {\noexpand\csname XINT_g43\expandafter\noexpand\csname XINT_g41\endcsname}%
\expandafter\edef\csname XINT_g45\endcsname
 {\noexpand\csname XINT_g44\expandafter\noexpand\csname XINT_g41\endcsname}%
\expandafter\edef\csname XINT_g46\endcsname
 {\noexpand\csname XINT_g45\expandafter\noexpand\csname XINT_g41\endcsname}%
\expandafter\edef\csname XINT_g47\endcsname
 {\noexpand\csname XINT_g46\expandafter\noexpand\csname XINT_g41\endcsname}%
\expandafter\edef\csname XINT_g48\endcsname
 {\noexpand\csname XINT_g47\expandafter\noexpand\csname XINT_g41\endcsname}%
\expandafter\let\csname XINT_g50\endcsname\endcsname
\expandafter\edef\csname XINT_g51\endcsname
 {\noexpand\csname XINT_g48\expandafter\noexpand\csname XINT_g41\endcsname}%
\expandafter\edef\csname XINT_g52\endcsname
 {\noexpand\csname XINT_g51\expandafter\noexpand\csname XINT_g51\endcsname}%
\expandafter\edef\csname XINT_g53\endcsname
 {\noexpand\csname XINT_g52\expandafter\noexpand\csname XINT_g51\endcsname}%
\expandafter\edef\csname XINT_g54\endcsname
 {\noexpand\csname XINT_g53\expandafter\noexpand\csname XINT_g51\endcsname}%
\expandafter\edef\csname XINT_g55\endcsname
 {\noexpand\csname XINT_g54\expandafter\noexpand\csname XINT_g51\endcsname}%
\expandafter\edef\csname XINT_g56\endcsname
 {\noexpand\csname XINT_g55\expandafter\noexpand\csname XINT_g51\endcsname}%
```

```
\expandafter\edef\csname XINT_g57\endcsname
 {\noexpand\csname XINT_g56\expandafter\noexpand\csname XINT_g51\endcsname}%
\expandafter\edef\csname XINT_g58\endcsname
 {\noexpand\csname XINT_g57\expandafter\noexpand\csname XINT_g51\endcsname}%
\expandafter\let\csname XINT_g60\endcsname\endcsname
\expandafter\edef\csname XINT_g61\endcsname
 {\noexpand\csname XINT_g58\expandafter\noexpand\csname XINT_g51\endcsname}%
\expandafter\edef\csname XINT_g62\endcsname
 {\noexpand\csname XINT_g61\expandafter\noexpand\csname XINT_g61\endcsname}%
\expandafter\edef\csname XINT_g63\endcsname
 {\noexpand\csname XINT_g62\expandafter\noexpand\csname XINT_g61\endcsname}%
\expandafter\edef\csname XINT_g64\endcsname
 {\noexpand\csname XINT_g63\expandafter\noexpand\csname XINT_g61\endcsname}%
\expandafter\edef\csname XINT_g65\endcsname
 {\noexpand\csname XINT_g64\expandafter\noexpand\csname XINT_g61\endcsname}%
\expandafter\edef\csname XINT_g66\endcsname
 {\noexpand\csname XINT_g65\expandafter\noexpand\csname XINT_g61\endcsname}%
\expandafter\edef\csname XINT_g67\endcsname
 {\noexpand\csname XINT_g66\expandafter\noexpand\csname XINT_g61\endcsname}%
\expandafter\edef\csname XINT_g68\endcsname
 {\noexpand\csname XINT_g67\expandafter\noexpand\csname XINT_g61\endcsname}%
```

2.18 (WIP) \xintUniformDeviate

1.3b. See user manual for related information.

```
\ifdefined\xint_texuniformdeviate
     \expandafter\xint_firstoftwo
\else\expandafter\xint_secondoftwo
\fi
{%
  \def\xintUniformDeviate#1%
     {\the\numexpr\expandafter\XINT_uniformdeviate_sgnfork\the\numexpr#1\xint:}%
  \def\XINT_uniformdeviate_sgnfork#1%
  {%
     \if-#1\XINT_uniformdeviate_neg\fi \XINT_uniformdeviate{}#1%
  }%
  \def\XINT_uniformdeviate_neg\fi\XINT_uniformdeviate#1-%
  {%
      \fi-\numexpr\XINT_uniformdeviate\relax
  }%
  \def\XINT_uniformdeviate#1#2\xint:
  {%(
      \expandafter\XINT_uniformdeviate_a\the\numexpr%
                   -\xint_texuniformdeviate\xint_c_ii^vii%
                   -\xint_c_ii^vii*\xint_texuniformdeviate\xint_c_ii^vii%
                   -\xint_c_ii^xiv*\xint_texuniformdeviate\xint_c_ii^vii%
                   -\xint_c_ii^xxi*\xint_texuniformdeviate\xint_c_ii^vii%
                   +\xint_texuniformdeviate#2\xint:/#2)*#2\xint:+#2\fi\relax#1%
  }%
  \def\XINT_uniformdeviate_a #1\xint:
  {%
      \expandafter\XINT_uniformdeviate_b\the\numexpr#1-(#1%
  }%
  \def\XINT_uniformdeviate_b#1#2\xint:{#1#2\if-#1}%
}%
{%
  \def\xintUniformDeviate#1%
  {%
      \the\numexpr
      \XINT_expandableerror{No uniformdeviate at engine level, returning 0.}%
      0\relax
  }%
}%
```

2.19 \xintMessage, \ifxintverbose

1.2c. for use by \xintdefvar and \xintdeffunc of xintexpr.

1.2e. uses \write128 rather than \write16 for compatibility with future extended range of output streams, in LuaTeX in particular.

```
\def\xintMessage #1#2#3{%
    \immediate\write128{Package #1 #2: (on line \the\inputlineno)}%
    \immediate\write128{\space\space\space\space#3}%
}%
\newif\ifxintverbose
```

2.20 \ifxintglobaldefs, \XINT_global

1.3c.

```
\newif\ifxintglobaldefs
\def\XINT_global{\ifxintglobaldefs\global\fi}%
```

2.21 (WIP) Expandable error message

1.2l. but really belongs to next major release beyond 1.3.

This is copied over from l3kernel code. I am using \ ! / control sequence though, which must be left undefined. \xintError: would be 6 letters more already.

```
\def\XINT_expandableerror #1#2{%
    \def\XINT_expandableerror ##1{%
        \expandafter\expandafter\expandafter
        \XINT_expandableerror_continue\xint_firstofone{#2#1##1#1}}%
    \def\XINT_expandableerror_continue ##1#1##2#1{##1}%
}%
\begingroup\lccode`$ 32 \catcode`/ 11 \catcode`! 11 \catcode32 11 % $
\lowercase{\endgroup\XINT_expandableerror$\ ! /\let\ ! /\xint_undefined}% $
\XINT_restorecatcodes_endinput%
```

3 Package xinttools implementation

Release 1.09g of 2013/11/22 splits off xinttools.sty from xint.sty. Starting with 1.1, xinttools ceases being loaded automatically by xint.

3.1 Catcodes, ε-TEX and reload detection

The code for reload detection was initially copied from Heiko Oberdiek's packages, then modified.
The method for catcodes was also initially directly inspired by these packages.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode64=11   % @
  \catcode35=6    % #
  \catcode44=12   % ,
  \catcode45=12   % -
  \catcode46=12   % .
  \catcode58=12   % :
  \let\z\endgroup
  \expandafter\let\expandafter\x\csname ver@xinttools.sty\endcsname
  \expandafter\let\expandafter\w\csname ver@xintkernel.sty\endcsname
  \expandafter
    \ifx\csname PackageInfo\endcsname\relax
      \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
    \else
      \def\y#1#2{\PackageInfo{#1}{#2}}%
    \fi
  \expandafter
  \ifx\csname numexpr\endcsname\relax
     \y{xinttools}{\numexpr not available, aborting input}%
     \aftergroup\endinput
  \else
    \ifx\x\relax   % plain-TeX, first loading of xinttools.sty
      \ifx\w\relax % but xintkernel.sty not yet loaded.
         \def\z{\endgroup\input xintkernel.sty\relax}%
      \fi
    \else
      \def\empty {}%
      \ifx\x\empty % LaTeX, first loading,
      % variable is initialized, but \ProvidesPackage not yet seen
          \ifx\w\relax % xintkernel.sty not yet loaded.
            \def\z{\endgroup\RequirePackage{xintkernel}}%
          \fi
      \else
        \aftergroup\endinput % xinttools already loaded.
      \fi
    \fi
  \fi
\z%
\XINTsetupcatcodes% defined in xintkernel.sty
```

3.2 Package identification

```
\XINT_providespackage
\ProvidesPackage{xinttools}%
  [2018/06/17 1.3c Expandable and non-expandable utilities (JFB)]%
```

\XINT_toks is used in macros such as \xintFor. It is not used elsewhere in the xint bundle.

```
\newtoks\XINT_toks
\xint_firstofone{\let\XINT_sptoken= } %<- space here!
```

3.3 `\xintgodef`, `\xintgoodef`, `\xintgfdef`

1.09i. For use in \xintAssign.

```
\def\xintgodef  {\global\xintodef }%
\def\xintgoodef {\global\xintoodef }%
\def\xintgfdef  {\global\xintfdef }%
```

3.4 `\xintRevWithBraces`

New with 1.06. Makes the expansion of its argument and then reverses the resulting tokens or braced tokens, adding a pair of braces to each (thus, maintaining it when it was already there.) The reason for \xint:, here and in other locations, is in case #1 expands to nothing, the \romannumeral-`0 must be stopped

```
\def\xintRevWithBraces         {\romannumeral0\xintrevwithbraces }%
```

```
\def\xintRevWithBracesNoExpand {\romannumeral0\xintrevwithbracesnoexpand }%
\long\def\xintrevwithbraces #1%
{%
    \expandafter\XINT_revwbr_loop\expandafter{\expandafter}%
    \romannumeral`&&@#1\xint:\xint:\xint:\xint:%
                      \xint:\xint:\xint:\xint:\xint_bye
}%
\long\def\xintrevwithbracesnoexpand #1%
{%
    \XINT_revwbr_loop {}%
    #1\xint:\xint:\xint:\xint:%
      \xint:\xint:\xint:\xint:\xint_bye
}%
\long\def\XINT_revwbr_loop #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_xint: #9\XINT_revwbr_finish_a\xint:%
    \XINT_revwbr_loop {{#9}{#8}{#7}{#6}{#5}{#4}{#3}{#2}#1}%
}%
\long\def\XINT_revwbr_finish_a\xint:\XINT_revwbr_loop #1#2\xint_bye
{%
    \XINT_revwbr_finish_b #2\R\R\R\R\R\R\R\Z #1%
}%
\def\XINT_revwbr_finish_b #1#2#3#4#5#6#7#8\Z
{%
    \xint_gob_til_R
            #1\XINT_revwbr_finish_c \xint_gobble_viii
            #2\XINT_revwbr_finish_c \xint_gobble_vii
            #3\XINT_revwbr_finish_c \xint_gobble_vi
            #4\XINT_revwbr_finish_c \xint_gobble_v
            #5\XINT_revwbr_finish_c \xint_gobble_iv
            #6\XINT_revwbr_finish_c \xint_gobble_iii
            #7\XINT_revwbr_finish_c \xint_gobble_ii
            \R\XINT_revwbr_finish_c \xint_gobble_i\Z
}%
```

1.1c revisited this old code and improved upon the earlier endings.

```
\def\XINT_revwbr_finish_c#1{%
\def\XINT_revwbr_finish_c##1##2\Z{\expandafter#1##1}%
}\XINT_revwbr_finish_c{ }%
```

3.5 \xintZapFirstSpaces

1.09f, written [2013/11/01]. Modified (2014/10/21) for release 1.1 to correct the bug in case of an empty argument, or argument containing only spaces, which had been forgotten in first version. New version is simpler than the initial one. This macro does NOT expand its argument.

```
\def\xintZapFirstSpaces {\romannumeral0\xintzapfirstspaces }%
\def\xintzapfirstspaces#1{\long
\def\xintzapfirstspaces ##1{\XINT_zapbsp_a #1##1\xint:#1#1\xint:}%
}\xintzapfirstspaces{ }%
```

If the original #1 started with a space, the grabbed #1 is empty. Thus _again? will see #1=\xint_bye, and hand over control to _again which will loop back into \XINT_zapbsp_a, with one initial space less. If the original #1 did not start with a space, or was empty, then the #1 below will be a <sptoken>, then an extract of the original #1, not empty and not starting with a space, which contains what was up to the first <sp><sp> present in original #1, or, if none preexisted, <sptoken> and all of #1 (possibly empty) plus an ending \xint:. The added initial space will stop later the \romannumeral0. No brace stripping is possible. Control is handed over to \XINT_zapbsp_b which strips out the ending \xint:<sp><sp>\xint:

```
\def\XINT_zapbsp_a#1{\long\def\XINT_zapbsp_a ##1#1#1{%
  \XINT_zapbsp_again?##1\xint_bye\XINT_zapbsp_b ##1#1#1}%
}\XINT_zapbsp_a{ }%
\long\def\XINT_zapbsp_again? #1{\xint_bye #1\XINT_zapbsp_again }%
\xint_firstofone{\def\XINT_zapbsp_again\XINT_zapbsp_b} {\XINT_zapbsp_a }%
\long\def\XINT_zapbsp_b #1\xint:#2\xint:{#1}%
```

3.6 \xintZapLastSpaces

1.09f, written [2013/11/01].

```
\def\xintZapLastSpaces {\romannumeral0\xintzaplastspaces }%
\def\xintzaplastspaces#1{\long
\def\xintzaplastspaces ##1{\XINT_zapesp_a {}\empty##1#1#1\xint_bye\xint:}%
}\xintzaplastspaces{ }%
```

The \empty from \xintzaplastspaces is to prevent brace removal in the #2 below. The \expandafter chain removes it.

```
\xint_firstofone {\long\def\XINT_zapesp_a #1#2 } %<- second space here
    {\expandafter\XINT_zapesp_b\expandafter{#2}{#1}}%
```

Notice again an \empty added here. This is in preparation for possibly looping back to \XINT_zapesp_a. If the initial #1 had no <sp><sp>, the stuff however will not loop, because #3 will already be <some spaces>\xint_bye. Notice that this macro fetches all way to the ending \xint:. This looks not very efficient, but how often do we have to strip ending spaces from something which also has inner stretches of _multiple_ space tokens ?;-).

```
\long\def\XINT_zapesp_b #1#2#3\xint:%
    {\XINT_zapesp_end? #3\XINT_zapesp_e {#2#1}\empty #3\xint:}%
```

When we have been over all possible <sp><sp> things, we reach the ending space tokens, and #3 will be a bunch of spaces (possibly none) followed by \xint_bye. So the #1 in _end? will be \xint_bye. In all other cases #1 can not be \xint_bye (assuming naturally this token does nor arise in original input), hence control falls back to \XINT_zapesp_e which will loop back to \XINT_zapesp_a.

```
\long\def\XINT_zapesp_end? #1{\xint_bye #1\XINT_zapesp_end }%
```

We are done. The #1 here has accumulated all the previous material, and is stripped of its ending spaces, if any.

```
\long\def\XINT_zapesp_end\XINT_zapesp_e #1#2\xint:{ #1}%
```

We haven't yet reached the end, so we need to re-inject two space tokens after what we have gotten so far. Then we loop.

```
\def\XINT_zapesp_e#1{%
\long\def\XINT_zapesp_e ##1{\XINT_zapesp_a {##1#1#1}}%
}\XINT_zapesp_e{ }%
```

3.7 \xintZapSpaces

1.09f, written [2013/11/01]. Modified for 1.1, 2014/10/21 as it has the same bug as \xintZapFirstSpaces. We in effect do first \xintZapFirstSpaces, then \xintZapLastSpaces.

```
\def\xintZapSpaces {\romannumeral0\xintzapspaces }%
\def\xintzapspaces#1{%
\long\def\xintzapspaces ##1% like \xintZapFirstSpaces.
        {\XINT_zapsp_a #1##1\xint:#1#1\xint:}%
}\xintzapspaces{ }%
\def\XINT_zapsp_a#1{%
\long\def\XINT_zapsp_a ##1#1#1%
        {\XINT_zapsp_again?##1\xint_bye\XINT_zapsp_b##1#1#1}%
}\XINT_zapsp_a{ }%
\long\def\XINT_zapsp_again? #1{\xint_bye #1\XINT_zapsp_again }%
\xint_firstofone{\def\XINT_zapsp_again\XINT_zapsp_b} {\XINT_zapsp_a }%
\xint_firstofone{\def\XINT_zapsp_b} {\XINT_zapsp_c }%
\def\XINT_zapsp_c#1{%
\long\def\XINT_zapsp_c ##1\xint:##2\xint:%
        {\XINT_zapesp_a{}\empty ##1#1#1\xint_bye\xint:}%
}\XINT_zapsp_c{ }%
```

3.8 \xintZapSpacesB

1.09f, written [2013/11/01]. Strips up to one pair of braces (but then does not strip spaces inside).

```
\def\xintZapSpacesB {\romannumeral0\xintzapspacesb }%
\long\def\xintzapspacesb #1{\XINT_zapspb_one? #1\xint:\xint:%
                         \xint_bye\xintzapspaces {#1}}%
\long\def\XINT_zapspb_one? #1#2%
   {\xint_gob_til_xint: #1\XINT_zapspb_onlyspaces\xint:%
    \xint_gob_til_xint: #2\XINT_zapspb_bracedorone\xint:%
    \xint_bye {#1}}%
\def\XINT_zapspb_onlyspaces\xint:%
    \xint_gob_til_xint:\xint:\XINT_zapspb_bracedorone\xint:%
    \xint_bye #1\xint_bye\xintzapspaces #2{ }%
\long\def\XINT_zapspb_bracedorone\xint:%
    \xint_bye #1\xint:\xint_bye\xintzapspaces #2{ #1}%
```

3.9 \xintCSVtoList, \xintCSVtoListNonStripped

\xintCSVtoList transforms a,b,..,z into {a}{b}...{z}. The comma separated list may be a macro which is first f-expanded. First included in release 1.06. Here, use of \Z (and \R) perfectly safe.

[2013/11/02]: Starting with 1.09f, automatically filters items with \xintZapSpacesB to strip away all spaces around commas, and spaces at the start and end of the list. The original is kept as \xintCSVtoListNonStripped, and is faster. But ... it doesn't strip spaces.

```
\def\xintCSVtoList {\romannumeral0\xintcsvtolist }%
\long\def\xintcsvtolist #1{\expandafter\xintApply
           \expandafter\xintzapspacesb
           \expandafter{\romannumeral0\xintcsvtolistnonstripped{#1}}}%
```

```
\def\xintCSVtoListNoExpand {\romannumeral0\xintcsvtolistnoexpand }%
\long\def\xintcsvtolistnoexpand #1{\expandafter\xintApply
           \expandafter\xintzapspacesb
           \expandafter{\romannumeral0\xintcsvtolistnonstrippednoexpand{#1}}}%
\def\xintCSVtoListNonStripped {\romannumeral0\xintcsvtolistnonstripped }%
\def\xintCSVtoListNonStrippedNoExpand
         {\romannumeral0\xintcsvtolistnonstrippednoexpand }%
\long\def\xintcsvtolistnonstripped #1%
{%
    \expandafter\XINT_csvtol_loop_a\expandafter
    {\expandafter}\romannumeral`&&@#1%
        ,\xint_bye,\xint_bye,\xint_bye,\xint_bye
        ,\xint_bye,\xint_bye,\xint_bye,\xint_bye,\Z
}%
\long\def\xintcsvtolistnonstrippednoexpand #1%
{%
    \XINT_csvtol_loop_a
    {}#1,\xint_bye,\xint_bye,\xint_bye,\xint_bye
        ,\xint_bye,\xint_bye,\xint_bye,\xint_bye,\Z
}%
\long\def\XINT_csvtol_loop_a #1#2,#3,#4,#5,#6,#7,#8,#9,%
{%
    \xint_bye #9\XINT_csvtol_finish_a\xint_bye
    \XINT_csvtol_loop_b {#1}{{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}}%
}%
\long\def\XINT_csvtol_loop_b #1#2{\XINT_csvtol_loop_a {#1#2}}%
\long\def\XINT_csvtol_finish_a\xint_bye\XINT_csvtol_loop_b #1#2#3\Z
{%
    \XINT_csvtol_finish_b #3\R,\R,\R,\R,\R,\R,\R,\Z #2{#1}%
}%
```

1.1c revisits this old code and improves upon the earlier endings. But as the _d.. macros have already nine parameters, I needed the \expandafter and \xint_gob_til_Z in finish_b (compare \XINT_keep_endb, or also \XINT_RQ_end_b).

```
\def\XINT_csvtol_finish_b #1,#2,#3,#4,#5,#6,#7,#8\Z
{%
    \xint_gob_til_R
             #1\expandafter\XINT_csvtol_finish_dviii\xint_gob_til_Z
             #2\expandafter\XINT_csvtol_finish_dvii \xint_gob_til_Z
             #3\expandafter\XINT_csvtol_finish_dvi  \xint_gob_til_Z
             #4\expandafter\XINT_csvtol_finish_dv   \xint_gob_til_Z
             #5\expandafter\XINT_csvtol_finish_div  \xint_gob_til_Z
             #6\expandafter\XINT_csvtol_finish_diii \xint_gob_til_Z
             #7\expandafter\XINT_csvtol_finish_dii  \xint_gob_til_Z
             \R\XINT_csvtol_finish_di \Z
}%
\long\def\XINT_csvtol_finish_dviii #1#2#3#4#5#6#7#8#9{ #9}%
\long\def\XINT_csvtol_finish_dvii  #1#2#3#4#5#6#7#8#9{ #9{#1}}%
\long\def\XINT_csvtol_finish_dvi   #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}}%
\long\def\XINT_csvtol_finish_dv    #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}}%
\long\def\XINT_csvtol_finish_div   #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}{#4}}%
\long\def\XINT_csvtol_finish_diii  #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}{#4}{#5}}%
```

```
\long\def\XINT_csvtol_finish_dii   #1#2#3#4#5#6#7#8#9%
                                            { #9{#1}{#2}{#3}{#4}{#5}{#6}}%
\long\def\XINT_csvtol_finish_di\Z  #1#2#3#4#5#6#7#8#9%
                                            { #9{#1}{#2}{#3}{#4}{#5}{#6}{#7}}%
```

3.10 \xintListWithSep

```
1.04. \xintListWithSep {\sep}{{a}{b}...{z}} returns a \sep b \sep ....\sep z. It f-expands its
second argument. The 'sep' may be \par's: the macro \xintlistwithsep etc... are all declared long.
'sep' does not have to be a single token. It is not expanded. The "list" argument may be empty.
  \xintListWithSepNoExpand does not f-expand its second argument.
  This venerable macro from 1.04 remained unchanged for a long time and was finally refactored at
1.2p for increased speed. Tests done with a list of identical {\x} items and a sep of \z demon-
strated a speed increase of about:
   - 3x for 30 items,
   - 4.5x for 100 items,
   - 7.5x--8x for 1000 items.
```

```
\def\xintListWithSep         {\romannumeral0\xintlistwithsep }%
\def\xintListWithSepNoExpand {\romannumeral0\xintlistwithsepnoexpand }%
\long\def\xintlistwithsep #1#2%
    {\expandafter\XINT_lws\expandafter {\romannumeral`&&@#2}{#1}}%
\long\def\xintlistwithsepnoexpand #1#2%
{%
    \XINT_lws_loop_a {#1}#2{\xint_bye\XINT_lws_e_vi}%
       {\xint_bye\XINT_lws_e_v}{\xint_bye\XINT_lws_e_iv}%
       {\xint_bye\XINT_lws_e_iii}{\xint_bye\XINT_lws_e_ii}%
       {\xint_bye\XINT_lws_e_i}{\xint_bye\XINT_lws_e}%
       {\xint_bye\expandafter\space}\xint_bye
}%
\long\def\XINT_lws #1#2%
{%
    \XINT_lws_loop_a {#2}#1{\xint_bye\XINT_lws_e_vi}%
       {\xint_bye\XINT_lws_e_v}{\xint_bye\XINT_lws_e_iv}%
       {\xint_bye\XINT_lws_e_iii}{\xint_bye\XINT_lws_e_ii}%
       {\xint_bye\XINT_lws_e_i}{\xint_bye\XINT_lws_e}%
       {\xint_bye\expandafter\space}\xint_bye
}%
\long\def\XINT_lws_loop_a #1#2#3#4#5#6#7#8#9%
{%
    \xint_bye #9\xint_bye
    \XINT_lws_loop_b {#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}%
}%
\long\def\XINT_lws_loop_b #1#2#3#4#5#6#7#8#9%
{%
    \XINT_lws_loop_a {#1}{#2#1#3#1#4#1#5#1#6#1#7#1#8#1#9}%
}%
\long\def\XINT_lws_e_vi\xint_bye\XINT_lws_loop_b #1#2#3#4#5#6#7#8#9\xint_bye
    { #2#1#3#1#4#1#5#1#6#1#7#1#8}%
\long\def\XINT_lws_e_v\xint_bye\XINT_lws_loop_b #1#2#3#4#5#6#7#8\xint_bye
    { #2#1#3#1#4#1#5#1#6#1#7}%
\long\def\XINT_lws_e_iv\xint_bye\XINT_lws_loop_b #1#2#3#4#5#6#7\xint_bye
    { #2#1#3#1#4#1#5#1#6}%
```

```
\long\def\XINT_lws_e_iii\xint_bye\XINT_lws_loop_b #1#2#3#4#5#6\xint_bye
    { #2#1#3#1#4#1#5}%
\long\def\XINT_lws_e_ii\xint_bye\XINT_lws_loop_b #1#2#3#4#5\xint_bye
    { #2#1#3#1#4}%
\long\def\XINT_lws_e_i\xint_bye\XINT_lws_loop_b #1#2#3#4\xint_bye
    { #2#1#3}%
\long\def\XINT_lws_e\xint_bye\XINT_lws_loop_b #1#2#3\xint_bye
    { #2}%
```

3.11 \xintNthElt

First included in release 1.06. Last refactored in 1.2j.

\xintNthElt {i}{List} returns the i th item from List (one pair of braces removed). The list is first f-expanded. The \xintNthEltNoExpand does no expansion of its second argument. Both variants expand i inside \numexpr.

With i = 0, the number of items is returned using \xintLength but with the List argument f-expanded first.

Negative values return the |i|th element from the end.

When i is out of range, an empty value is returned.

```
\def\xintNthElt         {\romannumeral0\xintnthelt }%
\def\xintNthEltNoExpand {\romannumeral0\xintntheltnoexpand }%
\long\def\xintnthelt #1#2{\expandafter\XINT_nthelt_a\the\numexpr #1\expandafter.%
                         \expandafter{\romannumeral`&&@#2}}%
\def\xintntheltnoexpand #1{\expandafter\XINT_nthelt_a\the\numexpr #1.}%
\def\XINT_nthelt_a #1%
{%
    \xint_UDzerominusfork
        #1-\XINT_nthelt_zero
        0#1\XINT_nthelt_neg
         0-{\XINT_nthelt_pos #1}%
    \krof
}%
\def\XINT_nthelt_zero #1.{\xintlength }%
\long\def\XINT_nthelt_neg #1.#2%
{%
    \expandafter\XINT_nthelt_neg_a\the\numexpr\xint_c_i+\XINT_length_loop
    #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
      \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
      \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
    -#1.#2\xint_bye
}%
\def\XINT_nthelt_neg_a #1%
{%
    \xint_UDzerominusfork
        #1-\xint_bye_thenstop
        0#1\xint_bye_thenstop
         0-{}%
    \krof
    \expandafter\XINT_nthelt_neg_b
    \romannumeral\expandafter\XINT_gobble\the\numexpr-\xint_c_i+#1%
}%
\long\def\XINT_nthelt_neg_b #1#2\xint_bye{ #1}%
```

```
\long\def\XINT_nthelt_pos #1.#2%
{%
    \expandafter\XINT_nthelt_pos_done
    \romannumeral0\expandafter\XINT_trim_loop\the\numexpr#1-\xint_c_x.%
     #2\xint:\xint:\xint:\xint:\xint:%
       \xint:\xint:\xint:\xint:\xint:%
    \xint_bye
}%
\def\XINT_nthelt_pos_done #1{%
\long\def\XINT_nthelt_pos_done ##1##2\xint_bye{%
  \xint_gob_til_xint:##1\expandafter#1\xint_gobble_ii\xint:#1##1}%
}\XINT_nthelt_pos_done{ }%
```

3.12 \xintKeep

First included in release 1.09m.

\xintKeep{i}{L} f-expands its second argument L. It then grabs the first i items from L and discards the rest.

ATTENTION: **each such kept item is returned inside a brace pair** Use \xintKeepUnbraced to avoid that.

For i equal or larger to the number N of items in (expanded) L, the full L is returned (with braced items). For i=0, the macro returns an empty output. For i<0, the macro discards the first N-|i| items. No brace pairs added to the remaining items. For i is less or equal to -N, the full L is returned (with no braces added.)

\xintKeepNoExpand does not expand the L argument.

Prior to 1.2i the code proceeded along a loop with no pre-computation of the length of L, for the i>0 case. The faster 1.2i version takes advantage of novel \xintLengthUpTo from xintkernel.sty.

```
\def\xintKeep         {\romannumeral0\xintkeep }%
\def\xintKeepNoExpand {\romannumeral0\xintkeepnoexpand }%
\long\def\xintkeep #1#2{\expandafter\XINT_keep_a\the\numexpr #1\expandafter.%
                         \expandafter{\romannumeral`&&@#2}}%
\def\xintkeepnoexpand #1{\expandafter\XINT_keep_a\the\numexpr #1.}%
\def\XINT_keep_a #1%
{%
    \xint_UDzerominusfork
        #1-\XINT_keep_keepnone
        0#1\XINT_keep_neg
         0-{\XINT_keep_pos #1}%
    \krof
}%
\long\def\XINT_keep_keepnone .#1{ }%
\long\def\XINT_keep_neg #1.#2%
{%
    \expandafter\XINT_keep_neg_a\the\numexpr
    #1-\numexpr\XINT_length_loop
    #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
      \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
      \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye.#2%
}%
\def\XINT_keep_neg_a #1%
{%
    \xint_UDsignfork
```

```
        #1{\expandafter\space\romannumeral\XINT_gobble}%
         -\XINT_keep_keepall
    \krof
}%
\def\XINT_keep_keepall #1.{ }%
\long\def\XINT_keep_pos #1.#2%
{%
    \expandafter\XINT_keep_loop
    \the\numexpr#1-\XINT_lengthupto_loop
    #1.#2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
         \xint_c_vii\xint_c_vi\xint_c_v\xint_c_iv
         \xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye.%
    -\xint_c_viii.{}#2\xint_bye%
}%
\def\XINT_keep_loop #1#2.%
{%
    \xint_gob_til_minus#1\XINT_keep_loop_end-%
    \expandafter\XINT_keep_loop
    \the\numexpr#1#2-\xint_c_viii\expandafter.\XINT_keep_loop_pickeight
}%
\long\def\XINT_keep_loop_pickeight
     #1#2#3#4#5#6#7#8#9{{#1{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}}}%
\def\XINT_keep_loop_end-\expandafter\XINT_keep_loop
    \the\numexpr-#1-\xint_c_viii\expandafter.\XINT_keep_loop_pickeight
    {\csname XINT_keep_end#1\endcsname}%
\long\expandafter\def\csname XINT_keep_end1\endcsname
   #1#2#3#4#5#6#7#8#9\xint_bye { #1{#2}{#3}{#4}{#5}{#6}{#7}{#8}}%
\long\expandafter\def\csname XINT_keep_end2\endcsname
   #1#2#3#4#5#6#7#8\xint_bye { #1{#2}{#3}{#4}{#5}{#6}{#7}}%
\long\expandafter\def\csname XINT_keep_end3\endcsname
   #1#2#3#4#5#6#7\xint_bye { #1{#2}{#3}{#4}{#5}{#6}}%
\long\expandafter\def\csname XINT_keep_end4\endcsname
   #1#2#3#4#5#6\xint_bye { #1{#2}{#3}{#4}{#5}}%
\long\expandafter\def\csname XINT_keep_end5\endcsname
   #1#2#3#4#5\xint_bye { #1{#2}{#3}{#4}}%
\long\expandafter\def\csname XINT_keep_end6\endcsname
   #1#2#3#4\xint_bye { #1{#2}{#3}}%
\long\expandafter\def\csname XINT_keep_end7\endcsname
   #1#2#3\xint_bye { #1{#2}}%
\long\expandafter\def\csname XINT_keep_end8\endcsname
   #1#2\xint_bye { #1}%
```

3.13 \xintKeepUnbraced

1.2a. Same as \xintKeep but will *not* add (or maintain) brace pairs around the kept items when length(L)>i>0.

The name may cause a mis-understanding: for i<0, (i.e. keeping only trailing items), there is no brace removal at all happening.

Modified for 1.2i like \xintKeep.

```
\def\xintKeepUnbraced         {\romannumeral0\xintkeepunbraced }%
\def\xintKeepUnbracedNoExpand {\romannumeral0\xintkeepunbracednoexpand }%
\long\def\xintkeepunbraced #1#2%
```

```
    {\expandafter\XINT_keepunbr_a\the\numexpr #1\expandafter.%
                            \expandafter{\romannumeral`&&@#2}}%
\def\xintkeepunbracednoexpand #1%
   {\expandafter\XINT_keepunbr_a\the\numexpr #1.}%
\def\XINT_keepunbr_a #1%
{%
    \xint_UDzerominusfork
        #1-\XINT_keep_keepnone
        0#1\XINT_keep_neg
         0-{\XINT_keepunbr_pos #1}%
    \krof
}%
\long\def\XINT_keepunbr_pos #1.#2%
{%
    \expandafter\XINT_keepunbr_loop
    \the\numexpr#1-\XINT_lengthupto_loop
    #1.#2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
         \xint_c_vii\xint_c_vi\xint_c_v\xint_c_iv
         \xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye.%
    -\xint_c_viii.{}#2\xint_bye%
}%
\def\XINT_keepunbr_loop #1#2.%
{%
    \xint_gob_til_minus#1\XINT_keepunbr_loop_end-%
    \expandafter\XINT_keepunbr_loop
    \the\numexpr#1#2-\xint_c_viii\expandafter.\XINT_keepunbr_loop_pickeight
}%
\long\def\XINT_keepunbr_loop_pickeight
     #1#2#3#4#5#6#7#8#9{{#1#2#3#4#5#6#7#8#9}}%
\def\XINT_keepunbr_loop_end-\expandafter\XINT_keepunbr_loop
    \the\numexpr-#1-\xint_c_viii\expandafter.\XINT_keepunbr_loop_pickeight
    {\csname XINT_keepunbr_end#1\endcsname}%
\long\expandafter\def\csname XINT_keepunbr_end1\endcsname
   #1#2#3#4#5#6#7#8#9\xint_bye { #1#2#3#4#5#6#7#8}%
\long\expandafter\def\csname XINT_keepunbr_end2\endcsname
   #1#2#3#4#5#6#7#8\xint_bye { #1#2#3#4#5#6#7}%
\long\expandafter\def\csname XINT_keepunbr_end3\endcsname
   #1#2#3#4#5#6#7\xint_bye { #1#2#3#4#5#6}%
\long\expandafter\def\csname XINT_keepunbr_end4\endcsname
   #1#2#3#4#5#6\xint_bye { #1#2#3#4#5}%
\long\expandafter\def\csname XINT_keepunbr_end5\endcsname
   #1#2#3#4#5\xint_bye { #1#2#3#4}%
\long\expandafter\def\csname XINT_keepunbr_end6\endcsname
   #1#2#3#4\xint_bye { #1#2#3}%
\long\expandafter\def\csname XINT_keepunbr_end7\endcsname
   #1#2#3\xint_bye { #1#2}%
\long\expandafter\def\csname XINT_keepunbr_end8\endcsname
   #1#2\xint_bye { #1}%
```

3.14 \xintTrim

First included in release 1.09m.

\xintTrim{i}{L} f-expands its second argument L. It then removes the first i items from L and keeps the rest. For i equal or larger to the number N of items in (expanded) L, the macro returns an empty output. For i=0, the original (expanded) L is returned. For i<0, the macro proceeds from the tail. It thus removes the last |i| items, i.e. it keeps the first N-|i| items. For |i|>= N, the empty list is returned.

\xintTrimNoExpand does not expand the L argument.

Speed improvements with 1.2i for i<0 branch (which hands over to \xintKeep). Speed improvements with 1.2j for i>0 branch which gobbles items nine by nine despite not knowing in advance if it will go too far.

```
\def\xintTrim         {\romannumeral0\xinttrim }%
\def\xintTrimNoExpand {\romannumeral0\xinttrimnoexpand }%
\long\def\xinttrim #1#2{\expandafter\XINT_trim_a\the\numexpr #1\expandafter.%
                        \expandafter{\romannumeral`&&@#2}}%
\def\xinttrimnoexpand #1{\expandafter\XINT_trim_a\the\numexpr #1.}%
\def\XINT_trim_a #1%
{%
    \xint_UDzerominusfork
        #1-\XINT_trim_trimnone
        0#1\XINT_trim_neg
         0-{\XINT_trim_pos #1}%
    \krof
}%
\long\def\XINT_trim_trimnone .#1{ #1}%
\long\def\XINT_trim_neg #1.#2%
{%
    \expandafter\XINT_trim_neg_a\the\numexpr
    #1-\numexpr\XINT_length_loop
    #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
      \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
      \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
    .{}#2\xint_bye
}%
\def\XINT_trim_neg_a #1%
{%
    \xint_UDsignfork
        #1{\expandafter\XINT_keep_loop\the\numexpr-\xint_c_viii+}%
         -\XINT_trim_trimall
    \krof
}%
\def\XINT_trim_trimall#1{%
\def\XINT_trim_trimall {\expandafter#1\xint_bye}%
}\XINT_trim_trimall{ }%
```

This branch doesn't pre-evaluate the length of the list argument. Redone again for 1.2j, manages to trim nine by nine. Some non optimal looking aspect of the code is for allowing sharing with \xintNthElt.

```
\long\def\XINT_trim_pos #1.#2%
{%
    \expandafter\XINT_trim_pos_done\expandafter\space
    \romannumeral0\expandafter\XINT_trim_loop\the\numexpr#1-\xint_c_ix.%
     #2\xint:\xint:\xint:\xint:\xint:%
```

```
       \xint:\xint:\xint:\xint:\xint:%
    \xint_bye
}%
\def\XINT_trim_loop #1#2.%
{%
    \xint_gob_til_minus#1\XINT_trim_finish-%
    \expandafter\XINT_trim_loop\the\numexpr#1#2\XINT_trim_loop_trimnine
}%
\long\def\XINT_trim_loop_trimnine #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_xint: #9\XINT_trim_toofew\xint:-\xint_c_ix.%
}%
\def\XINT_trim_toofew\xint:{*\xint_c_}%
\def\XINT_trim_finish#1{%
\def\XINT_trim_finish-%
    \expandafter\XINT_trim_loop\the\numexpr-##1\XINT_trim_loop_trimnine
{%
    \expandafter\expandafter\expandafter#1%
    \csname xint_gobble_\romannumeral\numexpr\xint_c_ix-##1\endcsname
}}\XINT_trim_finish{ }%
\long\def\XINT_trim_pos_done #1\xint:#2\xint_bye {#1}%
```

3.15 \xintTrimUnbraced

1.2a. Modified in 1.2i like \xintTrim

```
\def\xintTrimUnbraced         {\romannumeral0\xinttrimunbraced }%
\def\xintTrimUnbracedNoExpand {\romannumeral0\xinttrimunbracednoexpand }%
\long\def\xinttrimunbraced #1#2%
    {\expandafter\XINT_trimunbr_a\the\numexpr #1\expandafter.%
                        \expandafter{\romannumeral`&&@#2}}%
\def\xinttrimunbracednoexpand #1%
    {\expandafter\XINT_trimunbr_a\the\numexpr #1.}%
\def\XINT_trimunbr_a #1%
{%
    \xint_UDzerominusfork
        #1-\XINT_trim_trimnone
        0#1\XINT_trimunbr_neg
         0-{\XINT_trim_pos #1}%
    \krof
}%
\long\def\XINT_trimunbr_neg #1.#2%
{%
    \expandafter\XINT_trimunbr_neg_a\the\numexpr
    #1-\numexpr\XINT_length_loop
    #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
      \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
      \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
    .{}#2\xint_bye
}%
\def\XINT_trimunbr_neg_a #1%
{%
    \xint_UDsignfork
```

```
        #1{\expandafter\XINT_keepunbr_loop\the\numexpr-\xint_c_viii+}%
         -\XINT_trim_trimall
    \krof
}%
```

3.16 \xintApply

\xintApply {\macro}{{a}{b}...{z}} returns {\macro{a}}...{\macro{b}} where each instance of \macro is f-expanded. The list itself is first f-expanded and may thus be a macro. Introduced with release 1.04.

```
\def\xintApply          {\romannumeral0\xintapply }%
\def\xintApplyNoExpand {\romannumeral0\xintapplynoexpand }%
\long\def\xintapply #1#2%
{%
    \expandafter\XINT_apply\expandafter {\romannumeral`&&@#2}%
    {#1}%
}%
\long\def\XINT_apply #1#2{\XINT_apply_loop_a {}{#2}#1\xint_bye }%
\long\def\xintapplynoexpand #1#2{\XINT_apply_loop_a {}{#1}#2\xint_bye }%
\long\def\XINT_apply_loop_a #1#2#3%
{%
    \xint_bye #3\XINT_apply_end\xint_bye
    \expandafter
    \XINT_apply_loop_b
    \expandafter {\romannumeral`&&@#2{#3}}{#1}{#2}%
}%
\long\def\XINT_apply_loop_b #1#2{\XINT_apply_loop_a {#2{#1}}}%
\long\def\XINT_apply_end\xint_bye\expandafter\XINT_apply_loop_b
    \expandafter #1#2#3{ #2}%
```

3.17 \xintApplyUnbraced

\xintApplyUnbraced {\macro}{{a}{b}...{z}} returns \macro{a}...\macro{z} where each instance of \macro is f-expanded using \romannumeral-`0. The second argument may be a macro as it is itself also f-expanded. No braces are added: this allows for example a non-expandable \def in \macro, without having to do \gdef. Introduced with release 1.06b.

```
\def\xintApplyUnbraced {\romannumeral0\xintapplyunbraced }%
\def\xintApplyUnbracedNoExpand {\romannumeral0\xintapplyunbracednoexpand }%
\long\def\xintapplyunbraced #1#2%
{%
    \expandafter\XINT_applyunbr\expandafter {\romannumeral`&&@#2}%
    {#1}%
}%
\long\def\XINT_applyunbr #1#2{\XINT_applyunbr_loop_a {}{#2}#1\xint_bye }%
\long\def\xintapplyunbracednoexpand #1#2%
   {\XINT_applyunbr_loop_a {}{#1}#2\xint_bye }%
\long\def\XINT_applyunbr_loop_a #1#2#3%
{%
    \xint_bye #3\XINT_applyunbr_end\xint_bye
    \expandafter\XINT_applyunbr_loop_b
    \expandafter {\romannumeral`&&@#2{#3}}{#1}{#2}%
}%
\long\def\XINT_applyunbr_loop_b #1#2{\XINT_applyunbr_loop_a {#2#1}}%
\long\def\XINT_applyunbr_end\xint_bye\expandafter\XINT_applyunbr_loop_b
    \expandafter #1#2#3{ #2}%
```

3.18 \xintSeq

1.09c. Without the optional argument puts stress on the input stack, should not be used to generated thousands of terms then.

```
\def\xintSeq {\romannumeral0\xintseq }%
\def\xintseq #1{\XINT_seq_chkopt  #1\xint_bye }%
\def\XINT_seq_chkopt #1%
{%
    \ifx [#1\expandafter\XINT_seq_opt
       \else\expandafter\XINT_seq_noopt
    \fi  #1%
}%
\def\XINT_seq_noopt #1\xint_bye #2%
{%
    \expandafter\XINT_seq\expandafter
       {\the\numexpr#1\expandafter}\expandafter{\the\numexpr #2}%
}%
\def\XINT_seq #1#2%
{%
   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
      \expandafter\xint_firstoftwo_thenstop
   \or
      \expandafter\XINT_seq_p
   \else
      \expandafter\XINT_seq_n
   \fi
   {#2}{#1}%
}%
\def\XINT_seq_p #1#2%
{%
    \ifnum #1>#2
      \expandafter\expandafter\expandafter\XINT_seq_p
    \else
      \expandafter\XINT_seq_e
    \fi
    \expandafter{\the\numexpr #1-\xint_c_i}{#2}{#1}%
}%
\def\XINT_seq_n #1#2%
{%
    \ifnum #1<#2
      \expandafter\expandafter\expandafter\XINT_seq_n
    \else
      \expandafter\XINT_seq_e
    \fi
     \expandafter{\the\numexpr #1+\xint_c_i}{#2}{#1}%
}%
\def\XINT_seq_e #1#2#3{ }%
```

```
\def\XINT_seq_opt [\xint_bye #1]#2#3%
{%
    \expandafter\XINT_seqo\expandafter
    {\the\numexpr #2\expandafter}\expandafter
    {\the\numexpr #3\expandafter}\expandafter
    {\the\numexpr #1}%
}%
\def\XINT_seqo #1#2%
{%
   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
      \expandafter\XINT_seqo_a
   \or
      \expandafter\XINT_seqo_pa
   \else
      \expandafter\XINT_seqo_na
   \fi
   {#1}{#2}%
}%
\def\XINT_seqo_a #1#2#3{ {#1}}%
\def\XINT_seqo_o #1#2#3#4{ #4}%
\def\XINT_seqo_pa #1#2#3%
{%
    \ifcase\ifnum #3=\xint_c_ 0\else\ifnum #3>\xint_c_ 1\else -1\fi\fi\space
           \expandafter\XINT_seqo_o
    \or
           \expandafter\XINT_seqo_pb
    \else
           \xint_afterfi{\expandafter\space\xint_gobble_iv}%
    \fi
    {#1}{#2}{#3}{{#1}}%
}%
\def\XINT_seqo_pb #1#2#3%
{%
    \expandafter\XINT_seqo_pc\expandafter{\the\numexpr #1+#3}{#2}{#3}%
}%
\def\XINT_seqo_pc #1#2%
{%
    \ifnum #1>#2
        \expandafter\XINT_seqo_o
    \else
        \expandafter\XINT_seqo_pd
    \fi
    {#1}{#2}%
}%
\def\XINT_seqo_pd #1#2#3#4{\XINT_seqo_pb {#1}{#2}{#3}{#4{#1}}}%
\def\XINT_seqo_na #1#2#3%
{%
    \ifcase\ifnum #3=\xint_c_ 0\else\ifnum #3>\xint_c_ 1\else -1\fi\fi\space
        \expandafter\XINT_seqo_o
    \or
        \xint_afterfi{\expandafter\space\xint_gobble_iv}%
    \else
```

```
        \expandafter\XINT_seqo_nb
    \fi
    {#1}{#2}{#3}{{#1}}%
}%
\def\XINT_seqo_nb #1#2#3%
{%
    \expandafter\XINT_seqo_nc\expandafter{\the\numexpr #1+#3}{#2}{#3}%
}%
\def\XINT_seqo_nc #1#2%
{%
    \ifnum #1<#2
        \expandafter\XINT_seqo_o
    \else
        \expandafter\XINT_seqo_nd
    \fi
    {#1}{#2}%
}%
\def\XINT_seqo_nd #1#2#3#4{\XINT_seqo_nb {#1}{#2}{#3}{#4{#1}}}%
```

3.19 \xintloop, \xintbreakloop, \xintbreakloopanddo, \xintloopskiptonext

1.09g [2013/11/22]. Made long with 1.09h.

```
\long\def\xintloop #1#2\repeat {#1#2\xintloop_again\fi\xint_gobble_i {#1#2}}%
\long\def\xintloop_again\fi\xint_gobble_i #1{\fi
                             #1\xintloop_again\fi\xint_gobble_i {#1}}%
\long\def\xintbreakloop #1\xintloop_again\fi\xint_gobble_i #2{}%
\long\def\xintbreakloopanddo #1#2\xintloop_again\fi\xint_gobble_i #3{#1}%
\long\def\xintloopskiptonext #1\xintloop_again\fi\xint_gobble_i #2{%
                        #2\xintloop_again\fi\xint_gobble_i {#2}}%
```

3.20 \xintiloop, \xintiloopindex, \xintbracediloopindex, \xintouteriloopindex, \xintbracedouteriloopindex, \xintbreakiloop, \xintbreakiloopanddo, \xintiloopskiptonext, \xintiloopskipandredo

1.09g [2013/11/22]. Made long with 1.09h.
«braced» variants added (2018/04/24) for 1.3b.

```
\def\xintiloop [#1+#2]{%
    \expandafter\xintiloop_a\the\numexpr #1\expandafter.\the\numexpr #2.}%
\long\def\xintiloop_a #1.#2.#3#4\repeat{%
    #3#4\xintiloop_again\fi\xint_gobble_iii {#1}{#2}{#3#4}}%
\def\xintiloop_again\fi\xint_gobble_iii #1#2{%
    \fi\expandafter\xintiloop_again_b\the\numexpr#1+#2.#2.}%
\long\def\xintiloop_again_b #1.#2.#3{%
    #3\xintiloop_again\fi\xint_gobble_iii {#1}{#2}{#3}}%
\long\def\xintbreakiloop #1\xintiloop_again\fi\xint_gobble_iii #2#3#4{}%
\long\def\xintbreakiloopanddo
     #1.#2\xintiloop_again\fi\xint_gobble_iii #3#4#5{#1}%
\long\def\xintiloopindex #1\xintiloop_again\fi\xint_gobble_iii #2%
                  {#2#1\xintiloop_again\fi\xint_gobble_iii {#2}}%
\long\def\xintbracediloopindex #1\xintiloop_again\fi\xint_gobble_iii #2%
```

```
    {{#2}#1\xintiloop_again\fi\xint_gobble_iii {#2}}%
\long\def\xintouteriloopindex #1\xintiloop_again
                          #2\xintiloop_again\fi\xint_gobble_iii #3%
   {#3#1\xintiloop_again #2\xintiloop_again\fi\xint_gobble_iii {#3}}%
\long\def\xintbracedouteriloopindex #1\xintiloop_again
                          #2\xintiloop_again\fi\xint_gobble_iii #3%
   {{#3}#1\xintiloop_again #2\xintiloop_again\fi\xint_gobble_iii {#3}}%
\long\def\xintiloopskiptonext #1\xintiloop_again\fi\xint_gobble_iii #2#3{%
    \expandafter\xintiloop_again_b \the\numexpr#2+#3.#3.}%
\long\def\xintiloopskipandredo #1\xintiloop_again\fi\xint_gobble_iii #2#3#4{%
    #4\xintiloop_again\fi\xint_gobble_iii {#2}{#3}{#4}}%
```

3.21 `\XINT_xflet`

1.09e [2013/10/29]: we f-expand unbraced tokens and swallow arising space tokens until the dust settles.

```
\def\XINT_xflet #1%
{%
    \def\XINT_xflet_macro {#1}\XINT_xflet_zapsp
}%
\def\XINT_xflet_zapsp
{%
    \expandafter\futurelet\expandafter\XINT_token
    \expandafter\XINT_xflet_sp?\romannumeral`&&@%
}%
\def\XINT_xflet_sp?
{%
    \ifx\XINT_token\XINT_sptoken
         \expandafter\XINT_xflet_zapsp
    \else\expandafter\XINT_xflet_zapspB
    \fi
}%
\def\XINT_xflet_zapspB
{%
    \expandafter\futurelet\expandafter\XINT_tokenB
    \expandafter\XINT_xflet_spB?\romannumeral`&&@%
}%
\def\XINT_xflet_spB?
{%
    \ifx\XINT_tokenB\XINT_sptoken
         \expandafter\XINT_xflet_zapspB
    \else\expandafter\XINT_xflet_eq?
    \fi
}%
\def\XINT_xflet_eq?
{%
    \ifx\XINT_token\XINT_tokenB
         \expandafter\XINT_xflet_macro
    \else\expandafter\XINT_xflet_zapsp
    \fi
}%
```

3.22 \xintApplyInline

1.09a: \xintApplyInline\macro{{a}{b}...{z}} has the same effect as executing \macro{a} and then applying again \xintApplyInline to the shortened list {{b}...{z}} until nothing is left. This is a non-expandable command which will result in quicker code than using \xintApplyUnbraced. It f-expands its second (list) argument first, which may thus be encapsulated in a macro.

Rewritten in 1.09c. Nota bene: uses catcode 3 Z as privated list terminator.

```
\catcode`Z 3
\long\def\xintApplyInline #1#2%
{%
  \long\expandafter\def\expandafter\XINT_inline_macro
  \expandafter ##\expandafter 1\expandafter {#1{##1}}%
  \XINT_xflet\XINT_inline_b #2Z% this Z has catcode 3
}%
\def\XINT_inline_b
{%
    \ifx\XINT_token Z\expandafter\xint_gobble_i
    \else\expandafter\XINT_inline_d\fi
}%
\long\def\XINT_inline_d #1%
{%
  \long\def\XINT_item{{#1}}\XINT_xflet\XINT_inline_e
}%
\def\XINT_inline_e
{%
    \ifx\XINT_token Z\expandafter\XINT_inline_w
    \else\expandafter\XINT_inline_f\fi
}%
\def\XINT_inline_f
{%
  \expandafter\XINT_inline_g\expandafter{\XINT_inline_macro {##1}}%
}%
\long\def\XINT_inline_g #1%
{%
   \expandafter\XINT_inline_macro\XINT_item
   \long\def\XINT_inline_macro ##1{#1}\XINT_inline_d
}%
\def\XINT_inline_w #1%
{%
   \expandafter\XINT_inline_macro\XINT_item
}%
```

3.23 \xintFor, \xintFor*, \xintBreakFor, \xintBreakForAndDo

1.09c [2013/10/09]: a new kind of loop which uses macro parameters #1, #2, #3, #4 rather than macros; while not expandable it survives executing code closing groups, like what happens in an alignment with the & character. When inserted in a macro for later use, the # character must be doubled.

The non-star variant works on a csv list, which it expands once, the star variant works on a token list, which it (repeatedly) f-expands.

1.09e adds \XINT_forever with \xintintegers, \xintdimensions, \xintrationals and \xintBreakFor, \xintBreakForAndDo, \xintifForFirst, \xintifForLast. On this occasion \xint_firstoftwo and

\xint_secondoftwo are made long.

1.09f: rewrites large parts of \xintFor code in order to filter the comma separated list via \xintCSVtoList which gets rid of spaces. The #1 in \XINT_for_forever? has an initial space token which serves two purposes: preventing brace stripping, and stopping the expansion made by \xintcsvtolist. If the \XINT_forever branch is taken, the added space will not be a problem there.

1.09f rewrites (2013/11/03) the code which now allows all macro parameters from #1 to #9 in \xintFor, \xintFor*, and \XINT_forever. 1.2i: slightly more robust \xintifForFirst/Last in case of nesting.

```
\def\XINT_tmpa #1#2{\ifnum #2<#1 \xint_afterfi {{#########2}}\fi}%
\def\XINT_tmpb #1#2{\ifnum #1<#2 \xint_afterfi {{#########2}}\fi}%
\def\XINT_tmpc #1%
{%
    \expandafter\edef \csname XINT_for_left#1\endcsname
                {\xintApplyUnbraced {\XINT_tmpa #1}{123456789}}%
    \expandafter\edef \csname XINT_for_right#1\endcsname
                {\xintApplyUnbraced {\XINT_tmpb #1}{123456789}}%
}%
\xintApplyInline \XINT_tmpc {123456789}%
\long\def\xintBreakFor      #1Z{}%
\long\def\xintBreakForAndDo #1#2Z{#1}%
\def\xintFor {\let\xintifForFirst\xint_firstoftwo
              \let\xintifForLast\xint_secondoftwo
              \futurelet\XINT_token\XINT_for_ifstar }%
\def\XINT_for_ifstar {\ifx\XINT_token*\expandafter\XINT_forx
                                 \else\expandafter\XINT_for \fi }%
\catcode`U 3 % with numexpr
\catcode`V 3 % with xintfrac.sty (xint.sty not enough)
\catcode`D 3 % with dimexpr
\def\XINT_flet_zapsp
{%
    \futurelet\XINT_token\XINT_flet_sp?
}%
\def\XINT_flet_sp?
{%
    \ifx\XINT_token\XINT_sptoken
         \xint_afterfi{\expandafter\XINT_flet_zapsp\romannumeral0}%
    \else\expandafter\XINT_flet_macro
    \fi
}%
\long\def\XINT_for #1#2in#3#4#5%
{%
    \expandafter\XINT_toks\expandafter
        {\expandafter\XINT_for_d\the\numexpr #2\relax {#5}}%
    \def\XINT_flet_macro {\expandafter\XINT_for_forever?\space}%
    \expandafter\XINT_flet_zapsp #3Z%
}%
\def\XINT_for_forever? #1Z%
{%
    \ifx\XINT_token U\XINT_to_forever\fi
    \ifx\XINT_token V\XINT_to_forever\fi
    \ifx\XINT_token D\XINT_to_forever\fi
    \expandafter\the\expandafter\XINT_toks\romannumeral0\xintcsvtolist {#1}Z%
```

```
}%
\def\XINT_to_forever\fi #1\xintcsvtolist #2{\fi \XINT_forever #2}%
\long\def\XINT_forx *#1#2in#3#4#5%
{%
    \expandafter\XINT_toks\expandafter
       {\expandafter\XINT_forx_d\the\numexpr #2\relax {#5}}%
    \XINT_xflet\XINT_forx_forever? #3Z%
}%
\def\XINT_forx_forever?
{%
    \ifx\XINT_token U\XINT_to_forxever\fi
    \ifx\XINT_token V\XINT_to_forxever\fi
    \ifx\XINT_token D\XINT_to_forxever\fi
    \XINT_forx_empty?
}%
\def\XINT_to_forxever\fi #1\XINT_forx_empty? {\fi \XINT_forever }%
\catcode`U 11
\catcode`D 11
\catcode`V 11
\def\XINT_forx_empty?
{%
    \ifx\XINT_token Z\expandafter\xintBreakFor\fi
    \the\XINT_toks
}%
\long\def\XINT_for_d #1#2#3%
{%
  \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
  \XINT_toks {{#3}}%
  \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
                     \the\XINT_toks   \csname XINT_for_right#1\endcsname }%
  \XINT_toks {\XINT_x\let\xintifForFirst\xint_secondoftwo
                     \let\xintifForLast\xint_secondoftwo\XINT_for_d #1{#2}}%
  \futurelet\XINT_token\XINT_for_last?
}%
\long\def\XINT_forx_d #1#2#3%
{%
  \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
  \XINT_toks {{#3}}%
  \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
                     \the\XINT_toks   \csname XINT_for_right#1\endcsname }%
  \XINT_toks {\XINT_x\let\xintifForFirst\xint_secondoftwo
                     \let\xintifForLast\xint_secondoftwo\XINT_forx_d #1{#2}}%
  \XINT_xflet\XINT_for_last?
}%
\def\XINT_for_last?
{%
  \ifx\XINT_token Z\expandafter\XINT_for_last?yes\fi
  \the\XINT_toks
}%
\def\XINT_for_last?yes
{%
 \let\xintifForLast\xint_firstoftwo
 \xintBreakForAndDo{\XINT_x\xint_gobble_i Z}%
}%
```

3.24 \XINT_forever, \xintintegers, \xintdimensions, \xintrationals

New with 1.09e. But this used inadvertently \xintiadd/\xintimul which have the unnecessary \xintnum overhead. Changed in 1.09f to use \xintiiadd/\xintiimul which do not have this overhead. Also 1.09f uses \xintZapSpacesB for the \xintrationals case to get rid of leading and ending spaces in the #4 and #5 delimited parameters of \XINT_forever_opt_a (for \xintintegers and \xintdimensions this is not necessary, due to the use of \numexpr resp. \dimexpr in \XINT_?expr_Ua, resp.\XINT_?expr_Da).

```
\catcode`U 3
\catcode`D 3
\catcode`V 3
\let\xintegers      U%
\let\xintintegers   U%
\let\xintdimensions D%
\let\xintrationals  V%
\def\XINT_forever #1%
{%
  \expandafter\XINT_forever_a
  \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi a\expandafter\endcsname
  \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi i\expandafter\endcsname
  \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi \endcsname
}%
\catcode`U 11
\catcode`D 11
\catcode`V 11
\def\XINT_?expr_Ua #1#2%
   {\expandafter{\expandafter\numexpr\the\numexpr #1\expandafter\relax
                               \expandafter\relax\expandafter}%
    \expandafter{\the\numexpr #2}}%
\def\XINT_?expr_Da #1#2%
   {\expandafter{\expandafter\dimexpr\number\dimexpr #1\expandafter\relax
                 \expandafter s\expandafter p\expandafter\relax\expandafter}%
    \expandafter{\number\dimexpr #2}}%
\catcode`Z 11
\def\XINT_?expr_Va #1#2%
{%
    \expandafter\XINT_?expr_Vb\expandafter
          {\romannumeral`&&@\xintrawwithzeros{\xintZapSpacesB{#2}}}%
          {\romannumeral`&&@\xintrawwithzeros{\xintZapSpacesB{#1}}}%
}%
\catcode`Z 3
\def\XINT_?expr_Vb #1#2{\expandafter\XINT_?expr_Vc #2.#1.}%
\def\XINT_?expr_Vc #1/#2.#3/#4.%
{%
     \xintifEq {#2}{#4}%
       {\XINT_?expr_Vf {#3}{#1}{#2}}%
       {\expandafter\XINT_?expr_Vd\expandafter
        {\romannumeral0\xintiimul {#2}{#4}}%
        {\romannumeral0\xintiimul {#1}{#4}}%
        {\romannumeral0\xintiimul {#2}{#3}}%
       }%
}%
\def\XINT_?expr_Vd #1#2#3{\expandafter\XINT_?expr_Ve\expandafter {#2}{#3}{#1}}%
\def\XINT_?expr_Ve #1#2{\expandafter\XINT_?expr_Vf\expandafter {#2}{#1}}%
\def\XINT_?expr_Vf #1#2#3{{#2/#3}{{0}{#1}{#2}{#3}}}%
\def\XINT_?expr_Ui {{\numexpr 1\relax}{1}}%
\def\XINT_?expr_Di {{\dimexpr 0pt\relax}{65536}}%
\def\XINT_?expr_Vi {{1/1}{0111}}%
\def\XINT_?expr_U #1#2%
   {\expandafter{\expandafter\numexpr\the\numexpr #1+#2\relax\relax}{#2}}%
\def\XINT_?expr_D #1#2%
   {\expandafter{\expandafter\dimexpr\the\numexpr #1+#2\relax sp\relax}{#2}}%
\def\XINT_?expr_V #1#2{\XINT_?expr_Vx #2}%
\def\XINT_?expr_Vx #1#2%
{%
     \expandafter\XINT_?expr_Vy\expandafter
        {\romannumeral0\xintiiadd {#1}{#2}}{#2}%
}%
\def\XINT_?expr_Vy #1#2#3#4%
{%
     \expandafter{\romannumeral0\xintiiadd {#3}{#1}/#4}{{#1}{#2}{#3}{#4}}%
}%
\def\XINT_forever_a #1#2#3#4%
{%
    \ifx #4[\expandafter\XINT_forever_opt_a
       \else\expandafter\XINT_forever_b
    \fi #1#2#3#4%
}%
\def\XINT_forever_b #1#2#3Z{\expandafter\XINT_forever_c\the\XINT_toks #2#3}%
\long\def\XINT_forever_c #1#2#3#4#5%
    {\expandafter\XINT_forever_d\expandafter #2#4#5{#3}Z}%
\def\XINT_forever_opt_a #1#2#3[#4+#5]#6Z%
{%
    \expandafter\expandafter\expandafter
    \XINT_forever_opt_c\expandafter\the\expandafter\XINT_toks
    \romannumeral`&&@#1{#4}{#5}#3%
}%
\long\def\XINT_forever_opt_c #1#2#3#4#5#6{\XINT_forever_d #2{#4}{#5}#6{#3}Z}%
\long\def\XINT_forever_d #1#2#3#4#5%
{%
  \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#5}%
  \XINT_toks {{#2}}%
  \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
                     \the\XINT_toks   \csname XINT_for_right#1\endcsname }%
  \XINT_x
  \let\xintifForFirst\xint_secondoftwo
  \let\xintifForLast\xint_secondoftwo
  \expandafter\XINT_forever_d\expandafter #1\romannumeral`&&@#4{#2}{#3}#4{#5}%
}%
```

3.25 \xintForpair, \xintForthree, \xintForfour

1.09c.

[2013/11/02] 1.09f \xintForpair delegate to \xintCSVtoList and its \xintZapSpacesB the handling of spaces. Does not share code with \xintFor anymore.

[2013/11/03] 1.09f: \xintForpair extended to accept #1#2, #2#3 etc... up to #8#9, \xintForthree, #1#2#3 up to #7#8#9, \xintForfour id.

1.2i: slightly more robust \xintifForFirst/Last in case of nesting.

```
\catcode`j 3
\long\def\xintForpair #1#2#3in#4#5#6%
{%
    \let\xintifForFirst\xint_firstoftwo
    \let\xintifForLast\xint_secondoftwo
    \XINT_toks  {\XINT_forpair_d #2{#6}}%
    \expandafter\the\expandafter\XINT_toks #4jZ%
}%
\long\def\XINT_forpair_d #1#2#3(#4)#5%
{%
  \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
  \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
  \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
      \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_i\endcsname}%
  \ifx #5j\expandafter\XINT_for_last?yes\fi
  \XINT_x
  \let\xintifForFirst\xint_secondoftwo
  \let\xintifForLast\xint_secondoftwo
  \XINT_forpair_d #1{#2}%
}%
\long\def\xintForthree #1#2#3in#4#5#6%
{%
    \let\xintifForFirst\xint_firstoftwo
    \let\xintifForLast\xint_secondoftwo
    \XINT_toks  {\XINT_forthree_d #2{#6}}%
    \expandafter\the\expandafter\XINT_toks #4jZ%
}%
\long\def\XINT_forthree_d #1#2#3(#4)#5%
{%
  \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
  \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
  \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
      \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_ii\endcsname}%
  \ifx #5j\expandafter\XINT_for_last?yes\fi
  \XINT_x
  \let\xintifForFirst\xint_secondoftwo
  \let\xintifForLast\xint_secondoftwo
  \XINT_forthree_d #1{#2}%
}%
\long\def\xintForfour #1#2#3in#4#5#6%
{%
    \let\xintifForFirst\xint_firstoftwo
    \let\xintifForLast\xint_secondoftwo
    \XINT_toks  {\XINT_forfour_d #2{#6}}%
    \expandafter\the\expandafter\XINT_toks #4jZ%
}%
\long\def\XINT_forfour_d #1#2#3(#4)#5%
{%
  \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
  \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
  \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
     \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_iii\endcsname}%
  \ifx #5j\expandafter\XINT_for_last?yes\fi
  \XINT_x
  \let\xintifForFirst\xint_secondoftwo
  \let\xintifForLast\xint_secondoftwo
  \XINT_forfour_d #1{#2}%
}%
\catcode`Z 11
\catcode`j 11
```

3.26 `\xintAssign`, `\xintAssignArray`, `\xintDigitsOf`

\xintAssign {a}{b}..{z}\to\A\B...\Z resp. \xintAssignArray {a}{b}..{z}\to\U.
  \xintDigitsOf=\xintAssignArray.
  1.1c 2015/09/12 has (belatedly) corrected some "features" of \xintAssign which didn't like the case of a space right before the "\to", or the case with the first token not an opening brace and the subsequent material containing brace groups. The new code handles gracefully these situations.

```
\def\xintAssign{\def\XINT_flet_macro {\XINT_assign_fork}\XINT_flet_zapsp }%
\def\XINT_assign_fork
{%
    \let\XINT_assign_def\def
    \ifx\XINT_token[\expandafter\XINT_assign_opt
              \else\expandafter\XINT_assign_a
    \fi
}%
\def\XINT_assign_opt [#1]%
{%
    \ifcsname #1def\endcsname
      \expandafter\let\expandafter\XINT_assign_def \csname #1def\endcsname
    \else
      \expandafter\let\expandafter\XINT_assign_def \csname xint#1def\endcsname
    \fi
    \XINT_assign_a
}%
\long\def\XINT_assign_a #1\to
{%
    \def\XINT_flet_macro{\XINT_assign_b}%
    \expandafter\XINT_flet_zapsp\romannumeral`&&@#1\xint:\to
}%
\long\def\XINT_assign_b
{%
    \ifx\XINT_token\bgroup
         \expandafter\XINT_assign_c
    \else\expandafter\XINT_assign_f
    \fi
```

```
}%
\long\def\XINT_assign_f #1\xint:\to #2%
{%
    \XINT_assign_def #2{#1}%
}%
\long\def\XINT_assign_c #1%
{%
    \def\xint_temp {#1}%
    \ifx\xint_temp\xint_bracedstopper
        \expandafter\XINT_assign_e
    \else
        \expandafter\XINT_assign_d
    \fi
}%
\long\def\XINT_assign_d #1\to #2%
{%
    \expandafter\XINT_assign_def\expandafter #2\expandafter{\xint_temp}%
    \XINT_assign_c #1\to
}%
\def\XINT_assign_e #1\to {}%
\def\xintRelaxArray #1%
{%
    \edef\XINT_restoreescapechar {\escapechar\the\escapechar\relax}%
    \escapechar -1
    \expandafter\def\expandafter\xint_arrayname\expandafter {\string #1}%
    \XINT_restoreescapechar
    \xintiloop [\csname\xint_arrayname 0\endcsname+-1]
      \global
      \expandafter\let\csname\xint_arrayname\xintiloopindex\endcsname\relax
      \ifnum \xintiloopindex > \xint_c_
    \repeat
    \global\expandafter\let\csname\xint_arrayname 00\endcsname\relax
    \global\let #1\relax
}%
\def\xintAssignArray{\def\XINT_flet_macro {\XINT_assignarray_fork}%
                     \XINT_flet_zapsp }%
\def\XINT_assignarray_fork
{%
    \let\XINT_assignarray_def\def
    \ifx\XINT_token[\expandafter\XINT_assignarray_opt
               \else\expandafter\XINT_assignarray
    \fi
}%
\def\XINT_assignarray_opt [#1]%
{%
    \ifcsname #1def\endcsname
      \expandafter\let\expandafter\XINT_assignarray_def \csname #1def\endcsname
    \else
      \expandafter\let\expandafter\XINT_assignarray_def
                                 \csname xint#1def\endcsname
    \fi
    \XINT_assignarray
}%
\long\def\XINT_assignarray #1\to #2%
{%
    \edef\XINT_restoreescapechar {\escapechar\the\escapechar\relax }%
    \escapechar -1
    \expandafter\def\expandafter\xint_arrayname\expandafter {\string #2}%
    \XINT_restoreescapechar
    \def\xint_itemcount {0}%
    \expandafter\XINT_assignarray_loop \romannumeral`&&@#1\xint:
    \csname\xint_arrayname 00\expandafter\endcsname
    \csname\xint_arrayname 0\expandafter\endcsname
    \expandafter {\xint_arrayname}#2%
}%
\long\def\XINT_assignarray_loop #1%
{%
    \def\xint_temp {#1}%
    \ifx\xint_temp\xint_bracedstopper
       \expandafter\def\csname\xint_arrayname 0\expandafter\endcsname
                    \expandafter{\the\numexpr\xint_itemcount}%
       \expandafter\expandafter\expandafter\XINT_assignarray_end
    \else
       \expandafter\def\expandafter\xint_itemcount\expandafter
                    {\the\numexpr\xint_itemcount+\xint_c_i}%
       \expandafter\XINT_assignarray_def
          \csname\xint_arrayname\xint_itemcount\expandafter\endcsname
             \expandafter{\xint_temp }%
       \expandafter\XINT_assignarray_loop
    \fi
}%
\def\XINT_assignarray_end #1#2#3#4%
{%
    \def #4##1%
    {%
        \romannumeral0\expandafter #1\expandafter{\the\numexpr ##1}%
    }%
    \def #1##1%
    {%
        \ifnum ##1<\xint_c_
            \xint_afterfi{\XINT_expandableerror{Array index negative: 0 > ##1} }%
        \else
        \xint_afterfi {%
              \ifnum ##1>#2
                  \xint_afterfi
                  {\XINT_expandableerror{Array index beyond range: ##1 > #2} }%
              \else\xint_afterfi
       {\expandafter\expandafter\expandafter\space\csname #3##1\endcsname}%
              \fi}%
        \fi
     }%
}%
\let\xintDigitsOf\xintAssignArray
```

3.27 \xintExpandArgs

1.3a. Added for the needs of user defined functions for the expression parsers. Should I re-code it to gain a bit in argument grabbing? Must be f-expandable.

```
\def\xintExpandArgs#1#2{\csname #1\expandafter\endcsname
    \romannumeral0\xintapply\xint_firstofone{#2}}%
```

3.28 CSV (non user documented) variants of Length, Keep, Trim, NthElt, Reverse

These routines are for use by \xintListSel:x:csv and \xintListSel:f:csv from xintexpr, and also for the reversed and len functions. Refactored for 1.2j release, following 1.2i updates to \xintKeep, \xintTrim, ...

These macros will remain undocumented in the user manual:

-- they exist primarily for internal use by the xintexpr parsers, hence don't have to be general purpose; for example, they a priori need to handle only catcode 12 tokens (not true in \xintNewExpr, though) hence they are not really worried about controlling brace stripping (nevertheless 1.2j has paid some secondary attention to it, see below.) They are not worried about normalizing leading spaces either, because none will be encountered when the macros are used as auxiliaries to the expression parsers.

-- crucial design elements may change in future:

1. whether the handled lists must have or not have a final comma. Currently, the model is the one of comma separated lists with **no** final comma. But this means that there can not be a distinction of principle between a truly empty list and a list which contains one item which turns out to be empty. More importantly it makes the coding more complicated as it is needed to distinguish the empty list from the single-item list, both lacking commas.

For the internal use of xintexpr, it would be ok to require all list items to be terminated by a comma, and this would bring quite some simplications here, but as initially I started with non-terminated lists, I have left it this way in the 1.2j refactoring.

2. the way to represent the empty list. I was tempted for matter of optimization and synchronization with xintexpr context to require the empty list to be always represented by a space token and to not let the macros admit a completely empty input. But there were complications so for the time being 1.2j does accept truly empty output (it is not distinguished from an input equal to a space token) and produces empty output for empty list. This means that the status of the «nil» object for the xintexpr parsers is not completely clarified (currently it is represented by a space token).

The original Python slicing code in xintexpr 1.1 used \xintCSVtoList and \xintListWithSep{,} to convert back and forth to token lists and apply \xintKeep/\xintTrim. Release 1.2g switched to devoted f-expandable macros added to xinttools. Release 1.2j refactored all these macros as a follow-up to 1.2i improvements to \xintKeep/\xintTrim. They were made \long on this occasion and auxiliary \xintLengthUpTo:f:csv was added.

Leading spaces in items are currently maintained as is by the 1.2j macros, even by \xintNthEltPy:f:csv, with the exception of the first item, as the list is f-expanded. Perhaps \xintNthEltPy:f:csv should remove a leading space if present in the picked item; anyway, there are no spaces for the lists handled internally by the Python slicer of xintexpr, except the «nil» object currently represented by exactly one space.

Kept items (with no leading spaces; but first item special as it will have lost a leading space due to f-expansion) will lose a brace pair under \xintKeep:f:csv if the first argument was positive and strictly less than the length of the list. This differs of course from \xintKeep (which always braces items it outputs when used with positive first argument) and also from \xintKeepUnbraced in the case when the whole list is kept. Actually the case of singleton list is special, and brace removal will happen then.

This behaviour was otherwise for releases earlier than 1.2j and may change again.

Directly usable names are provided, but these macros (and the behaviour as described above) are to be considered *unstable* for the time being.

3.28.1 \xintLength:f:csv

1.2g. Redone for 1.2j. Contrarily to \xintLength from xintkernel.sty, this one expands its argument.

```
\def\xintLength:f:csv {\romannumeral0\xintlength:f:csv}%
\def\xintlength:f:csv #1%
{\long\def\xintlength:f:csv ##1{%
    \expandafter#1\the\numexpr\expandafter\XINT_length:f:csv_a
    \romannumeral`&&@##1\xint:,\xint:,\xint:,\xint:,%
      \xint:,\xint:,\xint:,\xint:,\xint:,%
      \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
      \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
    \relax
}}\xintlength:f:csv { }%
```

Must first check if empty list.

```
\long\def\XINT_length:f:csv_a #1%
{%
    \xint_gob_til_xint: #1\xint_c_\xint_bye\xint:%
    \XINT_length:f:csv_loop #1%
}%
\long\def\XINT_length:f:csv_loop #1,#2,#3,#4,#5,#6,#7,#8,#9,%
{%
    \xint_gob_til_xint: #9\XINT_length:f:csv_finish\xint:%
    \xint_c_ix+\XINT_length:f:csv_loop
}%
\def\XINT_length:f:csv_finish\xint:\xint_c_ix+\XINT_length:f:csv_loop
    #1,#2,#3,#4,#5,#6,#7,#8,#9,{#9\xint_bye}%
```

3.28.2 \xintLengthUpTo:f:csv

1.2j. \xintLengthUpTo:f:csv{N}{comma-list}. No ending comma. Returns -0 if length>N, else returns difference N-length. **N must be non-negative!!**

Attention to the dot after \xint_bye for the loop interface.

```
\def\xintLengthUpTo:f:csv {\romannumeral0\xintlengthupto:f:csv}%
\long\def\xintlengthupto:f:csv #1#2%
{%
    \expandafter\XINT_lengthupto:f:csv_a
    \the\numexpr#1\expandafter.%
    \romannumeral`&&@#2\xint:,\xint:,\xint:,\xint:,%
         \xint:,\xint:,\xint:,\xint:,%
         \xint_c_viii,\xint_c_vii,\xint_c_vi,\xint_c_v,%
         \xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye.%
}%
```

Must first recognize if empty list. If this is the case, return N.

```
\long\def\XINT_lengthupto:f:csv_a #1.#2%
{%
    \xint_gob_til_xint: #2\XINT_lengthupto:f:csv_empty\xint:%
    \XINT_lengthupto:f:csv_loop_b #1.#2%
}%
\def\XINT_lengthupto:f:csv_empty\xint:%
    \XINT_lengthupto:f:csv_loop_b #1.#2\xint_bye.{ #1}%
\def\XINT_lengthupto:f:csv_loop_a #1%
{%
    \xint_UDsignfork
      #1\XINT_lengthupto:f:csv_gt
       -\XINT_lengthupto:f:csv_loop_b
    \krof #1%
}%
\long\def\XINT_lengthupto:f:csv_gt #1\xint_bye.{-0}%
\long\def\XINT_lengthupto:f:csv_loop_b #1.#2,#3,#4,#5,#6,#7,#8,#9,%
{%
    \xint_gob_til_xint: #9\XINT_lengthupto:f:csv_finish_a\xint:%
    \expandafter\XINT_lengthupto:f:csv_loop_a\the\numexpr #1-\xint_c_viii.%
}%
\def\XINT_lengthupto:f:csv_finish_a\xint:
    \expandafter\XINT_lengthupto:f:csv_loop_a
    \the\numexpr #1-\xint_c_viii.#2,#3,#4,#5,#6,#7,#8,#9,%
{%
    \expandafter\XINT_lengthupto:f:csv_finish_b\the\numexpr #1-#9\xint_bye
}%
\def\XINT_lengthupto:f:csv_finish_b #1#2.%
{%
    \xint_UDsignfork
       #1{-0}%
        -{ #1#2}%
    \krof
}%
```

3.28.3 \xintKeep:f:csv

1.2g 2016/03/17. Redone for 1.2j with use of \xintLengthUpTo:f:csv. Same code skeleton as \xintKeep but handling comma separated but non terminated lists has complications. The \xintKeep in case of a negative #1 uses \xintgobble, we don't have that for comma delimited items, hence we do a special loop here (this style of loop is surely competitive with xintgobble for a few dozens items and even more). The loop knows before starting that it will not go too far.

```
\def\xintKeep:f:csv {\romannumeral0\xintkeep:f:csv }%
\long\def\xintkeep:f:csv #1#2%
{%
    \expandafter\xint_gobble_thenstop
    \romannumeral0\expandafter\XINT_keep:f:csv_a
    \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
}%
\def\XINT_keep:f:csv_a #1%
{%
    \xint_UDzerominusfork
        #1-\XINT_keep:f:csv_keepnone
        0#1\XINT_keep:f:csv_neg
         0-{\XINT_keep:f:csv_pos #1}%
    \krof
}%
\long\def\XINT_keep:f:csv_keepnone .#1{,}%
\long\def\XINT_keep:f:csv_neg #1.#2%
{%
    \expandafter\XINT_keep:f:csv_neg_done\expandafter,%
    \romannumeral0%
    \expandafter\XINT_keep:f:csv_neg_a\the\numexpr
    #1-\numexpr\XINT_length:f:csv_a
    #2\xint:,\xint:,\xint:,\xint:,%
      \xint:,\xint:,\xint:,\xint:,\xint:,%
      \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
      \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
    .#2\xint_bye
}%
\def\XINT_keep:f:csv_neg_a #1%
{%
    \xint_UDsignfork
        #1{\expandafter\XINT_keep:f:csv_trimloop\the\numexpr-\xint_c_ix+}%
         -\XINT_keep:f:csv_keepall
    \krof
}%
\def\XINT_keep:f:csv_keepall #1.{ }%
\long\def\XINT_keep:f:csv_neg_done #1\xint_bye{#1}%
\def\XINT_keep:f:csv_trimloop #1#2.%
{%
    \xint_gob_til_minus#1\XINT_keep:f:csv_trimloop_finish-%
    \expandafter\XINT_keep:f:csv_trimloop
    \the\numexpr#1#2-\xint_c_ix\expandafter.\XINT_keep:f:csv_trimloop_trimnine
}%
\long\def\XINT_keep:f:csv_trimloop_trimnine #1,#2,#3,#4,#5,#6,#7,#8,#9,{}%
\def\XINT_keep:f:csv_trimloop_finish-%
    \expandafter\XINT_keep:f:csv_trimloop
    \the\numexpr-#1-\xint_c_ix\expandafter.\XINT_keep:f:csv_trimloop_trimnine
    {\csname XINT_trim:f:csv_finish#1\endcsname}%
\long\def\XINT_keep:f:csv_pos #1.#2%
{%
    \expandafter\XINT_keep:f:csv_pos_fork
    \romannumeral0\XINT_lengthupto:f:csv_a
    #1.#2\xint:,\xint:,\xint:,\xint:,%
         \xint:,\xint:,\xint:,\xint:,%
         \xint_c_viii,\xint_c_vii,\xint_c_vi,\xint_c_v,%
         \xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye.%
    .#1.{}#2\xint_bye%
}%
\def\XINT_keep:f:csv_pos_fork #1#2.%
{%
    \xint_UDsignfork
      #1{\expandafter\XINT_keep:f:csv_loop\the\numexpr-\xint_c_viii+}%
       -\XINT_keep:f:csv_pos_keepall
    \krof
}%
\long\def\XINT_keep:f:csv_pos_keepall #1.#2#3\xint_bye{,#3}%
\def\XINT_keep:f:csv_loop #1#2.%
{%
    \xint_gob_til_minus#1\XINT_keep:f:csv_loop_end-%
    \expandafter\XINT_keep:f:csv_loop
    \the\numexpr#1#2-\xint_c_viii\expandafter.\XINT_keep:f:csv_loop_pickeight
}%
\long\def\XINT_keep:f:csv_loop_pickeight
     #1#2,#3,#4,#5,#6,#7,#8,#9,{{#1,#2,#3,#4,#5,#6,#7,#8,#9}}%
\def\XINT_keep:f:csv_loop_end-\expandafter\XINT_keep:f:csv_loop
    \the\numexpr-#1-\xint_c_viii\expandafter.\XINT_keep:f:csv_loop_pickeight
    {\csname XINT_keep:f:csv_end#1\endcsname}%
\long\expandafter\def\csname XINT_keep:f:csv_end1\endcsname
   #1#2,#3,#4,#5,#6,#7,#8,#9\xint_bye {#1,#2,#3,#4,#5,#6,#7,#8}%
\long\expandafter\def\csname XINT_keep:f:csv_end2\endcsname
   #1#2,#3,#4,#5,#6,#7,#8\xint_bye {#1,#2,#3,#4,#5,#6,#7}%
\long\expandafter\def\csname XINT_keep:f:csv_end3\endcsname
   #1#2,#3,#4,#5,#6,#7\xint_bye {#1,#2,#3,#4,#5,#6}%
\long\expandafter\def\csname XINT_keep:f:csv_end4\endcsname
   #1#2,#3,#4,#5,#6\xint_bye {#1,#2,#3,#4,#5}%
\long\expandafter\def\csname XINT_keep:f:csv_end5\endcsname
   #1#2,#3,#4,#5\xint_bye {#1,#2,#3,#4}%
\long\expandafter\def\csname XINT_keep:f:csv_end6\endcsname
   #1#2,#3,#4\xint_bye {#1,#2,#3}%
\long\expandafter\def\csname XINT_keep:f:csv_end7\endcsname
   #1#2,#3\xint_bye {#1,#2}%
\long\expandafter\def\csname XINT_keep:f:csv_end8\endcsname
   #1#2\xint_bye {#1}%
```

3.28.4 \xintTrim:f:csv

1.2g 2016/03/17. Redone for 1.2j 2016/12/20 on the basis of new \xintTrim.

```
\def\xintTrim:f:csv {\romannumeral0\xinttrim:f:csv }%
\long\def\xinttrim:f:csv #1#2%
{%
    \expandafter\xint_gobble_thenstop
    \romannumeral0\expandafter\XINT_trim:f:csv_a
    \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
}%
\def\XINT_trim:f:csv_a #1%
{%
    \xint_UDzerominusfork
        #1-\XINT_trim:f:csv_trimnone
        0#1\XINT_trim:f:csv_neg
         0-{\XINT_trim:f:csv_pos #1}%
    \krof
}%
\long\def\XINT_trim:f:csv_trimnone .#1{,#1}%
\long\def\XINT_trim:f:csv_neg #1.#2%
{%
```

```
    \expandafter\XINT_trim:f:csv_neg_a\the\numexpr
    #1-\numexpr\XINT_length:f:csv_a
    #2\xint:,\xint:,\xint:,\xint:,%
      \xint:,\xint:,\xint:,\xint:,\xint:,%
      \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
      \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
    .{}#2\xint_bye
}%
\def\XINT_trim:f:csv_neg_a #1%
{%
    \xint_UDsignfork
        #1{\expandafter\XINT_keep:f:csv_loop\the\numexpr-\xint_c_viii+}%
         -\XINT_trim:f:csv_trimall
    \krof
}%
\def\XINT_trim:f:csv_trimall {\expandafter,\xint_bye}%
\long\def\XINT_trim:f:csv_pos #1.#2%
{%
    \expandafter\XINT_trim:f:csv_pos_done\expandafter,%
    \romannumeral0%
    \expandafter\XINT_trim:f:csv_loop\the\numexpr#1-\xint_c_ix.%
     #2\xint:,\xint:,\xint:,\xint:,\xint:,%
       \xint:,\xint:,\xint:,\xint:,\xint:\xint_bye
}%
\def\XINT_trim:f:csv_loop #1#2.%
{%
    \xint_gob_til_minus#1\XINT_trim:f:csv_finish-%
    \expandafter\XINT_trim:f:csv_loop\the\numexpr#1#2\XINT_trim:f:csv_loop_trimnine
}%
\long\def\XINT_trim:f:csv_loop_trimnine #1,#2,#3,#4,#5,#6,#7,#8,#9,%
{%
    \xint_gob_til_xint: #9\XINT_trim:f:csv_toofew\xint:-\xint_c_ix.%
}%
\def\XINT_trim:f:csv_toofew\xint:{*\xint_c_}%
\def\XINT_trim:f:csv_finish-%
    \expandafter\XINT_trim:f:csv_loop\the\numexpr-#1\XINT_trim:f:csv_loop_trimnine
{%
    \csname XINT_trim:f:csv_finish#1\endcsname
}%
\long\expandafter\def\csname XINT_trim:f:csv_finish1\endcsname
  #1,#2,#3,#4,#5,#6,#7,#8,{ }%
\long\expandafter\def\csname XINT_trim:f:csv_finish2\endcsname
  #1,#2,#3,#4,#5,#6,#7,{ }%
\long\expandafter\def\csname XINT_trim:f:csv_finish3\endcsname
  #1,#2,#3,#4,#5,#6,{ }%
\long\expandafter\def\csname XINT_trim:f:csv_finish4\endcsname
  #1,#2,#3,#4,#5,{ }%
\long\expandafter\def\csname XINT_trim:f:csv_finish5\endcsname
  #1,#2,#3,#4,{ }%
\long\expandafter\def\csname XINT_trim:f:csv_finish6\endcsname
  #1,#2,#3,{ }%
\long\expandafter\def\csname XINT_trim:f:csv_finish7\endcsname
```

```
  #1,#2,{ }%
\long\expandafter\def\csname XINT_trim:f:csv_finish8\endcsname
  #1,{ }%
\expandafter\let\csname XINT_trim:f:csv_finish9\endcsname\space
\long\def\XINT_trim:f:csv_pos_done #1\xint:#2\xint_bye{#1}%
```

3.28.5 \xintNthEltPy:f:csv

Counts like Python starting at zero. Last refactored with 1.2j. Attention, makes currently no effort at removing leading spaces in the picked item.

```
\def\xintNthEltPy:f:csv {\romannumeral0\xintntheltpy:f:csv }%
\long\def\xintntheltpy:f:csv #1#2%
{%
   \expandafter\XINT_nthelt:f:csv_a
   \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
}%
\def\XINT_nthelt:f:csv_a #1%
{%
    \xint_UDsignfork
        #1\XINT_nthelt:f:csv_neg
         -\XINT_nthelt:f:csv_pos
    \krof #1%
}%
\long\def\XINT_nthelt:f:csv_neg -#1.#2%
{%
    \expandafter\XINT_nthelt:f:csv_neg_fork
    \the\numexpr\XINT_length:f:csv_a
    #2\xint:,\xint:,\xint:,\xint:,%
      \xint:,\xint:,\xint:,\xint:,\xint:,%
      \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
      \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
    -#1.#2,\xint_bye
}%
\def\XINT_nthelt:f:csv_neg_fork #1%
{%
    \if#1-\expandafter\xint_bye_thenstop\fi
    \expandafter\XINT_nthelt:f:csv_neg_done
    \romannumeral0%
    \expandafter\XINT_keep:f:csv_trimloop\the\numexpr-\xint_c_ix+#1%
}%
\long\def\XINT_nthelt:f:csv_neg_done#1,#2\xint_bye{ #1}%
\long\def\XINT_nthelt:f:csv_pos #1.#2%
{%
    \expandafter\XINT_nthelt:f:csv_pos_done
    \romannumeral0%
    \expandafter\XINT_trim:f:csv_loop\the\numexpr#1-\xint_c_ix.%
    #2\xint:,\xint:,\xint:,\xint:,\xint:,%
       \xint:,\xint:,\xint:,\xint:,\xint:,\xint_bye
}%
\def\XINT_nthelt:f:csv_pos_done #1{%
\long\def\XINT_nthelt:f:csv_pos_done ##1,##2\xint_bye{%
  \xint_gob_til_xint:##1\XINT_nthelt:f:csv_pos_cleanup\xint:#1##1}%
}\XINT_nthelt:f:csv_pos_done{ }%
```

This strange thing is in case the picked item was the last one, hence there was an ending \xint: (we could not put a comma earlier for matters of not confusing empty list with a singleton list), and we do this here to activate brace-stripping of item as all other items may be brace-stripped if picked. This is done for coherence. Of course, in the context of the xintexpr.sty parsers, there are no braces in list items...

```
\xint_firstofone{\long\def\XINT_nthelt:f:csv_pos_cleanup\xint:} %
   #1\xint:{ #1}%
```

3.28.6 \xintReverse:f:csv

1.2g. Contrarily to \xintReverseOrder from xintkernel.sty, this one expands its argument. Handles empty list too. 2016/03/17. Made \long for 1.2j.

```
\def\xintReverse:f:csv {\romannumeral0\xintreverse:f:csv }%
\long\def\xintreverse:f:csv #1%
{%
    \expandafter\XINT_reverse:f:csv_loop
    \expandafter{\expandafter}\romannumeral`&&@#1,%
      \xint:,%
        \xint_bye,\xint_bye,\xint_bye,\xint_bye,%
        \xint_bye,\xint_bye,\xint_bye,\xint_bye,%
      \xint:
}%
\long\def\XINT_reverse:f:csv_loop #1#2,#3,#4,#5,#6,#7,#8,#9,%
{%
    \xint_bye #9\XINT_reverse:f:csv_cleanup\xint_bye
    \XINT_reverse:f:csv_loop {,#9,#8,#7,#6,#5,#4,#3,#2#1}%
}%
\long\def\XINT_reverse:f:csv_cleanup\xint_bye\XINT_reverse:f:csv_loop #1#2\xint:
{%
    \XINT_reverse:f:csv_finish #1%
}%
\long\def\XINT_reverse:f:csv_finish #1\xint:,{ }%
```

3.28.7 \xintFirstItem:f:csv

Added with 1.2k for use by first() in \xintexpr-essions, and some amount of compatibility with \xintNewExpr.

```
\def\xintFirstItem:f:csv {\romannumeral0\xintfirstitem:f:csv}%
\long\def\xintfirstitem:f:csv #1%
{%
    \expandafter\XINT_first:f:csv_a\romannumeral`&&@#1,\xint_bye
}%
\long\def\XINT_first:f:csv_a #1,#2\xint_bye{ #1}%
```

3.28.8 \xintLastItem:f:csv

Added with 1.2k, based on and sharing code with xintkernel's \xintLastItem from 1.2i. Output empty if input empty. f-expands its argument (hence first item, if not protected.) For use by last() in \xintexpr-essions with to some extent \xintNewExpr compatibility.

```
\def\xintLastItem:f:csv {\romannumeral0\xintlastitem:f:csv}%
\long\def\xintlastitem:f:csv #1%
{%
    \expandafter\XINT_last:f:csv_loop\expandafter{\expandafter}\expandafter.%
    \romannumeral`&&@#1,%
    \xint:\XINT_last_loop_enda,\xint:\XINT_last_loop_endb,%
    \xint:\XINT_last_loop_endc,\xint:\XINT_last_loop_endd,%
    \xint:\XINT_last_loop_ende,\xint:\XINT_last_loop_endf,%
    \xint:\XINT_last_loop_endg,\xint:\XINT_last_loop_endh,\xint_bye
}%
\long\def\XINT_last:f:csv_loop #1.#2,#3,#4,#5,#6,#7,#8,#9,%
{%
    \xint_gob_til_xint: #9%
        {#8}{#7}{#6}{#5}{#4}{#3}{#2}{#1}\xint:
    \XINT_last:f:csv_loop {#9}.%
}%
```

3.28.9 Public names for the undocumented csv macros: \xintCSVLength, \xintCSVKeep, \xintCSVTrim, \xintCSVNthEltPy, \xintCSVReverse, \xintCSVFirstItem, \xintCSVLastItem

Completely unstable macros: currently they expand the list argument and want no final comma. But for matters of xintexpr.sty I could as well decide to require a final comma, and then I could simplify implementation but of course this would break the macros if used with current functionalities.

```
\let\xintCSVLength   \xintLength:f:csv
\let\xintCSVKeep     \xintKeep:f:csv
\let\xintCSVTrim     \xintTrim:f:csv
\let\xintCSVNthEltPy \xintNthEltPy:f:csv
\let\xintCSVReverse  \xintReverse:f:csv
\let\xintCSVFirstItem\xintFirstItem:f:csv
\let\xintCSVLastItem \xintLastItem:f:csv
\let\XINT_tmpa\relax \let\XINT_tmpb\relax \let\XINT_tmpc\relax
\XINT_restorecatcodes_endinput%
```

4 Package xintcore implementation

Got split off from xint with release 1.1.

The core arithmetic routines have been entirely rewritten for release 1.2. The 1.2i and 1.2l brought again some improvements.

The commenting continues (2018/06/17) to be very sparse: actually it got worse than ever with release 1.2. I will possibly add comments at a later date, but for the time being the new routines are not commented at all.

1.3 removes all macros which were deprecated at 1.2o.

4.1 Catcodes, ε-TEX and reload detection

The code for reload detection was initially copied from Heiko Oberdiek's packages, then modified.

The method for catcodes was also initially directly inspired by these packages.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5     % ^^M
  \endlinechar=13 %
  \catcode123=1    % {
  \catcode125=2    % }
  \catcode64=11    % @
  \catcode35=6     % #
  \catcode44=12    % ,
  \catcode45=12    % -
  \catcode46=12    % .
  \catcode58=12    % :
```

```
  \let\z\endgroup
  \expandafter\let\expandafter\x\csname ver@xintcore.sty\endcsname
  \expandafter\let\expandafter\w\csname ver@xintkernel.sty\endcsname
  \expandafter
    \ifx\csname PackageInfo\endcsname\relax
      \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
    \else
      \def\y#1#2{\PackageInfo{#1}{#2}}%
    \fi
  \expandafter
  \ifx\csname numexpr\endcsname\relax
     \y{xintcore}{\numexpr not available, aborting input}%
     \aftergroup\endinput
  \else
    \ifx\x\relax   % plain-TeX, first loading of xintcore.sty
      \ifx\w\relax % but xintkernel.sty not yet loaded.
         \def\z{\endgroup\input xintkernel.sty\relax}%
      \fi
    \else
      \def\empty {}%
      \ifx\x\empty % LaTeX, first loading,
      % variable is initialized, but \ProvidesPackage not yet seen
          \ifx\w\relax % xintkernel.sty not yet loaded.
            \def\z{\endgroup\RequirePackage{xintkernel}}%
          \fi
      \else
        \aftergroup\endinput % xintkernel already loaded.
      \fi
    \fi
  \fi
\z%
\XINTsetupcatcodes% defined in xintkernel.sty
```

4.2 Package identification

```
\XINT_providespackage
\ProvidesPackage{xintcore}%
  [2018/06/17 1.3c Expandable arithmetic on big integers (JFB)]%
```

4.3 (WIP!) Error conditions and exceptions

As per the Mike Cowlishaw/IBM's General Decimal Arithmetic Specification
http://speleotrove.com/decimal/decarith.html
and the Python3 implementation in its Decimal module.

Clamped, ConversionSyntax, DivisionByZero, DivisionImpossible, DivisionUndefined, Inexact, InsufficientStorage, InvalidContext, InvalidOperation, Overflow, Inexact, Rounded, Subnormal, Underflow.

X3.274 rajoute LostDigits

Python rajoute FloatOperation (et n'inclut pas InsufficientStorage)

quote de decarith.pdf: The Clamped, Inexact, Rounded, and Subnormal conditions can coincide with each other or with other conditions. In these cases then any trap enabled for another condition takes precedence over (is handled before) all of these, any Subnormal trap takes precedence over Inexact, any Inexact trap takes precedence over Rounded, and any Rounded trap takes precedence over Clamped.

WORK IN PROGRESS ! (1.2l, 2017/07/26)

I follow the Python terminology: a trapped signal means it raises an exception which for us means an expandable error message with some possible user interaction. In this WIP state, the interaction is commented out. A non-trapped signal or condition would activate a (presumably silent) handler.

Here, no signal-raising condition is "ignored" and all are "trapped" which means that error handlers are never activated, thus left in garbage state in the code.

Various conditions can raise the same signal.

Only signals, not conditions, raise Flags.

If a signal is ignored it does not raise a Flag, but it activates the signal handler (by default now no signal is ignored.)

If a signal is not ignored it raises a Flag and then if it is not trapped it activates the handler of the _condition_.

If trapped (which is default now) an «exception» is raised, which means an expandable error message (I copied over the LaTeX3 code for expandable error messages, basically) interrupts the TeX run. In future, user input could be solicited, but currently this is commented out.

For now macros to reset flags are done but without public interface nor documentation.

Only four conditions are currently possibly encountered:

- InvalidOperation
- DivisionByZero
- DivisionUndefined (which signals InvalidOperation)
- Underflow

I did it quickly, anyhow this will become more palpable when some of the Decimal Specification is actually implemented. The plan is to first do the X3.274 norm, then more complete implementation will follow... perhaps...

```
\csname XINT_Clamped_istrapped\endcsname
\csname XINT_ConversionSyntax_istrapped\endcsname
\csname XINT_DivisionByZero_istrapped\endcsname
\csname XINT_DivisionImpossible_istrapped\endcsname
\csname XINT_DivisionUndefined_istrapped\endcsname
\csname XINT_InvalidOperation_istrapped\endcsname
\csname XINT_Overflow_istrapped\endcsname
\csname XINT_Underflow_istrapped\endcsname
```

```
\catcode`- 11
\def\XINT_ConversionSyntax-signal  {{InvalidOperation}}%
\let\XINT_DivisionImpossible-signal\XINT_ConversionSyntax-signal
\let\XINT_DivisionUndefined-signal \XINT_ConversionSyntax-signal
\let\XINT_InvalidContext-signal    \XINT_ConversionSyntax-signal
\catcode`- 12
\def\XINT_signalcondition #1{\expandafter\XINT_signalcondition_a
    \romannumeral0\ifcsname XINT_#1-signal\endcsname
                    \xint_dothis{\csname XINT_#1-signal\endcsname}%
                  \fi\xint_orthat{{#1}}{#1}}%
\def\XINT_signalcondition_a #1#2#3#4#5{% copied over from Python Decimal module
% #1=signal, #2=condition, #3=explanation for user,
% #4=context for error handlers, #5=used
    \ifcsname XINT_#1_isignoredflag\endcsname
       \xint_dothis{\csname XINT_#1.handler\endcsname {#4}}%
    \fi
    \expandafter\xint_gobble_i\csname XINT_#1Flag_ON\endcsname
    \unless\ifcsname XINT_#1_istrapped\endcsname
```

```
      \xint_dothis{\csname XINT_#2.handler\endcsname {#4}}%
    \fi
    \xint_orthat{%
      % the flag raised is named after the signal #1, but we show condition #2
      \XINT_expandableerror{#2 (hit <RET> thrice)}%
      \XINT_expandableerror{#3}%
      \XINT_expandableerror{next: #5}%
      % not for X3.274
      %\XINT_expandableerror{<RET>, or I\xintUse{...}<RET>, or I\xintCTRLC<RET>}%
      \xint_firstofone_thenstop{#5}%
    }%
}%
%% \let\xintUse\xint_firstofthree_thenstop % defined in xint.sty
\def\XINT_ifFlagRaised #1{%
    \ifcsname XINT_#1Flag_ON\endcsname
        \expandafter\xint_firstoftwo
    \else
        \expandafter\xint_secondoftwo
    \fi}%
\def\XINT_resetFlag #1%
    {\expandafter\let\csname XINT_#1Flag_ON\endcsname\XINT_undefined}%
\def\XINT_resetFlags {% WIP
    \XINT_resetFlag{InvalidOperation}% also from DivisionUndefined
    \XINT_resetFlag{DivisionByZero}%
    \XINT_resetFlag{Underflow}% (\xintiiPow with negative exponent)
    \XINT_resetFlag{Overflow}%   not encountered so far in xint code 1.21
    % .. others ..
}%
\def\XINT_RaiseFlag #1{\expandafter\xint_gobble_i\csname XINT_#1Flag_ON\endcsname}%
```

NOT IMPLEMENTED! WORK IN PROGRESS! (ALL SIGNALS TRAPPED, NO HANDLERS USED)

```
\catcode`. 11
\let\XINT_Clamped.handler\xint_firstofone % WIP
\def\XINT_InvalidOperation.handler#1{_NaN}% WIP
\def\XINT_ConversionSyntax.handler#1{_NaN}% WIP
\def\XINT_DivisionByZero.handler#1{_SignedInfinity(#1)}% WIP
\def\XINT_DivisionImpossible.handler#1{_NaN}% WIP
\def\XINT_DivisionUndefined.handler#1{_NaN}%  WIP
\let\XINT_Inexact.handler\xint_firstofone  %  WIP
\def\XINT_InvalidContext.handler#1{_NaN}%     WIP
\let\XINT_Rounded.handler\xint_firstofone  %  WIP
\let\XINT_Subnormal.handler\xint_firstofone%  WIP
\def\XINT_Overflow.handler#1{_NaN}%  WIP
\def\XINT_Underflow.handler#1{_NaN}% WIP
\catcode`. 12
```

4.4 Counts for holding needed constants

```
\ifdefined\m@ne\let\xint_c_mone\m@ne
          \else\csname newcount\endcsname\xint_c_mone \xint_c_mone -1 \fi
\ifdefined\xint_c_x^viii\else
\csname newcount\endcsname\xint_c_x^viii \xint_c_x^viii   100000000
\fi
```

```
\ifdefined\xint_c_x^ix\else
\csname newcount\endcsname\xint_c_x^ix   \xint_c_x^ix    1000000000
\fi
\newcount\xint_c_x^viii_mone        \xint_c_x^viii_mone    99999999
\newcount\xint_c_xii_e_viii          \xint_c_xii_e_viii  1200000000
\newcount\xint_c_xi_e_viii_mone  \xint_c_xi_e_viii_mone  1099999999
```

Routines handling integers as lists of token digits

Routines handling big integers which are lists of digit tokens with no special additional structure.

Some routines do not accept non properly terminated inputs like "\the\numexpr1", or "\the\mathcode`\-", others do.

These routines or their sub-routines are mainly for internal usage.

4.5 \XINT_cuz_small

\XINT_cuz_small removes leading zeroes from the first eight digits. Expands following \romannumeral0. At least one digit is produced.

```
\def\XINT_cuz_small#1{%
\def\XINT_cuz_small ##1##2##3##4##5##6##7##8%
{%
    \expandafter#1\the\numexpr ##1##2##3##4##5##6##7##8\relax
}}\XINT_cuz_small{ }%
```

4.6 \xintNum, \xintiNum

For example \xintNum {----+-+++---+----000000000000003}

Very old routine got completely rewritten at 1.2l.

New code uses \numexpr governed expansion and fixes some issues of former version particularly regarding inputs of the \numexpr...\relax type without \the or \number prefix, and/or possibly no terminating \relax.

\xintiNum{\numexpr 1}\foo in earlier versions caused premature expansion of \foo.

\xintiNum{\the\numexpr 1} was ok, but a bit luckily so.

Also, up to 1.2k inclusive, the macro fetched tokens eight by eight, and not nine by nine as is done now. I have no idea why.

\xintNum gets redefined by xintfrac.

```
\def\xintiNum {\romannumeral0\xintinum }%
\def\xintinum #1%
{%
    \expandafter\XINT_num_cleanup\the\numexpr\expandafter\XINT_num_loop
    \romannumeral`&&@#1\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z
}%
\def\xintNum {\romannumeral0\xintnum }%
\let\xintnum\xintinum
\def\XINT_num #1%
{%
    \expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
    #1\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z
}%
\def\XINT_num_loop #1#2#3#4#5#6#7#8#9%
```

```
{%
    \xint_gob_til_xint: #9\XINT_num_end\xint:
    #1#2#3#4#5#6#7#8#9%
    \ifnum \numexpr #1#2#3#4#5#6#7#8#9+\xint_c_ = \xint_c_
```

means that so far only signs encountered, (if syntax is legal) then possibly zeroes or a terminated or not terminated \numexpr evaluating to zero In that latter case a correct zero will be produced in the end.

```
      \expandafter\XINT_num_loop
    \else
```

non terminated \numexpr (with nine tokens total) are safe as after \fi, there is then \xint:

```
     \expandafter\relax
    \fi
}%
\def\XINT_num_end\xint:#1\xint:{#1+\xint_c_\xint:}% empty input ok
\def\XINT_num_cleanup #1\xint:#2\Z { #1}%
```

4.7 \xintiiSgn

1.2l made \xintiiSgn robust against non terminated input.
1.2o deprecates here \xintSgn (it requires xintfrac.sty).

```
\def\xintiiSgn {\romannumeral0\xintiisgn }%
\def\xintiisgn #1%
{%
    \expandafter\XINT_sgn \romannumeral`&&@#1\xint:
}%
\def\XINT_sgn #1#2\xint:
{%
    \xint_UDzerominusfork
      #1-{ 0}%
      0#1{-1}%
       0-{ 1}%
    \krof
}%
\def\XINT_Sgn #1#2\xint:
{%
    \xint_UDzerominusfork
      #1-{0}%
      0#1{-1}%
       0-{1}%
    \krof
}%
\def\XINT_cntSgn #1#2\xint:
{%
    \xint_UDzerominusfork
      #1-\xint_c_
      0#1\xint_c_mone
       0-\xint_c_i
    \krof
}%
```

4.8 \xintiiOpp

Attention, \xintiiOpp non robust against non terminated inputs. Reason is I don't want to have to grab a delimiter at the end, as everything happens "upfront".

```
\def\xintiiOpp {\romannumeral0\xintiiopp }%
\def\xintiiopp #1%
{%
    \expandafter\XINT_opp \romannumeral`&&@#1%
}%
\def\XINT_Opp #1{\romannumeral0\XINT_opp #1}%
\def\XINT_opp #1%
{%
    \xint_UDzerominusfork
      #1-{ 0}%      zero
      0#1{ }%     negative
       0-{ -#1}%  positive
    \krof
}%
```

4.9 \xintiiAbs

Attention \xintiiAbs non robust against non terminated input.

```
\def\xintiiAbs {\romannumeral0\xintiiabs }%
\def\xintiiabs #1%
{%
    \expandafter\XINT_abs \romannumeral`&&@#1%
}%
\def\XINT_abs #1%
{%
    \xint_UDsignfork
      #1{ }%
       -{ #1}%
    \krof
}%
```

4.10 \xintFDg

FIRST DIGIT.
  1.2l: \xintiiFDg made robust against non terminated input.
  1.2o deprecates \xintiiFDg, gives to \xintFDg former meaning of \xintiiFDg.

```
\def\xintFDg {\romannumeral0\xintfdg }%
\def\xintfdg #1{\expandafter\XINT_fdg \romannumeral`&&@#1\xint:\Z}%
\def\XINT_FDg #1%
   {\romannumeral0\expandafter\XINT_fdg\romannumeral`&&@\xintnum{#1}\xint:\Z }%
\def\XINT_fdg #1#2#3\Z
{%
    \xint_UDzerominusfork
      #1-{ 0}%   zero
      0#1{ #2}%  negative
       0-{ #1}%  positive
    \krof
}%
```

4.11 \xintLDg

LAST DIGIT.
  Rewritten for 1.2i (2016/12/10). Surprisingly perhaps, it is faster than \xintLastItem from xintkernel.sty despite the \numexpr operations.
  1.2o deprecates \xintiiLDg, gives to \xintLDg former meaning of \xintiiLDg.
  Attention \xintLDg non robust against non terminated input.

```
\def\xintLDg {\romannumeral0\xintldg }%
\def\xintldg #1{\expandafter\XINT_ldg_fork\romannumeral`&&@#1%
    \XINT_ldg_c{}{}{}{}{}{}{}{}{}\xint_bye\relax}%
\def\XINT_ldg_fork #1%
{%
    \xint_UDsignfork
      #1\XINT_ldg
       -{\XINT_ldg#1}%
    \krof
}%
\def\XINT_ldg #1{%
\def\XINT_ldg ##1##2##3##4##5##6##7##8##9%
   {\expandafter#1%
    \the\numexpr##9##8##7##6##5##4##3##2##1*\xint_c_+\XINT_ldg_a##9}%
}\XINT_ldg{ }%
\def\XINT_ldg_a#1#2{\XINT_ldg_cbye#2\XINT_ldg_d#1\XINT_ldg_c\XINT_ldg_b#2}%
\def\XINT_ldg_b#1#2#3#4#5#6#7#8#9{#9#8#7#6#5#4#3#2#1*\xint_c_+\XINT_ldg_a#9}%
\def\XINT_ldg_c    #1#2\xint_bye{#1}%
\def\XINT_ldg_cbye #1\XINT_ldg_c{}%
\def\XINT_ldg_d#1#2\xint_bye{#1}%
```

4.12 \xintDouble

Attention \xintDouble non robust against non terminated input.

```
\def\xintDouble {\romannumeral0\xintdouble}%
\def\xintdouble #1{\expandafter\XINT_dbl_fork\romannumeral`&&@#1%
                   \xint_bye2345678\xint_bye*\xint_c_ii\relax}%
\def\XINT_dbl_fork #1%
{%
    \xint_UDsignfork
     #1\XINT_dbl_neg
      -\XINT_dbl
    \krof #1%
}%
\def\XINT_dbl_neg-{\expandafter-\romannumeral0\XINT_dbl}%
\def\XINT_dbl #1{%
\def\XINT_dbl ##1##2##3##4##5##6##7##8%
   {\expandafter#1\the\numexpr##1##2##3##4##5##6##7##8\XINT_dbl_a}%
}\XINT_dbl{ }%
\def\XINT_dbl_a #1#2#3#4#5#6#7#8%
   {\expandafter\XINT_dbl_e\the\numexpr 1#1#2#3#4#5#6#7#8\XINT_dbl_a}%
```

```
\def\XINT_dbl_e#1{*\xint_c_ii\if#13+\xint_c_i\fi\relax}%
```

4.13 \xintHalf

Attention \xintHalf non robust against non terminated input.

```
\def\xintHalf {\romannumeral0\xinthalf}%
\def\xinthalf #1{\expandafter\XINT_half_fork\romannumeral`&&@#1%
    \xint_bye\xint_Bye345678\xint_bye
    *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax}%
\def\XINT_half_fork #1%
{%
    \xint_UDsignfork
     #1\XINT_half_neg
      -\XINT_half
    \krof #1%
}%
\def\XINT_half_neg-{\xintiiopp\XINT_half}%
\def\XINT_half #1{%
\def\XINT_half ##1##2##3##4##5##6##7##8%
   {\expandafter#1\the\numexpr(##1##2##3##4##5##6##7##8\XINT_half_a}%
}\XINT_half{ }%
\def\XINT_half_a#1{\xint_Bye#1\xint_bye\XINT_half_b#1}%
\def\XINT_half_b #1#2#3#4#5#6#7#8%
   {\expandafter\XINT_half_e\the\numexpr(1#1#2#3#4#5#6#7#8\XINT_half_a}%
\def\XINT_half_e#1{*\xint_c_v+#1-\xint_c_v)\relax}%
```

4.14 \xintInc

1.2i much delayed complete rewrite in 1.2 style.

As we take 9 by 9 with the input save stack at 5000 this allows a bit less than 9 times 2500 = 22500 digits on input.

Attention \xintInc non robust against non terminated input.

```
\def\xintInc {\romannumeral0\xintinc}%
\def\xintinc #1{\expandafter\XINT_inc_fork\romannumeral`&&@#1%
                 \xint_bye23456789\xint_bye+\xint_c_i\relax}%
\def\XINT_inc_fork #1%
{%
    \xint_UDsignfork
      #1\XINT_inc_neg
       -\XINT_inc
    \krof #1%
}%
\def\XINT_inc_neg-#1\xint_bye#2\relax
   {\xintiiopp\XINT_dec #1\XINT_dec_bye234567890\xint_bye}%
\def\XINT_inc #1{%
\def\XINT_inc ##1##2##3##4##5##6##7##8##9%
   {\expandafter#1\the\numexpr##1##2##3##4##5##6##7##8##9\XINT_inc_a}%
}\XINT_inc{ }%
\def\XINT_inc_a #1#2#3#4#5#6#7#8#9%
   {\expandafter\XINT_inc_e\the\numexpr 1#1#2#3#4#5#6#7#8#9\XINT_inc_a}%
\def\XINT_inc_e#1{\if#12+\xint_c_i\fi\relax}%
```

4.15 \xintDec

1.2i much delayed complete rewrite in the 1.2 style. Things are a bit more complicated than \xintInc because 2999999999 is too big for TeX.

Attention \xintDec non robust against non terminated input.

```
\def\xintDec {\romannumeral0\xintdec}%
\def\xintdec #1{\expandafter\XINT_dec_fork\romannumeral`&&@#1%
                \XINT_dec_bye234567890\xint_bye}%
\def\XINT_dec_fork #1%
{%
    \xint_UDsignfork
      #1\XINT_dec_neg
       -\XINT_dec
    \krof #1%
}%
\def\XINT_dec_neg-#1\XINT_dec_bye#2\xint_bye
   {\expandafter-%
    \romannumeral0\XINT_inc #1\xint_bye23456789\xint_bye+\xint_c_i\relax}%
\def\XINT_dec #1{%
\def\XINT_dec ##1##2##3##4##5##6##7##8##9%
   {\expandafter#1\the\numexpr##1##2##3##4##5##6##7##8##9\XINT_dec_a}%
}\XINT_dec{ }%
\def\XINT_dec_a #1#2#3#4#5#6#7#8#9%
   {\expandafter\XINT_dec_e\the\numexpr 1#1#2#3#4#5#6#7#8#9\XINT_dec_a}%
\def\XINT_dec_bye #1\XINT_dec_a#2#3\xint_bye
         {\if#20-\xint_c_ii\relax+\else-\fi\xint_c_i\relax}%
\def\XINT_dec_e#1{\unless\if#11\xint_dothis{-\xint_c_i#1}\fi\xint_orthat\relax}%
```

4.16 \xintDSL

DECIMAL SHIFT LEFT (=MULTIPLICATION PAR 10). Rewritten for 1.2i. This was very old code... I never came back to it, but I should have rewritten it long time ago.

Attention \xintDSL non robust against non terminated input.

```
\def\xintDSL {\romannumeral0\xintdsl }%
\def\xintdsl #1{\expandafter\XINT_dsl\romannumeral`&&@#10}%
\def\XINT_dsl#1{%
\def\XINT_dsl ##1{\xint_gob_til_zero ##1\xint_dsl_zero 0#1##1}%
}\XINT_dsl{ }%
\def\xint_dsl_zero 0 0{ }%
```

4.17 \xintDSR

Decimal shift right, truncates towards zero. Rewritten for 1.2i. Limited to 22483 digits on input.

Attention \xintDSR non robust against non terminated input.

```
\def\xintDSR{\romannumeral0\xintdsr}%
\def\xintdsr #1{\expandafter\XINT_dsr_fork\romannumeral`&&@#1%
    \xint_bye\xint_Bye3456789\xint_bye+\xint_c_v)/\xint_c_x-\xint_c_i\relax}%
\def\XINT_dsr_fork #1%
{%
    \xint_UDsignfork
```

```
      #1\XINT_dsr_neg
       -\XINT_dsr
    \krof #1%
}%
\def\XINT_dsr_neg-{\xintiiopp\XINT_dsr}%
\def\XINT_dsr #1{%
\def\XINT_dsr ##1##2##3##4##5##6##7##8##9%
   {\expandafter#1\the\numexpr(##1##2##3##4##5##6##7##8##9\XINT_dsr_a}%
}\XINT_dsr{ }%
\def\XINT_dsr_a#1{\xint_Bye#1\xint_bye\XINT_dsr_b#1}%
\def\XINT_dsr_b #1#2#3#4#5#6#7#8#9%
   {\expandafter\XINT_dsr_e\the\numexpr(1#1#2#3#4#5#6#7#8#9\XINT_dsr_a}%
\def\XINT_dsr_e #1{)\relax}%
```

4.18 \xintDSRr

New with 1.2i. Decimal shift right, rounds away from zero; done in the 1.2 spirit (with much delay, sorry). Used by \xintRound, \xintDivRound.

This is about the first time I am happy that the division in \numexpr rounds!

Attention \xintDSRr non robust against non terminated input.

```
\def\xintDSRr{\romannumeral0\xintdsrr}%
\def\xintdsrr #1{\expandafter\XINT_dsrr_fork\romannumeral`&&@#1%
                \xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax}%
\def\XINT_dsrr_fork #1%
{%
    \xint_UDsignfork
      #1\XINT_dsrr_neg
       -\XINT_dsrr
    \krof #1%
}%
\def\XINT_dsrr_neg-{\xintiiopp\XINT_dsrr}%
\def\XINT_dsrr #1{%
\def\XINT_dsrr ##1##2##3##4##5##6##7##8##9%
   {\expandafter#1\the\numexpr##1##2##3##4##5##6##7##8##9\XINT_dsrr_a}%
}\XINT_dsrr{ }%
\def\XINT_dsrr_a#1{\xint_Bye#1\xint_bye\XINT_dsrr_b#1}%
\def\XINT_dsrr_b #1#2#3#4#5#6#7#8#9%
   {\expandafter\XINT_dsrr_e\the\numexpr1#1#2#3#4#5#6#7#8#9\XINT_dsrr_a}%
\let\XINT_dsrr_e\XINT_inc_e
```

Blocks of eight digits

The lingua of release 1.2.

4.19 \XINT_cuz

This (launched by \romannumeral0) iterately removes all leading zeroes from a sequence of 8N digits ended by \R.

Rewritten for 1.2l, now uses \numexpr governed expansion and \ifnum test rather than delimited gobbling macros.

Note 2015/11/28: with only four digits the gob_til_fourzeroes had proved in some old testing faster than \ifnum test. But with eight digits, the execution times are much closer, as I tested back then.

```
\def\XINT_cuz #1{%
\def\XINT_cuz {\expandafter#1\the\numexpr\XINT_cuz_loop}%
}\XINT_cuz{ }%
\def\XINT_cuz_loop #1#2#3#4#5#6#7#8#9%
{%
    #1#2#3#4#5#6#7#8%
       \xint_gob_til_R #9\XINT_cuz_hitend\R
       \ifnum #1#2#3#4#5#6#7#8>\xint_c_
             \expandafter\XINT_cuz_cleantoend
       \else\expandafter\XINT_cuz_loop
       \fi #9%
}%
\def\XINT_cuz_hitend\R #1\R{\relax}%
\def\XINT_cuz_cleantoend #1\R{\relax #1}%
```

4.20 \XINT_cuz_byviii

This removes eight by eight leading zeroes from a sequence of 8N digits ended by \R. Thus, we still have 8N digits on output. Expansion started by \romannumeral0

```
\def\XINT_cuz_byviii #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_R #9\XINT_cuz_byviii_e \R
    \xint_gob_til_eightzeroes #1#2#3#4#5#6#7#8\XINT_cuz_byviii_z 00000000%
    \XINT_cuz_byviii_done #1#2#3#4#5#6#7#8#9%
}%
\def\XINT_cuz_byviii_z 00000000\XINT_cuz_byviii_done 00000000{\XINT_cuz_byviii}%
\def\XINT_cuz_byviii_done #1\R { #1}%
\def\XINT_cuz_byviii_e\R #1\XINT_cuz_byviii_done #2\R{ #2}%
```

4.21 \XINT_unsep_loop

This is used as

```
   \the\numexpr0\XINT_unsep_loop (blocks of 1<8digits>!)
                 \xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax
```

It removes the 1's and !'s, and outputs the 8N digits with a 0 token as as prefix which will have to be cleaned out by caller.

Actually it does not matter whether the blocks contain really 8 digits, all that matters is that they have 1 as first digit (and at most 9 digits after that to obey the TeX-\numexpr bound).

Done at 1.2l for usage by other macros. The similar code in earlier releases was strangely in O(N^2) style, apparently to avoid some memory constraints. But these memory constraints related to \numexpr chaining seems to be in many places in xint code base. The 1.2l version is written in the 1.2i style of \xintInc etc... and is compatible with some 1! block without digits among the treated blocks, they will disappear.

```
\def\XINT_unsep_loop #1!#2!#3!#4!#5!#6!#7!#8!#9!%
{%
    \expandafter\XINT_unsep_clean
    \the\numexpr #1\expandafter\XINT_unsep_clean
    \the\numexpr #2\expandafter\XINT_unsep_clean
    \the\numexpr #3\expandafter\XINT_unsep_clean
    \the\numexpr #4\expandafter\XINT_unsep_clean
    \the\numexpr #5\expandafter\XINT_unsep_clean
    \the\numexpr #6\expandafter\XINT_unsep_clean
    \the\numexpr #7\expandafter\XINT_unsep_clean
    \the\numexpr #8\expandafter\XINT_unsep_clean
    \the\numexpr #9\XINT_unsep_loop
}%
\def\XINT_unsep_clean 1{\relax}%
```

4.22 \XINT_unsep_cuzsmall

```
This is used as
   \romannumeral0\XINT_unsep_cuzsmall (blocks of 1<8d>!)
                  \xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax
It removes the 1's and !'s, and removes the leading zeroes *of the first block*.
  Redone for 1.2l: the 1.2 variant was strangely in O(N^2) style.
```

```
\def\XINT_unsep_cuzsmall
{%
    \expandafter\XINT_unsep_cuzsmall_x\the\numexpr0\XINT_unsep_loop
}%
\def\XINT_unsep_cuzsmall_x #1{%
\def\XINT_unsep_cuzsmall_x 0##1##2##3##4##5##6##7##8%
{%
    \expandafter#1\the\numexpr ##1##2##3##4##5##6##7##8\relax
}}\XINT_unsep_cuzsmall_x{ }%
```

4.23 \XINT_div_unsepQ

This is used by division to remove separators from the produced quotient. The quotient is produced in the correct order. The routine will also remove leading zeroes. An extra initial block of 8 zeroes is possible and thus if present must be removed. Then the next eight digits must be cleaned of leading zeroes. Attention that there might be a single block of 8 zeroes. Expansion launched by \romannumeral0.
  Rewritten for 1.2l in 1.2i style.

```
\def\XINT_div_unsepQ_delim {\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax\Z}%
\def\XINT_div_unsepQ
{%
    \expandafter\XINT_div_unsepQ_x\the\numexpr0\XINT_unsep_loop
}%
\def\XINT_div_unsepQ_x #1{%
\def\XINT_div_unsepQ_x 0##1##2##3##4##5##6##7##8##9%
{%
    \xint_gob_til_Z ##9\XINT_div_unsepQ_one\Z
    \xint_gob_til_eightzeroes ##1##2##3##4##5##6##7##8\XINT_div_unsepQ_y 00000000%
    \expandafter#1\the\numexpr ##1##2##3##4##5##6##7##8\relax ##9%
}}\XINT_div_unsepQ_x{ }%
\def\XINT_div_unsepQ_y #1{%
\def\XINT_div_unsepQ_y ##1\relax ##2##3##4##5##6##7##8##9%
{%
```

```
    \expandafter#1\the\numexpr ##2##3##4##5##6##7##8##9\relax
}}\XINT_div_unsepQ_y{ }%
\def\XINT_div_unsepQ_one#1\expandafter{\expandafter}%
```

4.24 \XINT_div_unsepR

This is used by division to remove separators from the produced remainder. The remainder is here in correct order. It must be cleaned of leading zeroes, possibly all the way.

Also rewritten for 1.21, the 1.2 version was O(N^2) style.

Terminator \xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax\R

We have a need for something like \R because it is not guaranteed the thing is not actually zero.

```
\def\XINT_div_unsepR
{%
    \expandafter\XINT_div_unsepR_x\the\numexpr0\XINT_unsep_loop
}%
\def\XINT_div_unsepR_x#1{%
\def\XINT_div_unsepR_x 0{\expandafter#1\the\numexpr\XINT_cuz_loop}%
}\XINT_div_unsepR_x{ }%
```

4.25 \XINT_zeroes_forviii

\romannumeral0\XINT_zeroes_forviii #1\R\R\R\R\R\R\R\R{10}0000001\W
produces a string of k 0's such that k+length(#1) is smallest bigger multiple of eight.

```
\def\XINT_zeroes_forviii #1#2#3#4#5#6#7#8%
{%
    \xint_gob_til_R #8\XINT_zeroes_forviii_end\R\XINT_zeroes_forviii
}%
\def\XINT_zeroes_forviii_end#1{%
\def\XINT_zeroes_forviii_end\R\XINT_zeroes_forviii ##1##2##3##4##5##6##7##8##9\W
{%
    \expandafter#1\xint_gob_til_one ##2##3##4##5##6##7##8%
}}\XINT_zeroes_forviii_end{ }%
```

4.26 \XINT_sepbyviii_Z

This is used as
  \the\numexpr\XINT_sepbyviii_Z <8Ndigits>\XINT_sepbyviii_Z_end 2345678\relax
It produces 1<8d>!...1<8d>!1;!

Prior to 1.21 it used \Z as terminator not the semi-colon (hence the name). The switch to ; was done at a time I thought perhaps I would use an internal format maintaining such 8 digits blocks, and this has to be compatible with the \csname...\endcsname encapsulation in \xintexpr parsers.

```
\def\XINT_sepbyviii_Z #1#2#3#4#5#6#7#8%
{%
    1#1#2#3#4#5#6#7#8\expandafter!\the\numexpr\XINT_sepbyviii_Z
}%
\def\XINT_sepbyviii_Z_end #1\relax {;!}%
```

4.27 \XINT_sepbyviii_andcount

```
This is used as
   \the\numexpr\XINT_sepbyviii_andcount <8Ndigits>%
       \XINT_sepbyviii_end 2345678\relax
       \xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!%
       \xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_\W
It will produce
   1<8d>!1<8d>!....1<8d>!1\xint:<count of blocks>\xint:
Used by \XINT_div_prepare_g for \XINT_div_prepare_h, and also by \xintiiCmp.
```

```
\def\XINT_sepbyviii_andcount
{%
    \expandafter\XINT_sepbyviii_andcount_a\the\numexpr\XINT_sepbyviii
}%
\def\XINT_sepbyviii #1#2#3#4#5#6#7#8%
{%
    1#1#2#3#4#5#6#7#8\expandafter!\the\numexpr\XINT_sepbyviii
}%
\def\XINT_sepbyviii_end #1\relax {\relax\XINT_sepbyviii_andcount_end!}%
\def\XINT_sepbyviii_andcount_a {\XINT_sepbyviii_andcount_b \xint_c_\xint:}%
\def\XINT_sepbyviii_andcount_b #1\xint:#2!#3!#4!#5!#6!#7!#8!#9!%
{%
    #2\expandafter!\the\numexpr#3\expandafter!\the\numexpr#4\expandafter
    !\the\numexpr#5\expandafter!\the\numexpr#6\expandafter!\the\numexpr
    #7\expandafter!\the\numexpr#8\expandafter!\the\numexpr#9\expandafter!\the\numexpr
    \expandafter\XINT_sepbyviii_andcount_b\the\numexpr #1+\xint_c_viii\xint:%
}%
\def\XINT_sepbyviii_andcount_end #1\XINT_sepbyviii_andcount_b\the\numexpr
    #2+\xint_c_viii\xint:#3#4\W {\expandafter\xint:\the\numexpr #2+#3\xint:}%
```

4.28 \XINT_rsepbyviii

```
This is used as
   \the\numexpr1\XINT_rsepbyviii <8Ndigits>%
                \XINT_rsepbyviii_end_A 2345678%
                \XINT_rsepbyviii_end_B 2345678\relax UV%
and will produce
   1<8digits>!1<8digits>\xint:1<8digits>!...
```

where the original digits are organized by eight, and the order inside successive pairs of blocks separated by \xint: has been reversed. Output ends either in 1<8d>!1<8d>\xint:1U\xint: (even) or 1<8d>!1<8d>\xint:1V!1<8d>\xint: (odd)

The U an V should be \numexpr1 stoppers (or will expand and be ended by !). This macro is currently (1.2..1.2l) exclusively used in combination with \XINT_sepandrev_andcount or \XINT_sepandrev.

```
\def\XINT_rsepbyviii #1#2#3#4#5#6#7#8%
{%
    \XINT_rsepbyviii_b {#1#2#3#4#5#6#7#8}%
}%
\def\XINT_rsepbyviii_b #1#2#3#4#5#6#7#8#9%
{%
    #2#3#4#5#6#7#8#9\expandafter!\the\numexpr
    1#1\expandafter\xint:\the\numexpr 1\XINT_rsepbyviii
}%
\def\XINT_rsepbyviii_end_B #1\relax #2#3{#2\xint:}%
\def\XINT_rsepbyviii_end_A #11#2\expandafter #3\relax #4#5{#5!1#2\xint:}%
```

4.29 \XINT_sepandrev

```
This is used typically as
   \romannumeral0\XINT_sepandrev <8Ndigits>%
                 \XINT_rsepbyviii_end_A 2345678%
                 \XINT_rsepbyviii_end_B 2345678\relax UV%
                 \R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\W
and will produce
   1<8digits>!1<8digits>!1<8digits>!...
```

where the blocks have been globally reversed. The UV here are only place holders (must be \numexpr1 stoppers) to share same syntax as \XINT_sepandrev_andcount, they are gobbled (#2 in \XINT_sepandrev_done).

```
\def\XINT_sepandrev
{%
    \expandafter\XINT_sepandrev_a\the\numexpr 1\XINT_rsepbyviii
}%
\def\XINT_sepandrev_a {\XINT_sepandrev_b {}}%
\def\XINT_sepandrev_b #1#2\xint:#3\xint:#4\xint:#5\xint:#6\xint:#7\xint:#8\xint:#9\xint:%
{%
    \xint_gob_til_R #9\XINT_sepandrev_end\R
    \XINT_sepandrev_b {#9!#8!#7!#6!#5!#4!#3!#2!#1}%
}%
\def\XINT_sepandrev_end\R\XINT_sepandrev_b #1#2\W {\XINT_sepandrev_done #1}%
\def\XINT_sepandrev_done #11#2!{ }%
```

4.30 \XINT_sepandrev_andcount

```
This is used typically as
   \romannumeral0\XINT_sepandrev_andcount <8Ndigits>%
                \XINT_rsepbyviii_end_A 2345678%
                \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
                \R\xint:\xint_c_xii \R\xint:\xint_c_x  \R\xint:\xint_c_viii \R\xint:\xint_c_vi
                \R\xint:\xint_c_iv  \R\xint:\xint_c_ii \R\xint:\xint_c_\W
and will produce
   <length>.1<8digits>!1<8digits>!1<8digits>!...
where the blocks have been globally reversed and <length> is the number of blocks.
```

```
\def\XINT_sepandrev_andcount
{%
    \expandafter\XINT_sepandrev_andcount_a\the\numexpr 1\XINT_rsepbyviii
}%
\def\XINT_sepandrev_andcount_a {\XINT_sepandrev_andcount_b 0!{}}%
\def\XINT_sepandrev_andcount_b #1!#2#3\xint:#4\xint:#5\xint:#6\xint:#7\xint:#8\xint:#9\xint:%
{%
    \xint_gob_til_R #9\XINT_sepandrev_andcount_end\R
    \expandafter\XINT_sepandrev_andcount_b \the\numexpr #1+\xint_c_i!%
    {#9!#8!#7!#6!#5!#4!#3!#2}%
}%
```

```
\def\XINT_sepandrev_andcount_end\R
    \expandafter\XINT_sepandrev_andcount_b\the\numexpr #1+\xint_c_i!#2#3#4\W
{\expandafter\XINT_sepandrev_andcount_done\the\numexpr #3+\xint_c_xiv*#1!#2}%
\def\XINT_sepandrev_andcount_done#1{%
\def\XINT_sepandrev_andcount_done##1!##21##3!{\expandafter#1\the\numexpr##1-##3\xint:}%
}\XINT_sepandrev_andcount_done{ }%
```

4.31 \XINT_rev_nounsep

This is used as
  \romannumeral0\XINT_rev_nounsep {}<blocks 1<8d>!>\R!\R!\R!\R!\R!\R!\R!\R!\W
It reverses the blocks, keeping the 1's and ! separators. Used multiple times in the division algorithm. The inserted {} here is not optional.

```
\def\XINT_rev_nounsep #1#2!#3!#4!#5!#6!#7!#8!#9!%
{%
    \xint_gob_til_R #9\XINT_rev_nounsep_end\R
    \XINT_rev_nounsep {#9!#8!#7!#6!#5!#4!#3!#2!#1}%
}%
\def\XINT_rev_nounsep_end\R\XINT_rev_nounsep #1#2\W {\XINT_rev_nounsep_done #1}%
\def\XINT_rev_nounsep_done #11{ 1}%
```

4.32 \XINT_unrevbyviii

Used as \romannumeral0\XINT_unrevbyviii 1<8d>!....1<8d>! terminated by
  1;!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
The \romannumeral in unrevbyviii_a is for special effects (expand some token which was put as 1<token>! at the end of the original blocks). This mechanism is used by 1.2 subtraction (still true for 1.2l).

```
\def\XINT_unrevbyviii #11#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
{%
    \xint_gob_til_R #9\XINT_unrevbyviii_a\R
    \XINT_unrevbyviii {#9#8#7#6#5#4#3#2#1}%
}%
\def\XINT_unrevbyviii_a#1{%
\def\XINT_unrevbyviii_a\R\XINT_unrevbyviii ##1##2\W
    {\expandafter#1\romannumeral`&&@\xint_gob_til_sc ##1}%
}\XINT_unrevbyviii_a{ }%
```

Can work with shorter ending pattern: 1;!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W but the longer one of unrevbyviii is ok here too. Used currently (1.2) only by addition, now (1.2c) with long ending pattern. Does the final clean up of leading zeroes contrarily to general \XINT_unrevbyviii.

```
\def\XINT_smallunrevbyviii 1#1!1#2!1#3!1#4!1#5!1#6!1#7!1#8!#9\W%
{%
    \expandafter\XINT_cuz_small\xint_gob_til_sc #8#7#6#5#4#3#2#1%
}%
```

Core arithmetic

The four operations have been rewritten entirely for release 1.2. The new routines works with separated blocks of eight digits. They all measure first the lengths of the arguments, even addition and subtraction (this was not the case with xintcore.sty 1.1 or earlier.)

The technique of chaining \the\numexpr induces a limitation on the maximal size depending on the size of the input save stack and the maximum expansion depth. For the current (TL2015) settings (5000, resp. 10000), the induced limit for addition of numbers is at 19968 and for multiplication it is observed to be 19959 (valid as of 2015/10/07).

Side remark: I tested that \the\numexpr was more efficient than \number. But it reduced the allowable numbers for addition from 19976 digits to 19968 digits.

4.33 \xintiiAdd

1.2l: \xintiiAdd made robust against non terminated input.

```
\def\xintiiAdd   {\romannumeral0\xintiiadd }%
\def\xintiiadd #1{\expandafter\XINT_iiadd\romannumeral`&&@#1\xint:}%
\def\XINT_iiadd #1#2\xint:#3%
{%
    \expandafter\XINT_add_nfork\expandafter#1\romannumeral`&&@#3\xint:#2\xint:
}%
\def\XINT_iadd #1#2\xint:#3%
{%
    \expandafter\XINT_add_nfork\expandafter
    #1\romannumeral0\xintnum{#3}\xint:#2\xint:
}%
\def\XINT_add_fork #1#2\xint:#3\xint:{\XINT_add_nfork #1#3\xint:#2\xint:}%
\def\XINT_add_nfork #1#2%
{%
    \xint_UDzerofork
      #1\XINT_add_firstiszero
      #2\XINT_add_secondiszero
       0{}%
    \krof
    \xint_UDsignsfork
          #1#2\XINT_add_minusminus
           #1-\XINT_add_minusplus
           #2-\XINT_add_plusminus
            --\XINT_add_plusplus
    \krof #1#2%
}%
\def\XINT_add_firstiszero  #1\krof 0#2#3\xint:#4\xint:{ #2#3}%
\def\XINT_add_secondiszero #1\krof #20#3\xint:#4\xint:{ #2#4}%
\def\XINT_add_minusminus   #1#2%
   {\expandafter-\romannumeral0\XINT_add_pp_a {}{}}%
\def\XINT_add_minusplus    #1#2{\XINT_sub_mm_a {}#2}%
\def\XINT_add_plusminus    #1#2%
   {\expandafter\XINT_opp\romannumeral0\XINT_sub_mm_a #1{}}%
\def\XINT_add_pp_a #1#2#3\xint:
{%
  \expandafter\XINT_add_pp_b
      \romannumeral0\expandafter\XINT_sepandrev_andcount
      \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
      #2#3\XINT_rsepbyviii_end_A 2345678%
          \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
              \R\xint:\xint_c_xii \R\xint:\xint_c_x  \R\xint:\xint_c_viii \R\xint:\xint_c_vi
```

```
               \R\xint:\xint_c_iv  \R\xint:\xint_c_ii \R\xint:\xint_c_\W
   \X #1%
}%
\let\XINT_add_plusplus \XINT_add_pp_a
\def\XINT_add_pp_b #1\xint:#2\X #3\xint:
{%
    \expandafter\XINT_add_checklengths
    \the\numexpr #1\expandafter\xint:%
    \romannumeral0\expandafter\XINT_sepandrev_andcount
    \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
    #3\XINT_rsepbyviii_end_A 2345678%
      \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
               \R\xint:\xint_c_xii \R\xint:\xint_c_x  \R\xint:\xint_c_viii \R\xint:\xint_c_vi
               \R\xint:\xint_c_iv  \R\xint:\xint_c_ii \R\xint:\xint_c_\W
     1;!1;!1;!1;!\W #21;!1;!1;!1;!\W
     1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
}%
```

I keep #1.#2. to check if at most 6 + 6 base 10^8 digits which can be treated faster for final reverse. But is this overhead at all useful ?

```
\def\XINT_add_checklengths #1\xint:#2\xint:%
{%
    \ifnum #2>#1
       \expandafter\XINT_add_exchange
    \else
       \expandafter\XINT_add_A
    \fi
    #1\xint:#2\xint:%
}%
\def\XINT_add_exchange #1\xint:#2\xint:#3\W #4\W
{%
    \XINT_add_A #2\xint:#1\xint:#4\W #3\W
}%
\def\XINT_add_A #1\xint:#2\xint:%
{%
    \ifnum #1>\xint_c_vi
          \expandafter\XINT_add_aa
    \else \expandafter\XINT_add_aa_small
    \fi
}%
\def\XINT_add_aa {\expandafter\XINT_add_out\the\numexpr\XINT_add_a \xint_c_ii}%
\def\XINT_add_out{\expandafter\XINT_cuz_small\romannumeral0\XINT_unrevbyviii {}}%
\def\XINT_add_aa_small
    {\expandafter\XINT_smallunrevbyviii\the\numexpr\XINT_add_a \xint_c_ii}%
```

2 as first token of #1 stands for "no carry", 3 will mean a carry (we are adding 1<8digits> to 1<8digits>.) Version 1.2c has terminators of the shape 1;!, replacing the \Z! used in 1.2.

Call: \the\numexpr\XINT_add_a 2#11;!1;!1;!1;!\W #21;!1;!1;!1;!\W where #1 and #2 are blocks of 1<8d>!, and #1 is at most as long as #2. This last requirement is a bit annoying (if one wants to do recursive algorithms but not have to check lengths), and I will probably remove it at some point.

Output: blocks of 1<8d>! representing the addition, (least significant first), and a final 1;!. In recursive algotithm this 1;! terminator can thus conveniently be reused as part of input terminator (up to the length problem).

```
\def\XINT_add_a #1!#2!#3!#4!#5\W
                #6!#7!#8!#9!%
{%
    \XINT_add_b
        #1!#6!#2!#7!#3!#8!#4!#9!%
        #5\W
}%
\def\XINT_add_b #11#2#3!#4!%
{%
    \xint_gob_til_sc #2\XINT_add_bi ;%
    \expandafter\XINT_add_c\the\numexpr#1+1#2#3+#4-\xint_c_ii\xint:%
}%
\def\XINT_add_bi;\expandafter\XINT_add_c
    \the\numexpr#1+#2+#3-\xint_c_ii\xint:#4!#5!#6!#7!#8!#9!\W
{%
    \XINT_add_k #1#3!#5!#7!#9!%
}%
\def\XINT_add_c #1#2\xint:%
{%
    1#2\expandafter!\the\numexpr\XINT_add_d #1%
}%
\def\XINT_add_d #11#2#3!#4!%
{%
    \xint_gob_til_sc #2\XINT_add_di ;%
    \expandafter\XINT_add_e\the\numexpr#1+1#2#3+#4-\xint_c_ii\xint:%
}%
\def\XINT_add_di;\expandafter\XINT_add_e
    \the\numexpr#1+#2+#3-\xint_c_ii\xint:#4!#5!#6!#7!#8\W
{%
    \XINT_add_k #1#3!#5!#7!%
}%
\def\XINT_add_e #1#2\xint:%
{%
    1#2\expandafter!\the\numexpr\XINT_add_f #1%
}%
\def\XINT_add_f #11#2#3!#4!%
{%
    \xint_gob_til_sc #2\XINT_add_fi ;%
    \expandafter\XINT_add_g\the\numexpr#1+1#2#3+#4-\xint_c_ii\xint:%
}%
\def\XINT_add_fi;\expandafter\XINT_add_g
    \the\numexpr#1+#2+#3-\xint_c_ii\xint:#4!#5!#6\W
{%
    \XINT_add_k #1#3!#5!%
}%
\def\XINT_add_g #1#2\xint:%
{%
    1#2\expandafter!\the\numexpr\XINT_add_h #1%
}%
\def\XINT_add_h #11#2#3!#4!%
{%
    \xint_gob_til_sc #2\XINT_add_hi ;%
    \expandafter\XINT_add_i\the\numexpr#1+1#2#3+#4-\xint_c_ii\xint:%
}%
\def\XINT_add_hi;%
    \expandafter\XINT_add_i\the\numexpr#1+#2+#3-\xint_c_ii\xint:#4\W
{%
    \XINT_add_k #1#3!%
}%
\def\XINT_add_i #1#2\xint:%
{%
    1#2\expandafter!\the\numexpr\XINT_add_a #1%
}%
```

```
\def\XINT_add_k #1{\if #12\expandafter\XINT_add_ke\else\expandafter\XINT_add_l \fi}%
\def\XINT_add_ke #11;#2\W {\XINT_add_kf #11;!}%
\def\XINT_add_kf 1{1\relax }%
\def\XINT_add_l 1#1#2{\xint_gob_til_sc #1\XINT_add_lf ;\XINT_add_m 1#1#2}%
\def\XINT_add_lf #1\W {1\relax 00000001!1;!}%
\def\XINT_add_m #1!{\expandafter\XINT_add_n\the\numexpr\xint_c_i+#1\xint:}%
\def\XINT_add_n #1#2\xint:{1#2\expandafter!\the\numexpr\XINT_add_o #1}%
```

Here 2 stands for "carry", and 1 for "no carry" (we have been adding 1 to 1<8digits>.)

```
\def\XINT_add_o #1{\if #12\expandafter\XINT_add_l\else\expandafter\XINT_add_ke \fi}%
```

4.34 \xintiiCmp

Moved from xint.sty to xintcore.sty and rewritten for 1.2l.

1.2l's \xintiiCmp is robust against non terminated input.

1.2o deprecates \xintCmp, with xintfrac loaded it will get overwritten anyhow.

```
\def\xintiiCmp   {\romannumeral0\xintiicmp }%
\def\xintiicmp #1{\expandafter\XINT_iicmp\romannumeral`&&@#1\xint:}%
\def\XINT_iicmp #1#2\xint:#3%
{%
    \expandafter\XINT_cmp_nfork\expandafter #1\romannumeral`&&@#3\xint:#2\xint:
}%
\def\XINT_icmp #1#2\xint:#3%
{%
    \expandafter\XINT_cmp_nfork\expandafter #1\romannumeral0\xintnum{#3}\xint:#2\xint:
}%
\def\XINT_cmp_nfork #1#2%
{%
    \xint_UDzerofork
      #1\XINT_cmp_firstiszero
      #2\XINT_cmp_secondiszero
       0{}%
    \krof
    \xint_UDsignsfork
          #1#2\XINT_cmp_minusminus
           #1-\XINT_cmp_minusplus
           #2-\XINT_cmp_plusminus
            --\XINT_cmp_plusplus
    \krof #1#2%
}%
```

```
\def\XINT_cmp_firstiszero  #1\krof 0#2#3\xint:#4\xint:
{%
    \xint_UDzerominusfork
      #2-{ 0}%
      0#2{ 1}%
       0-{ -1}%
    \krof
}%
\def\XINT_cmp_secondiszero #1\krof #20#3\xint:#4\xint:
{%
    \xint_UDzerominusfork
      #2-{ 0}%
      0#2{ -1}%
       0-{ 1}%
    \krof
}%
\def\XINT_cmp_plusminus    #1\xint:#2\xint:{ 1}%
\def\XINT_cmp_minusplus    #1\xint:#2\xint:{ -1}%
\def\XINT_cmp_minusminus
    --{\expandafter\XINT_opp\romannumeral0\XINT_cmp_plusplus {}{}}%
\def\XINT_cmp_plusplus  #1#2#3\xint:
{%
    \expandafter\XINT_cmp_pp
    \the\numexpr\expandafter\XINT_sepbyviii_andcount
    \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
    #2#3\XINT_sepbyviii_end 2345678\relax
        \xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!%
        \xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_\W
    #1%
}%
\def\XINT_cmp_pp #1\xint:#2\xint:#3\xint:
{%
    \expandafter\XINT_cmp_checklengths
    \the\numexpr #2\expandafter\xint:%
    \the\numexpr\expandafter\XINT_sepbyviii_andcount
    \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
    #3\XINT_sepbyviii_end 2345678\relax
        \xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!%
        \xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_\W
    #1;!1;!1;!1;!\W
}%
\def\XINT_cmp_checklengths #1\xint:#2\xint:#3\xint:
{%
    \ifnum #1=#3
       \expandafter\xint_firstoftwo
    \else
       \expandafter\xint_secondoftwo
    \fi
    \XINT_cmp_a {\XINT_cmp_distinctlengths {#1}{#3}}#2;!1;!1;!1;!\W
}%
\def\XINT_cmp_distinctlengths #1#2#3\W #4\W
{%
```

```
    \ifnum #1>#2
        \expandafter\xint_firstoftwo
    \else
        \expandafter\xint_secondoftwo
    \fi
    { -1}{ 1}%
}%
\def\XINT_cmp_a 1#1!1#2!1#3!1#4!#5\W 1#6!1#7!1#8!1#9!%
{%
    \xint_gob_til_sc #1\XINT_cmp_equal ;%
    \ifnum #1>#6 \XINT_cmp_gt\fi
    \ifnum #1<#6 \XINT_cmp_lt\fi
    \xint_gob_til_sc #2\XINT_cmp_equal ;%
    \ifnum #2>#7 \XINT_cmp_gt\fi
    \ifnum #2<#7 \XINT_cmp_lt\fi
    \xint_gob_til_sc #3\XINT_cmp_equal ;%
    \ifnum #3>#8 \XINT_cmp_gt\fi
    \ifnum #3<#8 \XINT_cmp_lt\fi
    \xint_gob_til_sc #4\XINT_cmp_equal ;%
    \ifnum #4>#9 \XINT_cmp_gt\fi
    \ifnum #4<#9 \XINT_cmp_lt\fi
    \XINT_cmp_a #5\W
}%
\def\XINT_cmp_lt#1{\def\XINT_cmp_lt\fi ##1\W ##2\W {\fi#1-1}}\XINT_cmp_lt{ }%
\def\XINT_cmp_gt#1{\def\XINT_cmp_gt\fi ##1\W ##2\W {\fi#11}}\XINT_cmp_gt{ }%
\def\XINT_cmp_equal #1\W #2\W { 0}%
```

4.35 \xintiiSub

Entirely rewritten for 1.2.

Refactored at 1.2l. I was initially aiming at clinching some internal format of the type 1<8digits>!....1<8digits>! for chaining the arithmetic operations (as a preliminary step to decided upon some internal format for xintfrac macros), thus I wanted to uniformize delimiters in particular and have some core macros inputting and outputting such formats. But the way division is implemented makes it currently very hard to obtain a satisfactory solution. For subtraction I got there almost, but there was added overhead and, as the core sub-routine still assumed the shorter number will be positioned first, one would need to record the length also in the basic internal format, or add the overhead to not make assumption on which one is shorter. I thus but back-tracked my steps but in passing I improved the efficiency (probably) in the worst case branch.

Sadly this 1.2l refactoring left an extra ! in macro \XINT_sub_l_Ida. This bug shows only in rare circumstances which escaped out test suite :(Fixed at 1.2q.

The other reason for backtracking was in relation with the decimal numbers. Having a core format in base 10^8 but ultimately the radix is actually 10 leads to complications. I could use radix 10^8 for \xintiiexpr only, but then I need to make it compatible with sub-\xintiiexpr in \xintexpr, etc... there are many issues of this type.

I considered also an approach like in the 1.2l \xintiiCmp, but decided to stick with the method here for now.

```
\def\xintiiSub   {\romannumeral0\xintiisub }%
\def\xintiisub #1{\expandafter\XINT_iisub\romannumeral`&&@#1\xint:}%
\def\XINT_iisub #1#2\xint:#3%
{%
    \expandafter\XINT_sub_nfork\expandafter
    #1\romannumeral`&&@#3\xint:#2\xint:
}%
\def\XINT_isub #1#2\xint:#3%
{%
   \expandafter\XINT_sub_nfork\expandafter
   #1\romannumeral0\xintnum{#3}\xint:#2\xint:
}%
\def\XINT_sub_nfork #1#2%
{%
    \xint_UDzerofork
      #1\XINT_sub_firstiszero
      #2\XINT_sub_secondiszero
       0{}%
    \krof
    \xint_UDsignsfork
          #1#2\XINT_sub_minusminus
           #1-\XINT_sub_minusplus
           #2-\XINT_sub_plusminus
            --\XINT_sub_plusplus
    \krof #1#2%
}%
\def\XINT_sub_firstiszero  #1\krof 0#2#3\xint:#4\xint:{\XINT_opp #2#3}%
\def\XINT_sub_secondiszero #1\krof #20#3\xint:#4\xint:{ #2#4}%
\def\XINT_sub_plusminus    #1#2{\XINT_add_pp_a #1{}}%
\def\XINT_sub_plusplus   #1#2%
   {\expandafter\XINT_opp\romannumeral0\XINT_sub_mm_a #1#2}%
\def\XINT_sub_minusplus    #1#2%
   {\expandafter-\romannumeral0\XINT_add_pp_a {}#2}%
\def\XINT_sub_minusminus #1#2{\XINT_sub_mm_a {}{}}%
\def\XINT_sub_mm_a  #1#2#3\xint:
{%
  \expandafter\XINT_sub_mm_b
      \romannumeral0\expandafter\XINT_sepandrev_andcount
      \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
      #2#3\XINT_rsepbyviii_end_A 2345678%
          \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
              \R\xint:\xint_c_xii \R\xint:\xint_c_x  \R\xint:\xint_c_viii \R\xint:\xint_c_vi
              \R\xint:\xint_c_iv  \R\xint:\xint_c_ii \R\xint:\xint_c_\W
  \X #1%
}%
\def\XINT_sub_mm_b #1\xint:#2\X #3\xint:
{%
    \expandafter\XINT_sub_checklengths
    \the\numexpr #1\expandafter\xint:%
    \romannumeral0\expandafter\XINT_sepandrev_andcount
    \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
    #3\XINT_rsepbyviii_end_A 2345678%
      \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
              \R\xint:\xint_c_xii \R\xint:\xint_c_x  \R\xint:\xint_c_viii \R\xint:\xint_c_vi
              \R\xint:\xint_c_iv  \R\xint:\xint_c_ii \R\xint:\xint_c_\W
      1;!1;!1;!1;!\W
    #21;!1;!1;!1;!\W
    1;!1\R!1\R!1\R!1\R!%
    1\R!1\R!1\R!1\R!\W
}%
\def\XINT_sub_checklengths #1\xint:#2\xint:%
{%
    \ifnum #2>#1
       \expandafter\XINT_sub_exchange
    \else
       \expandafter\XINT_sub_aa
    \fi
}%
\def\XINT_sub_exchange #1\W #2\W
{%
    \expandafter\XINT_opp\romannumeral0\XINT_sub_aa #2\W #1\W
}%
\def\XINT_sub_aa
{%
    \expandafter\XINT_sub_out\the\numexpr\XINT_sub_a\xint_c_i
}%
```

The post-processing (clean-up of zeros, or rescue of situation with A-B where actually B turns out bigger than A) will be done by a macro which depends on circumstances and will be initially last token before the reversion done by \XINT_unrevbyviii.

```
\def\XINT_sub_out {\XINT_unrevbyviii{}}%
```

1 as first token of #1 stands for "no carry", 0 will mean a carry.

```
 Call: \the\numexpr
       \XINT_sub_a 1#11;!1;!1;!1;!\W
                    #21;!1;!1;!1;!\W
```

where #1 and #2 are blocks of 1<8d>!, #1 (=B) *must* be at most as long as #2 (=A), (in radix 10^8) and the routine wants to compute #2-#1 = A - B

1.2l uses 1;! delimiters to match those of addition (and multiplication). But in the end I reverted the code branch which made it possible to chain such operations keeping internal format in 8 digits blocks throughout.

\numexpr governed expansion stops with various possibilities:

- Type Ia: #1 shorter than #2, no final carry
- Type Ib: #1 shorter than #2, a final carry but next block of #2 > 1
- Type Ica: #1 shorter than #2, a final carry, next block of #2 is final and = 1
- Type Icb: as Ica except that 00000001 block from #2 was not final
- Type Id: #1 shorter than #2, a final carry, next block of #2 = 0
- Type IIa: #1 same length as #2, turns out it was <= #2.
- Type IIb: #1 same length as #2, but turned out > #2.

Various type of post actions are then needed:

- Ia: clean up of zeros in most significant block of 8 digits
- Ib: as Ia
- Ic: there may be significant blocks of 8 zeros to clean up from result. Only case Ica may have arbitrarily many of them, case Icb has only one such block.
- Id: blocks of 99999999 may propagate and there might a be final zero block created which has to be cleaned up.
- IIa: arbitrarily many zeros might have to be removed.
- IIb: We wanted #2-#1 = - (#1-#2), but we got 10^{8N}+#2 -#1 = 10^{8N}-(#1-#2). We need to do the correction then we are as in IIa situation, except that final result can not be zero.

The 1.2l method for this correction is (presumably, testing takes lots of time, which I do not have) more efficient than in 1.2 release.

```
\def\XINT_sub_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
{%
    \XINT_sub_b
    #1!#6!#2!#7!#3!#8!#4!#9!%
    #5\W
}%
```

As 1.2l code uses 1<8digits>! blocks one has to be careful with the carry digit 1 or 0: A #11#2#3 pattern would result into an empty #1 if the carry digit which is upfront is 1, rather than setting #1=1.

```
\def\XINT_sub_b #1#2#3#4!#5!%
{%
    \xint_gob_til_sc #3\XINT_sub_bi ;%
    \expandafter\XINT_sub_c\the\numexpr#1+1#5-#3#4-\xint_c_i\xint:%
}%
\def\XINT_sub_c 1#1#2\xint:%
{%
    1#2\expandafter!\the\numexpr\XINT_sub_d #1%
}%
\def\XINT_sub_d #1#2#3#4!#5!%
{%
    \xint_gob_til_sc #3\XINT_sub_di ;%
    \expandafter\XINT_sub_e\the\numexpr#1+1#5-#3#4-\xint_c_i\xint:
}%
\def\XINT_sub_e 1#1#2\xint:%
{%
    1#2\expandafter!\the\numexpr\XINT_sub_f #1%
}%
\def\XINT_sub_f #1#2#3#4!#5!%
{%
    \xint_gob_til_sc #3\XINT_sub_fi ;%
    \expandafter\XINT_sub_g\the\numexpr#1+1#5-#3#4-\xint_c_i\xint:
}%
\def\XINT_sub_g 1#1#2\xint:%
{%
    1#2\expandafter!\the\numexpr\XINT_sub_h #1%
}%
\def\XINT_sub_h #1#2#3#4!#5!%
{%
    \xint_gob_til_sc #3\XINT_sub_hi ;%
    \expandafter\XINT_sub_i\the\numexpr#1+1#5-#3#4-\xint_c_i\xint:
}%
\def\XINT_sub_i 1#1#2\xint:%
{%
    1#2\expandafter!\the\numexpr\XINT_sub_a #1%
}%
\def\XINT_sub_bi;%
    \expandafter\XINT_sub_c\the\numexpr#1+1#2-#3\xint:
    #4!#5!#6!#7!#8!#9!\W
```

```
{%
    \XINT_sub_k #1#2!#5!#7!#9!%
}%
\def\XINT_sub_di;%
    \expandafter\XINT_sub_e\the\numexpr#1+1#2-#3\xint:
    #4!#5!#6!#7!#8\W
{%
    \XINT_sub_k #1#2!#5!#7!%
}%
\def\XINT_sub_fi;%
    \expandafter\XINT_sub_g\the\numexpr#1+1#2-#3\xint:
    #4!#5!#6\W
{%
    \XINT_sub_k #1#2!#5!%
}%
\def\XINT_sub_hi;%
    \expandafter\XINT_sub_i\the\numexpr#1+1#2-#3\xint:
    #4\W
{%
    \XINT_sub_k #1#2!%
}%
```

B terminated. Have we reached the end of A (necessarily at least as long as B) ? (we are computing A-B, digits of B come first).

If not, then we are certain that even if there is carry it will not propagate beyond the end of A. But it may propagate far transforming chains of 00000000 into 99999999, and if it does go to the final block which possibly is just 1<00000001>!, we will have those eight zeros to clean up.

If A and B have the same length (in base 10^8) then arbitrarily many zeros might have to be cleaned up, and if A<B, the whole result will have to be complemented first.

```
\def\XINT_sub_k #1#2#3%
{%
    \xint_gob_til_sc #3\XINT_sub_p;\XINT_sub_l #1#2#3%
}%
\def\XINT_sub_l #1%
   {\xint_UDzerofork #1\XINT_sub_l_carry 0\XINT_sub_l_Ia\krof}%
\def\XINT_sub_l_Ia 1#1;!#2\W{1\relax#1;!1\XINT_sub_fix_none!}%


\def\XINT_sub_l_carry 1#1!{\ifcase #1
         \expandafter \XINT_sub_l_Id
    \or  \expandafter \XINT_sub_l_Ic
    \else\expandafter \XINT_sub_l_Ib\fi 1#1!}%
\def\XINT_sub_l_Ib #1;#2\W {-\xint_c_i+#1;!1\XINT_sub_fix_none!}%
\def\XINT_sub_l_Ic 1#1!1#2#3!#4;#5\W
{%
    \xint_gob_til_sc #2\XINT_sub_l_Ica;%
    1\relax 00000000!1#2#3!#4;!1\XINT_sub_fix_none!%
}%
```

We need to add some extra delimiters at the end for post-action by \XINT_num, so we first grab the material up to \W

```
\def\XINT_sub_l_Ica#1\W
```

```
{%
    1;!1\XINT_sub_fix_cuz!%
    1;!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
    \xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z
}%
\def\XINT_sub_l_Id 1#1!%
    {199999999\expandafter!\the\numexpr \XINT_sub_l_Id_a}%
\def\XINT_sub_l_Id_a 1#1!{\ifcase #1
         \expandafter \XINT_sub_l_Id
    \or  \expandafter \XINT_sub_l_Id_b
    \else\expandafter \XINT_sub_l_Ib\fi 1#1!}%
\def\XINT_sub_l_Id_b 1#1!1#2#3!#4;#5\W
{%
    \xint_gob_til_sc #2\XINT_sub_l_Ida;%
    1\relax 00000000!1#2#3!#4;!1\XINT_sub_fix_none!%
}%
\def\XINT_sub_l_Ida#1\XINT_sub_fix_none{1;!1\XINT_sub_fix_none}%
```

This is the case where both operands have same 10^8-base length.

We were handling A-B but perhaps B>A. The situation with A=B is also annoying because we then have to clean up all zeros but don't know where to stop (if A>B the first non-zero 8 digits block would tell use when).

Here again we need to grab #3\W to position the actually used terminating delimiters.

```
\def\XINT_sub_p;\XINT_sub_l #1#2\W #3\W
{%
    \xint_UDzerofork
       #1{1;!1\XINT_sub_fix_neg!%
          1;!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
          \xint_bye2345678\xint_bye1099999988\relax}% A - B, B > A
        0{1;!1\XINT_sub_fix_cuz!%
          1;!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W}%
    \krof
    \xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z
}%
```

Routines for post-processing after reversal, and removal of separators. It is a matter of cleaning up zeros, and possibly in the bad case to take a complement before that.

```
\def\XINT_sub_fix_none;{\XINT_cuz_small}%
\def\XINT_sub_fix_cuz ;{\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop}%
```

Case with A and B same number of digits in base 10^8 and B>A.

1.2l subtle chaining on the model of the 1.2i rewrite of \xintInc and similar routines. After taking complement, leading zeroes need to be cleaned up as in B<=A branch.

```
\def\XINT_sub_fix_neg;%
{%
    \expandafter-\romannumeral0\expandafter
    \XINT_sub_comp_finish\the\numexpr\XINT_sub_comp_loop
}%
\def\XINT_sub_comp_finish 0{\XINT_sub_fix_cuz;}%
\def\XINT_sub_comp_loop #1#2#3#4#5#6#7#8%
{%
```

```
    \expandafter\XINT_sub_comp_clean
    \the\numexpr \xint_c_xi_e_viii_mone-#1#2#3#4#5#6#7#8\XINT_sub_comp_loop
}%
```

#1 = 0 signifie une retenue, #1 = 1 pas de retenue, ce qui ne peut arriver que tant qu'il n'y a que des zéros du côté non significatif. Lorsqu'on est revenu au début on a forcément une retenue.

```
\def\XINT_sub_comp_clean 1#1{+#1\relax}%
```

4.36 \xintiiMul

Completely rewritten for 1.2.

1.2l: \xintiiMul made robust against non terminated input.

```
\def\xintiiMul {\romannumeral0\xintiimul }%
\def\xintiimul #1%
{%
    \expandafter\XINT_iimul\romannumeral`&&@#1\xint:
}%
\def\XINT_iimul #1#2\xint:#3%
{%
    \expandafter\XINT_mul_nfork\expandafter #1\romannumeral`&&@#3\xint:#2\xint:
}%
```

(1.2) I have changed the fork, and it complicates matters elsewhere.

```
\def\XINT_mul_fork #1#2\xint:#3\xint:{\XINT_mul_nfork #1#3\xint:#2\xint:}%
\def\XINT_mul_nfork #1#2%
{%
    \xint_UDzerofork
      #1\XINT_mul_zero
      #2\XINT_mul_zero
       0{}%
    \krof
    \xint_UDsignsfork
          #1#2\XINT_mul_minusminus
           #1-\XINT_mul_minusplus
           #2-\XINT_mul_plusminus
            --\XINT_mul_plusplus
    \krof #1#2%
}%
\def\XINT_mul_zero  #1\krof #2#3\xint:#4\xint:{ 0}%
\def\XINT_mul_minusminus   #1#2{\XINT_mul_plusplus {}{}}%
\def\XINT_mul_minusplus    #1#2%
   {\expandafter-\romannumeral0\XINT_mul_plusplus {}#2}%
\def\XINT_mul_plusminus    #1#2%
   {\expandafter-\romannumeral0\XINT_mul_plusplus #1{}}%
\def\XINT_mul_plusplus #1#2#3\xint:
{%
  \expandafter\XINT_mul_pre_b
      \romannumeral0\expandafter\XINT_sepandrev_andcount
      \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
      #2#3\XINT_rsepbyviii_end_A 2345678%
          \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
```

```
              \R\xint:\xint_c_xii \R\xint:\xint_c_x  \R\xint:\xint_c_viii \R\xint:\xint_c_vi
              \R\xint:\xint_c_iv  \R\xint:\xint_c_ii \R\xint:\xint_c_\W
  \W #1%
}%
\def\XINT_mul_pre_b #1\xint:#2\W #3\xint:
{%
    \expandafter\XINT_mul_checklengths
    \the\numexpr #1\expandafter\xint:%
    \romannumeral0\expandafter\XINT_sepandrev_andcount
    \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
    #3\XINT_rsepbyviii_end_A 2345678%
      \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
              \R\xint:\xint_c_xii \R\xint:\xint_c_x  \R\xint:\xint_c_viii \R\xint:\xint_c_vi
              \R\xint:\xint_c_iv  \R\xint:\xint_c_ii \R\xint:\xint_c_\W
     1;!\W #21;!%
    1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
}%
```

Cooking recipe, 2015/10/05.

```
\def\XINT_mul_checklengths #1\xint:#2\xint:%
{%
    \ifnum #2=\xint_c_i\expandafter\XINT_mul_smallbyfirst\fi
    \ifnum #1=\xint_c_i\expandafter\XINT_mul_smallbysecond\fi
    \ifnum #2<#1
       \ifnum \numexpr (#2-\xint_c_i)*(#1-#2)<383
          \XINT_mul_exchange
       \fi
    \else
       \ifnum \numexpr (#1-\xint_c_i)*(#2-#1)>383
          \XINT_mul_exchange
       \fi
    \fi
    \XINT_mul_start
}%
\def\XINT_mul_smallbyfirst #1\XINT_mul_start 1#2!1;!\W
{%
    \ifnum#2=\xint_c_i\expandafter\XINT_mul_oneisone\fi
    \ifnum#2<\xint_c_xxii\expandafter\XINT_mul_verysmall\fi
    \expandafter\XINT_mul_out\the\numexpr\XINT_smallmul 1#2!%
}%
\def\XINT_mul_smallbysecond #1\XINT_mul_start #2\W 1#3!1;!%
{%
    \ifnum#3=\xint_c_i\expandafter\XINT_mul_oneisone\fi
    \ifnum#3<\xint_c_xxii\expandafter\XINT_mul_verysmall\fi
    \expandafter\XINT_mul_out\the\numexpr\XINT_smallmul 1#3!#2%
}%
\def\XINT_mul_oneisone #1!{\XINT_mul_out }%
\def\XINT_mul_verysmall\expandafter\XINT_mul_out
                       \the\numexpr\XINT_smallmul 1#1!%
    {\expandafter\XINT_mul_out\the\numexpr\XINT_verysmallmul 0\xint:#1!}%
\def\XINT_mul_exchange #1\XINT_mul_start #2\W #31;!%
   {\fi\fi\XINT_mul_start #31;!\W #2}%
```

```
\def\XINT_mul_start
   {\expandafter\XINT_mul_out\the\numexpr\XINT_mul_loop 100000000!1;!\W}%
\def\XINT_mul_out
   {\expandafter\XINT_cuz_small\romannumeral0\XINT_unrevbyviii {}}%
```

Call:
\the\numexpr \XINT_mul_loop 100000000!1;!\W #11;!\W #21;!
where #1 and #2 are (globally reversed) blocks 1<8d>!. Its is generally more efficient if #1 is the shorter one, but a better recipe is implemented in \XINT_mul_checklengths. One may call \XINT_mul_loop directly (but multiplication by zero will produce many 100000000! blocks on output).

Ends after having produced: 1<8d>!....1<8d>!1;!. The last 8-digits block is significant one. It can not be 100000000! except if the loop was called with a zero operand.

Thus \XINT_mul_loop can be conveniently called directly in recursive routines, as the output terminator can serve as input terminator, we can arrange to not have to grab the whole thing again.

```
\def\XINT_mul_loop #1\W #2\W 1#3!%
{%
    \xint_gob_til_sc #3\XINT_mul_e ;%
    \expandafter\XINT_mul_a\the\numexpr \XINT_smallmul 1#3!#2\W
    #1\W #2\W
}%
```

Each of #1 and #2 brings its 1;! for \XINT_add_a.

```
\def\XINT_mul_a #1\W #2\W
{%
    \expandafter\XINT_mul_b\the\numexpr
    \XINT_add_a \xint_c_ii #21;!1;!1;!\W #11;!1;!1;!\W\W
}%
\def\XINT_mul_b 1#1!{1#1\expandafter!\the\numexpr\XINT_mul_loop }%
\def\XINT_mul_e;#1\W 1#2\W #3\W {1\relax #2}%
```

1.2 small and mini multiplication in base 10^8 with carry. Used by the main multiplication routines. But division, float factorial, etc.. have their own variants as they need output with specific constraints.

The minimulwc has 1<8digits carry>.<4 high digits>.<4 low digits!<8digits>.

It produces a block 1<8d>! and then jump back into \XINT_smallmul_a with the new 8digits carry as argument. The \XINT_smallmul_a fetches a new 1<8d>! block to multiply, and calls back \XINT_minimul_wc having stored the multiplicand for re-use later. When the loop terminates, the final carry is checked for being nul, and in all cases the output is terminated by a 1;!

Multiplication by zero will produce blocks of zeros.

```
\def\XINT_minimulwc_a 1#1\xint:#2\xint:#3!#4#5#6#7#8\xint:%
{%
    \expandafter\XINT_minimulwc_b
    \the\numexpr \xint_c_x^ix+#1+#3*#8\xint:
                     #3*#4#5#6#7+#2*#8\xint:
                           #2*#4#5#6#7\xint:%
}%
\def\XINT_minimulwc_b 1#1#2#3#4#5#6\xint:#7\xint:%
{%
    \expandafter\XINT_minimulwc_c
    \the\numexpr \xint_c_x^ix+#1#2#3#4#5+#7\xint:#6\xint:%
}%
\def\XINT_minimulwc_c 1#1#2#3#4#5#6\xint:#7\xint:#8\xint:%
{%
    1#6#7\expandafter!%
    \the\numexpr\expandafter\XINT_smallmul_a
    \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8\xint:%
}%
\def\XINT_smallmul 1#1#2#3#4#5!{\XINT_smallmul_a 100000000\xint:#1#2#3#4\xint:#5!}%
\def\XINT_smallmul_a #1\xint:#2\xint:#3!1#4!%
{%
    \xint_gob_til_sc #4\XINT_smallmul_e;%
    \XINT_minimulwc_a #1\xint:#2\xint:#3!#4\xint:#2\xint:#3!%
}%
\def\XINT_smallmul_e;\XINT_minimulwc_a 1#1\xint:#2;#3!%
    {\xint_gob_til_eightzeroes #1\XINT_smallmul_f 000000001\relax #1!1;!}%
\def\XINT_smallmul_f 000000001\relax 00000000!1{1\relax}%


\def\XINT_verysmallmul #1\xint:#2!1#3!%
{%
    \xint_gob_til_sc #3\XINT_verysmallmul_e;%
    \expandafter\XINT_verysmallmul_a
    \the\numexpr #2*#3+#1\xint:#2!%
}%
\def\XINT_verysmallmul_e;\expandafter\XINT_verysmallmul_a\the\numexpr
    #1+#2#3\xint:#4!%
{\xint_gob_til_zero #2\XINT_verysmallmul_f 0\xint_c_x^viii+#2#3!1;!}%
\def\XINT_verysmallmul_f #1!1{1\relax}%
\def\XINT_verysmallmul_a #1#2\xint:%
{%
    \unless\ifnum #1#2<\xint_c_x^ix
    \expandafter\XINT_verysmallmul_bi\else
    \expandafter\XINT_verysmallmul_bj\fi
    \the\numexpr \xint_c_x^ix+#1#2\xint:%
}%
\def\XINT_verysmallmul_bj{\expandafter\XINT_verysmallmul_cj }%
\def\XINT_verysmallmul_cj 1#1#2\xint:%
    {1#2\expandafter!\the\numexpr\XINT_verysmallmul #1\xint:}%
\def\XINT_verysmallmul_bi\the\numexpr\xint_c_x^ix+#1#2#3\xint:%
    {1#3\expandafter!\the\numexpr\XINT_verysmallmul #1#2\xint:}%
```

Used by division and by squaring, not by multiplication itself.

This routine does not loop, it only does one mini multiplication with input format <4 high digits>.<4 low digits>!<8 digits>!, and on output 1<8d>!1<8d>!, with least significant block first.

```
\def\XINT_minimul_a #1\xint:#2!#3#4#5#6#7!%
{%
    \expandafter\XINT_minimul_b
    \the\numexpr \xint_c_x^viii+#2*#7\xint:#2*#3#4#5#6+#1*#7\xint:#1*#3#4#5#6\xint:%
}%
\def\XINT_minimul_b 1#1#2#3#4#5\xint:#6\xint:%
{%
    \expandafter\XINT_minimul_c
    \the\numexpr \xint_c_x^ix+#1#2#3#4+#6\xint:#5\xint:%
}%
\def\XINT_minimul_c 1#1#2#3#4#5#6\xint:#7\xint:#8\xint:%
{%
    1#6#7\expandafter!\the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8!%
}%
```

4.37 \xintiiDivision

Completely rewritten for 1.2.

WARNING: some comments below try to describe the flow of tokens but they date back to xint 1.09j and I updated them on the fly while doing the 1.2 version. As the routine now works in base 10^8, not 10^4 and "drops" the quotient digits,rather than store them upfront as the earlier code, I may well have not correctly converted all such comments. At the last minute some previously #1 became stuff like #1#2#3#4, then of course the old comments describing what the macro parameters stand for are necessarily wrong.

Side remark: the way tokens are grouped was not essentially modified in 1.2, although the situation has changed. It was fine-tuned in xint 1.0/1.1 but the context has changed, and perhaps I should revisit this. As a corollary to the fact that quotient digits are now left behind thanks to the chains of \numexpr, some macros which in 1.0/1.1 fetched up to 9 parameters now need handle less such parameters. Thus, some rationale for the way the code was structured has disappeared.

1.2l: \xintiiDivision et al. made robust against non terminated input.

#1 = A, #2 = B. On calcule le quotient et le reste dans la division euclidienne de A par B: A=BQ+R, 0<= R < |B|.

```
\def\xintiiDivision   {\romannumeral0\xintiidivision }%
\def\xintiidivision  #1{\expandafter\XINT_iidivision \romannumeral`&&@#1\xint:}%
\def\XINT_iidivision #1#2\xint:#3{\expandafter\XINT_iidivision_a\expandafter #1%
                              \romannumeral`&&@#3\xint:#2\xint:}%
```

On regarde les signes de A et de B.

```
\def\XINT_iidivision_a #1#2% #1 de A, #2 de B.
{%
    \if0#2\xint_dothis{\XINT_iidivision_divbyzero #1#2}\fi
    \if0#1\xint_dothis\XINT_iidivision_aiszero\fi
    \if-#2\xint_dothis{\expandafter\XINT_iidivision_bneg
                        \romannumeral0\XINT_iidivision_bpos #1}\fi
    \xint_orthat{\XINT_iidivision_bpos #1#2}%
}%
\def\XINT_iidivision_divbyzero#1#2#3\xint:#4\xint:
   {\if0#1\xint_dothis{\XINT_signalcondition{DivisionUndefined}}\fi
         \xint_orthat{\XINT_signalcondition{DivisionByZero}}%
    {Division of #1#4 by #2#3}{}{{0}{0}}}%
\def\XINT_iidivision_aiszero #1\xint:#2\xint:{{0}{0}}%
\def\XINT_iidivision_bneg #1% q->-q, r unchanged
                           {\expandafter{\romannumeral0\XINT_opp #1}}%
\def\XINT_iidivision_bpos #1%
{%
    \xint_UDsignfork
            #1\XINT_iidivision_aneg
             -{\XINT_iidivision_apos #1}%
    \krof
}%
```

Donc attention malgré son nom \XINT_div_prepare va jusqu'au bout. C'est donc en fait l'entrée principale (pour B>0, A>0) mais elle va regarder si B est < 10^8 et s'il vaut alors 1 ou 2, et si A < 10^8. Dans tous les cas le résultat est produit sous la forme {Q}{R}, avec Q et R sous leur forme final. On doit ensuite ajuster si le B ou le A initial était négatif. Je n'ai pas fait beaucoup d'efforts pour être un minimum efficace si A ou B n'est pas positif.

```
\def\XINT_iidivision_apos #1#2\xint:#3\xint:{\XINT_div_prepare {#2}{#1#3}}%
\def\XINT_iidivision_aneg #1\xint:#2\xint:
   {\expandafter
    \XINT_iidivision_aneg_b\romannumeral0\XINT_div_prepare {#1}{#2}{#1}}%
\def\XINT_iidivision_aneg_b #1#2{\if0\XINT_Sgn #2\xint:
                                  \expandafter\XINT_iidivision_aneg_rzero
                               \else
                                  \expandafter\XINT_iidivision_aneg_rpos
                               \fi {#1}{#2}}%
\def\XINT_iidivision_aneg_rzero #1#2#3{{-#1}{0}}% necessarily q was >0
\def\XINT_iidivision_aneg_rpos #1%
{%
    \expandafter\XINT_iidivision_aneg_end\expandafter
               {\expandafter-\romannumeral0\xintinc {#1}}% q-> -(1+q)
}%
\def\XINT_iidivision_aneg_end #1#2#3%
{%
     \expandafter\xint_exchangetwo_keepbraces
     \expandafter{\romannumeral0\XINT_sub_mm_a {}{}#3\xint:#2\xint:}{#1}% r-> b-r
}%
```

Le diviseur B va être étendu par des zéros pour que sa longueur soit multiple de huit. Les zéros seront mis du côté non significatif.

```
\def\XINT_div_prepare #1%
{%
    \XINT_div_prepare_a #1\R\R\R\R\R\R\R\R {10}0000001\W !{#1}%
}%
\def\XINT_div_prepare_a #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_R #9\XINT_div_prepare_small\R
    \XINT_div_prepare_b #9%
}%
```

B a au plus huit chiffres. On se débarrasse des trucs superflus. Si B>0 n'est ni 1 ni 2, le point d'entrée est \XINT_div_small_a {B}{A} (avec un A positif).

```
\def\XINT_div_prepare_small\R #1!#2%
{%
    \ifcase #2
    \or\expandafter\XINT_div_BisOne
    \or\expandafter\XINT_div_BisTwo
    \else\expandafter\XINT_div_small_a
    \fi {#2}%
}%
\def\XINT_div_BisOne #1#2{{#2}{0}}%
\def\XINT_div_BisTwo #1#2%
{%
```

```
    \expandafter\expandafter\expandafter\XINT_div_BisTwo_a
    \ifodd\xintLDg{#2} \expandafter1\else \expandafter0\fi {#2}%
}%
\def\XINT_div_BisTwo_a #1#2%
{%
    \expandafter{\romannumeral0\XINT_half
     #2\xint_bye\xint_Bye345678\xint_bye
     *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax}{#1}%
}%
```

B a au plus huit chiffres et est au moins 3. On va l'utiliser directement, sans d'abord le multiplier par une puissance de 10 pour qu'il ait 8 chiffres.

```
\def\XINT_div_small_a #1#2%
{%
    \expandafter\XINT_div_small_b
    \the\numexpr #1/\xint_c_ii\expandafter
    \xint:\the\numexpr \xint_c_x^viii+#1\expandafter!%
    \romannumeral0%
    \XINT_div_small_ba #2\R\R\R\R\R\R\R\R{10}0000001\W
       #2\XINT_sepbyviii_Z_end 2345678\relax
}%
```

Le #2 poursuivra l'expansion par \XINT_div_dosmallsmall ou par \XINT_smalldivx_a suivi de \XINT_sdiv_out.

```
\def\XINT_div_small_b #1!#2{#2#1!}%
```

On ajoute des zéros avant A, puis on le prépare sous la forme de blocs 1<8d>! Au passage on repère le cas d'un A<10^8.

```
\def\XINT_div_small_ba #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_R #9\XINT_div_smallsmall\R
    \expandafter\XINT_div_dosmalldiv
    \the\numexpr\expandafter\XINT_sepbyviii_Z
           \romannumeral0\XINT_zeroes_forviii
    #1#2#3#4#5#6#7#8#9%
}%
```

Si A<10^8, on va poursuivre par \XINT_div_dosmallsmall round(B/2).10^8+B!{A}. On fait la division directe par \numexpr. Le résultat est produit sous la forme {Q}{R}.

```
\def\XINT_div_smallsmall\R
    \expandafter\XINT_div_dosmalldiv
    \the\numexpr\expandafter\XINT_sepbyviii_Z
    \romannumeral0\XINT_zeroes_forviii #1\R #2\relax
   {{\XINT_div_dosmallsmall}{#1}}%
\def\XINT_div_dosmallsmall #1\xint:1#2!#3%
{%
    \expandafter\XINT_div_smallsmallend
    \the\numexpr (#3+#1)/#2-\xint_c_i\xint:#2\xint:#3\xint:%
}%
\def\XINT_div_smallsmallend #1\xint:#2\xint:#3\xint:{\expandafter
    {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #3-#1*#2}}%
```

Si A>=10^8, il est maintenant sous la forme 1<8d>!...1<8d>!1;! avec plus significatifs en premier. Donc on poursuit par
\expandafter\XINT_sdiv_out\the\numexpr\XINT_smalldivx_a x.1B!1<8d>!...1<8d>!1;! avec x =round(B/2), 1B=10^8+B.

```
\def\XINT_div_dosmalldiv
    {{\expandafter\XINT_sdiv_out\the\numexpr\XINT_smalldivx_a}}%
```

Ici B est au moins 10^8, on détermine combien de zéros lui adjoindre pour qu'il soit de longueur 8N.

```
\def\XINT_div_prepare_b
   {\expandafter\XINT_div_prepare_c\romannumeral0\XINT_zeroes_forviii }%
\def\XINT_div_prepare_c #1!%
{%
     \XINT_div_prepare_d  #1.00000000!{#1}%
}%
\def\XINT_div_prepare_d #1#2#3#4#5#6#7#8#9%
{%
    \expandafter\XINT_div_prepare_e\xint_gob_til_dot #1#2#3#4#5#6#7#8#9!%
}%
\def\XINT_div_prepare_e #1!#2!#3#4%
{%
    \XINT_div_prepare_f #4#3\X {#1}{#3}%
}%
```

attention qu'on calcule ici x'=x+1 (x = huit premiers chiffres du diviseur) et que si x=99999999, x' aura donc 9 chiffres, pas compatible avec div_mini (avant 1.2, x avait 4 chiffres, et on faisait la division avec x' dans un \numexpr). Bon, facile à dire après avoir laissé passer ce bug dans 1.2. C'est le problème lorsqu'au lieu de tout refaire à partir de zéro on recycle d'anciennes routines qui avaient un contexte différent.

```
\def\XINT_div_prepare_f #1#2#3#4#5#6#7#8#9\X
{%
    \expandafter\XINT_div_prepare_g
     \the\numexpr  #1#2#3#4#5#6#7#8+\xint_c_i\expandafter
    \xint:\the\numexpr (#1#2#3#4#5#6#7#8+\xint_c_i)/\xint_c_ii\expandafter
    \xint:\the\numexpr #1#2#3#4#5#6#7#8\expandafter
    \xint:\romannumeral0\XINT_sepandrev_andcount
    #1#2#3#4#5#6#7#8#9\XINT_rsepbyviii_end_A 2345678%
                      \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
              \R\xint:\xint_c_xii \R\xint:\xint_c_x  \R\xint:\xint_c_viii \R\xint:\xint_c_vi
              \R\xint:\xint_c_iv  \R\xint:\xint_c_ii \R\xint:\xint_c_\W
    \X
}%
\def\XINT_div_prepare_g #1\xint:#2\xint:#3\xint:#4\xint:#5\X #6#7#8%
{%
    \expandafter\XINT_div_prepare_h
    \the\numexpr\expandafter\XINT_sepbyviii_andcount
    \romannumeral0\XINT_zeroes_forviii #8#7\R\R\R\R\R\R\R\R{10}0000001\W
    #8#7\XINT_sepbyviii_end 2345678\relax
     \xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!%
     \xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_\W
    {#1}{#2}{#3}{#4}{#5}{#6}%
}%
\def\XINT_div_prepare_h #11\xint:#2\xint:#3#4#5#6%#7#8%
{%
    \XINT_div_start_a {#2}{#6}{#1}{#3}{#4}{#5}%{#7}{#8}%
}%
```

L, K, A, x',y,x, B, «c». Attention que K est diminué de 1 plus loin. Comme xint 1.2 a déjà repéré K=1, on a ici au minimum K=2. Attention B est à l'envers, A est à l'endroit et les deux avec séparateurs. Attention que ce n'est pas ici qu'on boucle mais en \XINT_div_I_a.

```
\def\XINT_div_start_a #1#2%
{%
    \ifnum #1 < #2
      \expandafter\XINT_div_zeroQ
    \else
      \expandafter\XINT_div_start_b
    \fi
    {#1}{#2}%
}%
\def\XINT_div_zeroQ #1#2#3#4#5#6#7%
{%
    \expandafter\XINT_div_zeroQ_end
    \romannumeral0\XINT_unsep_cuzsmall
    #3\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax\xint:
}%
\def\XINT_div_zeroQ_end #1\xint:#2%
    {\expandafter{\expandafter0\expandafter}\XINT_div_cleanR #1#2\xint:}%
```

L, K, A, x',y,x, B, «c»->K.A.x{LK{x'y}x}B«c»

```
\def\XINT_div_start_b #1#2#3#4#5#6%
{%
    \expandafter\XINT_div_finish\the\numexpr
    \XINT_div_start_c {#2}\xint:#3\xint:{#6}{{#1}{#2}{{#4}{#5}}{#6}}%
}%
\def\XINT_div_finish
{%
    \expandafter\XINT_div_finish_a \romannumeral`&&@\XINT_div_unsepQ
}%
\def\XINT_div_finish_a #1\Z #2\xint:{\XINT_div_finish_b #2\xint:{#1}}%
```

Ici ce sont routines de fin. Le reste déjà nettoyé. R.Q«c».

```
\def\XINT_div_finish_b #1%
{%
    \if0#1%
       \expandafter\XINT_div_finish_bRzero
    \else
       \expandafter\XINT_div_finish_bRpos
    \fi
    #1%
}%
\def\XINT_div_finish_bRzero 0\xint:#1#2{{#1}{0}}%
\def\XINT_div_finish_bRpos #1\xint:#2#3%
```

```
{%
    \expandafter\xint_exchangetwo_keepbraces\XINT_div_cleanR  #1#3\xint:{#2}%
}%
\def\XINT_div_cleanR #100000000\xint:{{#1}}%
```

Kalpha.A.x{LK{x'y}x}, B, «c», au début #2=alpha est vide. On fait une boucle pour prendre K unités de A (on a au moins L égal à K) et les mettre dans alpha.

```
\def\XINT_div_start_c #1%
{%
    \ifnum #1>\xint_c_vi
       \expandafter\XINT_div_start_ca
    \else
       \expandafter\XINT_div_start_cb
    \fi {#1}%
}%
\def\XINT_div_start_ca #1#2\xint:#3!#4!#5!#6!#7!#8!#9!%
{%
    \expandafter\XINT_div_start_c\expandafter
    {\the\numexpr #1-\xint_c_vii}#2#3!#4!#5!#6!#7!#8!#9!\xint:%
}%
\def\XINT_div_start_cb #1%
   {\csname XINT_div_start_c_\romannumeral\numexpr#1\endcsname}%
\def\XINT_div_start_c_i   #1\xint:#2!%
    {\XINT_div_start_c_   #1#2!\xint:}%
\def\XINT_div_start_c_ii  #1\xint:#2!#3!%
    {\XINT_div_start_c_   #1#2!#3!\xint:}%
\def\XINT_div_start_c_iii #1\xint:#2!#3!#4!%
    {\XINT_div_start_c_   #1#2!#3!#4!\xint:}%
\def\XINT_div_start_c_iv  #1\xint:#2!#3!#4!#5!%
    {\XINT_div_start_c_   #1#2!#3!#4!#5!\xint:}%
\def\XINT_div_start_c_v   #1\xint:#2!#3!#4!#5!#6!%
    {\XINT_div_start_c_   #1#2!#3!#4!#5!#6!\xint:}%
\def\XINT_div_start_c_vi  #1\xint:#2!#3!#4!#5!#6!#7!%
    {\XINT_div_start_c_   #1#2!#3!#4!#5!#6!#7!\xint:}%
```

#1=a, #2=alpha (de longueur K, à l'endroit).#3=reste de A.#4=x, #5={LK{x'y}x},#6=B,«c» -> a, x, alpha, B, {00000000}, L, K, {x'y},x, alpha'=reste de A, B«c».

```
\def\XINT_div_start_c_ 1#1!#2\xint:#3\xint:#4#5#6%
{%
    \XINT_div_I_a {#1}{#4}{1#1!#2}{#6}{00000000}#5{#3}{#6}%
}%
```

Ceci est le point de retour de la boucle principale. a, x, alpha, B, q0, L, K, {x'y}, x, alpha', B«c»

```
\def\XINT_div_I_a #1#2%
{%
    \expandafter\XINT_div_I_b\the\numexpr #1/#2\xint:{#1}{#2}%
}%
\def\XINT_div_I_b #1%
{%
    \xint_gob_til_zero #1\XINT_div_I_czero 0\XINT_div_I_c #1%
}%
```

On intercepte petit quotient nul: #1=a, x, alpha, B, #5=q0, L, K, {x'y}, x, alpha', B«c» -> on lâche un q puis {alpha} L, K, {x'y}, x, alpha', B«c».

```
\def\XINT_div_I_czero 0\XINT_div_I_c 0\xint:#1#2#3#4#5{1#5\XINT_div_I_g {#3}}%
\def\XINT_div_I_c #1\xint:#2#3%
{%
    \expandafter\XINT_div_I_da\the\numexpr #2-#1*#3\xint:#1\xint:{#2}{#3}%
}%
```

r.q.alpha, B, q0, L, K, {x'y}, x, alpha', B«c»

```
\def\XINT_div_I_da #1\xint:%
{%
    \ifnum #1>\xint_c_ix
       \expandafter\XINT_div_I_dP
    \else
       \ifnum #1<\xint_c_
        \expandafter\expandafter\expandafter\XINT_div_I_dN
       \else
        \expandafter\expandafter\expandafter\XINT_div_I_db
       \fi
    \fi
}%
```

attention très mauvaises notations avec _b et _db.

```
\def\XINT_div_I_dN #1\xint:%
{%
    \expandafter\XINT_div_I_b\the\numexpr #1-\xint_c_i\xint:%
}%
\def\XINT_div_I_db #1\xint:#2#3#4#5%
{%
    \expandafter\XINT_div_I_dc\expandafter #1%
    \romannumeral0\expandafter\XINT_div_sub\expandafter
       {\romannumeral0\XINT_rev_nounsep {}#4\R!\R!\R!\R!\R!\R!\R!\R!\W}%
       {\the\numexpr\XINT_div_verysmallmul #1!#51;!}%
    \Z {#4}{#5}%
}%
```

La soustraction spéciale renvoie simplement - si le chiffre q est trop grand. On invoque dans ce cas I_dP.

```
\def\XINT_div_I_dc #1#2%
{%
    \if-#2\expandafter\XINT_div_I_dd\else\expandafter\XINT_div_I_de\fi
     #1#2%
}%
\def\XINT_div_I_dd #1-\Z
{%
    \if #11\expandafter\XINT_div_I_dz\fi
    \expandafter\XINT_div_I_dP\the\numexpr #1-\xint_c_i\xint: XX%
}%
\def\XINT_div_I_dz #1XX#2#3#4%
{%
```

```
    1#4\XINT_div_I_g {#2}%
}%
\def\XINT_div_I_de #1#2\Z #3#4#5{1#5+#1\XINT_div_I_g {#2}}%
```

q.alpha, B, q0, L, K, {x'y},x, alpha'B«c» (q=0 has been intercepted) -> 1nouveauq.nouvel alpha, L, K, {x'y}, x, alpha',B«c»

```
\def\XINT_div_I_dP #1\xint:#2#3#4#5#6%
{%
    1#6+#1\expandafter\XINT_div_I_g\expandafter
    {\romannumeral0\expandafter\XINT_div_sub\expandafter
      {\romannumeral0\XINT_rev_nounsep {}#4\R!\R!\R!\R!\R!\R!\R!\R!\W}%
      {\the\numexpr\XINT_div_verysmallmul #1!#51;!}%
    }%
}%
```

1#1=nouveau q. nouvel alpha, L, K, {x'y},x,alpha', BQ«c»

#1=q,#2=nouvel alpha,#3=L, #4=K, #5={x'y}, #6=x, #7= alpha',#8=B, «c» -> on laisse q puis {x'y}alpha.alpha'.{{x'y}xKL}B«c»

```
\def\XINT_div_I_g #1#2#3#4#5#6#7%
{%
     \expandafter !\the\numexpr
     \ifnum#2=#3
          \expandafter\XINT_div_exittofinish
     \else
          \expandafter\XINT_div_I_h
     \fi
     {#4}#1\xint:#6\xint:{{#4}{#5}{#3}{#2}}{#7}%
}%
```

{x'y}alpha.alpha'.{{x'y}xKL}B«c» -> Attention retour à l'envoyeur ici par terminaison des \the\numexpr. On doit reprendre le Q déjà sorti, qui n'a plus de séparateurs, ni de leading 1. Ensuite R sans leading zeros.«c»

```
\def\XINT_div_exittofinish #1#2\xint:#3\xint:#4#5%
{%
    1\expandafter\expandafter\expandafter!\expandafter\XINT_div_unsepQ_delim
    \romannumeral0\XINT_div_unsepR #2#3%
    \xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax\R\xint:
}%
```

ATTENTION DESCRIPTION OBSOLÈTE. #1={x'y}alpha.#2!#3=reste de A. #4={{x'y},x,K,L},#5=B,«c» devient {x'y},alpha sur K+4 chiffres.B, {{x'y},x,K,L}, #6= nouvel alpha',B,«c»

```
\def\XINT_div_I_h #1\xint:#2!#3\xint:#4#5%
{%
    \XINT_div_II_b #1#2!\xint:{#5}{#4}{#3}{#5}%
}%
```

{x'y}alpha.B, {{x'y},x,K,L}, nouveau alpha',B,«c»

```
\def\XINT_div_II_b #11#2!#3!%
{%
```

```
    \xint_gob_til_eightzeroes #2\XINT_div_II_skipc 00000000%
    \XINT_div_II_c #1{1#2}{#3}%
}%
```

x'y{100000000}{1<8>}reste de alpha.#6=B,#7={{x'y},x,K,L}, alpha',B, «c» -> {x'y}x,K,L (à diminuer de 4), {alpha sur K}B{q1=00000000}{alpha'}B,«c»

```
\def\XINT_div_II_skipc 00000000\XINT_div_II_c #1#2#3#4#5\xint:#6#7%
{%
    \XINT_div_II_k #7{#4!#5}{#6}{00000000}%
}%
```

x'ya->1qx'yalpha.B, {{x'y},x,K,L}, nouveau alpha',B, «c». En fait, attention, ici #3 et #4 sont les 16 premiers chiffres du numérateur,sous la forme blocs 1<8chiffres>.

```
\def\XINT_div_II_c #1#2#3#4%
{%
     \expandafter\XINT_div_II_d\the\numexpr\XINT_div_xmini
     #1\xint:#2!#3!#4!{#1}{#2}#3!#4!%
}%
\def\XINT_div_xmini #1%
{%
    \xint_gob_til_one #1\XINT_div_xmini_a 1\XINT_div_mini #1%
}%
\def\XINT_div_xmini_a 1\XINT_div_mini 1#1%
{%
    \xint_gob_til_zero #1\XINT_div_xmini_b 0\XINT_div_mini 1#1%
}%
\def\XINT_div_xmini_b 0\XINT_div_mini 10#1#2#3#4#5#6#7%
{%
    \xint_gob_til_zero #7\XINT_div_xmini_c 0\XINT_div_mini 10#1#2#3#4#5#6#7%
}%
```

x'=10^8 and we return #1=1<8digits>.

```
\def\XINT_div_xmini_c 0\XINT_div_mini 100000000\xint:50000000!#1!#2!{#1!}%
```

1 suivi de q1 sur huit chiffres! #2=x', #3=y, #4=alpha.#5=B, {{x'y},x,K,L}, alpha', B, «c» --> nouvel alpha.x',y,B,q1,{{x'y},x,K,L}, alpha', B, «c»

```
\def\XINT_div_II_d 1#1#2#3#4#5!#6#7#8\xint:#9%
{%
    \expandafter\XINT_div_II_e
    \romannumeral0\expandafter\XINT_div_sub\expandafter
      {\romannumeral0\XINT_rev_nounsep {}#8\R!\R!\R!\R!\R!\R!\R!\R!\W}%
      {\the\numexpr\XINT_div_smallmul_a 100000000\xint:#1#2#3#4\xint:#5!#91;!}%
    \xint:{#6}{#7}{#9}{#1#2#3#4#5}%
}%
```

alpha.x',y,B,q1, {{x'y},x,K,L}, alpha', B, «c». Attention la soustraction spéciale doit maintenir les blocs 1<8>!

```
\def\XINT_div_II_e 1#1!%
{%
    \xint_gob_til_eightzeroes #1\XINT_div_II_skipf 00000000%
    \XINT_div_II_f 1#1!%
}%
```

100000000! alpha sur K chiffres.#2=x',#3=y,#4=B,#5=q1, #6={{x'y},x,K,L}, #7=alpha',B«c» -> {x'y}x,K,L (à diminuer de 1), {alpha sur K}B{q1}{alpha'}B«c»

```
\def\XINT_div_II_skipf 00000000\XINT_div_II_f 100000000!#1\xint:#2#3#4#5#6%
{%
    \XINT_div_II_k #6{#1}{#4}{#5}%
}%
```

1<a1>!1<a2>!, alpha (sur K+1 blocs de 8). x', y, B, q1, {{x'y},x,K,L}, alpha', B,«c».
Here also we are dividing with x' which could be 10^8 in the exceptional case x=99999999. Must intercept it before sending to \XINT_div_mini.

```
\def\XINT_div_II_f #1!#2!#3\xint:%
{%
    \XINT_div_II_fa {#1!#2!}{#1!#2!#3}%
}%
\def\XINT_div_II_fa #1#2#3#4%
{%
    \expandafter\XINT_div_II_g \the\numexpr\XINT_div_xmini #3\xint:#4!#1{#2}%
}%
```

#1=q, #2=alpha (K+4), #3=B, #4=q1, {{x'y},x,K,L}, alpha', BQ«c» -> 1 puis nouveau q sur 8 chiffres. nouvel alpha sur K blocs, B, {{x'y},x,K,L}, alpha',B«c»

```
\def\XINT_div_II_g 1#1#2#3#4#5!#6#7#8%
{%
    \expandafter \XINT_div_II_h
    \the\numexpr 1#1#2#3#4#5+#8\expandafter\expandafter\expandafter
    \xint:\expandafter\expandafter\expandafter
    {\expandafter\xint_gob_til_exclam
     \romannumeral0\expandafter\XINT_div_sub\expandafter
       {\romannumeral0\XINT_rev_nounsep {}#6\R!\R!\R!\R!\R!\R!\R!\R!\W}%
       {\the\numexpr\XINT_div_smallmul_a 100000000\xint:#1#2#3#4\xint:#5!#71;!}}%
    {#7}%
}%
```

1 puis nouveau q sur 8 chiffres, #2=nouvel alpha sur K blocs, #3=B, #4={{x'y},x,K,L} avec L à ajuster, alpha', BQ«c» -> {x'y}x,K,L à diminuer de 1, {alpha}B{q}, alpha', BQ«c»

```
\def\XINT_div_II_h 1#1\xint:#2#3#4%
{%
    \XINT_div_II_k #4{#2}{#3}{#1}%
}%
```

{x'y}x,K,L à diminuer de 1, alpha, B{q}alpha',B«c» ->nouveau L.K,x',y,x,alpha.B,q,alpha',B,«c» ->{LK{x'y}x},x,a,alpha.B,q,alpha',B,«c»

```
\def\XINT_div_II_k #1#2#3#4#5%
{%
    \expandafter\XINT_div_II_l \the\numexpr #4-\xint_c_i\xint:{#3}#1{#2}#5\xint:%
}%
\def\XINT_div_II_l #1\xint:#2#3#4#51#6!%
{%
    \XINT_div_II_m {{#1}{#2}{{#3}{#4}}{#5}}{#5}{#6}1#6!%
}%
```

{LK{x'y}x},x,a,alpha.B{q}alpha'B -> a, x, alpha, B, q, L, K, {x'y}, x, alpha', B«c»

```
\def\XINT_div_II_m #1#2#3#4\xint:#5#6%
{%
     \XINT_div_I_a {#3}{#2}{#4}{#5}{#6}#1%
}%
```

This multiplication is exactly like \XINT_smallmul -- apart from not inserting an ending 1;! --, but keeps ever a vanishing ending carry.

```
\def\XINT_div_minimulwc_a 1#1\xint:#2\xint:#3!#4#5#6#7#8\xint:%
{%
    \expandafter\XINT_div_minimulwc_b
    \the\numexpr \xint_c_x^ix+#1+#3*#8\xint:#3*#4#5#6#7+#2*#8\xint:#2*#4#5#6#7\xint:%
}%
\def\XINT_div_minimulwc_b 1#1#2#3#4#5#6\xint:#7\xint:%
{%
    \expandafter\XINT_div_minimulwc_c
    \the\numexpr \xint_c_x^ix+#1#2#3#4#5+#7\xint:#6\xint:%
}%
\def\XINT_div_minimulwc_c 1#1#2#3#4#5#6\xint:#7\xint:#8\xint:%
{%
    1#6#7\expandafter!%
    \the\numexpr\expandafter\XINT_div_smallmul_a
    \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8\xint:%
}%
\def\XINT_div_smallmul_a #1\xint:#2\xint:#3!1#4!%
{%
    \xint_gob_til_sc #4\XINT_div_smallmul_e;%
    \XINT_div_minimulwc_a #1\xint:#2\xint:#3!#4\xint:#2\xint:#3!%
}%
\def\XINT_div_smallmul_e;\XINT_div_minimulwc_a 1#1\xint:#2;#3!{1\relax #1!}%
```

Special very small multiplication for division. We only need to cater for multiplicands from 1 to 9. The ending is different from standard verysmallmul, a zero carry is not suppressed. And no final 1;! is added. If multiplicand is just 1 let's not forget to add the zero carry 100000000! at the end.

```
\def\XINT_div_verysmallmul #1%
   {\xint_gob_til_one #1\XINT_div_verysmallisone 1\XINT_div_verysmallmul_a 0\xint:#1}%
\def\XINT_div_verysmallisone 1\XINT_div_verysmallmul_a 0\xint:1!1#11;!%
   {1\relax #1100000000!}%
\def\XINT_div_verysmallmul_a #1\xint:#2!1#3!%
{%
    \xint_gob_til_sc #3\XINT_div_verysmallmul_e;%
    \expandafter\XINT_div_verysmallmul_b
    \the\numexpr \xint_c_x^ix+#2*#3+#1\xint:#2!%
}%
\def\XINT_div_verysmallmul_b 1#1#2\xint:%
    {1#2\expandafter!\the\numexpr\XINT_div_verysmallmul_a #1\xint:}%
\def\XINT_div_verysmallmul_e;#1;+#2#3!{1\relax 0000000#2!}%
```

Special subtraction for division purposes. If the subtracted thing turns out to be bigger, then just return a -. If not, then we must reverse the result, keeping the separators.

```
\def\XINT_div_sub #1#2%
{%
    \expandafter\XINT_div_sub_clean
    \the\numexpr\expandafter\XINT_div_sub_a\expandafter
    1#2;!;!;!;!;!\W #1;!;!;!;!;!\W
}%
\def\XINT_div_sub_clean #1-#2#3\W
{%
    \if1#2\expandafter\XINT_rev_nounsep\else\expandafter\XINT_div_sub_neg\fi
    {}#1\R!\R!\R!\R!\R!\R!\R!\R!\W
}%
\def\XINT_div_sub_neg #1\W { -}%
\def\XINT_div_sub_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
{%
    \XINT_div_sub_b #1!#6!#2!#7!#3!#8!#4!#9!#5\W
}%
\def\XINT_div_sub_b #1#2#3!#4!%
{%
    \xint_gob_til_sc #4\XINT_div_sub_bi ;%
    \expandafter\XINT_div_sub_c\the\numexpr#1-#3+1#4-\xint_c_i\xint:%
}%
\def\XINT_div_sub_c 1#1#2\xint:%
{%
    1#2\expandafter!\the\numexpr\XINT_div_sub_d #1%
}%
\def\XINT_div_sub_d #1#2#3!#4!%
{%
    \xint_gob_til_sc #4\XINT_div_sub_di ;%
    \expandafter\XINT_div_sub_e\the\numexpr#1-#3+1#4-\xint_c_i\xint:%
}%
\def\XINT_div_sub_e 1#1#2\xint:%
{%
    1#2\expandafter!\the\numexpr\XINT_div_sub_f #1%
}%
\def\XINT_div_sub_f #1#2#3!#4!%
{%
    \xint_gob_til_sc #4\XINT_div_sub_fi ;%
    \expandafter\XINT_div_sub_g\the\numexpr#1-#3+1#4-\xint_c_i\xint:%
}%
\def\XINT_div_sub_g 1#1#2\xint:%
{%
    1#2\expandafter!\the\numexpr\XINT_div_sub_h #1%
}%
\def\XINT_div_sub_h #1#2#3!#4!%
{%
    \xint_gob_til_sc #4\XINT_div_sub_hi ;%
    \expandafter\XINT_div_sub_i\the\numexpr#1-#3+1#4-\xint_c_i\xint:%
}%
\def\XINT_div_sub_i 1#1#2\xint:%
{%
    1#2\expandafter!\the\numexpr\XINT_div_sub_a #1%
}%
```

```
\def\XINT_div_sub_bi;%
    \expandafter\XINT_div_sub_c\the\numexpr#1-#2+#3\xint:#4!#5!#6!#7!#8!#9!;!\W
{%
    \XINT_div_sub_l #1#2!#5!#7!#9!%
}%
\def\XINT_div_sub_di;%
    \expandafter\XINT_div_sub_e\the\numexpr#1-#2+#3\xint:#4!#5!#6!#7!#8\W
{%
    \XINT_div_sub_l #1#2!#5!#7!%
}%
\def\XINT_div_sub_fi;%
    \expandafter\XINT_div_sub_g\the\numexpr#1-#2+#3\xint:#4!#5!#6\W
{%
    \XINT_div_sub_l #1#2!#5!%
}%
\def\XINT_div_sub_hi;%
    \expandafter\XINT_div_sub_i\the\numexpr#1-#2+#3\xint:#4\W
{%
    \XINT_div_sub_l #1#2!%
}%
\def\XINT_div_sub_l #1%
{%
   \xint_UDzerofork
      #1{-2\relax}%
       0\XINT_div_sub_r
   \krof
}%
\def\XINT_div_sub_r #1!%
{%
    -\ifnum 0#1=\xint_c_ 1\else2\fi\relax
}%
```

Ici B<10^8 (et est >2). On exécute
\expandafter\XINT_sdiv_out\the\numexpr\XINT_smalldivx_a x.1B!1<8d>!...1<8d>!1;!
avec x =round(B/2), 1B=10^8+B, et A déjà en blocs 1<8d>! (non renversés). Le \the\numexpr\XINT_smalldivx_a va produire Q\Z R\W avec un R<10^8, et un Q sous forme de blocs 1<8d>! terminé par 1! et nécessitant le nettoyage du premier bloc. Dans cette branche le B n'a pas été multiplié par une puissance de 10, il peut avoir moins de huit chiffres.

```
\def\XINT_sdiv_out #1;!#2!%
    {\expandafter
     {\romannumeral0\XINT_unsep_cuzsmall
      #1\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax}%
     {#2}}%
```

La toute première étape fait la première division pour être sûr par la suite d'avoir un premier bloc pour A qui sera < B.

```
\def\XINT_smalldivx_a #1\xint:1#2!1#3!%
{%
    \expandafter\XINT_smalldivx_b
    \the\numexpr (#3+#1)/#2-\xint_c_i!#1\xint:#2!#3!%
}%
\def\XINT_smalldivx_b #1#2!%
```

```
{%
    \if0#1\else
          \xint_c_x^viii+#1#2\xint_afterfi{\expandafter!\the\numexpr}\fi
    \XINT_smalldiv_c #1#2!%
}%
\def\XINT_smalldiv_c #1!#2\xint:#3!#4!%
{%
    \expandafter\XINT_smalldiv_d\the\numexpr #4-#1*#3!#2\xint:#3!%
}%
```

On va boucler ici: #1 est un reste, #2 est x.B (avec B sans le 1 mais sur huit chiffres). #3#4 est le premier bloc qui reste de A. Si on a terminé avec A, alors #1 est le reste final. Le quotient lui est terminé par un 1! ce 1! disparaîtra dans le nettoyage par \XINT_unsep_cuzsmall.

```
\def\XINT_smalldiv_d #1!#2!1#3#4!%
{%
    \xint_gob_til_sc #3\XINT_smalldiv_end ;%
    \XINT_smalldiv_e #1!#2!1#3#4!%
}%
\def\XINT_smalldiv_end;\XINT_smalldiv_e #1!#2!1;!{1!;!#1!}%
```

Il est crucial que le reste #1 est < #3. J'ai documenté cette routine dans le fichier où j'ai préparé 1.2, il faudra transférer ici. Il n'est pas nécessaire pour cette routine que le diviseur B ait au moins 8 chiffres. Mais il doit être < 10^8.

```
\def\XINT_smalldiv_e #1!#2\xint:#3!%
{%
    \expandafter\XINT_smalldiv_f\the\numexpr
    \xint_c_xi_e_viii_mone+#1*\xint_c_x^viii/#3!#2\xint:#3!#1!%
}%
\def\XINT_smalldiv_f 1#1#2#3#4#5#6!#7\xint:#8!%
{%
     \xint_gob_til_zero #1\XINT_smalldiv_fz 0%
     \expandafter\XINT_smalldiv_g
     \the\numexpr\XINT_minimul_a #2#3#4#5\xint:#6!#8!#2#3#4#5#6!#7\xint:#8!%
}%
\def\XINT_smalldiv_fz 0%
    \expandafter\XINT_smalldiv_g\the\numexpr\XINT_minimul_a
    9999\xint:9999!#1!99999999!#2!0!1#3!%
{%
    \XINT_smalldiv_i \xint:#3!\xint_c_!#2!%
}%
\def\XINT_smalldiv_g 1#1!1#2!#3!#4!#5!#6!%
{%
    \expandafter\XINT_smalldiv_h\the\numexpr 1#6-#1\xint:#2!#5!#3!#4!%
}%
\def\XINT_smalldiv_h 1#1#2\xint:#3!#4!%
{%
    \expandafter\XINT_smalldiv_i\the\numexpr #4-#3+#1-\xint_c_i\xint:#2!%
}%
\def\XINT_smalldiv_i #1\xint:#2!#3!#4\xint:#5!%
{%
    \expandafter\XINT_smalldiv_j\the\numexpr (#1#2+#4)/#5-\xint_c_i!#3!#1#2!#4\xint:#5!%
}%
```

```
\def\XINT_smalldiv_j #1!#2!%
{%
    \xint_c_x^viii+#1+#2\expandafter!\the\numexpr\XINT_smalldiv_k
    #1!%
}%
```

On boucle vers \XINT_smalldiv_d.

```
\def\XINT_smalldiv_k #1!#2!#3\xint:#4!%
{%
    \expandafter\XINT_smalldiv_d\the\numexpr #2-#1*#4!#3\xint:#4!%
}%
```

Cette routine fait la division euclidienne d'un nombre de seize chiffres par #1 = C = diviseur sur huit chiffres >= 10^7, avec #2 = sa moitié utilisée dans \numexpr pour contrebalancer l'arrondi (ARRRRRRGGGGGHHHH) fait par /. Le nombre divisé XY = X*10^8+Y se présente sous la forme 1<8chiffres>!1<8chiffres>! avec plus significatif en premier.

Seul le quotient est calculé, pas le reste. En effet la routine de division principale va utiliser ce quotient pour déterminer le "grand" reste, et le petit reste ici ne nous serait d'à peu près aucune utilité.

ATTENTION UNIQUEMENT UTILISÉ POUR DES SITUATIONS OÙ IL EST GARANTI QUE X < C ! (et C au moins 10^7) le quotient euclidien de X*10^8+Y par C sera donc < 10^8. Il sera renvoyé sous la forme 1<8chiffres>.

```
\def\XINT_div_mini #1\xint:#2!1#3!%
{%
    \expandafter\XINT_div_mini_a\the\numexpr
    \xint_c_xi_e_viii_mone+#3*\xint_c_x^viii/#1!#1\xint:#2!#3!%
}%
```

Note (2015/10/08). Attention à la différence dans l'ordre des arguments avec ce que je vois en dans \XINT_smalldiv_f. Je ne me souviens plus du tout s'il y a une raison quelconque.

```
\def\XINT_div_mini_a 1#1#2#3#4#5#6!#7\xint:#8!%
{%
     \xint_gob_til_zero #1\XINT_div_mini_w 0%
     \expandafter\XINT_div_mini_b
     \the\numexpr\XINT_minimul_a #2#3#4#5\xint:#6!#7!#2#3#4#5#6!#7\xint:#8!%
}%
\def\XINT_div_mini_w 0%
    \expandafter\XINT_div_mini_b\the\numexpr\XINT_minimul_a
    9999\xint:9999!#1!99999999!#2\xint:#3!00000000!#4!%
{%
    \xint_c_x^viii_mone+(#4+#3)/#2!%
}%
\def\XINT_div_mini_b 1#1!1#2!#3!#4!#5!#6!%
{%
    \expandafter\XINT_div_mini_c
    \the\numexpr 1#6-#1\xint:#2!#5!#3!#4!%
}%
\def\XINT_div_mini_c 1#1#2\xint:#3!#4!%
{%
    \expandafter\XINT_div_mini_d
    \the\numexpr #4-#3+#1-\xint_c_i\xint:#2!%
}%
\def\XINT_div_mini_d #1\xint:#2!#3!#4\xint:#5!%
{%
    \xint_c_x^viii_mone+#3+(#1#2+#5)/#4!%
}%
```

Derived arithmetic

4.38 \xintiiQuo, \xintiiRem

```
\def\xintiiQuo {\romannumeral0\xintiiquo }%
\def\xintiiRem {\romannumeral0\xintiirem }%
\def\xintiiquo
   {\expandafter\xint_firstoftwo_thenstop\romannumeral0\xintiidivision }%
\def\xintiirem
   {\expandafter\xint_secondoftwo_thenstop\romannumeral0\xintiidivision }%
```

4.39 \xintiiDivRound

1.1, transferred from first release of bnumexpr. Rewritten for 1.2. Ending rewritten for 1.2i. (new \xintDSRr).

1.2l: \xintiiDivRound made robust against non terminated input.

```
\def\xintiiDivRound   {\romannumeral0\xintiidivround }%
\def\xintiidivround #1{\expandafter\XINT_iidivround\romannumeral`&&@#1\xint:}%
\def\XINT_idivround #1#2\xint:#3%
    {\expandafter\XINT_iidivround_a\expandafter #1%
                 \romannumeral0\xintnum{#3}\xint:#2\xint:}%
\def\XINT_iidivround #1#2\xint:#3%
    {\expandafter\XINT_iidivround_a\expandafter #1\romannumeral`&&@#3\xint:#2\xint:}%
\def\XINT_iidivround_a #1#2% #1 de A, #2 de B.
{%
    \if0#2\xint_dothis{\XINT_iidivround_divbyzero#1#2}\fi
    \if0#1\xint_dothis\XINT_iidivround_aiszero\fi
    \if-#2\xint_dothis{\XINT_iidivround_bneg #1}\fi
          \xint_orthat{\XINT_iidivround_bpos #1#2}%
}%
\def\XINT_iidivround_divbyzero #1#2#3\xint:#4\xint:
   {\XINT_signalcondition{DivisionByZero}{Division of #1#4 by #2#3}{}{0}}%
\def\XINT_iidivround_aiszero   #1\xint:#2\xint:{ 0}%
\def\XINT_iidivround_bpos #1%
{%
    \xint_UDsignfork
            #1{\xintiiopp\XINT_iidivround_pos {}}%
             -{\XINT_iidivround_pos #1}%
    \krof
}%
\def\XINT_iidivround_bneg #1%
{%
    \xint_UDsignfork
            #1{\XINT_iidivround_pos {}}%
             -{\xintiiopp\XINT_iidivround_pos #1}%
    \krof
}%
```

```
\def\XINT_iidivround_pos #1#2\xint:#3\xint:
{%
    \expandafter\expandafter\expandafter\XINT_dsrr
    \expandafter\xint_firstoftwo
    \romannumeral0\XINT_div_prepare {#2}{#1#30}%
    \xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax
}%
```

4.40 \xintiiDivTrunc

1.2l: \xintiiDivTrunc made robust against non terminated input.

```
\def\xintiiDivTrunc   {\romannumeral0\xintiidivtrunc }%
\def\xintiidivtrunc #1{\expandafter\XINT_iidivtrunc\romannumeral`&&@#1\xint:}%
\def\XINT_iidivtrunc #1#2\xint:#3{\expandafter\XINT_iidivtrunc_a\expandafter #1%
                              \romannumeral`&&@#3\xint:#2\xint:}%
\def\XINT_iidivtrunc_a #1#2% #1 de A, #2 de B.
{%
    \if0#2\xint_dothis{\XINT_iidivtrunc_divbyzero#1#2}\fi
    \if0#1\xint_dothis\XINT_iidivtrunc_aiszero\fi
    \if-#2\xint_dothis{\XINT_iidivtrunc_bneg #1}\fi
          \xint_orthat{\XINT_iidivtrunc_bpos #1#2}%
}%
```

Attention to not move DivRound code beyond that point.

```
\let\XINT_iidivtrunc_divbyzero\XINT_iidivround_divbyzero
\let\XINT_iidivtrunc_aiszero  \XINT_iidivround_aiszero
\def\XINT_iidivtrunc_bpos #1%
{%
    \xint_UDsignfork
            #1{\xintiiopp\XINT_iidivtrunc_pos {}}%
             -{\XINT_iidivtrunc_pos #1}%
    \krof
}%
\def\XINT_iidivtrunc_bneg #1%
{%
    \xint_UDsignfork
            #1{\XINT_iidivtrunc_pos {}}%
             -{\xintiiopp\XINT_iidivtrunc_pos #1}%
    \krof
}%
\def\XINT_iidivtrunc_pos #1#2\xint:#3\xint:
    {\expandafter\xint_firstoftwo_thenstop
     \romannumeral0\XINT_div_prepare {#2}{#1#3}}%
```

4.41 \xintiiModTrunc

Renamed from \xintiiMod to \xintiiModTrunc at 1.2p.

```
\def\xintiiModTrunc {\romannumeral0\xintiimodtrunc }%
\def\xintiimodtrunc #1{\expandafter\XINT_iimodtrunc\romannumeral`&&@#1\xint:}%
\def\XINT_iimodtrunc #1#2\xint:#3{\expandafter\XINT_iimodtrunc_a\expandafter #1%
```

```
                              \romannumeral`&&@#3\xint:#2\xint:}%
\def\XINT_iimodtrunc_a #1#2% #1 de A, #2 de B.
{%
    \if0#2\xint_dothis{\XINT_iimodtrunc_divbyzero#1#2}\fi
    \if0#1\xint_dothis\XINT_iimodtrunc_aiszero\fi
    \if-#2\xint_dothis{\XINT_iimodtrunc_bneg #1}\fi
          \xint_orthat{\XINT_iimodtrunc_bpos #1#2}%
}%
```

Attention to not move DivRound code beyond that point. A bit of abuse here for divbyzero defaulted-to value, which happily works in both.

```
\let\XINT_iimodtrunc_divbyzero\XINT_iidivround_divbyzero
\let\XINT_iimodtrunc_aiszero  \XINT_iidivround_aiszero
\def\XINT_iimodtrunc_bpos #1%
{%
    \xint_UDsignfork
            #1{\xintiiopp\XINT_iimodtrunc_pos {}}%
             -{\XINT_iimodtrunc_pos #1}%
    \krof
}%
\def\XINT_iimodtrunc_bneg #1%
{%
    \xint_UDsignfork
            #1{\xintiiopp\XINT_iimodtrunc_pos {}}%
             -{\XINT_iimodtrunc_pos #1}%
    \krof
}%
\def\XINT_iimodtrunc_pos #1#2\xint:#3\xint:
    {\expandafter\xint_secondoftwo_thenstop\romannumeral0\XINT_div_prepare
      {#2}{#1#3}}%
```

4.42 \xintiiDivMod

1.2p. Associated with floored division like Python's divmod.

```
\def\xintiiDivMod   {\romannumeral0\xintiidivmod }%
\def\xintiidivmod #1{\expandafter\XINT_iidivmod\romannumeral`&&@#1\xint:}%
\def\XINT_iidivmod #1#2\xint:#3{\expandafter\XINT_iidivmod_a\expandafter #1%
                              \romannumeral`&&@#3\xint:#2\xint:}%
\def\XINT_iidivmod_a #1#2% #1 de A, #2 de B.
{%
    \if0#2\xint_dothis{\XINT_iidivmod_divbyzero#1#2}\fi
    \if0#1\xint_dothis\XINT_iidivmod_aiszero\fi
    \if-#2\xint_dothis{\XINT_iidivmod_bneg #1}\fi
          \xint_orthat{\XINT_iidivmod_bpos #1#2}%
}%
\def\XINT_iidivmod_divbyzero #1#2\xint:#3\xint:
{%
    \XINT_signalcondition{DivisionByZero}{Division by #2 of #1#3}{}%
    {{0}{0}}% à revoir...
}%
\def\XINT_iidivmod_aiszero #1#2\xint:#3\xint:{{0}{0}}%
```

```
\def\XINT_iidivmod_bneg #1%
{%
    \expandafter\XINT_iidivmod_bneg_finish
    \romannumeral0\xint_UDsignfork
            #1{\XINT_iidivmod_bpos {}}%
             -{\XINT_iidivmod_bpos {-#1}}%
    \krof
}%
\def\XINT_iidivmod_bneg_finish#1#2%
{%
    \expandafter\xint_exchangetwo_keepbraces\expandafter
    {\romannumeral0\xintiiopp#2}{#1}%
}%
\def\XINT_iidivmod_bpos #1#2\xint:#3\xint:{\xintiidivision{#1#3}{#2}}%
```

4.43 \xintiiDivFloor

1.2p. For bnumexpr actually, because \xintiiexpr could use \xintDivFloor which also outputs an integer in strict format.

```
\def\xintiiDivFloor {\romannumeral0\xintiidivfloor}%
\def\xintiidivfloor {\expandafter\xint_firstoftwo_thenstop
                \romannumeral0\xintiidivmod}%
```

4.44 \xintiiMod

Associated with floored division at 1.2p. Formerly was associated with truncated division.

```
\def\xintiiMod {\romannumeral0\xintiimod}%
\def\xintiimod {\expandafter\xint_secondoftwo_thenstop
                \romannumeral0\xintiidivmod}%
```

4.45 \xintiiSqr

1.2l: \xintiiSqr made robust against non terminated input.

```
\def\xintiiSqr {\romannumeral0\xintiisqr }%
\def\xintiisqr #1%
{%
    \expandafter\XINT_sqr\romannumeral0\xintiiabs{#1}\xint:
}%
\def\XINT_sqr #1\xint:
{%
    \expandafter\XINT_sqr_a
      \romannumeral0\expandafter\XINT_sepandrev_andcount
      \romannumeral0\XINT_zeroes_forviii #1\R\R\R\R\R\R\R\R{10}0000001\W
      #1\XINT_rsepbyviii_end_A 2345678%
        \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
              \R\xint:\xint_c_xii \R\xint:\xint_c_x  \R\xint:\xint_c_viii \R\xint:\xint_c_vi
              \R\xint:\xint_c_iv  \R\xint:\xint_c_ii \R\xint:\xint_c_\W
      \xint:
}%
```

1.2c \XINT_mul_loop can now be called directly even with small arguments, thus the following check is not anymore a necessity.

```
\def\XINT_sqr_a #1\xint:
{%
    \ifnum #1=\xint_c_i \expandafter\XINT_sqr_small
                   \else\expandafter\XINT_sqr_start\fi
}%
\def\XINT_sqr_small 1#1#2#3#4#5!\xint:
{%
    \ifnum #1#2#3#4#5<46341 \expandafter\XINT_sqr_verysmall\fi
    \expandafter\XINT_sqr_small_out
    \the\numexpr\XINT_minimul_a #1#2#3#4\xint:#5!#1#2#3#4#5!%
}%
\def\XINT_sqr_verysmall#1{%
\def\XINT_sqr_verysmall
    \expandafter\XINT_sqr_small_out\the\numexpr\XINT_minimul_a ##1!##2!%
    {\expandafter#1\the\numexpr ##2*##2\relax}%
}\XINT_sqr_verysmall{ }%
\def\XINT_sqr_small_out 1#1!1#2!%
{%
    \XINT_cuz #2#1\R
}%
```

An ending 1;! is produced on output for \XINT_mul_loop and gets incorporated to the delimiter needed by the \XINT_unrevbyviii done by \XINT_mul_out.

```
\def\XINT_sqr_start #1\xint:
{%
    \expandafter\XINT_mul_out
    \the\numexpr\XINT_mul_loop
                100000000!1;!\W #11;!\W #11;!%
    1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
}%
```

4.46 \xintiiPow

The exponent is not limited but with current default settings of tex memory, with xint 1.2, the maximal exponent for 2^N is N = 2^17 = 131072.

1.2f Modifies the initial steps: 1) in order to be able to let more easily \xintiPow use \xintNum on the exponent once xintfrac.sty is loaded; 2) also because I noticed it was not very well coded. And it did only a \numexpr on the exponent, contradicting the documentation related to the "i" convention in names.

1.2l: \xintiiPow made robust against non terminated input.

```
\def\xintiiPow {\romannumeral0\xintiipow }%
\def\xintiipow #1#2%
{%
    \expandafter\xint_pow\the\numexpr #2\expandafter
    .\romannumeral`&&@#1\xint:
}%
\def\xint_pow #1.#2%#3\xint:
{%
```

```
    \xint_UDzerominusfork
      #2-\XINT_pow_AisZero
      0#2\XINT_pow_Aneg
      0-{\XINT_pow_Apos #2}%
    \krof {#1}%
}%
\def\XINT_pow_AisZero #1#2\xint:
{%
     \ifcase\XINT_cntSgn #1\xint:
         \xint_afterfi { 1}%
     \or
         \xint_afterfi { 0}%
     \else
         \xint_afterfi
        {\XINT_signalcondition{DivisionByZero}{Zero to power #1}{}{0}}%
     \fi
}%
\def\XINT_pow_Aneg #1%
{%
   \ifodd #1
       \expandafter\XINT_opp\romannumeral0%
   \fi
   \XINT_pow_Apos {}{#1}%
}%
\def\XINT_pow_Apos #1#2{\XINT_pow_Apos_a {#2}#1}%
\def\XINT_pow_Apos_a #1#2#3%
{%
    \xint_gob_til_xint: #3\XINT_pow_Apos_short\xint:
    \XINT_pow_AatleastTwo {#1}#2#3%
}%
\def\XINT_pow_Apos_short\xint:\XINT_pow_AatleastTwo #1#2\xint:
{%
    \ifcase #2
         \xintError:thiscannothappen
    \or  \expandafter\XINT_pow_AisOne
    \else\expandafter\XINT_pow_AatleastTwo
    \fi {#1}#2\xint:
}%
\def\XINT_pow_AisOne #1\xint:{ 1}%
\def\XINT_pow_AatleastTwo #1%
{%
    \ifcase\XINT_cntSgn #1\xint:
        \expandafter\XINT_pow_BisZero
    \or
        \expandafter\XINT_pow_I_in
    \else
        \expandafter\XINT_pow_BisNegative
    \fi
    {#1}%
}%
\def\XINT_pow_BisNegative #1\xint:{\XINT_signalcondition{Underflow}{Inverse power
    can not be represented by an integer}{}{0}}%
```

```
\def\XINT_pow_BisZero #1\xint:{ 1}%
```

B = #1 > 0, A = #2 > 1. Earlier code checked if size of B did not exceed a given limit (for example 131000).

```
\def\XINT_pow_I_in #1#2\xint:
{%
    \expandafter\XINT_pow_I_loop
    \the\numexpr #1\expandafter\xint:%
    \romannumeral0\expandafter\XINT_sepandrev
    \romannumeral0\XINT_zeroes_forviii #2\R\R\R\R\R\R\R\R{10}0000001\W
    #2\XINT_rsepbyviii_end_A 2345678%
      \XINT_rsepbyviii_end_B 2345678\relax XX%
    \R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\W
    1;!\W
    1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
}%
\def\XINT_pow_I_loop #1\xint:%
{%
    \ifnum #1 = \xint_c_i\expandafter\XINT_pow_I_exit\fi
    \ifodd #1
       \expandafter\XINT_pow_II_in
    \else
       \expandafter\XINT_pow_I_squareit
    \fi #1\xint:%
}%
\def\XINT_pow_I_exit \ifodd #1\fi #2\xint:#3\W {\XINT_mul_out #3}%
```

The 1.2c \XINT_mul_loop can be called directly even with small arguments, hence the "butcheckifsmall" is not a necessity as it was earlier with 1.2. On 2^30, it does bring roughly a 40% time gain though, and 30% gain for 2^60. The overhead on big computations should be negligible.

```
\def\XINT_pow_I_squareit #1\xint:#2\W%
{%
    \expandafter\XINT_pow_I_loop
    \the\numexpr #1/\xint_c_ii\expandafter\xint:%
    \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W
}%
\def\XINT_pow_mulbutcheckifsmall #1!1#2%
{%
    \xint_gob_til_sc #2\XINT_pow_mul_small;%
    \XINT_mul_loop 100000000!1;!\W #1!1#2%
}%
\def\XINT_pow_mul_small;\XINT_mul_loop
    100000000!1;!\W 1#1!1;!\W
{%
    \XINT_smallmul 1#1!%
}%
\def\XINT_pow_II_in #1\xint:#2\W
{%
    \expandafter\XINT_pow_II_loop
    \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter\xint:%
    \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W #2\W
}%
```

```
\def\XINT_pow_II_loop #1\xint:%
{%
    \ifnum #1 = \xint_c_i\expandafter\XINT_pow_II_exit\fi
    \ifodd #1
       \expandafter\XINT_pow_II_odda
    \else
       \expandafter\XINT_pow_II_even
    \fi #1\xint:%
}%
\def\XINT_pow_II_exit\ifodd #1\fi #2\xint:#3\W #4\W
{%
    \expandafter\XINT_mul_out
    \the\numexpr\XINT_pow_mulbutcheckifsmall #4\W #3%
}%
\def\XINT_pow_II_even #1\xint:#2\W
{%
    \expandafter\XINT_pow_II_loop
    \the\numexpr #1/\xint_c_ii\expandafter\xint:%
    \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W
}%
\def\XINT_pow_II_odda #1\xint:#2\W #3\W
{%
    \expandafter\XINT_pow_II_oddb
    \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter\xint:%
    \the\numexpr\XINT_pow_mulbutcheckifsmall #3\W #2\W #2\W
}%
\def\XINT_pow_II_oddb #1\xint:#2\W #3\W
{%
    \expandafter\XINT_pow_II_loop
    \the\numexpr #1\expandafter\xint:%
    \the\numexpr\XINT_pow_mulbutcheckifsmall #3\W #3\W #2\W
}%
```

4.47 \xintiiFac

Moved here from xint.sty with release 1.2 (to be usable by \bnumexpr).

An \xintiFac is needed by xintexpr.sty. Prior to 1.2o it was defined here as an alias to \xintiiFac, then redefined by xintfrac to use \xintNum. This was incoherent. Contrarily to other similarly named macros, \xintiiFac uses \numexpr on its input. This is also incoherent with the naming scheme, alas.

Partially rewritten with release 1.2 to benefit from the inner format of the 1.2 multiplication.

With current default settings of the etex memory and a.t.t.o.w (11/2015) the maximal possible computation is 5971! (which has 19956 digits).

Note (end november 2015): I also tried out a quickly written recursive (binary split) implementation

```
\catcode`_ 11
\catcode`^ 11
\long\def\xint_firstofthree  #1#2#3{#1}%
\long\def\xint_secondofthree #1#2#3{#2}%
\long\def\xint_thirdofthree  #1#2#3{#3}%
% quickly written factorial using binary split recursive method
\def\tFac   {\romannumeral-`0\tfac }%
```

```
    \def\tfac #1{\expandafter\XINT_mul_out
                 \romannumeral-`0\ufac {1}{#1}1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W}%
    \def\ufac #1#2{\ifcase\numexpr#2-#1\relax
                     \expandafter\xint_firstofthree
                   \or
                     \expandafter\xint_secondofthree
                   \else
                     \expandafter\xint_thirdofthree
                   \fi
                   {\the\numexpr\xint_c_x^viii+#1!1;!}%
                   {\the\numexpr\xint_c_x^viii+#1*#2!1;!}%
                   {\expandafter\vfac\the\numexpr (#1+#2)/\xint_c_ii.#1.#2.}%
    }%
    \def\vfac #1.#2.#3.%
    {%
        \expandafter
        \wfac\expandafter
            {\romannumeral-`0\expandafter
             \ufac\expandafter{\the\numexpr #1+\xint_c_i}{#3}}%
            {\ufac {#2}{#1}}%
    }%
    \def\wfac #1#2{\expandafter\zfac\romannumeral-`0#2\W #1}%
    \def\zfac {\the\numexpr\XINT_mul_loop 100000000!1;!\W }% core multiplication...
    \catcode`_ 8
    \catcode`^ 7
```

and I was quite surprised that it was only about 1.6x--2x slower in the range N=200 to 2000 than the \xintiiFac here which attempts to be smarter...

Note (2017, 1.21): I found out some code comment of mine that the code here should be more in the style of \xintiiBinomial, but I left matters untouched.

1.2o modifies \xintiFac to be coherent with \xintiBinomial: only with xintfrac.sty loaded does it use \xintNum. It is documented only as macro of xintfrac.sty, not as macro of xint.sty.

```
\def\xintiiFac {\romannumeral0\xintiifac }%
\def\xintiifac #1{\expandafter\XINT_fac_fork\the\numexpr#1.}%
\def\XINT_fac_fork #1#2.%
{%
    \xint_UDzerominusfork
     #1-\XINT_fac_zero
     0#1\XINT_fac_neg
      0-\XINT_fac_checksize
    \krof #1#2.%
}%
\def\XINT_fac_zero #1.{ 1}%
\def\XINT_fac_neg  #1.{\XINT_signalcondition{InvalidOperation}{Factorial of
    negative: (#1)!}{}{0}}%
\def\XINT_fac_checksize #1.%
{%
    \ifnum #1>\xint_c_x^iv \xint_dothis{\XINT_fac_toobig #1.}\fi
    \ifnum #1>465 \xint_dothis{\XINT_fac_bigloop_a   #1.}\fi
    \ifnum #1>101 \xint_dothis{\XINT_fac_medloop_a   #1.\XINT_mul_out}\fi
                  \xint_orthat{\XINT_fac_smallloop_a #1.\XINT_mul_out}%
    1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
```

```
}%
\def\XINT_fac_toobig #1.#2\W{\XINT_signalcondition{InvalidOperation}{Factorial
    of too big argument: #1 > 10000}{}{0}}%
\def\XINT_fac_bigloop_a #1.%
{%
    \expandafter\XINT_fac_bigloop_b \the\numexpr
    #1+\xint_c_i-\xint_c_ii*((#1-464)/\xint_c_ii).#1.%
}%
\def\XINT_fac_bigloop_b #1.#2.%
{%
    \expandafter\XINT_fac_medloop_a
        \the\numexpr #1-\xint_c_i.{\XINT_fac_bigloop_loop #1.#2.}%
}%
\def\XINT_fac_bigloop_loop #1.#2.%
{%
    \ifnum #1>#2 \expandafter\XINT_fac_bigloop_exit\fi
    \expandafter\XINT_fac_bigloop_loop
    \the\numexpr #1+\xint_c_ii\expandafter.%
    \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_bigloop_mul #1!%
}%
\def\XINT_fac_bigloop_exit #1!{\XINT_mul_out}%
\def\XINT_fac_bigloop_mul #1!%
{%
    \expandafter\XINT_smallmul
        \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
}%
\def\XINT_fac_medloop_a #1.%
{%
    \expandafter\XINT_fac_medloop_b
        \the\numexpr #1+\xint_c_i-\xint_c_iii*((#1-100)/\xint_c_iii).#1.%
}%
\def\XINT_fac_medloop_b #1.#2.%
{%
    \expandafter\XINT_fac_smallloop_a
        \the\numexpr #1-\xint_c_i.{\XINT_fac_medloop_loop #1.#2.}%
}%
\def\XINT_fac_medloop_loop #1.#2.%
{%
    \ifnum #1>#2 \expandafter\XINT_fac_loop_exit\fi
    \expandafter\XINT_fac_medloop_loop
    \the\numexpr #1+\xint_c_iii\expandafter.%
    \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_medloop_mul #1!%
}%
\def\XINT_fac_medloop_mul #1!%
{%
    \expandafter\XINT_smallmul
    \the\numexpr
        \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
}%
\def\XINT_fac_smallloop_a #1.%
{%
    \csname
```

```
        XINT_fac_smallloop_\the\numexpr #1-\xint_c_iv*(#1/\xint_c_iv)\relax
    \endcsname #1.%
}%
\expandafter\def\csname XINT_fac_smallloop_1\endcsname #1.%
{%
    \XINT_fac_smallloop_loop 2.#1.100000001!1;!%
}%
\expandafter\def\csname XINT_fac_smallloop_-2\endcsname #1.%
{%
    \XINT_fac_smallloop_loop 3.#1.100000002!1;!%
}%
\expandafter\def\csname XINT_fac_smallloop_-1\endcsname #1.%
{%
    \XINT_fac_smallloop_loop 4.#1.100000006!1;!%
}%
\expandafter\def\csname XINT_fac_smallloop_0\endcsname #1.%
{%
    \XINT_fac_smallloop_loop 5.#1.1000000024!1;!%
}%
\def\XINT_fac_smallloop_loop #1.#2.%
{%
    \ifnum #1>#2 \expandafter\XINT_fac_loop_exit\fi
    \expandafter\XINT_fac_smallloop_loop
    \the\numexpr #1+\xint_c_iv\expandafter.%
    \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_smallloop_mul #1!%
}%
\def\XINT_fac_smallloop_mul #1!%
{%
    \expandafter\XINT_smallmul
    \the\numexpr
        \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
}%
\def\XINT_fac_loop_exit #1!#2;!#3{#3#2;!}%
```

4.48 \XINT_useiimessage

1.2o

```
\def\XINT_useiimessage #1% used in LaTeX only
{%
    \XINT_ifFlagRaised {#1}%
    {\@backslashchar#1
     (load xintfrac or use \@backslashchar xintii\xint_gobble_iv#1!)\MessageBreak}%
    {}%
}%
\XINT_restorecatcodes_endinput%
```

5 Package xint implementation

With release 1.1 the core arithmetic routines \xintiiAdd, \xintiiSub, \xintiiMul, \xintiiQuo, \xintiiPow were separated to be the main component of the then new xintcore.

At 1.3 the macros deprecated at 1.2o got all removed.

1.3b adds randomness related macros.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5     % ^^M
  \endlinechar=13 %
  \catcode123=1    % {
  \catcode125=2    % }
  \catcode64=11    % @
  \catcode35=6     % #
  \catcode44=12    % ,
  \catcode45=12    % -
  \catcode46=12    % .
  \catcode58=12    % :
  \let\z\endgroup
  \expandafter\let\expandafter\x\csname ver@xint.sty\endcsname
  \expandafter\let\expandafter\w\csname ver@xintcore.sty\endcsname
  \expandafter
    \ifx\csname PackageInfo\endcsname\relax
      \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
    \else
      \def\y#1#2{\PackageInfo{#1}{#2}}%
    \fi
  \expandafter
  \ifx\csname numexpr\endcsname\relax
     \y{xint}{\numexpr not available, aborting input}%
     \aftergroup\endinput
  \else
    \ifx\x\relax   % plain-TeX, first loading of xintcore.sty
      \ifx\w\relax % but xintkernel.sty not yet loaded.
         \def\z{\endgroup\input xintcore.sty\relax}%
      \fi
    \else
      \def\empty {}%
      \ifx\x\empty % LaTeX, first loading,
      % variable is initialized, but \ProvidesPackage not yet seen
          \ifx\w\relax % xintcore.sty not yet loaded.
            \def\z{\endgroup\RequirePackage{xintcore}}%
          \fi
      \else
        \aftergroup\endinput % xint already loaded.
      \fi
    \fi
  \fi
\z%
\XINTsetupcatcodes% defined in xintkernel.sty (loaded by xintcore.sty)
```

5.1 Package identification

```
\XINT_providespackage
\ProvidesPackage{xint}%
  [2018/06/17 1.3c Expandable operations on big integers (JFB)]%
```

5.2 More token management

```
\long\def\xint_firstofthree  #1#2#3{#1}%
\long\def\xint_secondofthree #1#2#3{#2}%
\long\def\xint_thirdofthree  #1#2#3{#3}%
\long\def\xint_firstofthree_thenstop  #1#2#3{ #1}% 1.09i
\long\def\xint_secondofthree_thenstop #1#2#3{ #2}%
\long\def\xint_thirdofthree_thenstop  #1#2#3{ #3}%
```

5.3 (WIP) A constant needed by **\xintRandomDigits** et al.

```
\ifdefined\xint_texuniformdeviate
  \unless\ifdefined\xint_c_nine_x^viii
      \csname newcount\endcsname\xint_c_nine_x^viii
      \xint_c_nine_x^viii 900000000
  \fi
\fi
```

5.4 \xintLen, \xintiLen

\xintLen gets extended to fractions by xintfrac.sty: A/B is given length len(A)+len(B)-1 (somewhat arbitrary). It applies \xintNum to its argument. A minus sign is accepted and ignored.

For parallelism with \xintiNum/\xintNum, 1.2o defines \xintiLen.

\xintLen gets redefined by xintfrac.

```
\def\xintiLen {\romannumeral0\xintilen }%
\def\xintilen #1{\def\xintilen ##1%
{%
    \expandafter#1\the\numexpr
    \expandafter\XINT_len_fork\romannumeral0\xintinum{##1}%
      \xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
      \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
      \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye\relax
}}\xintilen{ }%
\def\xintLen {\romannumeral0\xintlen }%
\let\xintlen\xintilen
```

```
\def\XINT_len_fork #1%
{%
    \expandafter\XINT_length_loop\xint_UDsignfork#1{}-#1\krof
}%
```

5.5 \xintReverseDigits

1.2.

This puts digits in reverse order, not suppressing leading zeros after reverse. Despite lacking the "ii" in its name, it does not apply \xintNum to its argument (contrarily to \xintLen, this is not very coherent).

1.2l variant is robust against non terminated \the\numexpr input.

This macro is currently not used elsewhere in xint code.

```
\def\xintReverseDigits {\romannumeral0\xintreversedigits }%
\def\xintreversedigits #1%
{%
    \expandafter\XINT_revdigits\romannumeral`&&@#1%
     {\XINT_microrevsep_end\W}\XINT_microrevsep_end
      \XINT_microrevsep_end\XINT_microrevsep_end
      \XINT_microrevsep_end\XINT_microrevsep_end
      \XINT_microrevsep_end\XINT_microrevsep_end\XINT_microrevsep_end\Z
    1\Z!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
}%
\def\XINT_revdigits #1%
{%
    \xint_UDsignfork
      #1{\expandafter-\romannumeral0\XINT_revdigits_a}%
       -{\XINT_revdigits_a #1}%
    \krof
}%
\def\XINT_revdigits_a
{%
```

```
    \expandafter\XINT_revdigits_b\expandafter{\expandafter}%
    \the\numexpr\XINT_microrevsep
}%
\def\XINT_microrevsep #1#2#3#4#5#6#7#8#9%
{%
    1#9#8#7#6#5#4#3#2#1\expandafter!\the\numexpr\XINT_microrevsep
}%
\def\XINT_microrevsep_end #1\W #2\expandafter #3\Z{\relax#2!}%
\def\XINT_revdigits_b #11#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
{%
    \xint_gob_til_R #9\XINT_revdigits_end\R
                  \XINT_revdigits_b {#9#8#7#6#5#4#3#2#1}%
}%
\def\XINT_revdigits_end#1{%
\def\XINT_revdigits_end\R\XINT_revdigits_b ##1##2\W
   {\expandafter#1\xint_gob_til_Z ##1}%
}\XINT_revdigits_end{ }%
\let\xintRev\xintReverseDigits
```

5.6 \xintiiE

Originally was used in \xintiiexpr. Transferred from xintfrac for 1.1. Code rewritten for 1.2i. \xintiiE{x}{e} extends x with e zeroes if e is positive and simply outputs x if e is zero or negative. Attention, le comportement pour e < 0 ne doit pas être modifié car \xintMod et autres macros en dépendent.

```
\def\xintiiE {\romannumeral0\xintiie }%
\def\xintiie #1#2%
   {\expandafter\XINT_iie_fork\the\numexpr #2\expandafter.\romannumeral`&&@#1;}%
\def\XINT_iie_fork #1%
{%
    \xint_UDsignfork
      #1\XINT_iie_neg
       -\XINT_iie_a
    \krof #1%
}%
```

le #2 a le bon pattern terminé par ; #1=0 est OK pour \XINT_rep.

```
\def\XINT_iie_a #1.%
 {\expandafter\XINT_dsx_append\romannumeral\XINT_rep #1\endcsname 0.}%
\def\XINT_iie_neg #1.#2;{ #2}%
```

5.7 \xintDecSplit

DECIMAL SPLIT

The macro \xintDecSplit {x}{A} cuts A which is composed of digits (leading zeroes ok, but no sign) (*) into two (each possibly empty) pieces L and R. The concatenation LR always reproduces A.

The position of the cut is specified by the first argument x. If x is zero or positive the cut location is x slots to the left of the right end of the number. If x becomes equal to or larger than the length of the number then L becomes empty. If x is negative the location of the cut is |x| slots to the right of the left end of the number.

(*) versions earlier than 1.2i first replaced A with its absolute value. This is not the case anymore. This macro should NOT be used for A with a leading sign (+ or -).

Entirely rewritten for 1.2i (2016/12/11).

Attention: \xintDecSplit not robust against non terminated second argument.

```
\def\xintDecSplit {\romannumeral0\xintdecsplit }%
\def\xintdecsplit #1#2%
{%
    \expandafter\XINT_split_finish
    \romannumeral0\expandafter\XINT_split_xfork
    \the\numexpr #1\expandafter.\romannumeral`&&@#2%
    \xint_bye2345678\xint_bye..%
}%
\def\XINT_split_finish  #1.#2.{{#1}{#2}}%

\def\XINT_split_xfork #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_split_zerosplit
      0#1\XINT_split_fromleft
       0-{\XINT_split_fromright #1}%
    \krof
}%
\def\XINT_split_zerosplit .#1\xint_bye#2\xint_bye..{ #1..}%
\def\XINT_split_fromleft
    {\expandafter\XINT_split_fromleft_a\the\numexpr\xint_c_viii-}%
\def\XINT_split_fromleft_a #1%
{%
    \xint_UDsignfork
      #1\XINT_split_fromleft_b
       -{\XINT_split_fromleft_end_a #1}%
    \krof
}%
\def\XINT_split_fromleft_b #1.#2#3#4#5#6#7#8#9%
{%
    \expandafter\XINT_split_fromleft_clean
    \the\numexpr1#2#3#4#5#6#7#8#9\expandafter
    \XINT_split_fromleft_a\the\numexpr\xint_c_viii-#1.%
}%
\def\XINT_split_fromleft_end_a #1.%
{%
    \expandafter\XINT_split_fromleft_clean
    \the\numexpr1\csname XINT_split_fromleft_end#1\endcsname
}%
\def\XINT_split_fromleft_clean 1{ }%
\expandafter\def\csname XINT_split_fromleft_end7\endcsname #1%
   {#1\XINT_split_fromleft_end_b}%
\expandafter\def\csname XINT_split_fromleft_end6\endcsname #1#2%
   {#1#2\XINT_split_fromleft_end_b}%
\expandafter\def\csname XINT_split_fromleft_end5\endcsname #1#2#3%
   {#1#2#3\XINT_split_fromleft_end_b}%
\expandafter\def\csname XINT_split_fromleft_end4\endcsname #1#2#3#4%
```

```
   {#1#2#3#4\XINT_split_fromleft_end_b}%
\expandafter\def\csname XINT_split_fromleft_end3\endcsname #1#2#3#4#5%
   {#1#2#3#4#5\XINT_split_fromleft_end_b}%
\expandafter\def\csname XINT_split_fromleft_end2\endcsname #1#2#3#4#5#6%
   {#1#2#3#4#5#6\XINT_split_fromleft_end_b}%
\expandafter\def\csname XINT_split_fromleft_end1\endcsname #1#2#3#4#5#6#7%
   {#1#2#3#4#5#6#7\XINT_split_fromleft_end_b}%
\expandafter\def\csname XINT_split_fromleft_end0\endcsname #1#2#3#4#5#6#7#8%
   {#1#2#3#4#5#6#7#8\XINT_split_fromleft_end_b}%
```

```
\def\XINT_split_fromleft_end_b #1\xint_bye#2\xint_bye.{.#1}% puis .
\def\XINT_split_fromright #1.#2\xint_bye
{%
    \expandafter\XINT_split_fromright_a
    \the\numexpr#1-\numexpr\XINT_length_loop
    #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
      \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
      \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
    .#2\xint_bye
}%
```

```
\def\XINT_split_fromright_a #1%
{%
    \xint_UDsignfork
      #1\XINT_split_fromleft
       -\XINT_split_fromright_Lempty
    \krof
}%
\def\XINT_split_fromright_Lempty #1.#2\xint_bye#3..{.#2.}%
```

5.8 \xintDecSplitL

```
\def\xintDecSplitL {\romannumeral0\xintdecsplitl }%
\def\xintdecsplitl #1#2%
{%
    \expandafter\XINT_splitl_finish
    \romannumeral0\expandafter\XINT_split_xfork
    \the\numexpr #1\expandafter.\romannumeral`&&@#2%
    \xint_bye2345678\xint_bye..%
}%
\def\XINT_splitl_finish #1.#2.{ #1}%
```

5.9 \xintDecSplitR

```
\def\xintDecSplitR {\romannumeral0\xintdecsplitr }%
\def\xintdecsplitr #1#2%
{%
    \expandafter\XINT_splitr_finish
    \romannumeral0\expandafter\XINT_split_xfork
    \the\numexpr #1\expandafter.\romannumeral`&&@#2%
    \xint_bye2345678\xint_bye..%
}%
\def\XINT_splitr_finish #1.#2.{ #2}%
```

5.10 \xintDSHr

DECIMAL SHIFTS \xintDSH {x}{A}
si x <= 0, fait A -> A.10^(|x|). si x > 0, et A >=0, fait A -> quo(A,10^(x))
si x > 0, et A < 0, fait A -> -quo(-A,10^(x))
(donc pour x > 0 c'est comme DSR itéré x fois)
\xintDSHr donne le `reste' (si x<=0 donne zéro).
  Badly named macros.
  Rewritten for 1.2i, this was old code and \xintDSx has changed interface.

```
\def\xintDSHr {\romannumeral0\xintdshr }%
\def\xintdshr #1#2%
{%
    \expandafter\XINT_dshr_fork\the\numexpr#1\expandafter.\romannumeral`&&@#2;%
}%
\def\XINT_dshr_fork #1%
{%
    \xint_UDzerominusfork
      0#1\XINT_dshr_xzeroorneg
      #1-\XINT_dshr_xzeroorneg
       0-\XINT_dshr_xpositive
    \krof #1%
}%
\def\XINT_dshr_xzeroorneg #1;{ 0}%
\def\XINT_dshr_xpositive
{%
    \expandafter\xint_secondoftwo_thenstop\romannumeral0\XINT_dsx_xisPos
}%
```

5.11 \xintDSH

```
\def\xintDSH {\romannumeral0\xintdsh }%
\def\xintdsh #1#2%
{%
    \expandafter\XINT_dsh_fork\the\numexpr#1\expandafter.\romannumeral`&&@#2;%
}%
\def\XINT_dsh_fork #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_dsh_xiszero
      0#1\XINT_dsx_xisNeg_checkA
       0-{\XINT_dsh_xisPos #1}%
    \krof
}%
\def\XINT_dsh_xiszero #1.#2;{ #2}%
\def\XINT_dsh_xisPos
```

```
{%
  \expandafter\xint_firstoftwo_thenstop\romannumeral0\XINT_dsx_xisPos
}%
```

5.12 \xintDSx

```
--> Attention le cas x=0 est traité dans la même catégorie que x > 0 <--
   si x < 0, fait A -> A.10^(|x|)
   si x >=  0, et A >=0, fait A -> {quo(A,10^(x))}{rem(A,10^(x))}
   si x >=  0, et A < 0, d'abord on calcule {quo(-A,10^(x))}{rem(-A,10^(x))}
      puis, si le premier n'est pas nul on lui donne le signe -
            si le premier est nul on donne le signe - au second.
On peut donc toujours reconstituer l'original A par 10^x Q \pm R où il faut prendre le signe plus
si Q est positif ou nul et le signe moins si Q est strictement négatif.
  Rewritten for 1.2i, this was old code.
```

```
\def\xintDSx {\romannumeral0\xintdsx }%
\def\xintdsx #1#2%
{%
    \expandafter\XINT_dsx_fork\the\numexpr#1\expandafter.\romannumeral`&&@#2;%
}%
\def\XINT_dsx_fork #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_dsx_xisZero
      0#1\XINT_dsx_xisNeg_checkA
       0-{\XINT_dsx_xisPos #1}%
    \krof
}%
\def\XINT_dsx_xisZero #1.#2;{{#2}{0}}%
\def\XINT_dsx_xisNeg_checkA #1.#2%
{%
    \xint_gob_til_zero #2\XINT_dsx_xisNeg_Azero 0%
    \expandafter\XINT_dsx_append\romannumeral\XINT_rep #1\endcsname 0.#2%
}%
\def\XINT_dsx_xisNeg_Azero #1;{ 0}%
\def\XINT_dsx_addzeros #1%
   {\expandafter\XINT_dsx_append\romannumeral\XINT_rep#1\endcsname0.}%
\def\XINT_dsx_addzerosnofuss #1%
   {\expandafter\XINT_dsx_append\romannumeral\xintreplicate{#1}0.}%
\def\XINT_dsx_append #1.#2;{ #2#1}%
\def\XINT_dsx_xisPos #1.#2%
{%
    \xint_UDzerominusfork
      #2-\XINT_dsx_AisZero
      0#2\XINT_dsx_AisNeg
       0-\XINT_dsx_AisPos
    \krof #1.#2%
}%
\def\XINT_dsx_AisZero #1;{{0}{0}}%
\def\XINT_dsx_AisNeg #1.-#2;%
{%
    \expandafter\XINT_dsx_AisNeg_checkiffirstempty
    \romannumeral0\XINT_split_xfork #1.#2\xint_bye2345678\xint_bye..%
}%

\def\XINT_dsx_AisNeg_checkiffirstempty #1%
{%
    \xint_gob_til_dot #1\XINT_dsx_AisNeg_finish_zero.%
    \XINT_dsx_AisNeg_finish_notzero #1%
}%
\def\XINT_dsx_AisNeg_finish_zero.\XINT_dsx_AisNeg_finish_notzero.#1.%
{%
    \expandafter\XINT_dsx_end
    \expandafter {\romannumeral0\XINT_num {-#1}}{0}%
}%
\def\XINT_dsx_AisNeg_finish_notzero #1.#2.%
{%
    \expandafter\XINT_dsx_end
    \expandafter {\romannumeral0\XINT_num {#2}}{-#1}%
}%

\def\XINT_dsx_AisPos #1.#2;%
{%
    \expandafter\XINT_dsx_AisPos_finish
    \romannumeral0\XINT_split_xfork #1.#2\xint_bye2345678\xint_bye..%
}%

\def\XINT_dsx_AisPos_finish #1.#2.%
{%
    \expandafter\XINT_dsx_end
    \expandafter {\romannumeral0\XINT_num {#2}}%
                 {\romannumeral0\XINT_num {#1}}%
}%
\def\XINT_dsx_end #1#2{\expandafter{#2}{#1}}%
```

5.13 \xintiiEq

no \xintiieq.

```
\def\xintiiEq #1#2{\romannumeral0\xintiiifeq{#1}{#2}{1}{0}}%
```

5.14 \xintiiNotEq

Pour xintexpr. Pas de version en lowercase.

```
\def\xintiiNotEq #1#2{\romannumeral0\xintiiifeq {#1}{#2}{0}{1}}%
```

5.15 \xintiiGeq

```
PLUS GRAND OU ÉGAL attention compare les **valeurs absolues**
  1.2l made \xintiiGeq robust against non terminated items.
  1.2l rewrote \xintiiCmp, but forgot to handle \xintiiGeq too. Done at 1.2m.
  This macro should have been called \xintGEq for example.
```

```
\def\xintiiGeq   {\romannumeral0\xintiigeq }%
\def\xintiigeq #1{\expandafter\XINT_iigeq\romannumeral`&&@#1\xint:}%
\def\XINT_iigeq #1#2\xint:#3%
{%
    \expandafter\XINT_geq_fork\expandafter #1\romannumeral`&&@#3\xint:#2\xint:
}%

\def\XINT_geq #1#2\xint:#3%
{%
    \expandafter\XINT_geq_fork\expandafter #1\romannumeral0\xintnum{#3}\xint:#2\xint:
}%
\def\XINT_geq_fork #1#2%
{%
    \xint_UDzerofork
      #1\XINT_geq_firstiszero
      #2\XINT_geq_secondiszero
       0{}%
    \krof
    \xint_UDsignsfork
          #1#2\XINT_geq_minusminus
           #1-\XINT_geq_minusplus
           #2-\XINT_geq_plusminus
            --\XINT_geq_plusplus
    \krof #1#2%
}%
\def\XINT_geq_firstiszero  #1\krof 0#2#3\xint:#4\xint:
                               {\xint_UDzerofork #2{ 1}0{ 0}\krof }%
\def\XINT_geq_secondiszero #1\krof #20#3\xint:#4\xint:{ 1}%
\def\XINT_geq_plusminus    #1-{\XINT_geq_plusplus #1{}}%
\def\XINT_geq_minusplus    -#1{\XINT_geq_plusplus {}#1}%
\def\XINT_geq_minusminus    --{\XINT_geq_plusplus {}{}}%
\def\XINT_geq_plusplus
   {\expandafter\XINT_geq_finish\romannumeral0\XINT_cmp_plusplus}%
\def\XINT_geq_finish #1{\if-#1\expandafter\XINT_geq_no
                          \else\expandafter\XINT_geq_yes\fi}%
\def\XINT_geq_no 1{ 0}%
\def\XINT_geq_yes { 1}%
```

5.16 \xintiiGt

```
\def\xintiiGt #1#2{\romannumeral0\xintiiifgt{#1}{#2}{1}{0}}%
```

5.17 \xintiiLt

```
\def\xintiiLt #1#2{\romannumeral0\xintiiiflt{#1}{#2}{1}{0}}%
```

5.18 \xintiiGtorEq

```
\def\xintiiGtorEq #1#2{\romannumeral0\xintiiiflt {#1}{#2}{0}{1}}%
```

5.19 \xintiiLtorEq

```
\def\xintiiLtorEq #1#2{\romannumeral0\xintiiifgt {#1}{#2}{0}{1}}%
```

5.20 \xintiiIsZero

1.09a. restyled in 1.09i. 1.1 adds \xintiiIsZero, etc... for optimization in \xintexpr

```
\def\xintiiIsZero {\romannumeral0\xintiiiszero }%
\def\xintiiiszero #1{\if0\xintiiSgn{#1}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%
```

5.21 \xintiiIsNotZero

1.09a. restyled in 1.09i. 1.1 adds \xintiiIsZero, etc... for optimization in \xintexpr

```
\def\xintiiIsNotZero {\romannumeral0\xintiiisnotzero }%
\def\xintiiisnotzero
          #1{\if0\xintiiSgn{#1}\xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi}%
```

5.22 \xintiiIsOne

Added in 1.03. 1.09a defines \xintIsOne. 1.1a adds \xintiiIsOne.
  \XINT_isOne rewritten for 1.2g. Works with expanded strict integers, positive or negative.

```
\def\xintiiIsOne {\romannumeral0\xintiiisone }%
\def\xintiiisone #1{\expandafter\XINT_isone\romannumeral`&&@#1XY}%
\def\XINT_isone #1#2#3Y%
{%
    \unless\if#2X\xint_dothis{ 0}\fi
    \unless\if#11\xint_dothis{ 0}\fi
    \xint_orthat{ 1}%
}%
\def\XINT_isOne #1{\XINT_is_One#1XY}%
\def\XINT_is_One #1#2#3Y%
{%
    \unless\if#2X\xint_dothis0\fi
    \unless\if#11\xint_dothis0\fi
    \xint_orthat1%
}%
```

5.23 \xintiiOdd

\xintOdd is needed for the xintexpr-essions even() and odd() functions (and also by \xintNewExpr).

```
\def\xintiiOdd {\romannumeral0\xintiiodd }%
\def\xintiiodd #1%
{%
    \ifodd\xintLDg{#1} %<- intentional space
        \xint_afterfi{ 1}%
    \else
        \xint_afterfi{ 0}%
    \fi
}%
```

5.24 \xintiiEven

```
\def\xintiiEven {\romannumeral0\xintiieven }%
\def\xintiieven #1%
{%
    \ifodd\xintLDg{#1} %<- intentional space
        \xint_afterfi{ 0}%
    \else
        \xint_afterfi{ 1}%
    \fi
}%
```

5.25 \xintiiMON

MINUS ONE TO THE POWER N

```
\def\xintiiMON {\romannumeral0\xintiimon }%
\def\xintiimon #1%
{%
    \ifodd\xintLDg {#1} %<- intentional space
        \xint_afterfi{ -1}%
    \else
        \xint_afterfi{ 1}%
    \fi
}%
```

5.26 \xintiiMMON

MINUS ONE TO THE POWER N-1

```
\def\xintiiMMON {\romannumeral0\xintiimmon }%
\def\xintiimmon #1%
{%
    \ifodd\xintLDg {#1} %<- intentional space
        \xint_afterfi{ 1}%
    \else
        \xint_afterfi{ -1}%
    \fi
}%
```

5.27 \xintSgnFork

Expandable three-way fork added in 1.07. The argument #1 must expand to non-self-ending -1,0 or 1. 1.09i with _thenstop.

```
\def\xintSgnFork {\romannumeral0\xintsgnfork }%
\def\xintsgnfork #1%
{%
    \ifcase #1 \expandafter\xint_secondofthree_thenstop
            \or\expandafter\xint_thirdofthree_thenstop
          \else\expandafter\xint_firstofthree_thenstop
    \fi
}%
```

5.28 \xintiiifSgn

Expandable three-way fork added in 1.09a. Branches expandably depending on whether <0, =0, >0. Choice of branch guaranteed in two steps.

1.09i has \xint_firstofthreeafterstop (now _thenstop) etc for faster expansion.

1.1 adds \xintiiifSgn for optimization in xintexpr-essions. Should I move them to xintcore? (for bnumexpr)

```
\def\xintiiifSgn {\romannumeral0\xintiiifsgn }%
\def\xintiiifsgn #1%
{%
    \ifcase \xintiiSgn{#1}
               \expandafter\xint_secondofthree_thenstop
            \or\expandafter\xint_thirdofthree_thenstop
          \else\expandafter\xint_firstofthree_thenstop
    \fi
}%
```

5.29 \xintiiifCmp

1.09e \xintifCmp {n}{m}{if n<m}{if n=m}{if n>m}. 1.1a adds ii variant

```
\def\xintiiifCmp {\romannumeral0\xintiiifcmp }%
\def\xintiiifcmp #1#2%
{%
    \ifcase\xintiiCmp {#1}{#2}
               \expandafter\xint_secondofthree_thenstop
            \or\expandafter\xint_thirdofthree_thenstop
          \else\expandafter\xint_firstofthree_thenstop
    \fi
}%
```

5.30 \xintiiifEq

1.09a \xintifEq {n}{m}{YES if n=m}{NO if n<>m}. 1.1a adds ii variant

```
\def\xintiiifEq {\romannumeral0\xintiiifeq }%
\def\xintiiifeq #1#2%
{%
    \if0\xintiiCmp{#1}{#2}%
               \expandafter\xint_firstoftwo_thenstop
          \else\expandafter\xint_secondoftwo_thenstop
    \fi
}%
```

5.31 \xintiiifGt

1.09a \xintifGt {n}{m}{YES if n>m}{NO if n<=m}. 1.1a adds ii variant

```
\def\xintiiifGt {\romannumeral0\xintiiifgt }%
\def\xintiiifgt #1#2%
{%
    \if1\xintiiCmp{#1}{#2}%
```

```
                \expandafter\xint_firstoftwo_thenstop
          \else\expandafter\xint_secondoftwo_thenstop
    \fi
}%
```

5.32 \xintiiifLt

1.09a \xintifLt {n}{m}{YES if n<m}{NO if n>=m}. Restyled in 1.09i. 1.1a adds ii variant

```
\def\xintiiifLt {\romannumeral0\xintiiiflt }%
\def\xintiiiflt #1#2%
{%
    \ifnum\xintiiCmp{#1}{#2}<\xint_c_
          \expandafter\xint_firstoftwo_thenstop
    \else \expandafter\xint_secondoftwo_thenstop
    \fi
}%
```

5.33 \xintiiifZero

Expandable two-way fork added in 1.09a. Branches expandably depending on whether the argument is zero (branch A) or not (branch B). 1.09i restyling. By the way it appears (not thoroughly tested, though) that \if tests are faster than \ifnum tests. 1.1 adds ii versions.

1.2o deprecates \xintifZero.

```
\def\xintiiifZero {\romannumeral0\xintiiifzero }%
\def\xintiiifzero #1%
{%
    \if0\xintiiSgn{#1}%
       \expandafter\xint_firstoftwo_thenstop
    \else
       \expandafter\xint_secondoftwo_thenstop
    \fi
}%
```

5.34 \xintiiifNotZero

```
\def\xintiiifNotZero {\romannumeral0\xintiiifnotzero }%
\def\xintiiifnotzero #1%
{%
    \if0\xintiiSgn{#1}%
       \expandafter\xint_secondoftwo_thenstop
    \else
       \expandafter\xint_firstoftwo_thenstop
    \fi
}%
```

5.35 \xintiiifOne

added in 1.09i. 1.1a adds \xintiiifOne.

```
\def\xintiiifOne {\romannumeral0\xintiiifone }%
\def\xintiiifone #1%
{%
```

```
    \if1\xintiiIsOne{#1}%
       \expandafter\xint_firstoftwo_thenstop
    \else
       \expandafter\xint_secondoftwo_thenstop
    \fi
}%
```

5.36 `\xintiiifOdd`

1.09e. Restyled in 1.09i. 1.1a adds `\xintiiifOdd`.

```
\def\xintiiifOdd {\romannumeral0\xintiiifodd }%
\def\xintiiifodd #1%
{%
    \if\xintiiOdd{#1}1%
       \expandafter\xint_firstoftwo_thenstop
    \else
       \expandafter\xint_secondoftwo_thenstop
    \fi
}%
```

5.37 `\xintifTrueAelseB`, `\xintifFalseAelseB`

1.09i. 1.2i has removed deprecated `\xintifTrueFalse`, `\xintifTrue`.

1.2o uses `\xintiiifNotZero`, see comments to `\xintAND` etc... This will work fine with arguments being nested xintfrac.sty macros, without the overhead of `\xintNum` or `\xintRaw` parsing.

```
\def\xintifTrueAelseB {\romannumeral0\xintiiifnotzero}%
\def\xintifFalseAelseB{\romannumeral0\xintiiifzero}%
```

5.38 `\xintIsTrue`, `\xintIsFalse`

1.09c. Suppressed at 1.2o. They seem not to have been documented, fortunately.

```
%\let\xintIsTrue \xintIsNotZero
%\let\xintIsFalse\xintIsZero
```

5.39 `\xintNOT`

1.09c. But it should have been called `\xintNOT`, not `\xintNot`. Former denomination deprecated at 1.2o. Besides, the macro is now defined as ii-type.

```
\def\xintNOT{\romannumeral0\xintiiiszero}%
```

5.40 `\xintAND`, `\xintOR`, `\xintXOR`

Added with 1.09a. But they used `\xintSgn`, etc... rather than `\xintiiSgn`. This brings `\xintNum` overhead which is not really desired, and which is not needed for use by xintexpr.sty. At 1.2o I modify them to use only ii macros. This is enough for sign or zeroness even for xintfrac format, as manipulated inside the `\xintexpr`. Big hesitation whether there should be however `\xintiiAND` outputting 1 or 0 versus an `\xintAND` outputting 1[0] versus 0[0] for example.

```
\def\xintAND {\romannumeral0\xintand }%
```

```
\def\xintand #1#2{\if0\xintiiSgn{#1}\expandafter\xint_firstoftwo
                             \else\expandafter\xint_secondoftwo\fi
                  { 0}{\xintiiisnotzero{#2}}}%
\def\xintOR {\romannumeral0\xintor }%
\def\xintor #1#2{\if0\xintiiSgn{#1}\expandafter\xint_firstoftwo
                            \else\expandafter\xint_secondoftwo\fi
                 {\xintiiisnotzero{#2}}{ 1}}%
\def\xintXOR {\romannumeral0\xintxor }%
\def\xintxor #1#2{\if\xintiiIsZero{#1}\xintiiIsZero{#2}%
                     \xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi }%
```

5.41 \xintANDof

New with 1.09a. \xintANDof works also with an empty list. Empty items however are not accepted.

1.2l made \xintANDof robust against non terminated items.

1.2o's \xintifTrueAelseB is now an ii macro, actually.

This macro as well as ORof and XORof are actually not used by xintexpr, which has its own csv handling macros.

```
\def\xintANDof      {\romannumeral0\xintandof }%
\def\xintandof    #1{\expandafter\XINT_andof_a\romannumeral`&&@#1\xint:}%
\def\XINT_andof_a #1{\expandafter\XINT_andof_b\romannumeral`&&@#1!}%
\def\XINT_andof_b #1%
           {\xint_gob_til_xint: #1\XINT_andof_e\xint:\XINT_andof_c #1}%
\def\XINT_andof_c #1!%
           {\xintifTrueAelseB {#1}{\XINT_andof_a}{\XINT_andof_no}}%
\def\XINT_andof_no #1\xint:{ 0}%
\def\XINT_andof_e  #1!{ 1}%
```

5.42 \xintORof

New with 1.09a. Works also with an empty list. Empty items however are not accepted.

1.2l made \xintORof robust against non terminated items.

```
\def\xintORof      {\romannumeral0\xintorof }%
\def\xintorof    #1{\expandafter\XINT_orof_a\romannumeral`&&@#1\xint:}%
\def\XINT_orof_a #1{\expandafter\XINT_orof_b\romannumeral`&&@#1!}%
\def\XINT_orof_b #1%
           {\xint_gob_til_xint: #1\XINT_orof_e\xint:\XINT_orof_c #1}%
\def\XINT_orof_c #1!%
           {\xintifTrueAelseB {#1}{\XINT_orof_yes}{\XINT_orof_a}}%
\def\XINT_orof_yes #1\xint:{ 1}%
\def\XINT_orof_e   #1!{ 0}%
```

5.43 \xintXORof

New with 1.09a. Works with an empty list, too. Empty items however are not accepted. \XINT_xorof_c more efficient in 1.09i.

1.2l made \xintXORof robust against non terminated items.

```
\def\xintXORof      {\romannumeral0\xintxorof }%
\def\xintxorof    #1{\expandafter\XINT_xorof_a\expandafter
                     0\romannumeral`&&@#1\xint:}%
```

```
\def\XINT_xorof_a #1#2{\expandafter\XINT_xorof_b\romannumeral`&&@#2!#1}%
\def\XINT_xorof_b #1%
           {\xint_gob_til_xint: #1\XINT_xorof_e\xint:\XINT_xorof_c #1}%
\def\XINT_xorof_c #1!#2%
           {\xintifTrueAelseB {#1}{\if #20\xint_afterfi{\XINT_xorof_a 1}%
                                   \else\xint_afterfi{\XINT_xorof_a 0}\fi}%
                                  {\XINT_xorof_a #2}%
           }%
\def\XINT_xorof_e #1!#2{ #2}%
```

5.44 \xintiiMax

At 1.2m, a long-standing bug was fixed: \xintiiMax had the overhead of applying \xintNum to its arguments due to use of a sub-macro of \xintGeq code to which this overhead was added at some point.
  And on this occasion I reduced even more number of times input is grabbed.

```
\def\xintiiMax {\romannumeral0\xintiimax }%
\def\xintiimax #1%
{%
    \expandafter\xint_iimax \romannumeral`&&@#1\xint:
}%
\def\xint_iimax #1\xint:#2%
{%
    \expandafter\XINT_max_fork\romannumeral`&&@#2\xint:#1\xint:
}%
```

#3#4 vient du *premier*, #1#2 vient du *second*. I have renamed the sub-macros at 1.2m because the terminology was quite counter-intuitive; there was no bug, but still.

```
\def\XINT_max_fork #1#2\xint:#3#4\xint:
{%
    \xint_UDsignsfork
          #1#3\XINT_max_minusminus  % A < 0, B < 0
           #1-\XINT_max_plusminus   % B < 0, A >= 0
           #3-\XINT_max_minusplus   % A < 0, B >= 0
            --{\xint_UDzerosfork
                     #1#3\XINT_max_zerozero % A = B = 0
                      #10\XINT_max_pluszero % B = 0, A > 0
                      #30\XINT_max_zeroplus % A = 0, B > 0
                       00\XINT_max_plusplus % A, B > 0
                     \krof }%
    \krof
    #3#1#2\xint:#4\xint:
      \expandafter\xint_firstoftwo_thenstop
    \else
      \expandafter\xint_secondoftwo_thenstop
    \fi
    {#3#4}{#1#2}%
}%
```

Refactored at 1.2m for avoiding grabbing arguments. Position of inputs shared with iiCmp and iiGeq code.

```
\def\XINT_max_zerozero  #1\fi{\xint_firstoftwo_thenstop }%
```

```
\def\XINT_max_zeroplus  #1\fi{\xint_secondoftwo_thenstop }%
\def\XINT_max_pluszero  #1\fi{\xint_firstoftwo_thenstop }%
\def\XINT_max_minusplus #1\fi{\xint_secondoftwo_thenstop }%
\def\XINT_max_plusminus #1\fi{\xint_firstoftwo_thenstop }%
\def\XINT_max_plusplus
{%
    \if1\romannumeral0\XINT_geq_plusplus
}%
```

Premier des testés |A|=-A, second est |B|=-B. On veut le max(A,B), c'est donc A si |A|<|B| (ou |A|=|B|, mais peu importe alors). Donc on peut faire cela avec \unless. Simple.

```
\def\XINT_max_minusminus --%
{%
    \unless\if1\romannumeral0\XINT_geq_plusplus{}{}%
}%
```

5.45 \xintiiMin

\xintnum added New with 1.09a. I add \xintiiMin in 1.1 and mark as deprecated \xintMin, renamed \xintiMin. \xintMin NOW REMOVED (1.2, as \xintMax, \xintMaxof), only provided by \xintfracnameimp.

At 1.2m, a long-standing bug was fixed: \xintiiMin had the overhead of applying \xintNum to its arguments due to use of a sub-macro of \xintGeq code to which this overhead was added at some point.

And on this occasion I reduced even more number of times input is grabbed.

```
\def\xintiiMin {\romannumeral0\xintiimin }%
\def\xintiimin #1%
{%
    \expandafter\xint_iimin \romannumeral`&&@#1\xint:
}%
\def\xint_iimin #1\xint:#2%
{%
    \expandafter\XINT_min_fork\romannumeral`&&@#2\xint:#1\xint:
}%
\def\XINT_min_fork #1#2\xint:#3#4\xint:
{%
    \xint_UDsignsfork
          #1#3\XINT_min_minusminus  % A < 0, B < 0
           #1-\XINT_min_plusminus   % B < 0, A >= 0
           #3-\XINT_min_minusplus   % A < 0, B >= 0
            --{\xint_UDzerosfork
                      #1#3\XINT_min_zerozero % A = B = 0
                       #10\XINT_min_pluszero % B = 0, A > 0
                       #30\XINT_min_zeroplus % A = 0, B > 0
                        00\XINT_min_plusplus % A, B > 0
                      \krof }%
    \krof
    #3#1#2\xint:#4\xint:
      \expandafter\xint_secondoftwo_thenstop
    \else
      \expandafter\xint_firstoftwo_thenstop
    \fi
    {#3#4}{#1#2}%
}%
\def\XINT_min_zerozero  #1\fi{\xint_firstoftwo_thenstop }%
\def\XINT_min_zeroplus  #1\fi{\xint_firstoftwo_thenstop }%
\def\XINT_min_pluszero  #1\fi{\xint_secondoftwo_thenstop }%
\def\XINT_min_minusplus #1\fi{\xint_firstoftwo_thenstop }%
\def\XINT_min_plusminus #1\fi{\xint_secondoftwo_thenstop }%
\def\XINT_min_plusplus
{%
    \if1\romannumeral0\XINT_geq_plusplus
}%
\def\XINT_min_minusminus --%
{%
    \unless\if1\romannumeral0\XINT_geq_plusplus{}{}%
}%
```

5.46 \xintiiMaxof

New with 1.09a. 1.2 has NO MORE \xintMaxof, requires \xintfracname. 1.2a adds \xintiiMaxof, as \xintiiMaxof:csv is not public.
  NOT compatible with empty list.
  1.2l made \xintiiMaxof robust against non terminated items.

```
\def\xintiiMaxof      {\romannumeral0\xintiimaxof }%
\def\xintiimaxof    #1{\expandafter\XINT_iimaxof_a\romannumeral`&&@#1\xint:}%
\def\XINT_iimaxof_a #1{\expandafter\XINT_iimaxof_b\romannumeral`&&@#1!}%
\def\XINT_iimaxof_b #1!#2%
           {\expandafter\XINT_iimaxof_c\romannumeral`&&@#2!{#1}!}%
\def\XINT_iimaxof_c #1%
           {\xint_gob_til_xint: #1\XINT_iimaxof_e\xint:\XINT_iimaxof_d #1}%
\def\XINT_iimaxof_d #1!%
           {\expandafter\XINT_iimaxof_b\romannumeral0\xintiimax {#1}}%
\def\XINT_iimaxof_e #1!#2!{ #2}%
```

5.47 \xintiiMinof

1.09a. 1.2a adds \xintiiMinof which was lacking.

```
\def\xintiiMinof      {\romannumeral0\xintiiminof }%
\def\xintiiminof    #1{\expandafter\XINT_iiminof_a\romannumeral`&&@#1\xint:}%
\def\XINT_iiminof_a #1{\expandafter\XINT_iiminof_b\romannumeral`&&@#1!}%
\def\XINT_iiminof_b #1!#2%
           {\expandafter\XINT_iiminof_c\romannumeral`&&@#2!{#1}!}%
\def\XINT_iiminof_c #1%
           {\xint_gob_til_xint: #1\XINT_iiminof_e\xint:\XINT_iiminof_d #1}%
\def\XINT_iiminof_d #1!%
           {\expandafter\XINT_iiminof_b\romannumeral0\xintiimin {#1}}%
\def\XINT_iiminof_e #1!#2!{ #2}%
```

5.48 \xintiiSum

\xintiiSum {{a}{b}...{z}}

```
\def\xintiiSum {\romannumeral0\xintiisum }%
\def\xintiisum #1{\expandafter\XINT_sumexpr\romannumeral`&&@#1\xint:}%
\def\XINT_sumexpr {\XINT_sum_loop_a 0\Z }%
\def\XINT_sum_loop_a #1\Z #2%
    {\expandafter\XINT_sum_loop_b \romannumeral`&&@#2\xint:#1\xint:\Z}%
\def\XINT_sum_loop_b #1%
    {\xint_gob_til_xint: #1\XINT_sum_finished\xint:\XINT_sum_loop_c #1}%
\def\XINT_sum_loop_c
    {\expandafter\XINT_sum_loop_a\romannumeral0\XINT_add_fork }%
\def\XINT_sum_finished\xint:\XINT_sum_loop_c\xint:\xint:#1\xint:\Z{ #1}%
```

5.49 \xintiiPrd

\xintiiPrd {{a}...{z}}

```
\def\xintiiPrd {\romannumeral0\xintiiprd }%
\def\xintiiprd #1{\expandafter\XINT_prdexpr\romannumeral`&&@#1\xint:}%
\def\XINT_prdexpr {\XINT_prod_loop_a 1\Z }%
\def\XINT_prod_loop_a #1\Z #2%
    {\expandafter\XINT_prod_loop_b\romannumeral`&&@#2\xint:#1\xint:\Z}%
\def\XINT_prod_loop_b #1%
    {\xint_gob_til_xint: #1\XINT_prod_finished\xint:\XINT_prod_loop_c #1}%
\def\XINT_prod_loop_c
    {\expandafter\XINT_prod_loop_a\romannumeral0\XINT_mul_fork }%
\def\XINT_prod_finished\xint:\XINT_prod_loop_c\xint:\xint:#1\xint:\Z { #1}%
```

5.50 \xintiiSquareRoot

First done with 1.08.

1.1 added \xintiiSquareRoot.

1.1a added \xintiiSqrtR.

1.2f (2016/03/01-02-03) has rewritten the implementation, the underlying mathematics remaining about the same. The routine is much faster for inputs having up to 16 digits (because it does it all with \numexpr directly now), and also much faster for very long inputs (because it now fetches only the needed new digits after the first 16 (or 17) ones, via the geometric sequence 16, then 32, then 64, etc...; earlier version did the computations with all remaining digits after a suitable starting point with correct 4 or 5 leading digits). Note however that the fetching of tokens is via intrinsically O(N^2) macros, hence inevitably inputs with thousands of digits start being treated less well.

Actually there is some room for improvements, one could prepare better input X for the upcoming treatment of fetching its digits by 16, then 32, then 64, etc...

Incidently, as \xintiiSqrt uses subtraction and subtraction was broken from 1.2 to 1.2c, then for another reason from 1.2c to 1.2f, it could get wrong in certain (relatively rare) cases. There was also a bug that made it unneedlessly slow for odd number of digits on input.

1.2f also modifies \xintFloatSqrt in xintfrac.sty which now has more code in common with here and benefits from the same speed improvements.

1.2k belatedly corrects the output to {1}{1} and not 11 when input is zero. As braces are used in all other cases they should have been used here too.

Also, 1.2k adds an \xintiSqrtR macro, for coherence as \xintiSqrt is defined (and mentioned in user manual.)

```
\def\xintiiSquareRoot {\romannumeral0\xintiisquareroot }%
\def\xintiisquareroot #1{\expandafter\XINT_sqrt_checkin\romannumeral`&&@#1\xint:}%
```

```
\def\XINT_sqrt_checkin #1%
{%
    \xint_UDzerominusfork
     #1-\XINT_sqrt_iszero
     0#1\XINT_sqrt_isneg
      0-\XINT_sqrt
    \krof #1%
}%
\def\XINT_sqrt_iszero #1\xint:{{1}{1}}%
\def\XINT_sqrt_isneg  #1\xint:{\XINT_signalcondition{InvalidOperation}{square
    root of negative: #1}{}{{0}{0}}}%
\def\XINT_sqrt #1\xint:
{%
    \expandafter\XINT_sqrt_start\romannumeral0\xintlength {#1}.#1.%
}%
\def\XINT_sqrt_start #1.%
{%
    \ifnum #1<\xint_c_x\xint_dothis\XINT_sqrt_small_a\fi
    \xint_orthat\XINT_sqrt_big_a #1.%
}%
\def\XINT_sqrt_small_a #1.{\XINT_sqrt_a #1.\XINT_sqrt_small_d }%
\def\XINT_sqrt_big_a   #1.{\XINT_sqrt_a #1.\XINT_sqrt_big_d   }%
\def\XINT_sqrt_a #1.%
{%
   \ifodd #1
     \expandafter\XINT_sqrt_bO
   \else
     \expandafter\XINT_sqrt_bE
   \fi
   #1.%
}%

\def\XINT_sqrt_bE #1.#2#3#4%
{%
    \XINT_sqrt_c {#3#4}#2{#1}#3#4%
}%

\def\XINT_sqrt_bO #1.#2#3%
{%
    \XINT_sqrt_c #3#2{#1}#3%
}%

\def\XINT_sqrt_c #1#2%
{%
    \expandafter #2%
    \the\numexpr \ifnum #1>\xint_c_ii
                 \ifnum #1>\xint_c_vi
                 \ifnum #1>12 \ifnum #1>20 \ifnum #1>30
                 \ifnum #1>42 \ifnum #1>56 \ifnum #1>72
                 \ifnum #1>90
      10\else 9\fi \else 8\fi \else 7\fi \else 6\fi \else 5\fi
```

```
        \else 4\fi \else 3\fi \else 2\fi \else 1\fi .%
}%
\def\XINT_sqrt_small_d #1.#2%
{%
   \expandafter\XINT_sqrt_small_e
   \the\numexpr #1\ifcase \numexpr #2/\xint_c_ii-\xint_c_i\relax
                   \or 0\or 00\or 000\or 0000\fi .%
}%
\def\XINT_sqrt_small_e #1.#2.%
{%
   \expandafter\XINT_sqrt_small_ea\the\numexpr #1*#1-#2.#1.%
}%
\def\XINT_sqrt_small_ea #1%
{%
    \if0#1\xint_dothis\XINT_sqrt_small_ez\fi
    \if-#1\xint_dothis\XINT_sqrt_small_eb\fi
    \xint_orthat\XINT_sqrt_small_f #1%
}%
\def\XINT_sqrt_small_ez 0.#1.{\expandafter{\the\numexpr#1+\xint_c_i
         \expandafter}\expandafter{\the\numexpr #1*\xint_c_ii+\xint_c_i}}%
\def\XINT_sqrt_small_eb -#1.#2.%
{%
    \expandafter\XINT_sqrt_small_ec \the\numexpr
    (#1-\xint_c_i+#2)/(\xint_c_ii*#2).#1.#2.%
}%
\def\XINT_sqrt_small_ec #1.#2.#3.%
{%
    \expandafter\XINT_sqrt_small_f \the\numexpr
      -#2+\xint_c_ii*#3*#1+#1*#1\expandafter.\the\numexpr #3+#1.%
}%
\def\XINT_sqrt_small_f #1.#2.%
{%
   \expandafter\XINT_sqrt_small_g
   \the\numexpr (#1+#2)/(\xint_c_ii*#2)-\xint_c_i.#1.#2.%
}%
\def\XINT_sqrt_small_g #1#2.%
{%
    \if 0#1%
       \expandafter\XINT_sqrt_small_end
    \else
       \expandafter\XINT_sqrt_small_h
    \fi
    #1#2.%
}%
```

```
\def\XINT_sqrt_small_h #1.#2.#3.%
{%
    \expandafter\XINT_sqrt_small_f
    \the\numexpr #2-\xint_c_ii*#1*#3+#1*#1\expandafter.%
    \the\numexpr #3-#1.%
}%
\def\XINT_sqrt_small_end #1.#2.#3.{{#3}{#2}}%

\def\XINT_sqrt_big_d #1.#2%
{%
   \ifodd #2 \xint_dothis{\expandafter\XINT_sqrt_big_eO}\fi
   \xint_orthat{\expandafter\XINT_sqrt_big_eE}%

   \the\numexpr (#2-\xint_c_i)/\xint_c_ii.#1;%
}%

\def\XINT_sqrt_big_eE  #1;#2#3#4#5#6#7#8#9%
{%
    \XINT_sqrt_big_eE_a #1;{#2#3#4#5#6#7#8#9}%
}%

\def\XINT_sqrt_big_eE_a #1.#2;#3%
{%
    \expandafter\XINT_sqrt_bigormed_f
    \romannumeral0\XINT_sqrt_small_e #2000.#3.#1;%
}%

\def\XINT_sqrt_big_eO #1;#2#3#4#5#6#7#8#9%
{%
    \XINT_sqrt_big_eO_a #1;{#2#3#4#5#6#7#8#9}%
}%
\def\XINT_sqrt_big_eO_a #1.#2;#3#4%
{%
    \expandafter\XINT_sqrt_bigormed_f
    \romannumeral0\XINT_sqrt_small_e #20000.#3#4.#1;%
}%

\def\XINT_sqrt_bigormed_f #1#2#3;%
{%
    \ifnum#3<\xint_c_ix
          \xint_dothis {\csname XINT_sqrt_med_f\romannumeral#3\endcsname}%
    \fi
    \xint_orthat\XINT_sqrt_big_f #1.#2.#3;%
}%
\def\XINT_sqrt_med_fv   {\XINT_sqrt_med_fa .}%
\def\XINT_sqrt_med_fvi  {\XINT_sqrt_med_fa 0.}%
\def\XINT_sqrt_med_fvii {\XINT_sqrt_med_fa 00.}%
\def\XINT_sqrt_med_fviii{\XINT_sqrt_med_fa 000.}%
```

```
\def\XINT_sqrt_med_fa #1.#2.#3.#4;%
{%
    \expandafter\XINT_sqrt_med_fb
    \the\numexpr (#30#1-5#1)/(\xint_c_ii*#2).#1.#2.#3.%
}%

\def\XINT_sqrt_med_fb #1.#2.#3.#4.#5.%
{%
    \expandafter\XINT_sqrt_small_ea
    \the\numexpr (#40#2-\xint_c_ii*#3*#1)*10#2+(#1*#1-#5)\expandafter.%
    \the\numexpr #30#2-#1.%
}%

\def\XINT_sqrt_big_f #1;#2#3#4#5#6#7#8#9%
{%
    \XINT_sqrt_big_fa #1;{#2#3#4#5#6#7#8#9}%
}%

\def\XINT_sqrt_big_fa #1.#2.#3;#4%
{%
    \expandafter\XINT_sqrt_big_ga
    \the\numexpr #3-\xint_c_viii\expandafter.%
    \romannumeral0\XINT_sqrt_med_fa 000.#1.#2.;#4.%
}%

\def\XINT_sqrt_big_ga #1.#2#3%
{%
    \ifnum #1>\xint_c_viii
      \expandafter\XINT_sqrt_big_gb\else
      \expandafter\XINT_sqrt_big_ka
    \fi #1.#3.#2.%
}%

\def\XINT_sqrt_big_gb #1.#2.#3.%
{%
    \expandafter\XINT_sqrt_big_gc
    \the\numexpr (\xint_c_ii*#2-\xint_c_i)*\xint_c_x^viii/(\xint_c_iv*#3).%
    #3.#2.#1;%
}%

\def\XINT_sqrt_big_gc #1.#2.#3.%
{%
    \expandafter\XINT_sqrt_big_gd
    \romannumeral0\xintiiadd
        {\xintiiSub {#300000000}{\xintDouble{\xintiiMul{#2}{#1}}}00000000}%
        {\xintiiSqr {#1}}.%
    \romannumeral0\xintiisub{#200000000}{#1}.%
}%
```

```
\def\XINT_sqrt_big_gd #1.#2.%
{%
    \expandafter\XINT_sqrt_big_ge #2.#1.%
}%

\def\XINT_sqrt_big_ge #1;#2#3#4#5#6#7#8#9%
   {\XINT_sqrt_big_gf #1.#2#3#4#5#6#7#8#9;}%
\def\XINT_sqrt_big_gf #1;#2#3#4#5#6#7#8#9%
   {\XINT_sqrt_big_gg #1#2#3#4#5#6#7#8#9.}%

\def\XINT_sqrt_big_gg #1.#2.#3.#4.%
{%
    \expandafter\XINT_sqrt_big_gloop
    \expandafter\xint_c_xvi\expandafter.%
    \the\numexpr #3-\xint_c_viii\expandafter.%
    \romannumeral0\xintiisub {#2}{\xintiNum{#4}}.#1.%
}%

\def\XINT_sqrt_big_gloop #1.#2.%
{%
    \unless\ifnum #1<#2 \xint_dothis\XINT_sqrt_big_ka \fi
    \xint_orthat{\XINT_sqrt_big_gi #1.}#2.%
}%

\def\XINT_sqrt_big_gi #1.%
{%
    \expandafter\XINT_sqrt_big_gj\romannumeral\xintreplicate{#1}0.#1.%
}%

\def\XINT_sqrt_big_gj #1.#2.#3.#4.#5.%
{%
    \expandafter\XINT_sqrt_big_gk
    \romannumeral0\xintiidivision {#4#1}%
                  {\XINT_dbl #5\xint_bye2345678\xint_bye*\xint_c_ii\relax}.%
    #1.#5.#2.#3.%
}%

\def\XINT_sqrt_big_gk #1#2.#3.#4.%
{%
    \expandafter\XINT_sqrt_big_gl
    \romannumeral0\xintiiadd {#2#3}{\xintiiSqr{#1}}.%
    \romannumeral0\xintiisub {#4#3}{#1}.%
}%

\def\XINT_sqrt_big_gl #1.#2.%
{%
    \expandafter\XINT_sqrt_big_gm #2.#1.%
}%
```

```
\def\XINT_sqrt_big_gm #1.#2.#3.#4.#5.%
{%
    \expandafter\XINT_sqrt_big_gn

    \romannumeral0\XINT_split_fromleft\xint_c_ii*#3.#5\xint_bye2345678\xint_bye..%
    #1.#2.#3.#4.%
}%

\def\XINT_sqrt_big_gn #1.#2.#3.#4.#5.#6.%
{%
    \expandafter\XINT_sqrt_big_gloop
    \the\numexpr \xint_c_ii*#5\expandafter.%
    \the\numexpr #6-#5\expandafter.%
    \romannumeral0\xintiisub{#4}{\xintiNum{#1}}.#3.#2.%
}%

\def\XINT_sqrt_big_ka #1.#2.#3.#4.%
{%
    \expandafter\XINT_sqrt_big_kb

    \romannumeral0\XINT_dsx_addzeros {#1}#3;.%
    \romannumeral0\xintiisub
      {\XINT_dsx_addzerosnofuss {\xint_c_ii*#1}#2;}%
      {\xintiNum{#4}}.%
}%
\def\XINT_sqrt_big_kb #1.#2.%
{%
    \expandafter\XINT_sqrt_big_kc #2.#1.%
}%

\def\XINT_sqrt_big_kc #1%
{%
    \if0#1\xint_dothis\XINT_sqrt_big_kz\fi
    \xint_orthat\XINT_sqrt_big_kloop #1%
}%
\def\XINT_sqrt_big_kz 0.#1.%
{%
    \expandafter\XINT_sqrt_big_kend
    \romannumeral0%
    \xintinc{\XINT_dbl#1\xint_bye2345678\xint_bye*\xint_c_ii\relax}.#1.%
}%
\def\XINT_sqrt_big_kend #1.#2.%
{%
    \expandafter{\romannumeral0\xintinc{#2}}{#1}%
}%

\def\XINT_sqrt_big_kloop #1.#2.%
{%
    \expandafter\XINT_sqrt_big_ke
    \romannumeral0\xintiidivision{#1}%
     {\romannumeral0\XINT_dbl #2\xint_bye2345678\xint_bye*\xint_c_ii\relax}{#2}%
}%
```

```
\def\XINT_sqrt_big_ke #1%
{%
    \if0\XINT_Sgn #1\xint:
          \expandafter \XINT_sqrt_big_end
    \else \expandafter \XINT_sqrt_big_kf
    \fi {#1}%
}%

\def\XINT_sqrt_big_kf #1#2#3%
{%
    \expandafter\XINT_sqrt_big_kg
    \romannumeral0\xintiisub {#3}{#1}.%
    \romannumeral0\xintiiadd {#2}{\xintiiSqr {#1}}.%
}%
\def\XINT_sqrt_big_kg #1.#2.%
{%
   \expandafter\XINT_sqrt_big_kloop #2.#1.%
}%

\def\XINT_sqrt_big_end #1#2#3{{#3}{#2}}%
```

5.51 \xintiiSqrt, \xintiiSqrtR

```
\def\xintiiSqrt  {\romannumeral0\xintiisqrt  }%
\def\xintiisqrt  {\expandafter\XINT_sqrt_post\romannumeral0\xintiisquareroot  }%
\def\XINT_sqrt_post #1#2{\XINT_dec #1\XINT_dec_bye234567890\xint_bye}%
\def\xintiiSqrtR {\romannumeral0\xintiisqrtr }%
\def\xintiisqrtr {\expandafter\XINT_sqrtr_post\romannumeral0\xintiisquareroot }%
```

N = (#1)^2 - #2 avec #1 le plus petit possible et #2>0 (hence #2<2*#1). (#1-.5)^2=#1^2-#1+.25=N+#2-#1+.25. Si 0<#2<#1, <= N-0.75<N, donc rounded->#1 si #2>=#1, (#1-.5)^2>=N+.25>N, donc rounded->#1-1.

```
\def\XINT_sqrtr_post #1#2%
   {\xintiiifLt {#2}{#1}{ #1}{\XINT_dec #1\XINT_dec_bye234567890\xint_bye}}%
```

5.52 \xintiiBinomial

2015/11/28-29 for 1.2f.

2016/11/19 for 1.2h: I truly can't understand why I hard-coded last year an error-message for arguments outside of the range for binomial formula. Naturally there should be no error but a rather a 0 return value for binomial(x,y), if y<0 or x<y !

I really lack some kind of infinity or NaN value.

1.2o deprecates \xintiBinomial. (which xintfrac.sty redefined to use \xintNum)

```
\def\xintiiBinomial {\romannumeral0\xintiibinomial }%
\def\xintiibinomial #1#2%
{%
    \expandafter\XINT_binom_pre\the\numexpr #1\expandafter.\the\numexpr #2.%
}%
\def\XINT_binom_pre #1.#2.%
{%
    \expandafter\XINT_binom_fork \the\numexpr#1-#2.#2.#1.%
}%
```

k.x-k.x. I hesitated to restrict maximal allowed value of x to 10000. Finally I don't. But due to using small multiplication and small division, x must have at most eight digits. If x>=2^31 an arithmetic overflow error will have happened already.

```
\def\XINT_binom_fork #1#2.#3#4.#5#6.%
{%
    \if-#5\xint_dothis{\XINT_signalcondition{InvalidOperation}{Binomial with
        negative first arg: #5#6}{}{0}}\fi
    \if-#1\xint_dothis{ 0}\fi
    \if-#3\xint_dothis{ 0}\fi
    \if0#1\xint_dothis{ 1}\fi
    \if0#3\xint_dothis{ 1}\fi
    \ifnum #5#6>\xint_c_x^viii_mone\xint_dothis
       {\XINT_signalcondition{InvalidOperation}{Binomial with too
           large argument: 99999999 < #5#6}{}{0}}\fi
    \ifnum #1#2>#3#4  \xint_dothis{\XINT_binom_a #1#2.#3#4.}\fi
                      \xint_orthat{\XINT_binom_a #3#4.#1#2.}%
}%
```

x-k.k. avec 0<k<x, k<=x-k. Les divisions produiront en extra après le quotient un terminateur 1!\Z!0!. On va procéder par petite multiplication suivie par petite division. Donc ici on met le 1!\Z!0! pour amorcer.

Le \xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax est le terminateur pour le \XINT_unsep_cuzsmall final.

```
\def\XINT_binom_a #1.#2.%
{%
    \expandafter\XINT_binom_b\the\numexpr \xint_c_i+#1.1.#2.100000001!1!;!0!%
}%
```

y=x-k+1.j=1.k. On va évaluer par y/1*(y+1)/2*(y+2)/3 etc... On essaie de regrouper de manière à utiliser au mieux \numexpr. On peut aller jusqu'à x=10000 car 9999*10000<10^8. 463*464*465=99896880, 98*99*100*101=97990200. On va vérifier à chaque étape si on dépasse un seuil. Le style de l'implémentation diffère de celui que j'avais utilisé pour \xintiiFac. On pourrait tout-à-fait avoir une verybigloop, mais bon. Je rajoute aussi un verysmall. Le traitement est un peu différent pour elle afin d'aller jusqu'à x=29 (et pas seulement 26 si je suivais le modèle des autres, mais je veux pouvoir faire binomial(29,1), binomial(29,2), ... en vsmall).

```
\def\XINT_binom_b #1.%
{%
    \ifnum #1>9999 \xint_dothis\XINT_binom_vbigloop \fi
    \ifnum #1>463  \xint_dothis\XINT_binom_bigloop   \fi
    \ifnum #1>98   \xint_dothis\XINT_binom_medloop   \fi
    \ifnum #1>29   \xint_dothis\XINT_binom_smallloop \fi
                   \xint_orthat\XINT_binom_vsmallloop #1.%
}%
```

y.j.k. Au départ on avait x-k+1.1.k. Ensuite on a des blocs 1<8d>! donnant le résultat intermédiaire, dans l'ordre, et à la fin on a 1!1;!0!. Dans smallloop on peut prendre 4 par 4.

```
\def\XINT_binom_smallloop #1.#2.#3.%
{%
    \ifcase\numexpr #3-#2\relax
        \expandafter\XINT_binom_end_
    \or \expandafter\XINT_binom_end_i
    \or \expandafter\XINT_binom_end_ii
    \or \expandafter\XINT_binom_end_iii
    \else\expandafter\XINT_binom_smallloop_a
    \fi #1.#2.#3.%
}%
```

Ça m'ennuie un peu de reprendre les #1, #2, #3 ici. On a besoin de \numexpr pour \XINT_binom_div, mais de \romannumeral0 pour le unsep après \XINT_binom_mul.

```
\def\XINT_binom_smallloop_a #1.#2.#3.%
{%
    \expandafter\XINT_binom_smallloop_b
    \the\numexpr #1+\xint_c_iv\expandafter.%
    \the\numexpr #2+\xint_c_iv\expandafter.%
    \the\numexpr #3\expandafter.%
    \the\numexpr\expandafter\XINT_binom_div
    \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
    !\romannumeral0\expandafter\XINT_binom_mul
    \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
}%
\def\XINT_binom_smallloop_b #1.%
{%
    \ifnum #1>98  \expandafter\XINT_binom_medloop   \else
                  \expandafter\XINT_binom_smallloop \fi #1.%
}%
```

Ici on prend trois par trois.

```
\def\XINT_binom_medloop #1.#2.#3.%
{%
    \ifcase\numexpr #3-#2\relax
        \expandafter\XINT_binom_end_
    \or \expandafter\XINT_binom_end_i
    \or \expandafter\XINT_binom_end_ii
    \else\expandafter\XINT_binom_medloop_a
    \fi #1.#2.#3.%
}%
\def\XINT_binom_medloop_a #1.#2.#3.%
{%
    \expandafter\XINT_binom_medloop_b
    \the\numexpr #1+\xint_c_iii\expandafter.%
    \the\numexpr #2+\xint_c_iii\expandafter.%
    \the\numexpr #3\expandafter.%
    \the\numexpr\expandafter\XINT_binom_div
        \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
    !\romannumeral0\expandafter\XINT_binom_mul
        \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
}%
\def\XINT_binom_medloop_b #1.%
{%
    \ifnum #1>463 \expandafter\XINT_binom_bigloop   \else
                  \expandafter\XINT_binom_medloop   \fi #1.%
}%
```

Ici on prend deux par deux.

```
\def\XINT_binom_bigloop #1.#2.#3.%
{%
    \ifcase\numexpr #3-#2\relax
        \expandafter\XINT_binom_end_
    \or \expandafter\XINT_binom_end_i
    \else\expandafter\XINT_binom_bigloop_a
    \fi #1.#2.#3.%
}%
\def\XINT_binom_bigloop_a #1.#2.#3.%
{%
    \expandafter\XINT_binom_bigloop_b
    \the\numexpr #1+\xint_c_ii\expandafter.%
    \the\numexpr #2+\xint_c_ii\expandafter.%
    \the\numexpr #3\expandafter.%
    \the\numexpr\expandafter\XINT_binom_div
        \the\numexpr #2*(#2+\xint_c_i)\expandafter
    !\romannumeral0\expandafter\XINT_binom_mul
        \the\numexpr #1*(#1+\xint_c_i)!%
}%
\def\XINT_binom_bigloop_b #1.%
{%
    \ifnum #1>9999 \expandafter\XINT_binom_vbigloop  \else
                   \expandafter\XINT_binom_bigloop   \fi #1.%
}%
```

Et finalement un par un.

```
\def\XINT_binom_vbigloop #1.#2.#3.%
{%
    \ifnum #3=#2
         \expandafter\XINT_binom_end_
    \else\expandafter\XINT_binom_vbigloop_a
    \fi #1.#2.#3.%
}%
\def\XINT_binom_vbigloop_a #1.#2.#3.%
{%
    \expandafter\XINT_binom_vbigloop
    \the\numexpr #1+\xint_c_i\expandafter.%
    \the\numexpr #2+\xint_c_i\expandafter.%
    \the\numexpr #3\expandafter.%
    \the\numexpr\expandafter\XINT_binom_div\the\numexpr #2\expandafter
    !\romannumeral0\XINT_binom_mul #1!%
}%
```

y.j.k. La partie very small. y est au plus 26 (non 29 mais retesté dans \XINT_binom_vsmallloop_a), et tous les binomial(29,n) sont <10^8. On peut donc faire y(y+1)(y+2)(y+3) et aussi il y a le fait que etex fait a*b/c en double precision. Pour ne pas bifurquer à la fin sur smallloop, si n=27, 27, ou 29 on procède un peu différemment des autres boucles. Si je testais aussi #1 après #3-#2 pour les autres il faudrait des terminaisons différentes.

```
\def\XINT_binom_vsmallloop #1.#2.#3.%
{%
```

```
    \ifcase\numexpr #3-#2\relax
        \expandafter\XINT_binom_vsmallend_
    \or \expandafter\XINT_binom_vsmallend_i
    \or \expandafter\XINT_binom_vsmallend_ii
    \or \expandafter\XINT_binom_vsmallend_iii
    \else\expandafter\XINT_binom_vsmallloop_a
    \fi #1.#2.#3.%
}%
\def\XINT_binom_vsmallloop_a #1.%
{%
    \ifnum #1>26  \expandafter\XINT_binom_smallloop_a  \else
                  \expandafter\XINT_binom_vsmallloop_b \fi #1.%
}%
\def\XINT_binom_vsmallloop_b #1.#2.#3.%
{%
    \expandafter\XINT_binom_vsmallloop
    \the\numexpr #1+\xint_c_iv\expandafter.%
    \the\numexpr #2+\xint_c_iv\expandafter.%
    \the\numexpr #3\expandafter.%
    \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
    \the\numexpr  #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
    !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
}%
```

```
\def\XINT_binom_mul #1!#21!;!0!%
{%
    \expandafter\XINT_rev_nounsep\expandafter{\expandafter}%
    \the\numexpr\expandafter\XINT_smallmul
    \the\numexpr\xint_c_x^viii+#1\expandafter
    !\romannumeral0\XINT_rev_nounsep {}1;!#2%
    \R!\R!\R!\R!\R!\R!\R!\R!\W
    \R!\R!\R!\R!\R!\R!\R!\R!\W
    1;!%
}%
\def\XINT_binom_div #1!1;!%
{%
    \expandafter\XINT_smalldivx_a
    \the\numexpr #1/\xint_c_ii\expandafter\xint:
    \the\numexpr \xint_c_x^viii+#1!%
}%
```

Vaguement envisagé d'éviter le 10^8+ mais bon.

```
\def\XINT_binom_vsmallmuldiv #1!#2!1#3!{\xint_c_x^viii+#2*#3/#1!}%
```

On a des terminaisons communes aux trois situations small, med, big, et on est sûr de pouvoir faire les multiplications dans \numexpr, car on vient ici *après* avoir comparé à 9999 ou 463 ou 98.

```
\def\XINT_binom_end_iii #1.#2.#3.%
{%
    \expandafter\XINT_binom_finish
    \the\numexpr\expandafter\XINT_binom_div
        \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
    !\romannumeral0\expandafter\XINT_binom_mul
        \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
}%
\def\XINT_binom_end_ii #1.#2.#3.%
{%
    \expandafter\XINT_binom_finish
    \the\numexpr\expandafter\XINT_binom_div
        \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
    !\romannumeral0\expandafter\XINT_binom_mul
        \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
}%
\def\XINT_binom_end_i #1.#2.#3.%
{%
    \expandafter\XINT_binom_finish
    \the\numexpr\expandafter\XINT_binom_div
        \the\numexpr #2*(#2+\xint_c_i)\expandafter
    !\romannumeral0\expandafter\XINT_binom_mul
        \the\numexpr #1*(#1+\xint_c_i)!%
}%
\def\XINT_binom_end_ #1.#2.#3.%
{%
    \expandafter\XINT_binom_finish
    \the\numexpr\expandafter\XINT_binom_div\the\numexpr #2\expandafter
    !\romannumeral0\XINT_binom_mul #1!%
}%
\def\XINT_binom_finish #1;!0!%
   {\XINT_unsep_cuzsmall #1\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax}%
```

Duplication de code seulement pour la boucle avec très petits coeffs, mais en plus on fait au maximum des possibilités. (on pourrait tester plus le résultat déjà obtenu).

```
\def\XINT_binom_vsmallend_iii #1.%
{%
    \ifnum #1>26  \expandafter\XINT_binom_end_iii \else
                  \expandafter\XINT_binom_vsmallend_iiib \fi #1.%
}%
\def\XINT_binom_vsmallend_iiib #1.#2.#3.%
{%
    \expandafter\XINT_binom_vsmallfinish
    \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
    \the\numexpr  #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
    !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
}%
\def\XINT_binom_vsmallend_ii #1.%
{%
    \ifnum #1>27  \expandafter\XINT_binom_end_ii \else
                  \expandafter\XINT_binom_vsmallend_iib \fi #1.%
}%
\def\XINT_binom_vsmallend_iib #1.#2.#3.%
{%
    \expandafter\XINT_binom_vsmallfinish
    \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
    \the\numexpr  #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
    !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
}%
\def\XINT_binom_vsmallend_i #1.%
{%
    \ifnum #1>28  \expandafter\XINT_binom_end_i \else
                  \expandafter\XINT_binom_vsmallend_ib \fi #1.%
}%
\def\XINT_binom_vsmallend_ib #1.#2.#3.%
{%
    \expandafter\XINT_binom_vsmallfinish
    \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
    \the\numexpr  #2*(#2+\xint_c_i)\expandafter
    !\the\numexpr #1*(#1+\xint_c_i)!%
}%
\def\XINT_binom_vsmallend_ #1.%
{%
    \ifnum #1>29  \expandafter\XINT_binom_end_ \else
                  \expandafter\XINT_binom_vsmallend_b \fi #1.%
}%
\def\XINT_binom_vsmallend_b #1.#2.#3.%
{%
    \expandafter\XINT_binom_vsmallfinish
    \the\numexpr\XINT_binom_vsmallmuldiv #2!#1!%
}%
\def\XINT_binom_vsmallfinish#1{%
\def\XINT_binom_vsmallfinish1##1!1!;!0!{\expandafter#1\the\numexpr##1\relax}%
}\XINT_binom_vsmallfinish{ }%
```

5.53 \xintiiPFactorial

2015/11/29 for 1.2f. Partial factorial pfac(a,b)=(a+1)...b, only for non-negative integers with a<=b<10^8.

1.2h (2016/11/20) removes the non-negativity condition. It was a bit unfortunate that the code raised \xintError:OutOfRangePFac if 0<=a<=b<10^8 was violated. The rule now applied is to interpret pfac(a,b) as the product for a<j<=b (not as a ratio of Gamma function), hence if a>=b, return 1 because of an empty product. If a<b: if a<0, return 0 for b>=0 and (-1)^(b-a) times |b|...(|a|-1) for b<0. But only for the range 0<= a <= b < 10^8 is the macro result to be considered as stable.

```
\def\xintiiPFactorial {\romannumeral0\xintiipfactorial }%
\def\xintiipfactorial #1#2%
{%
    \expandafter\XINT_pfac_fork\the\numexpr#1\expandafter.\the\numexpr #2.%
}%
\def\xintPFactorial{\romannumeral0\xintpfactorial}%
\let\xintpfactorial\xintiipfactorial
```

Code is a simplified version of the one for \xintiiBinomial, with no attempt at implementing a "very small" branch.

```
\def\XINT_pfac_fork #1#2.#3#4.%
{%
    \unless\ifnum #1#2<#3#4 \xint_dothis\XINT_pfac_one\fi
    \if-#3\xint_dothis\XINT_pfac_neg\fi
    \if-#1\xint_dothis\XINT_pfac_zero\fi
    \ifnum #3#4>\xint_c_x^viii_mone\xint_dothis\XINT_pfac_outofrange\fi
    \xint_orthat \XINT_pfac_a #1#2.#3#4.%
}%
\def\XINT_pfac_outofrange #1.#2.%
   {\XINT_signalcondition{InvalidOperation}{PFactorial with
    too big second arg: 99999999 < #2}{}{0}}%
\def\XINT_pfac_one        #1.#2.{ 1}%
\def\XINT_pfac_zero       #1.#2.{ 0}%
\def\XINT_pfac_neg -#1.-#2.%
{%
    \ifnum #1>\xint_c_x^viii\xint_dothis\XINT_pfac_outofrange\fi
    \xint_orthat
   {\ifodd\numexpr#2-#1\relax\xint_afterfi{\expandafter-\romannumeral`&&@}\fi
    \expandafter\XINT_pfac_a }%
    \the\numexpr #2-\xint_c_i\expandafter.\the\numexpr#1-\xint_c_i.%
}%
\def\XINT_pfac_a #1.#2.%
{%
    \expandafter\XINT_pfac_b\the\numexpr \xint_c_i+#1.#2.100000001!1;!%
    1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
}%
\def\XINT_pfac_b #1.%
{%
    \ifnum #1>9999 \xint_dothis\XINT_pfac_vbigloop \fi
    \ifnum #1>463  \xint_dothis\XINT_pfac_bigloop   \fi
    \ifnum #1>98   \xint_dothis\XINT_pfac_medloop   \fi
                   \xint_orthat\XINT_pfac_smallloop #1.%
}%
\def\XINT_pfac_smallloop #1.#2.%
{%
    \ifcase\numexpr #2-#1\relax
        \expandafter\XINT_pfac_end_
    \or \expandafter\XINT_pfac_end_i
    \or \expandafter\XINT_pfac_end_ii
    \or \expandafter\XINT_pfac_end_iii
    \else\expandafter\XINT_pfac_smallloop_a
    \fi #1.#2.%
}%
\def\XINT_pfac_smallloop_a #1.#2.%
{%
    \expandafter\XINT_pfac_smallloop_b
    \the\numexpr #1+\xint_c_iv\expandafter.%
    \the\numexpr #2\expandafter.%
    \the\numexpr\expandafter\XINT_smallmul
    \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
}%
\def\XINT_pfac_smallloop_b #1.%
{%
    \ifnum #1>98  \expandafter\XINT_pfac_medloop   \else
                  \expandafter\XINT_pfac_smallloop \fi #1.%
}%
```

```
\def\XINT_pfac_medloop #1.#2.%
{%
    \ifcase\numexpr #2-#1\relax
        \expandafter\XINT_pfac_end_
    \or \expandafter\XINT_pfac_end_i
    \or \expandafter\XINT_pfac_end_ii
    \else\expandafter\XINT_pfac_medloop_a
    \fi #1.#2.%
}%
\def\XINT_pfac_medloop_a #1.#2.%
{%
    \expandafter\XINT_pfac_medloop_b
    \the\numexpr #1+\xint_c_iii\expandafter.%
    \the\numexpr #2\expandafter.%
    \the\numexpr\expandafter\XINT_smallmul
    \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
}%
\def\XINT_pfac_medloop_b #1.%
{%
    \ifnum #1>463 \expandafter\XINT_pfac_bigloop   \else
                  \expandafter\XINT_pfac_medloop   \fi #1.%
}%
\def\XINT_pfac_bigloop #1.#2.%
{%
    \ifcase\numexpr #2-#1\relax
        \expandafter\XINT_pfac_end_
    \or \expandafter\XINT_pfac_end_i
    \else\expandafter\XINT_pfac_bigloop_a
    \fi #1.#2.%
}%
\def\XINT_pfac_bigloop_a #1.#2.%
{%
    \expandafter\XINT_pfac_bigloop_b
    \the\numexpr #1+\xint_c_ii\expandafter.%
    \the\numexpr #2\expandafter.%
    \the\numexpr\expandafter
    \XINT_smallmul\the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
}%
\def\XINT_pfac_bigloop_b #1.%
{%
    \ifnum #1>9999 \expandafter\XINT_pfac_vbigloop  \else
                   \expandafter\XINT_pfac_bigloop   \fi #1.%
}%
\def\XINT_pfac_vbigloop #1.#2.%
{%
    \ifnum #2=#1
         \expandafter\XINT_pfac_end_
    \else\expandafter\XINT_pfac_vbigloop_a
    \fi #1.#2.%
}%
\def\XINT_pfac_vbigloop_a #1.#2.%
{%
```

```
    \expandafter\XINT_pfac_vbigloop
    \the\numexpr #1+\xint_c_i\expandafter.%
    \the\numexpr #2\expandafter.%
    \the\numexpr\expandafter\XINT_smallmul\the\numexpr\xint_c_x^viii+#1!%
}%
\def\XINT_pfac_end_iii #1.#2.%
{%
    \expandafter\XINT_mul_out
    \the\numexpr\expandafter\XINT_smallmul
    \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
}%
\def\XINT_pfac_end_ii #1.#2.%
{%
    \expandafter\XINT_mul_out
    \the\numexpr\expandafter\XINT_smallmul
    \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
}%
\def\XINT_pfac_end_i #1.#2.%
{%
    \expandafter\XINT_mul_out
    \the\numexpr\expandafter\XINT_smallmul
    \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
}%
\def\XINT_pfac_end_ #1.#2.%
{%
    \expandafter\XINT_mul_out
    \the\numexpr\expandafter\XINT_smallmul\the\numexpr \xint_c_x^viii+#1!%
}%
```

5.54 \xintBool, \xintToggle

1.09c

```
\def\xintBool #1{\romannumeral`&&@%
                 \csname if#1\endcsname\expandafter1\else\expandafter0\fi }%
\def\xintToggle #1{\romannumeral`&&@\iftoggle{#1}{1}{0}}%
```

5.55 (WIP) \xintRandomDigits

1.3b. See user manual. Whether this will be part of xintkernel, xintcore, or xint is yet to be decided.

```
\def\xintRandomDigits{\romannumeral0\xintrandomdigits}%
\def\xintrandomdigits#1%
{%
    \csname xint_gob_andstop_\expandafter\XINT_randomdigits\the\numexpr#1\xint:
}%
\def\XINT_randomdigits#1\xint:
{%
    \expandafter\XINT_randomdigits_a
    \the\numexpr(#1+\xint_c_iii)/\xint_c_viii\xint:#1\xint:
}%
\def\XINT_randomdigits_a#1\xint:#2\xint:
```

```
{%
    \romannumeral\numexpr\xint_c_viii*#1-#2\csname XINT_%
      \romannumeral\XINT_replicate #1\endcsname \csname
    XINT_rdg\endcsname
}%
\def\XINT_rdg
{%
    \expandafter\XINT_rdg_aux\the\numexpr%
                 \xint_c_nine_x^viii%
                                -\xint_texuniformdeviate\xint_c_ii^vii%
                 -\xint_c_ii^vii*\xint_texuniformdeviate\xint_c_ii^vii%
                 -\xint_c_ii^xiv*\xint_texuniformdeviate\xint_c_ii^vii%
                 -\xint_c_ii^xxi*\xint_texuniformdeviate\xint_c_ii^vii%
                 +\xint_texuniformdeviate\xint_c_x^viii%
                 \relax%
}%
\def\XINT_rdg_aux#1{XINT_rdg\endcsname}%
\let\XINT_XINT_rdg\endcsname
```

5.56 (WIP) \XINT_eightrandomdigits

1.3b.

```
\def\XINT_eightrandomdigits
{%
     \expandafter\xint_gobble_i\the\numexpr%
                 \xint_c_nine_x^viii%
                                -\xint_texuniformdeviate\xint_c_ii^vii%
                 -\xint_c_ii^vii*\xint_texuniformdeviate\xint_c_ii^vii%
                 -\xint_c_ii^xiv*\xint_texuniformdeviate\xint_c_ii^vii%
                 -\xint_c_ii^xxi*\xint_texuniformdeviate\xint_c_ii^vii%
                 +\xint_texuniformdeviate\xint_c_x^viii%
                 \relax%
}%
```

5.57 (WIP) \xintXRandomDigits

1.3b.

```
\def\xintXRandomDigits#1%
{%
    \csname xint_gobble_\expandafter\XINT_xrandomdigits\the\numexpr#1\xint:
}%
\def\XINT_xrandomdigits#1\xint:
{%
    \expandafter\XINT_xrandomdigits_a
    \the\numexpr(#1+\xint_c_iii)/\xint_c_viii\xint:#1\xint:
}%
\def\XINT_xrandomdigits_a#1\xint:#2\xint:
{%
    \romannumeral\numexpr\xint_c_viii*#1-#2\expandafter\endcsname
    \romannumeral`&&@\romannumeral
                    \XINT_replicate #1\endcsname\XINT_eightrandomdigits
}%
```

5.58 (WIP) `\xintiiRandRangeAtoB`

1.3b. Support for randrange() function.

Wee do it f-expandably for matters of \xintNewExpr etc... The \xintexpr will add \xintNum wrapper to possible fractional input. But \xintiiexpr will call as is.

TODO: ? implement third argument (STEP) TODO: \xintNum wrapper (which truncates) not so good in floatexpr. Use round?

It is an error if b<=a, as in Python.

```
\def\xintiiRandRangeAtoB{\romannumeral`&&@\xintiirandrangeAtoB}%
\def\xintiirandrangeAtoB#1%
{%
    \expandafter\XINT_randrangeAtoB_a\romannumeral`&&@#1\xint:
}%
\def\XINT_randrangeAtoB_a#1\xint:#2%
{%
    \xintiiadd{\expandafter\XINT_randrange
                \romannumeral0\xintiisub{#2}{#1}\xint:}%
               {#1}%
}%
```

5.59 (WIP) `\xintiiRandRange`

1.3b. Support for randrange().

```
\def\xintiiRandRange{\romannumeral`&&@\xintiirandrange}%
\def\xintiirandrange#1%
{%
    \expandafter\XINT_randrange\romannumeral`&&@#1\xint:
}%
\def\XINT_randrange #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_randrange_err:empty
      0#1\XINT_randrange_err:empty
       0-\XINT_randrange_a
    \krof #1%
}%
\def\XINT_randrange_err:empty#1\xint:
{%
    \XINT_expandableerror{Empty range for randrange.} 0%
}%
\def\XINT_randrange_a #1\xint:
{%
    \expandafter\XINT_randrange_b\romannumeral0\xintlength{#1}.#1\xint:
}%
\def\XINT_randrange_b #1.%
{%
    \ifnum#1<\xint_c_x\xint_dothis{\the\numexpr\XINT_uniformdeviate{}}\fi
    \xint_orthat{\XINT_randrange_c #1.}%
}%
```

```
\def\XINT_randrange_c #1.#2#3#4#5#6#7#8#9%
{%
    \expandafter\XINT_randrange_d
    \the\numexpr\expandafter\XINT_uniformdeviate\expandafter
       {\expandafter}\the\numexpr\xint_c_i+#2#3#4#5#6#7#8#9\xint:\xint:
    #2#3#4#5#6#7#8#9\xint:#1\xint:
}%
```

This raises following annex question: immediately after setting the seed is it possible for \xintUniformDeviate{N} where N>0 has exactly eight digits to return either 0 or N-1 ? It could be that this is never the case, then there is a bias in randrange(). Of course there are anyhow only 2^28 seeds so randrange(10^X) is by necessity biased when executed immediately after setting the seed, if X is at least 9.

```
\def\XINT_randrange_d #1\xint:#2\xint:
{%
    \ifnum#1=\xint_c_\xint_dothis\XINT_randrange_Z\fi
    \ifnum#1=#2 \xint_dothis\XINT_randrange_A\fi
    \xint_orthat\XINT_randrange_e #1\xint:
}%
\def\XINT_randrange_e #1\xint:#2\xint:#3\xint:
{%
    \the\numexpr#1\expandafter\relax
    \romannumeral0\xintrandomdigits{#2-\xint_c_viii}%
}%
```

This is quite unlikely to get executed but if it does it must pay attention to leading zeros, hence the \xintinum. We don't have to be overly obstinate about removing overheads...

```
\def\XINT_randrange_Z 0\xint:#1\xint:#2\xint:
{%
    \xintinum{\xintRandomDigits{#1-\xint_c_viii}}%
}%
```

Here too, overhead is not such a problem. The idea is that we got by extraordinary same first 8 digits as upper range bound so we pick at random the remaining needed digits in one go and compare with the upper bound. If too big, we start again with another random 8 leading digits in given range. No need to aim at any kind of efficiency for the check and loop back.

```
\def\XINT_randrange_A #1\xint:#2\xint:#3\xint:
{%
    \expandafter\XINT_randrange_B
    \romannumeral0\xintrandomdigits{#2-\xint_c_viii}\xint:
    #3\xint:#2.#1\xint:
}%
\def\XINT_randrange_B #1\xint:#2\xint:#3.#4\xint:
{%
    \xintiiifLt{#1}{#2}{\XINT_randrange_E}{\XINT_randrange_again}%
    #4#1\xint:#3.#4#2\xint:
}%
\def\XINT_randrange_E #1\xint:#2\xint:{ #1}%
\def\XINT_randrange_again #1\xint:{\XINT_randrange_c}%
```

5.60 Adjustments for engines without uniformdeviate primitive

1.3b.

```
\ifdefined\xint_texuniformdeviate
\else
  \def\xintrandomdigits#1%
  {%
      \XINT_expandableerror
      {No uniformdeviate at engine level, returning 0.} 0%
  }%
  \let\xintXRandomDigits\xintRandomDigits
  \def\XINT_randrange#1\xint:
  {%
      \XINT_expandableerror
      {No uniformdeviate at engine level, returning 0.} 0%
  }%
\fi
\XINT_restorecatcodes_endinput%
```

6 Package xintbinhex implementation

The commenting is currently (2018/06/17) very sparse.

The macros from 1.08 (2013/06/07) remained unchanged until their complete rewrite at 1.2m (2017/07/31).

At 1.2n dependencies on xintcore were removed, so now the package loads only xintkernel (this could have been done earlier).

Also at 1.2n, macros evolved again, the main improvements being in the increased allowable sizes of the input for \xintDecToHex, \xintDecToBin, \xintBinToHex. Use of \csname governed expansion at some places rather than \numexpr with some clean-up after it.

6.1 Catcodes, ε-TEX and reload detection

The code for reload detection was initially copied from Heiko Oberdiek's packages, then modified.

The method for catcodes was also initially directly inspired by these packages.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode64=11   % @
  \catcode35=6    % #
  \catcode44=12   % ,
  \catcode45=12   % -
  \catcode46=12   % .
  \catcode58=12   % :
  \let\z\endgroup
  \expandafter\let\expandafter\x\csname ver@xintbinhex.sty\endcsname
  \expandafter\let\expandafter\w\csname ver@xintkernel.sty\endcsname
  \expandafter
    \ifx\csname PackageInfo\endcsname\relax
      \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
    \else
      \def\y#1#2{\PackageInfo{#1}{#2}}%
    \fi
  \expandafter
  \ifx\csname numexpr\endcsname\relax
     \y{xintbinhex}{\numexpr not available, aborting input}%
     \aftergroup\endinput
  \else
    \ifx\x\relax   % plain-TeX, first loading of xintbinhex.sty
      \ifx\w\relax % but xintkernel.sty not yet loaded.
         \def\z{\endgroup\input xintkernel.sty\relax}%
      \fi
    \else
```

```
      \def\empty {}%
      \ifx\x\empty % LaTeX, first loading,
      % variable is initialized, but \ProvidesPackage not yet seen
          \ifx\w\relax % xintkernel.sty not yet loaded.
            \def\z{\endgroup\RequirePackage{xintkernel}}%
          \fi
      \else
        \aftergroup\endinput % xintbinhex already loaded.
      \fi
    \fi
  \fi
\z%
\XINTsetupcatcodes% defined in xintkernel.sty
```

6.2 Package identification

```
\XINT_providespackage
\ProvidesPackage{xintbinhex}%
  [2018/06/17 1.3c Expandable binary and hexadecimal conversions (JFB)]%
```

6.3 Constants, etc...

1.2n switches to \csname-governed expansion at various places.

```
\newcount\xint_c_ii^xv  \xint_c_ii^xv   32768
\newcount\xint_c_ii^xvi \xint_c_ii^xvi  65536
\def\XINT_tmpa #1{\ifx\relax#1\else
  \expandafter\edef\csname XINT_csdth_#1\endcsname
  {\endcsname\ifcase #1 0\or 1\or 2\or 3\or 4\or 5\or 6\or 7\or
              8\or 9\or A\or B\or C\or D\or E\or F\fi}%
  \expandafter\XINT_tmpa\fi }%
\XINT_tmpa {0}{1}{2}{3}{4}{5}{6}{7}{8}{9}{10}{11}{12}{13}{14}{15}\relax
\def\XINT_tmpa #1{\ifx\relax#1\else
  \expandafter\edef\csname XINT_csdtb_#1\endcsname
  {\endcsname\ifcase #1
   0000\or 0001\or 0010\or 0011\or 0100\or 0101\or 0110\or 0111\or
   1000\or 1001\or 1010\or 1011\or 1100\or 1101\or 1110\or 1111\fi}%
  \expandafter\XINT_tmpa\fi }%
\XINT_tmpa {0}{1}{2}{3}{4}{5}{6}{7}{8}{9}{10}{11}{12}{13}{14}{15}\relax
\let\XINT_tmpa\relax
\expandafter\def\csname XINT_csbth_0000\endcsname {\endcsname0}%
\expandafter\def\csname XINT_csbth_0001\endcsname {\endcsname1}%
\expandafter\def\csname XINT_csbth_0010\endcsname {\endcsname2}%
\expandafter\def\csname XINT_csbth_0011\endcsname {\endcsname3}%
\expandafter\def\csname XINT_csbth_0100\endcsname {\endcsname4}%
\expandafter\def\csname XINT_csbth_0101\endcsname {\endcsname5}%
\expandafter\def\csname XINT_csbth_0110\endcsname {\endcsname6}%
\expandafter\def\csname XINT_csbth_0111\endcsname {\endcsname7}%
\expandafter\def\csname XINT_csbth_1000\endcsname {\endcsname8}%
\expandafter\def\csname XINT_csbth_1001\endcsname {\endcsname9}%
\expandafter\def\csname XINT_csbth_1010\endcsname {\endcsname A}%
\expandafter\def\csname XINT_csbth_1011\endcsname {\endcsname B}%
\expandafter\def\csname XINT_csbth_1100\endcsname {\endcsname C}%
\expandafter\def\csname XINT_csbth_1101\endcsname {\endcsname D}%
```

```
\expandafter\def\csname XINT_csbth_1110\endcsname {\endcsname E}%
\expandafter\def\csname XINT_csbth_1111\endcsname {\endcsname F}%
\let\XINT_csbth_none \endcsname
\expandafter\def\csname XINT_cshtb_0\endcsname {\endcsname0000}%
\expandafter\def\csname XINT_cshtb_1\endcsname {\endcsname0001}%
\expandafter\def\csname XINT_cshtb_2\endcsname {\endcsname0010}%
\expandafter\def\csname XINT_cshtb_3\endcsname {\endcsname0011}%
\expandafter\def\csname XINT_cshtb_4\endcsname {\endcsname0100}%
\expandafter\def\csname XINT_cshtb_5\endcsname {\endcsname0101}%
\expandafter\def\csname XINT_cshtb_6\endcsname {\endcsname0110}%
\expandafter\def\csname XINT_cshtb_7\endcsname {\endcsname0111}%
\expandafter\def\csname XINT_cshtb_8\endcsname {\endcsname1000}%
\expandafter\def\csname XINT_cshtb_9\endcsname {\endcsname1001}%
\def\XINT_cshtb_A {\endcsname1010}%
\def\XINT_cshtb_B {\endcsname1011}%
\def\XINT_cshtb_C {\endcsname1100}%
\def\XINT_cshtb_D {\endcsname1101}%
\def\XINT_cshtb_E {\endcsname1110}%
\def\XINT_cshtb_F {\endcsname1111}%
\let\XINT_cshtb_none \endcsname
```

6.4 Helper macros

6.4.1 \XINT_zeroes_foriv

```
 \romannumeral0\XINT_zeroes_foriv #1\R{0\R}{00\R}{000\R}%
                                     \R{0\R}{00\R}{000\R}\R\W
```
expands to the <empty> or 0 or 00 or 000 needed which when adjoined to #1 extend it to length 4N.

```
\def\XINT_zeroes_foriv #1#2#3#4#5#6#7#8%
{%
    \xint_gob_til_R #8\XINT_zeroes_foriv_end\R\XINT_zeroes_foriv
}%
\def\XINT_zeroes_foriv_end\R\XINT_zeroes_foriv #1#2\W
   {\XINT_zeroes_foriv_done #1}%
\def\XINT_zeroes_foriv_done #1\R{ #1}%
```

6.5 \xintDecToHex

Complete rewrite at 1.2m in the 1.2 style. Also, 1.2m is robust against non terminated inputs.
  Improvements of coding at 1.2n, increased maximal size. Again some coding improvement at 1.2o, about 6% speed gain.
  An input without leading zeroes gives an output without leading zeroes.

```
\def\xintDecToHex {\romannumeral0\xintdectohex }%
\def\xintdectohex #1%
{%
    \expandafter\XINT_dth_checkin\romannumeral`&&@#1\xint:
}%
\def\XINT_dth_checkin #1%
{%
    \xint_UDsignfork
       #1\XINT_dth_neg
```

```
        -{\XINT_dth_main #1}%
     \krof
}%
\def\XINT_dth_neg {\expandafter-\romannumeral0\XINT_dth_main}%
\def\XINT_dth_main #1\xint:
{%
    \expandafter\XINT_dth_finish
    \romannumeral`&&@\expandafter\XINT_dthb_start
    \romannumeral0\XINT_zeroes_foriv
       #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
    #1\xint_bye\XINT_dth_tohex
}%
\def\XINT_dthb_start #1#2#3#4#5%
{%
    \xint_bye#5\XINT_dthb_small\xint_bye\XINT_dthb_start_a #1#2#3#4#5%
}%
\def\XINT_dthb_small\xint_bye\XINT_dthb_start_a #1\xint_bye#2{#2#1!}%
\def\XINT_dthb_start_a #1#2#3#4#5#6#7#8#9%
{%
    \expandafter\XINT_dthb_again\the\numexpr\expandafter\XINT_dthb_update
    \the\numexpr#1#2#3#4%
    \xint_bye#9\XINT_dthb_lastpass\xint_bye
    #5#6#7#8!\XINT_dthb_exclam\relax\XINT_dthb_nextfour #9%
}%
```

The 1.2n inserted exclamations marks, which when bumping back from \XINT_dthb_again gave rise to a \numexpr-loop which gathered the ! delimited arguments and inserted \expandafter\XINT_dthb_update\the\numexpr dynamically. The 1.2o trick is to insert it here immediately. Then at \XINT_dthb_again the \numexpr will trigger an already prepared chain.

The crux of the thing is handling of #3 at \XINT_dthb_update_a.

```
\def\XINT_dthb_exclam {!\XINT_dthb_exclam\relax
                       \expandafter\XINT_dthb_update\the\numexpr}%
\def\XINT_dthb_update #1!%
{%
    \expandafter\XINT_dthb_update_a
    \the\numexpr (#1+\xint_c_ii^xv)/\xint_c_ii^xvi-\xint_c_i\xint:
    #1\xint:%
}%
\def\XINT_dthb_update_a #1\xint:#2\xint:#3%
{%
    0000+#1\expandafter#3\the\numexpr#2-#1*\xint_c_ii^xvi
}%
```

1.2m and 1.2n had some unduly complicated ending pattern for \XINT_dthb_nextfour as inheritance of a loop needing ! separators which was pruned out at 1.2o (see previous comment).

```
\def\XINT_dthb_nextfour #1#2#3#4#5%
{%
    \xint_bye#5\XINT_dthb_lastpass\xint_bye
    #1#2#3#4!\XINT_dthb_exclam\relax\XINT_dthb_nextfour#5%
}%
\def\XINT_dthb_lastpass\xint_bye #1!#2\xint_bye#3{#1!#3!}%
\def\XINT_dth_tohex
```

```
{%
    \expandafter\expandafter\expandafter\XINT_dth_tohex_a\csname\XINT_tofourhex
}%
\def\XINT_dth_tohex_a\endcsname{!\XINT_dth_tohex!}%
\def\XINT_dthb_again #1!#2#3%
{%
    \ifx#3\relax
          \expandafter\xint_firstoftwo
      \else
          \expandafter\xint_secondoftwo
    \fi
    {\expandafter\XINT_dthb_again
     \the\numexpr
     \ifnum #1>\xint_c_
        \xint_afterfi{\expandafter\XINT_dthb_update\the\numexpr#1}%
     \fi}%
    {\ifnum #1>\xint_c_ \xint_dothis{#2#1!}\fi\xint_orthat{!#2!}}%
}%
\def\XINT_tofourhex #1!%
{%
    \expandafter\XINT_tofourhex_a
    \the\numexpr (#1+\xint_c_ii^vii)/\xint_c_ii^viii-\xint_c_i\xint:
    #1\xint:
}%
\def\XINT_tofourhex_a #1\xint:#2\xint:
{%
    \expandafter\XINT_tofourhex_c
    \the\numexpr (#1+\xint_c_viii)/\xint_c_xvi-\xint_c_i\xint:
    #1\xint:
    \the\numexpr #2-\xint_c_ii^viii*#1!%
}%
\def\XINT_tofourhex_c #1\xint:#2\xint:
{%
    XINT_csdth_#1%
    \csname XINT_csdth_\the\numexpr #2-\xint_c_xvi*#1\relax
    \csname \expandafter\XINT_tofourhex_d
}%
\def\XINT_tofourhex_d #1!%
{%
    \expandafter\XINT_tofourhex_e
    \the\numexpr (#1+\xint_c_viii)/\xint_c_xvi-\xint_c_i\xint:
    #1\xint:
}%
\def\XINT_tofourhex_e #1\xint:#2\xint:
{%
    XINT_csdth_#1%
    \csname XINT_csdth_\the\numexpr #2-\xint_c_xvi*#1\endcsname
}%
```

We only clean-up up to 3 zero hexadecimal digits, as output was produced in chunks of 4 hex digits. If input had no leading zero, output will have none either. If input had many leading zeroes, output will have some number (unspecified, but a recipe can be given...) of leading zeroes...

The coding is for varying a bit, I did not check if efficient, it does not matter.

```
\def\XINT_dth_finish !\XINT_dth_tohex!#1#2#3%
{%
    \unless\if#10\xint_dothis{ #1#2#3}\fi
    \unless\if#20\xint_dothis{ #2#3}\fi
    \unless\if#30\xint_dothis{ #3}\fi
    \xint_orthat{ }%
}%
```

6.6 \xintDecToBin

```
Complete rewrite at 1.2m in the 1.2 style. Also, 1.2m is robust against non terminated inputs.
  Revisited at 1.2n like in \xintDecToHex: increased maximal size.
  An input without leading zeroes gives an output without leading zeroes.
  Most of the code canvas is shared with \xintDecToHex.
```

```
\def\xintDecToBin {\romannumeral0\xintdectobin }%
\def\xintdectobin #1%
{%
    \expandafter\XINT_dtb_checkin\romannumeral`&&@#1\xint:
}%
\def\XINT_dtb_checkin #1%
{%
    \xint_UDsignfork
       #1\XINT_dtb_neg
        -{\XINT_dtb_main #1}%
     \krof
}%
\def\XINT_dtb_neg {\expandafter-\romannumeral0\XINT_dtb_main}%
\def\XINT_dtb_main #1\xint:
{%
    \expandafter\XINT_dtb_finish
    \romannumeral`&&@\expandafter\XINT_dthb_start
    \romannumeral0\XINT_zeroes_foriv
       #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
    #1\xint_bye\XINT_dtb_tobin
}%
\def\XINT_dtb_tobin
{%
    \expandafter\expandafter\expandafter\XINT_dtb_tobin_a\csname\XINT_tosixteenbits
}%
\def\XINT_dtb_tobin_a\endcsname{!\XINT_dtb_tobin!}%
\def\XINT_tosixteenbits #1!%
{%
    \expandafter\XINT_tosixteenbits_a
    \the\numexpr (#1+\xint_c_ii^vii)/\xint_c_ii^viii-\xint_c_i\xint:
    #1\xint:
}%
\def\XINT_tosixteenbits_a #1\xint:#2\xint:
{%
    \expandafter\XINT_tosixteenbits_c
    \the\numexpr (#1+\xint_c_viii)/\xint_c_xvi-\xint_c_i\xint:
    #1\xint:
    \the\numexpr #2-\xint_c_ii^viii*#1!%
}%
\def\XINT_tosixteenbits_c #1\xint:#2\xint:
{%
    XINT_csdtb_#1%
    \csname XINT_csdtb_\the\numexpr #2-\xint_c_xvi*#1\relax
    \csname \expandafter\XINT_tosixteenbits_d
}%
\def\XINT_tosixteenbits_d #1!%
{%
    \expandafter\XINT_tosixteenbits_e
    \the\numexpr (#1+\xint_c_viii)/\xint_c_xvi-\xint_c_i\xint:
    #1\xint:
}%
\def\XINT_tosixteenbits_e #1\xint:#2\xint:
{%
    XINT_csdtb_#1%
    \csname XINT_csdtb_\the\numexpr #2-\xint_c_xvi*#1\endcsname
}%
\def\XINT_dtb_finish !\XINT_dtb_tobin!#1#2#3#4#5#6#7#8%
{%
    \expandafter\XINT_dtb_finish_a\the\numexpr #1#2#3#4#5#6#7#8\relax
}%
\def\XINT_dtb_finish_a #1{%
\def\XINT_dtb_finish_a ##1##2##3##4##5##6##7##8##9%
{%
    \expandafter#1\the\numexpr ##1##2##3##4##5##6##7##8##9\relax
}}\XINT_dtb_finish_a { }%
```

6.7 \xintHexToDec

Completely (and belatedly) rewritten at 1.2m in the 1.2 style.

1.2m version robust against non terminated inputs, but there is no primitive from TeX which may generate hexadecimal digits and provoke expansion ahead, afaik, except of course if decimal digits are treated as hexadecimal. This robustness is not on purpose but from need to expand argument and then grab it again. So we do it safely.

Increased maximal size at 1.2n.

1.2m version robust against non terminated inputs.

An input without leading zeroes gives an output without leading zeroes.

```
\def\xintHexToDec {\romannumeral0\xinthextodec }%
\def\xinthextodec #1%
{%
    \expandafter\XINT_htd_checkin\romannumeral`&&@#1\xint:
}%
\def\XINT_htd_checkin #1%
{%
    \xint_UDsignfork
       #1\XINT_htd_neg
        -{\XINT_htd_main #1}%
     \krof
}%
\def\XINT_htd_neg {\expandafter-\romannumeral0\XINT_htd_main}%
\def\XINT_htd_main #1\xint:
```

```
{%
    \expandafter\XINT_htd_startb
    \the\numexpr\expandafter\XINT_htd_starta
    \romannumeral0\XINT_zeroes_foriv
       #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
    #1\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\relax
}%
\def\XINT_htd_starta #1#2#3#4{"#1#2#3#4+100000!}%
\def\XINT_htd_startb 1#1%
{%
    \if#10\expandafter\XINT_htd_startba\else
          \expandafter\XINT_htd_startbb
    \fi 1#1%
}%
\def\XINT_htd_startba 10#1!{\XINT_htd_again #1%
    \xint_bye!2!3!4!5!6!7!8!9!\xint_bye\XINT_htd_nextfour}%
\def\XINT_htd_startbb 1#1#2!{\XINT_htd_again #1!#2%
    \xint_bye!2!3!4!5!6!7!8!9!\xint_bye\XINT_htd_nextfour}%
```

It is a bit annoying to grab all to the end here. I have a version, modeled on the 1.2n variant of \xintDecToHex which solved that problem there, but it did not prove enough if at all faster in my brief testing and it had the defect of a reduced maximal allowed size of the input.

```
\def\XINT_htd_again #1\XINT_htd_nextfour #2%
{%
    \xint_bye #2\XINT_htd_finish\xint_bye
    \expandafter\XINT_htd_A\the\numexpr
    \XINT_htd_a #1\XINT_htd_nextfour #2%
}%
\def\XINT_htd_a #1!#2!#3!#4!#5!#6!#7!#8!#9!%
{%
    #1\expandafter\XINT_htd_update
    \the\numexpr #2\expandafter\XINT_htd_update
    \the\numexpr #3\expandafter\XINT_htd_update
    \the\numexpr #4\expandafter\XINT_htd_update
    \the\numexpr #5\expandafter\XINT_htd_update
    \the\numexpr #6\expandafter\XINT_htd_update
    \the\numexpr #7\expandafter\XINT_htd_update
    \the\numexpr #8\expandafter\XINT_htd_update
    \the\numexpr #9\expandafter\XINT_htd_update
    \the\numexpr \XINT_htd_a
}%
\def\XINT_htd_nextfour #1#2#3#4%
{%
    *\xint_c_ii^xvi+"#1#2#3#4+1000000000\relax\xint_bye!%
    2!3!4!5!6!7!8!9!\xint_bye\XINT_htd_nextfour
}%
```

If the innocent looking commented out #6 is left in the pattern as was the case at 1.2m, the maximal size becomes limited at 5538 digits, not 8298! (with parameter stack size = 10000.)

```
\def\XINT_htd_update 1#1#2#3#4#5%#6!%
{%
    *\xint_c_ii^xvi+10000#1#2#3#4#5!%#6!%
}%
\def\XINT_htd_A 1#1%
{%
    \if#10\expandafter\XINT_htd_Aa\else
          \expandafter\XINT_htd_Ab
    \fi 1#1%
}%
\def\XINT_htd_Aa 10#1#2#3#4{\XINT_htd_again #1#2#3#4!}%
\def\XINT_htd_Ab 1#1#2#3#4#5{\XINT_htd_again #1!#2#3#4#5!}%
\def\XINT_htd_finish\xint_bye
    \expandafter\XINT_htd_A\the\numexpr \XINT_htd_a #1\XINT_htd_nextfour
{%
    \expandafter\XINT_htd_finish_cuz\the\numexpr0\XINT_htd_unsep_loop #1%
}%
\def\XINT_htd_unsep_loop #1!#2!#3!#4!#5!#6!#7!#8!#9!%
{%
    \expandafter\XINT_unsep_clean
    \the\numexpr 1#1#2\expandafter\XINT_unsep_clean
    \the\numexpr 1#3#4\expandafter\XINT_unsep_clean
    \the\numexpr 1#5#6\expandafter\XINT_unsep_clean
    \the\numexpr 1#7#8\expandafter\XINT_unsep_clean
    \the\numexpr 1#9\XINT_htd_unsep_loop_a
}%
\def\XINT_htd_unsep_loop_a #1!#2!#3!#4!#5!#6!#7!#8!#9!%
{%
    #1\expandafter\XINT_unsep_clean
    \the\numexpr 1#2#3\expandafter\XINT_unsep_clean
    \the\numexpr 1#4#5\expandafter\XINT_unsep_clean
    \the\numexpr 1#6#7\expandafter\XINT_unsep_clean
    \the\numexpr 1#8#9\XINT_htd_unsep_loop
}%
\def\XINT_unsep_clean 1{\relax}% also in xintcore
\def\XINT_htd_finish_cuz #1{%
\def\XINT_htd_finish_cuz ##1##2##3##4##5%
   {\expandafter#1\the\numexpr ##1##2##3##4##5\relax}%
}\XINT_htd_finish_cuz{ }%
```

6.8 `\xintBinToDec`

Redone entirely for 1.2m. Starts by converting to hexadecimal first.
  Increased maximal size at 1.2n.
  An input without leading zeroes gives an output without leading zeroes.
  Robust against non-terminated input.

```
\def\xintBinToDec {\romannumeral0\xintbintodec }%
\def\xintbintodec #1%
{%
    \expandafter\XINT_btd_checkin\romannumeral`&&@#1\xint:
}%
\def\XINT_btd_checkin #1%
{%
    \xint_UDsignfork
       #1\XINT_btd_N
```

```
         -{\XINT_btd_main #1}%
      \krof
}%
\def\XINT_btd_N {\expandafter-\romannumeral0\XINT_btd_main }%
\def\XINT_btd_main #1\xint:
{%
    \csname XINT_btd_htd\csname\expandafter\XINT_bth_loop
    \romannumeral0\XINT_zeroes_foriv
       #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
    #1\xint_bye2345678\xint_bye none\endcsname\xint:
}%
\def\XINT_btd_htd #1\xint:
{%
    \expandafter\XINT_htd_startb
    \the\numexpr\expandafter\XINT_htd_starta
    \romannumeral0\XINT_zeroes_foriv
       #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
    #1\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\relax
}%
```

6.9 \xintBinToHex

Complete rewrite for 1.2m. But input for 1.2m version limited to about 13320 binary digits (expansion depth=10000).

Again redone for 1.2n for \csname governed expansion: increased maximal size.

Size of output is ceil(size(input)/4), leading zeroes in output (inherited from the input) are not trimmed.

An input without leading zeroes gives an output without leading zeroes.

Robust against non-terminated input.

```
\def\xintBinToHex {\romannumeral0\xintbintohex }%
\def\xintbintohex #1%
{%
    \expandafter\XINT_bth_checkin\romannumeral`&&@#1\xint:
}%
\def\XINT_bth_checkin #1%
{%
    \xint_UDsignfork
       #1\XINT_bth_N
         -{\XINT_bth_main #1}%
      \krof
}%
\def\XINT_bth_N {\expandafter-\romannumeral0\XINT_bth_main }%
\def\XINT_bth_main #1\xint:
{%
    \csname space\csname\expandafter\XINT_bth_loop
    \romannumeral0\XINT_zeroes_foriv
       #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
    #1\xint_bye2345678\xint_bye none\endcsname
}%
\def\XINT_bth_loop #1#2#3#4#5#6#7#8%
{%
            XINT_csbth_#1#2#3#4%
    \csname XINT_csbth_#5#6#7#8%
    \csname\XINT_bth_loop
}%
```

6.10 \xintHexToBin

Completely rewritten for 1.2m.

Attention this macro is not robust against arguments expanding after themselves.

Only up to three zeros are removed on front of output: if the input had a leading zero, there will be a leading zero (and then possibly 4n of them if inputs had more leading zeroes) on output.

Rewritten again at 1.2n for \csname governed expansion.

```
\def\xintHexToBin {\romannumeral0\xinthextobin }%
\def\xinthextobin #1%
{%
    \expandafter\XINT_htb_checkin\romannumeral`&&@#1%
    \xint_bye 23456789\xint_bye none\endcsname
}%
\def\XINT_htb_checkin #1%
{%
    \xint_UDsignfork
       #1\XINT_htb_N
        -{\XINT_htb_main #1}%
     \krof
}%
\def\XINT_htb_N {\expandafter-\romannumeral0\XINT_htb_main }%
\def\XINT_htb_main {\csname XINT_htb_cuz\csname\XINT_htb_loop}%
\def\XINT_htb_loop #1#2#3#4#5#6#7#8#9%
{%
             XINT_cshtb_#1%
     \csname XINT_cshtb_#2%
     \csname XINT_cshtb_#3%
     \csname XINT_cshtb_#4%
     \csname XINT_cshtb_#5%
     \csname XINT_cshtb_#6%
     \csname XINT_cshtb_#7%
     \csname XINT_cshtb_#8%
     \csname XINT_cshtb_#9%
     \csname \XINT_htb_loop
}%
\def\XINT_htb_cuz #1{%
\def\XINT_htb_cuz ##1##2##3##4%
   {\expandafter#1\the\numexpr##1##2##3##4\relax}%
}\XINT_htb_cuz { }%
```

6.11 \xintCHexToBin

The 1.08 macro had same functionality as \xintHexToBin, and slightly different code, the 1.2m version has the same code as \xintHexToBin except that it does not remove leading zeros from output: if the input had N hexadecimal digits, the output will have exactly 4N binary digits.

Rewritten again at 1.2n for \csname governed expansion.

```
\def\xintCHexToBin {\romannumeral0\xintchextobin }%
```

```
\def\xintchextobin #1%
{%
    \expandafter\XINT_chtb_checkin\romannumeral`&&@#1%
    \xint_bye 23456789\xint_bye none\endcsname
}%
\def\XINT_chtb_checkin #1%
{%
    \xint_UDsignfork
       #1\XINT_chtb_N
        -{\XINT_chtb_main #1}%
     \krof
}%
\def\XINT_chtb_N {\expandafter-\romannumeral0\XINT_chtb_main }%
\def\XINT_chtb_main {\csname space\csname\XINT_htb_loop}%
\XINT_restorecatcodes_endinput%
```

7 Package xintgcd implementation

The commenting is currently (2018/06/17) very sparse. Release 1.09h has modified a bit the \xintTypesetEuclideAlgorithm and \xintTypesetBezoutAlgorithm layout with respect to line indentation in particular. And they use the xinttools \xintloop rather than the Plain TEX or LATEX's \loop.

Since 1.1 the package only loads xintcore, not xint. And for the \xintTypesetEuclideAlgorithm and \xintTypesetBezoutAlgorithm macros to be functional the package xinttools needs to be loaded explicitely by the user.

Breaking change at 1.2p: \xintBezout{A}{B} formerly had output {A}{B}{U}{V}{D} with AU-BV=D, now it is {U}{V}{D} with AU+BV=D.

7.1 Catcodes, ε-TEX and reload detection

The code for reload detection was initially copied from Heiko Oberdiek's packages, then modified.

The method for catcodes was also initially directly inspired by these packages.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5     % ^^M
  \endlinechar=13 %
  \catcode123=1    % {
  \catcode125=2    % }
  \catcode64=11    % @
  \catcode35=6     % #
  \catcode44=12    % ,
  \catcode45=12    % -
  \catcode46=12    % .
  \catcode58=12    % :
  \let\z\endgroup
  \expandafter\let\expandafter\x\csname ver@xintgcd.sty\endcsname
  \expandafter\let\expandafter\w\csname ver@xintcore.sty\endcsname
  \expandafter
    \ifx\csname PackageInfo\endcsname\relax
      \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
    \else
      \def\y#1#2{\PackageInfo{#1}{#2}}%
    \fi
  \expandafter
  \ifx\csname numexpr\endcsname\relax
     \y{xintgcd}{\numexpr not available, aborting input}%
     \aftergroup\endinput
  \else
    \ifx\x\relax   % plain-TeX, first loading of xintgcd.sty
      \ifx\w\relax % but xintcore.sty not yet loaded.
         \def\z{\endgroup\input xintcore.sty\relax}%
      \fi
    \else
```

```
      \def\empty {}%
      \ifx\x\empty % LaTeX, first loading,
      % variable is initialized, but \ProvidesPackage not yet seen
          \ifx\w\relax % xintcore.sty not yet loaded.
            \def\z{\endgroup\RequirePackage{xintcore}}%
          \fi
      \else
        \aftergroup\endinput % xintgcd already loaded.
      \fi
    \fi
  \fi
\z%
\XINTsetupcatcodes% defined in xintkernel.sty
```

7.2 Package identification

```
\XINT_providespackage
\ProvidesPackage{xintgcd}%
  [2018/06/17 1.3c Euclide algorithm with xint package (JFB)]%
```

7.3 \xintGCD, \xintiiGCD

1.09a added \xintnum filtering from \xintiabs. This is a bit overhead but makes it easier for the gcd function in \xintexpr.

1.1a defines \xintiiGCD to avoid overhead in \xintiiexpr.

1.2p: but 1.2o deprecated \xintiAbs, and in fact \xintGCD should not have any \xintNum overhead for consistency with other xint macros. But well, it would be breaking change to modify this now. We can not use \xintAbs which will create a fraction A/1, so we use \xintNum directly.

```
\def\xintGCD {\romannumeral0\xintgcd }%
\def\xintgcd #1#2{\xintiigcd {\xintNum{#1}}{\xintNum{#2}}}%
\def\xintiiGCD {\romannumeral0\xintiigcd }%
\def\xintiigcd #1%
{%
    \expandafter\XINT_iigcd\expandafter{\romannumeral0\xintiiabs{#1}}%
}%
\def\XINT_iigcd #1#2%
{%
    \expandafter\XINT_gcd_fork\romannumeral0\xintiiabs{#2}\Z #1\Z
}%
```

Ici #3#4=A, #1#2=B

```
\def\XINT_gcd_fork #1#2\Z #3#4\Z
{%
    \xint_UDzerofork
      #1\XINT_gcd_BisZero
      #3\XINT_gcd_AisZero
       0\XINT_gcd_loop
    \krof
    {#1#2}{#3#4}%
}%
\def\XINT_gcd_AisZero #1#2{ #1}%
\def\XINT_gcd_BisZero #1#2{ #2}%
```

```
\def\XINT_gcd_CheckRem #1#2\Z
{%
    \xint_gob_til_zero #1\XINT_gcd_end0\XINT_gcd_loop {#1#2}%
}%
\def\XINT_gcd_end0\XINT_gcd_loop #1#2{ #2}%
```

#1=B, #2=A. \XINT_div_prepare{#1}{#2} divides A by B.

```
\def\XINT_gcd_loop #1#2%
{%
    \expandafter\expandafter\expandafter
        \XINT_gcd_CheckRem
    \expandafter\xint_secondoftwo
    \romannumeral0\XINT_div_prepare {#1}{#2}\Z
    {#1}%
}%
```

7.4 \xintLCM, \xintiiLCM

See comments of \xintGCD.

```
\def\xintLCM {\romannumeral0\xintlcm}%
\def\xintlcm #1#2{\xintiilcm{\xintNum{#1}}{\xintNum{#2}}}%
\def\xintiiLCM {\romannumeral0\xintiilcm}%
\def\xintiilcm #1%
{%
    \expandafter\XINT_iilcm\expandafter{\romannumeral0\xintiiabs{#1}}%
}%
\def\XINT_iilcm #1#2%
{%
    \expandafter\XINT_lcm_fork\romannumeral0\xintiiabs{#2}\Z #1\Z
}%
\def\XINT_lcm_fork #1#2\Z #3#4\Z
{%
    \xint_UDzerofork
      #1\XINT_lcm_BisZero
      #3\XINT_lcm_AisZero
       0\expandafter
    \krof
    \XINT_lcm_notzero\expandafter{\romannumeral0\XINT_gcd_loop {#1#2}{#3#4}}%
    {#1#2}{#3#4}%
}%
\def\XINT_lcm_AisZero #1#2#3#4#5{ 0}%
\def\XINT_lcm_BisZero #1#2#3#4#5{ 0}%
\def\XINT_lcm_notzero #1#2#3{\xintiimul {#2}{\xintiiQuo{#3}{#1}}}%
```

7.5 \xintBezout

\xintBezout{#1}{#2} produces {U}{V}{D} with UA+VB=D, D = PGCD(A,B) (non-positive), where #1 and #2 f-expand to big integers A and B.

I had not checked this macro for about three years when I realized in January 2017 that \xintBezout{A}{B} was buggy for the cases A = 0 or B = 0. I fixed that blemish in 1.2l but overlooked the other blemish that \xintBezout{A}{B} with A multiple of B produced a coefficient U as -0 in place of 0.

Hence I rewrote again for 1.2p. On this occasion I modified the output of the macro to be {U}{V}{D} with AU+BV=D, formerly it was {A}{B}{U}{V}{D} with AU - BV = D. This is quite breaking change!

Note in particular change of sign of V.

I don't know why I had designed this macro to contain {A}{B} in its output. Perhaps I initially intended to output {A//D}{B//D} (but forgot), as this is actually possible from outcome of the last iteration, with no need of actually dividing. Current code however arranges to skip this last update, as U and V are already furnished by the iteration prior to realizing that the last non-zero remainder was found.

Also 1.2l raised InvalidOperation if both A and B vanished, but I removed this behaviour at 1.2p.

```
\def\xintBezout {\romannumeral0\xintbezout }%
\def\xintbezout #1%
{%
    \expandafter\XINT_bezout\expandafter {\romannumeral0\xintnum{#1}}%
}%
\def\XINT_bezout #1#2%
{%
    \expandafter\XINT_bezout_fork \romannumeral0\xintnum{#2}\Z #1\Z
}%
```

#3#4 = A, #1#2=B. Micro improvement for 1.2l.

```
\def\XINT_bezout_fork #1#2\Z #3#4\Z
{%
    \xint_UDzerosfork
     #1#3\XINT_bezout_botharezero
      #10\XINT_bezout_secondiszero
      #30\XINT_bezout_firstiszero
       00\xint_UDsignsfork
    \krof
          #1#3\XINT_bezout_minusminus % A < 0, B < 0
           #1-\XINT_bezout_minusplus  % A > 0, B < 0
           #3-\XINT_bezout_plusminus  % A < 0, B > 0
            --\XINT_bezout_plusplus   % A > 0, B > 0
    \krof
    {#2}{#4}#1#3% #1#2=B, #3#4=A
}%
\def\XINT_bezout_botharezero #1\krof#2#300{{0}{0}{0}}%
\def\XINT_bezout_firstiszero #1\krof#2#3#4#5%
{%
    \xint_UDsignfork
      #4{{0}{-1}{#2}}%
       -{{0}{1}{#4#2}}%
    \krof
}%
\def\XINT_bezout_secondiszero #1\krof#2#3#4#5%
{%
    \xint_UDsignfork
       #5{{-1}{0}{#3}}%
        -{{1}{0}{#5#3}}%
    \krof
}%
```

#4#2= A < 0, #3#1 = B < 0

```
\def\XINT_bezout_minusminus #1#2#3#4%
{%
    \expandafter\XINT_bezout_mm_post
    \romannumeral0\expandafter\XINT_bezout_preloop_a
    \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
}%
\def\XINT_bezout_mm_post #1#2%
{%
    \expandafter\XINT_bezout_mm_postb\expandafter
    {\romannumeral0\xintiiopp{#2}}{\romannumeral0\xintiiopp{#1}}%
}%
\def\XINT_bezout_mm_postb #1#2{\expandafter{#2}{#1}}%
```

minusplus #4#2= A > 0, B < 0

```
\def\XINT_bezout_minusplus #1#2#3#4%
{%
    \expandafter\XINT_bezout_mp_post
    \romannumeral0\expandafter\XINT_bezout_preloop_a
    \romannumeral0\XINT_div_prepare {#1}{#4#2}{#1}%
}%
\def\XINT_bezout_mp_post #1#2%
{%
    \expandafter\xint_exchangetwo_keepbraces\expandafter
    {\romannumeral0\xintiiopp {#2}}{#1}%
}%
```

plusminus A < 0, B > 0

```
\def\XINT_bezout_plusminus #1#2#3#4%
{%
    \expandafter\XINT_bezout_pm_post
    \romannumeral0\expandafter\XINT_bezout_preloop_a
    \romannumeral0\XINT_div_prepare {#3#1}{#2}{#3#1}%
}%
\def\XINT_bezout_pm_post #1{\expandafter{\romannumeral0\xintiiopp{#1}}}%
```

plusplus, B = #3#1 > 0, A = #4#2 > 0

```
\def\XINT_bezout_plusplus #1#2#3#4%
{%
    \expandafter\XINT_bezout_preloop_a
    \romannumeral0\XINT_div_prepare {#3#1}{#4#2}{#3#1}%
}%
```

n = 0: BA1001 (B, A, e=1, vv, uu, v, u)
r(1)=B, r(0)=A, après n étapes {r(n+1)}{r(n)}{vv}{uu}{v}{u}
q(n) quotient de r(n-1) par r(n)
si reste nul, exit et renvoie U = -e*uu, V = e*vv, A*U+B*V=D
sinon mise à jour

```
    vv, v = q * vv + v, vv
    uu, u = q * uu + u, uu
    e = -e
  puis calcul quotient reste et itération
```

We arrange for \xintiiMul sub-routine to be called only with positive arguments, thus skipping some un-needed sign parsing there. For that though we have to screen out the special cases A divides B, or B divides A. And we first want to exchange A and B if A < B. These special cases are the only one possibly leading to U or V zero (for A and B positive which is the case here.) Thus the general case always leads to non-zero U and V's and assigning a final sign is done simply adding a - to one of them, with no fear of producing -0.

```
\def\XINT_bezout_preloop_a #1#2#3%
{%
    \if0#1\xint_dothis\XINT_bezout_preloop_exchange\fi
    \if0#2\xint_dothis\XINT_bezout_preloop_exit\fi
    \xint_orthat{\expandafter\XINT_bezout_loop_B}%
    \romannumeral0\XINT_div_prepare {#2}{#3}{#2}{#1}110%
}%
\def\XINT_bezout_preloop_exit
    \romannumeral0\XINT_div_prepare #1#2#3#4#5#6#7%
{%
    {0}{1}{#2}%
}%
\def\XINT_bezout_preloop_exchange
{%
    \expandafter\xint_exchangetwo_keepbraces
    \romannumeral0\expandafter\XINT_bezout_preloop_A
}%
\def\XINT_bezout_preloop_A #1#2#3#4%
{%
    \if0#2\xint_dothis\XINT_bezout_preloop_exit\fi
    \xint_orthat{\expandafter\XINT_bezout_loop_B}%
    \romannumeral0\XINT_div_prepare {#2}{#3}{#2}{#1}%
}%
\def\XINT_bezout_loop_B #1#2%
{%
    \if0#2\expandafter\XINT_bezout_exitA
     \else\expandafter\XINT_bezout_loop_C
    \fi {#1}{#2}%
}%
```

We use the fact that the \romannumeral-`0 (or equivalent) done by \xintiiadd will absorb the initial space token left by \XINT_mul_plusplus in its output.

We arranged for operands here to be always positive which is needed for \XINT_mul_plusplus entry point (last time I checked...). Admittedly this kind of optimization is not good for maintenance of code, but I can't resist temptation of limiting the shuffling around of tokens...

```
\def\XINT_bezout_loop_C #1#2#3#4#5#6#7%
{%
    \expandafter\XINT_bezout_loop_D\expandafter
        {\romannumeral0\xintiiadd{\XINT_mul_plusplus{}{}#1\xint:#4\xint:}{#6}}%
        {\romannumeral0\xintiiadd{\XINT_mul_plusplus{}{}#1\xint:#5\xint:}{#7}}%
    {#2}{#3}{#4}{#5}%
}%
\def\XINT_bezout_loop_D #1#2%
{%
    \expandafter\XINT_bezout_loop_E\expandafter{#2}{#1}%
}%
\def\XINT_bezout_loop_E #1#2#3#4%
{%
    \expandafter\XINT_bezout_loop_b
    \romannumeral0\XINT_div_prepare {#3}{#4}{#3}{#2}{#1}%
}%
\def\XINT_bezout_loop_b #1#2%
{%
    \if0#2\expandafter\XINT_bezout_exita
     \else\expandafter\XINT_bezout_loop_c
    \fi {#1}{#2}%
}%
\def\XINT_bezout_loop_c #1#2#3#4#5#6#7%
{%
    \expandafter\XINT_bezout_loop_d\expandafter
        {\romannumeral0\xintiiadd{\XINT_mul_plusplus{}{}#1\xint:#4\xint:}{#6}}%
        {\romannumeral0\xintiiadd{\XINT_mul_plusplus{}{}#1\xint:#5\xint:}{#7}}%
    {#2}{#3}{#4}{#5}%
}%
\def\XINT_bezout_loop_d #1#2%
{%
    \expandafter\XINT_bezout_loop_e\expandafter{#2}{#1}%
}%
\def\XINT_bezout_loop_e #1#2#3#4%
{%
    \expandafter\XINT_bezout_loop_B
    \romannumeral0\XINT_div_prepare {#3}{#4}{#3}{#2}{#1}%
}%
```

sortir U, V, D mais on a travaillé avec vv, uu, v, u dans cet ordre.
The code is structured so that #4 and #5 are guaranteed non-zero if we exit here, hence we can not create a -0 in output.

```
\def\XINT_bezout_exita #1#2#3#4#5#6#7{{-#5}{#4}{#3}}%
\def\XINT_bezout_exitA #1#2#3#4#5#6#7{{#5}{-#4}{#3}}%
```

7.6 \xintEuclideAlgorithm

Pour Euclide: {N}{A}{D=r(n)}{B}{q1}{r1}{q2}{r2}{q3}{r3}....{qN}{rN=0}
u<2n> = u<2n+3>u<2n+2> + u<2n+4> à la n ième étape.
Formerly, used \xintiabs, but got deprecated at 1.2o.

```
\def\xintEuclideAlgorithm {\romannumeral0\xinteuclidealgorithm }%
\def\xinteuclidealgorithm #1%
{%
    \expandafter\XINT_euc\expandafter{\romannumeral0\xintiiabs{\xintNum{#1}}}%
}%
\def\XINT_euc #1#2%
{%
```

```
    \expandafter\XINT_euc_fork\romannumeral0\xintiiabs{\xintNum{#2}}\Z #1\Z
}%
```

Ici #3#4=A, #1#2=B

```
\def\XINT_euc_fork #1#2\Z #3#4\Z
{%
    \xint_UDzerofork
      #1\XINT_euc_BisZero
      #3\XINT_euc_AisZero
       0\XINT_euc_a
    \krof
    {0}{#1#2}{#3#4}{{#3#4}{#1#2}}{}\Z
}%
```

Le {} pour protéger {{A}{B}} si on s'arrête après une étape (B divise A). On va renvoyer: {N}{A}{D=r(n)}{B}{q1}{r1}{q2}{r2}{q3}{r3}....{qN}{rN=0}

```
\def\XINT_euc_AisZero #1#2#3#4#5#6{{1}{0}{#2}{#2}{0}{0}}%
\def\XINT_euc_BisZero #1#2#3#4#5#6{{1}{0}{#3}{#3}{0}{0}}%
```

{n}{rn}{an}{{qn}{rn}}...{{A}{B}}{}\Z
a(n) = r(n-1). Pour n=0 on a juste {0}{B}{A}{{A}{B}}{}\Z
\XINT_div_prepare {u}{v} divise v par u

```
\def\XINT_euc_a #1#2#3%
{%
    \expandafter\XINT_euc_b\the\numexpr #1+\xint_c_i\expandafter.%
    \romannumeral0\XINT_div_prepare {#2}{#3}{#2}%
}%
```

{n+1}{q(n+1)}{r(n+1)}{rn}{{qn}{rn}}...

```
\def\XINT_euc_b #1.#2#3#4%
{%
    \XINT_euc_c #3\Z {#1}{#3}{#4}{{#2}{#3}}%
}%
```

r(n+1)\Z {n+1}{r(n+1)}{r(n)}{{q(n+1)}{r(n+1)}}{{qn}{rn}}...
Test si r(n+1) est nul.

```
\def\XINT_euc_c #1#2\Z
{%
    \xint_gob_til_zero #1\XINT_euc_end0\XINT_euc_a
}%
```

{n+1}{r(n+1)}{r(n)}{{q(n+1)}{r(n+1)}}...{}\Z Ici r(n+1) = 0. On arrête on se prépare à inverser {n+1}{0}{r(n)}{{q(n+1)}{r(n+1)}}.....{{q1}{r1}}{{A}{B}}{}\Z
On veut renvoyer: {N=n+1}{A}{D=r(n)}{B}{q1}{r1}{q2}{r2}{q3}{r3}....{qN}{rN=0}

```
\def\XINT_euc_end0\XINT_euc_a #1#2#3#4\Z%
{%
    \expandafter\XINT_euc_end_a
    \romannumeral0%
    \XINT_rord_main {}#4{{#1}{#3}}%
    \xint:
      \xint_bye\xint_bye\xint_bye\xint_bye
      \xint_bye\xint_bye\xint_bye\xint_bye
    \xint:
}%
\def\XINT_euc_end_a #1#2#3{{#1}{#3}{#2}}%
```

7.7 \xintBezoutAlgorithm

Pour Bezout: objectif, renvoyer
{N}{A}{0}{1}{D=r(n)}{B}{1}{0}{q1}{r1}{alpha1=q1}{beta1=1}
{q2}{r2}{alpha2}{beta2}....{qN}{rN=0}{alphaN=A/D}{betaN=B/D}
alpha0=1, beta0=0, alpha(-1)=0, beta(-1)=1

```
\def\xintBezoutAlgorithm {\romannumeral0\xintbezoutalgorithm }%
\def\xintbezoutalgorithm #1%
{%
    \expandafter \XINT_bezalg
    \expandafter{\romannumeral0\xintiiabs{\xintNum{#1}}}%
}%
\def\XINT_bezalg #1#2%
{%
    \expandafter\XINT_bezalg_fork\romannumeral0\xintiiabs{\xintNum{#2}}\Z #1\Z
}%
```

Ici #3#4=A, #1#2=B

```
\def\XINT_bezalg_fork #1#2\Z #3#4\Z
{%
    \xint_UDzerofork
      #1\XINT_bezalg_BisZero
      #3\XINT_bezalg_AisZero
       0\XINT_bezalg_a
    \krof
    0{#1#2}{#3#4}1001{{#3#4}{#1#2}}{}\Z
}%
\def\XINT_bezalg_AisZero #1#2#3\Z{{1}{0}{0}{1}{#2}{#2}{1}{0}{0}{0}{0}{1}}%
\def\XINT_bezalg_BisZero #1#2#3#4\Z{{1}{0}{0}{1}{#3}{#3}{1}{0}{0}{0}{0}{1}}%
```

pour préparer l'étape n+1 il faut {n}{r(n)}{r(n-1)}{alpha(n)}{beta(n)}{alpha(n-1)}{beta(n-1)} {{q(n)}{r(n)}{alpha(n)}{beta(n)}}... division de #3 par #2

```
\def\XINT_bezalg_a #1#2#3%
{%
    \expandafter\XINT_bezalg_b\the\numexpr #1+\xint_c_i\expandafter.%
    \romannumeral0\XINT_div_prepare {#2}{#3}{#2}%
}%
```

{n+1}{q(n+1)}{r(n+1)}{r(n)}{alpha(n)}{beta(n)}{alpha(n-1)}{beta(n-1)}...

```
\def\XINT_bezalg_b #1.#2#3#4#5#6#7#8%
{%
    \expandafter\XINT_bezalg_c\expandafter
     {\romannumeral0\xintiiadd {\xintiiMul {#6}{#2}}{#8}}%
     {\romannumeral0\xintiiadd {\xintiiMul {#5}{#2}}{#7}}%
```

```
      {#1}{#2}{#3}{#4}{#5}{#6}%
}%
```

{beta(n+1)}{alpha(n+1)}{n+1}{q(n+1)}{r(n+1)}{r(n)}{alpha(n)}{beta(n}}

```
\def\XINT_bezalg_c #1#2#3#4#5#6%
{%
    \expandafter\XINT_bezalg_d\expandafter {#2}{#3}{#4}{#5}{#6}{#1}%
}%
```

{alpha(n+1)}{n+1}{q(n+1)}{r(n+1)}{r(n)}{beta(n+1)}

```
\def\XINT_bezalg_d #1#2#3#4#5#6#7#8%
{%
    \XINT_bezalg_e #4\Z {#2}{#4}{#5}{#1}{#6}{#7}{#8}{{#3}{#4}{#1}{#6}}%
}%
```

r(n+1)\Z {n+1}{r(n+1)}{r(n)}{alpha(n+1)}{beta(n+1)}
{alpha(n)}{beta(n)}{q,r,alpha,beta(n+1)}
Test si r(n+1) est nul.

```
\def\XINT_bezalg_e #1#2\Z
{%
    \xint_gob_til_zero #1\XINT_bezalg_end0\XINT_bezalg_a
}%
```

Ici r(n+1) = 0. On arrête on se prépare à inverser.
{n+1}{r(n+1)}{r(n)}{alpha(n+1)}{beta(n+1)}{alpha(n)}{beta(n)}
{q,r,alpha,beta(n+1)}...{{A}{B}}{}\Z
On veut renvoyer
{N}{A}{0}{1}{D=r(n)}{B}{1}{0}{q1}{r1}{alpha1=q1}{beta1=1}
{q2}{r2}{alpha2}{beta2}....{qN}{rN=0}{alphaN=A/D}{betaN=B/D}

```
\def\XINT_bezalg_end0\XINT_bezalg_a #1#2#3#4#5#6#7#8\Z
{%
    \expandafter\XINT_bezalg_end_a
    \romannumeral0%
    \XINT_rord_main {}#8{{#1}{#3}}%
    \xint:
      \xint_bye\xint_bye\xint_bye\xint_bye
      \xint_bye\xint_bye\xint_bye\xint_bye
    \xint:
}%
```

{N}{D}{A}{B}{q1}{r1}{alpha1=q1}{beta1=1}{q2}{r2}{alpha2}{beta2}
....{qN}{rN=0}{alphaN=A/D}{betaN=B/D}
On veut renvoyer
{N}{A}{0}{1}{D=r(n)}{B}{1}{0}{q1}{r1}{alpha1=q1}{beta1=1}
{q2}{r2}{alpha2}{beta2}....{qN}{rN=0}{alphaN=A/D}{betaN=B/D}

```
\def\XINT_bezalg_end_a #1#2#3#4{{#1}{#3}{0}{1}{#2}{#4}{1}{0}}%
```

7.8 \xintGCDof

1.21 adds protection against items being non-terminated \the\numexpr...

```
\def\xintGCDof      {\romannumeral0\xintgcdof }%
\def\xintgcdof    #1{\expandafter\XINT_gcdof_a\romannumeral`&&@#1\xint:}%
\def\XINT_gcdof_a #1{\expandafter\XINT_gcdof_b\romannumeral`&&@#1!}%
\def\XINT_gcdof_b #1!#2{\expandafter\XINT_gcdof_c\romannumeral`&&@#2!{#1}!}%
\def\XINT_gcdof_c #1{\xint_gob_til_xint: #1\XINT_gcdof_e\xint:\XINT_gcdof_d #1}%
\def\XINT_gcdof_d #1!{\expandafter\XINT_gcdof_b\romannumeral0\xintgcd {#1}}%
\def\XINT_gcdof_e #1!#2!{ #2}%
```

7.9 \xintLCMof

New with 1.09a

1.21 adds protection against items being non-terminated \the\numexpr...

```
\def\xintLCMof      {\romannumeral0\xintlcmof }%
\def\xintlcmof    #1{\expandafter\XINT_lcmof_a\romannumeral`&&@#1\xint:}%
\def\XINT_lcmof_a #1{\expandafter\XINT_lcmof_b\romannumeral`&&@#1!}%
\def\XINT_lcmof_b #1!#2{\expandafter\XINT_lcmof_c\romannumeral`&&@#2!{#1}!}%
\def\XINT_lcmof_c #1{\xint_gob_til_xint: #1\XINT_lcmof_e\xint:\XINT_lcmof_d #1}%
\def\XINT_lcmof_d #1!{\expandafter\XINT_lcmof_b\romannumeral0\xintlcm {#1}}%
\def\XINT_lcmof_e #1!#2!{ #2}%
```

7.10 \xintTypesetEuclideAlgorithm

TYPESETTING
  Organisation:
  {N}{A}{D}{B}{q1}{r1}{q2}{r2}{q3}{r3}....{qN}{rN=0}
\U1 = N = nombre d'étapes, \U3 = PGCD, \U2 = A, \U4=B q1 = \U5, q2 = \U7 --> qn = \U<2n+3>, rn = \U<2n+4> bn = rn. B = r0. A=r(-1)
  r(n-2) = q(n)r(n-1)+r(n) (n e étape)
  \U{2n} = \U{2n+3} \times \U{2n+2} + \U{2n+4}, n e étape. (avec n entre 1 et N)
  1.09h uses \xintloop, and \par rather than \endgraf; and \par rather than \hfill\break

```
\def\xintTypesetEuclideAlgorithm {%
    \unless\ifdefined\xintAssignArray
       \errmessage
        {xintgcd: package xinttools is required for \string\xintTypesetEuclideAlgorithm}%
       \expandafter\xint_gobble_iii
    \fi
    \XINT_TypesetEuclideAlgorithm
}%
\def\XINT_TypesetEuclideAlgorithm #1#2%
{% l'algo remplace #1 et #2 par |#1| et |#2|
  \par
  \begingroup
    \xintAssignArray\xintEuclideAlgorithm {#1}{#2}\to\U
    \edef\A{\U2}\edef\B{\U4}\edef\N{\U1}%
    \setbox 0 \vbox{\halign {$##$\cr \A\cr \B \cr}}%
    \count 255 1
    \xintloop
      \indent\hbox to \wd 0 {\hfil$\U{\numexpr 2*\count255\relax}$}%
      ${} =  \U{\numexpr 2*\count255 + 3\relax}
      \times \U{\numexpr 2*\count255 + 2\relax}
          +  \U{\numexpr 2*\count255 + 4\relax}$%
    \ifnum \count255 < \N
      \par
      \advance \count255 1
    \repeat
  \endgroup
}%
```

7.11 \xintTypesetBezoutAlgorithm

Pour Bezout on a: {N}{A}{0}{1}{D=r(n)}{B}{1}{0}{q1}{r1}{alpha1=q1}{beta1=1}
{q2}{r2}{alpha2}{beta2}....{qN}{rN=0}{alphaN=A/D}{betaN=B/D}
  Donc 4N+8 termes: U1 = N, U2= A, U5=D, U6=B, q1 = U9, qn = U{4n+5}, n au moins 1
rn = U{4n+6}, n au moins -1
alpha(n) = U{4n+7}, n au moins -1
beta(n) = U{4n+8}, n au moins -1
  1.09h uses \xintloop, and \par rather than \endgraf; and no more \parindent0pt

```
\def\xintTypesetBezoutAlgorithm {%
    \unless\ifdefined\xintAssignArray
       \errmessage
        {xintgcd: package xinttools is required for \string\xintTypesetBezoutAlgorithm}%
       \expandafter\xint_gobble_iii
    \fi
    \XINT_TypesetBezoutAlgorithm
}%
\def\XINT_TypesetBezoutAlgorithm #1#2%
{%
  \par
  \begingroup
    \xintAssignArray\xintBezoutAlgorithm {#1}{#2}\to\BEZ
    \edef\A{\BEZ2}\edef\B{\BEZ6}\edef\N{\BEZ1}% A = |#1|, B = |#2|
    \setbox 0 \vbox{\halign {$##$\cr \A\cr \B \cr}}%
    \count255 1
    \xintloop
      \indent\hbox to \wd 0 {\hfil$\BEZ{4*\count255 - 2}$}%
      ${} =  \BEZ{4*\count255 + 5}
      \times \BEZ{4*\count255 + 2}
          +  \BEZ{4*\count255 + 6}$\hfill\break
      \hbox to \wd 0 {\hfil$\BEZ{4*\count255 +7}$}%
      ${} = \BEZ{4*\count255 + 5}
      \times \BEZ{4*\count255 + 3}
          +  \BEZ{4*\count255 - 1}$\hfill\break
      \hbox to \wd 0 {\hfil$\BEZ{4*\count255 +8}$}%
      ${} =  \BEZ{4*\count255 + 5}
      \times \BEZ{4*\count255 + 4}
          +  \BEZ{4*\count255 }$
      \par
    \ifnum \count255 < \N
    \advance \count255 1
  \repeat
    \edef\U{\BEZ{4*\N + 4}}%
    \edef\V{\BEZ{4*\N + 3}}%
    \edef\D{\BEZ5}%
    \ifodd\N
       $\U\times\A  - \V\times \B = -\D$%
    \else
       $\U\times\A - \V\times\B = \D$%
    \fi
    \par
  \endgroup
}%
\XINT_restorecatcodes_endinput%
```

8 Package xintfrac implementation

The commenting is currently (2018/06/17) very sparse.

8.1 Catcodes, ε-TEX and reload detection

The code for reload detection was initially copied from HEIKO OBERDIEK's packages, then modified.

The method for catcodes was also initially directly inspired by these packages.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode64=11   % @
  \catcode35=6    % #
  \catcode44=12   % ,
  \catcode45=12   % -
  \catcode46=12   % .
  \catcode58=12   % :
  \let\z\endgroup
  \expandafter\let\expandafter\x\csname ver@xintfrac.sty\endcsname
  \expandafter\let\expandafter\w\csname ver@xint.sty\endcsname
  \expandafter
    \ifx\csname PackageInfo\endcsname\relax
      \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
    \else
      \def\y#1#2{\PackageInfo{#1}{#2}}%
    \fi
  \expandafter
  \ifx\csname numexpr\endcsname\relax
     \y{xintfrac}{\numexpr not available, aborting input}%
     \aftergroup\endinput
  \else
    \ifx\x\relax   % plain-TeX, first loading of xintfrac.sty
      \ifx\w\relax % but xint.sty not yet loaded.
         \def\z{\endgroup\input xint.sty\relax}%
      \fi
    \else
      \def\empty {}%
      \ifx\x\empty % LaTeX, first loading,
      % variable is initialized, but \ProvidesPackage not yet seen
          \ifx\w\relax % xint.sty not yet loaded.
            \def\z{\endgroup\RequirePackage{xint}}%
          \fi
      \else
        \aftergroup\endinput % xintfrac already loaded.
      \fi
    \fi
  \fi
\z%
\XINTsetupcatcodes% defined in xintkernel.sty
```

8.2 Package identification

```
\XINT_providespackage
\ProvidesPackage{xintfrac}%
  [2018/06/17 1.3c Expandable operations on fractions (JFB)]%
```

8.3 \XINT_cntSgnFork

1.09i. Used internally, #1 must expand to \m@ne, \z@, or \@ne or equivalent. \XINT_cntSgnFork does not insert a romannumeral stopper.

```
\def\XINT_cntSgnFork #1%
{%
    \ifcase #1\expandafter\xint_secondofthree
            \or\expandafter\xint_thirdofthree
          \else\expandafter\xint_firstofthree
    \fi
}%
```

8.4 \xintLen

The used formula is disputable, the idea is that A/1 and A should have same length. Venerable code rewritten for 1.2i, following updates to \xintLength in xintkernel.sty. And sadly, I forgot on this occasion that this macro is not supposed to count the sign... Fixed in 1.2k.

```
\def\xintLen {\romannumeral0\xintlen }%
\def\xintlen #1%
{%
    \expandafter\XINT_flen\romannumeral0\XINT_infrac {#1}%
}%
\def\XINT_flen#1{\def\XINT_flen ##1##2##3%
{%
    \expandafter#1%


    \the\numexpr \XINT_abs##1+%
    \XINT_len_fork ##2##3\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
      \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
      \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye-\xint_c_i
    \relax
}}\XINT_flen{ }%
```

8.5 \XINT_outfrac

Months later (2014/10/22): perhaps I should document what this macro does before I forget? from {e}{N}{D} it outputs N/D[e], checking in passing if D=0 or if N=0. It also makes sure D is not < 0. I am not sure but I don't think there is any place in the code which could call \XINT_outfrac with a D < 0, but I should check.

```
\def\XINT_outfrac #1#2#3%
{%
    \ifcase\XINT_cntSgn #3\xint:
        \expandafter \XINT_outfrac_divisionbyzero
    \or
        \expandafter \XINT_outfrac_P
    \else
        \expandafter \XINT_outfrac_N
    \fi
    {#2}{#3}[#1]%
}%
\def\XINT_outfrac_divisionbyzero #1#2%
```

```
{%
    \XINT_signalcondition{DivisionByZero}{Division of #1 by #2}{}{0/1[0]}%
}%
\def\XINT_outfrac_P#1{%
\def\XINT_outfrac_P ##1##2%
   {\if0\XINT_Sgn ##1\xint:\expandafter\XINT_outfrac_Zero\fi#1##1/##2}%
}\XINT_outfrac_P{ }%
\def\XINT_outfrac_Zero #1[#2]{ 0/1[0]}%
\def\XINT_outfrac_N #1#2%
{%
    \expandafter\XINT_outfrac_N_a\expandafter
    {\romannumeral0\XINT_opp #2}{\romannumeral0\XINT_opp #1}%
}%
\def\XINT_outfrac_N_a #1#2%
{%
    \expandafter\XINT_outfrac_P\expandafter {#2}{#1}%
}%
```

8.6 \XINT_inFrac

Parses fraction, scientific notation, etc... and produces {n}{A}{B} corresponding to A/B times 10^n. No reduction to smallest terms.

Extended in 1.07 to accept scientific notation on input. With lowercase e only. The \xintexpr parser does accept uppercase E also. Ah, by the way, perhaps I should at least say what this macro does? (belated addition 2014/10/22...), before I forget! It prepares the fraction in the internal format {exponent}{Numerator}{Denominator} where Denominator is at least 1.

2015/10/09: this venerable macro from the very early days (1.03, 2013/04/14) has gotten a lifting for release 1.2. There were two kinds of issues:

1) use of \W, \Z, \T delimiters was very poor choice as this could clash with user input,

2) the new \XINT_frac_gen handles macros (possibly empty) in the input as general as \A.\Be\C/\D.\Ee\F. The earlier version would not have expanded the \B or \E: digits after decimal mark were constrained to arise from expansion of the first token. Thus the 1.03 original code would have expanded only \A, \D, \C, and \F for this input.

This reminded me think I should revisit the remaining earlier portions of code, as I was still learning TeX coding when I wrote them.

Also I thought about parsing even faster the A/B[N] input, not expanding B, but this turned out to clash with some established uses in the documentation such as 1/\xintiiSqr{...}[0]. For the implementation, careful here about potential brace removals with parameter patterns such as like #1/#2#3[#4]for example.

While I was at it 1.2 added \numexpr parsing of the N, which earlier was restricted to be only explicit digits. I allowed [] with empty N, but the way I did it in 1.2 with \the\numexpr 0#1 was buggy, as it did not allow #1 to be a \count for example or itself a \numexpr (although such inputs were not previously allowed, I later turned out to use them in the code itself, e.g. the float factorial of version 1.2f). The better way would be \the\numexpr#1+\xint_c_ but 1.2f finally does only \the\numexpr #1 and #1 is not allowed to be empty.

The 1.2 \XINT_frac_gen had two locations with such a problematic \numexpr 0#1 which I replaced for 1.2f with \numexpr#1+\xint_c_.

Regarding calling the macro with an argument A[<expression>], a / inthe expression must be suitably hidden for example in \firstofone type constructs.

Note: when the numerator is found to be zero \XINT_inFrac *always* returns {0}{0}{1}. This behaviour must not change because 1.2g \xintFloat and XINTinFloat (for example) rely upon it: if the denominator on output is not 1, then \xintFloat assumes that the numerator is not zero.

As described in the manual, if the input contains a (final) [N] part, it is assumed that it is in the shape A[N] or A/B[N] with A (and B) not containing neither decimal mark nor scientific part, moreover B must be positive and A have at most one minus sign (and no plus sign). Else there will be errors, for example -0/2[0] would not be recognized as being zero at this stage and this could cause issues afterwards. When there is no ending [N] part, both numerator and denominator will be parsed for the more general format allowing decimal digits and scientific part and possibly multiple leading signs.

1.2l fixes frailty of \XINT_infrac (hence basically of all xintfrac macros) respective to non terminated \numexpr input: \xintRaw{\the\numexpr1} for example. The issue was that \numexpr sees the / and expands what's next. But even \numexpr 1// for example creates an error, and to my mind this is a defect of \numexpr. It should be able to trace back and see that / was used as delimiter not as operator. Anyway, I thus fixed this problem belatedly here regarding \XINT_infrac.

```
\def\XINT_inFrac {\romannumeral0\XINT_infrac }%
\def\XINT_infrac #1%
{%
    \expandafter\XINT_infrac_fork\romannumeral`&&@#1\xint:/\XINT_W[\XINT_W\XINT_T
}%
\def\XINT_infrac_fork #1[#2%
{%
    \xint_UDXINTWfork
      #2\XINT_frac_gen          % input has no brackets [N]
      \XINT_W\XINT_infrac_res_a % there is some [N], must be strict A[N] or A/B[N] input
    \krof
    #1[#2%
}%
\def\XINT_infrac_res_a #1%
{%
    \xint_gob_til_zero #1\XINT_infrac_res_zero 0\XINT_infrac_res_b #1%
}%
```

Note that input exponent is here ignored and forced to be zero.

```
\def\XINT_infrac_res_zero 0\XINT_infrac_res_b #1\XINT_T {{0}{0}{1}}%
\def\XINT_infrac_res_b #1/#2%
{%
    \xint_UDXINTWfork
     #2\XINT_infrac_res_ca      % it was A[N] input
     \XINT_W\XINT_infrac_res_cb % it was A/B[N] input
    \krof
    #1/#2%
}%
```

An empty [] is not allowed. (this was authorized in 1.2, removed in 1.2f). As nobody reads xint documentation, no one will have noticed the fleeting possibility.

```
\def\XINT_infrac_res_ca #1[#2]\xint:/\XINT_W[\XINT_W\XINT_T
   {\expandafter{\the\numexpr #2}{#1}{1}}%
\def\XINT_infrac_res_cb #1/#2[%
   {\expandafter\XINT_infrac_res_cc\romannumeral`&&@#2~#1[}%
\def\XINT_infrac_res_cc #1~#2[#3]\xint:/\XINT_W[\XINT_W\XINT_T
   {\expandafter{\the\numexpr #3}{#2}{#1}}%
```

8.7 \XINT_frac_gen

Extended in 1.07 to recognize and accept scientific notation both at the numerator and (possible) denominator. Only a lowercase e will do here, but uppercase E is possible within an \xintexpr..\relax

Completely rewritten for 1.2 2015/10/10. The parsing handles inputs such as \A.\Be\C/\D.\Ee\F where each of \A, \B, \D, and \E may need f-expansion and \C and \F will end up in \numexpr.

1.2f corrects an issue to allow \C and \F to be \count variable (or expressions with \numexpr): 1.2 did a bad \numexpr0#1 which allowed only explicit digits for expanded #1.

```
\def\XINT_frac_gen #1/#2%
{%
    \xint_UDXINTWfork
      #2\XINT_frac_gen_A      % there was no /
      \XINT_W\XINT_frac_gen_B % there was a /
    \krof
    #1/#2%
}%
```

Note that #1 is only expanded so far up to decimal mark or "e".

```
\def\XINT_frac_gen_A #1\xint:/\XINT_W [\XINT_W {\XINT_frac_gen_C 0~1!#1ee.\XINT_W }%
\def\XINT_frac_gen_B #1/#2\xint:/\XINT_W[%\XINT_W
{%
    \expandafter\XINT_frac_gen_Ba
    \romannumeral`&&@#2ee.\XINT_W\XINT_Z #1ee.%\XINT_W
}%
\def\XINT_frac_gen_Ba #1.#2%
{%
    \xint_UDXINTWfork
      #2\XINT_frac_gen_Bb
      \XINT_W\XINT_frac_gen_Bc
    \krof
    #1.#2%
}%
\def\XINT_frac_gen_Bb #1e#2e#3\XINT_Z
                {\expandafter\XINT_frac_gen_C\the\numexpr #2+\xint_c_~#1!}%
\def\XINT_frac_gen_Bc #1.#2e%
{%
    \expandafter\XINT_frac_gen_Bd\romannumeral`&&@#2.#1e%
}%
\def\XINT_frac_gen_Bd #1.#2e#3e#4\XINT_Z
{%
    \expandafter\XINT_frac_gen_C\the\numexpr #3-%
    \numexpr\XINT_length_loop
    #1\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
      \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
      \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
    ~#2#1!%
}%
\def\XINT_frac_gen_C #1!#2.#3%
{%
    \xint_UDXINTWfork
```

```
      #3\XINT_frac_gen_Ca
      \XINT_W\XINT_frac_gen_Cb
    \krof
    #1!#2.#3%
}%
\def\XINT_frac_gen_Ca #1~#2!#3e#4e#5\XINT_T
{%
    \expandafter\XINT_frac_gen_F\the\numexpr #4-#1\expandafter
    ~\romannumeral0\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
     #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z~#3~%
}%
\def\XINT_frac_gen_Cb #1.#2e%
{%
    \expandafter\XINT_frac_gen_Cc\romannumeral`&&@#2.#1e%
}%
\def\XINT_frac_gen_Cc #1.#2~#3!#4e#5e#6\XINT_T
{%
    \expandafter\XINT_frac_gen_F\the\numexpr #5-#2-%


    \numexpr\XINT_length_loop
    #1\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
      \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
      \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye


    \relax\expandafter~%
    \romannumeral0\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
    #3\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z
    ~#4#1~%
}%
\def\XINT_frac_gen_F #1~#2%
{%
    \xint_UDzerominusfork
      #2-\XINT_frac_gen_Gdivbyzero
      0#2{\XINT_frac_gen_G  -{}}%
       0-{\XINT_frac_gen_G {}#2}%
    \krof  #1~%
}%
\def\XINT_frac_gen_Gdivbyzero #1~~#2~%
{%
   \expandafter\XINT_frac_gen_Gdivbyzero_a
   \romannumeral0\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
   #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z~#1~%
}%
\def\XINT_frac_gen_Gdivbyzero_a #1~#2~%
{%
    \XINT_signalcondition{DivisionByZero}{Division of #1 by zero}{}{{#2}{#1}{0}}%
}%
\def\XINT_frac_gen_G #1#2#3~#4~#5~%
{%
    \expandafter\XINT_frac_gen_Ga
    \romannumeral0\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
    #1#5\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z~#3~{#2#4}%
```

```
}%
\def\XINT_frac_gen_Ga #1#2~#3~%
{%
    \xint_gob_til_zero #1\XINT_frac_gen_zero 0%
    {#3}{#1#2}%
}%
\def\XINT_frac_gen_zero 0#1#2#3{{0}{0}{1}}%
```

8.8 \XINT_factortens

This is the core macro for \xintREZ. To be used as \romannumeral0\XINT_factortens{...}. Output is A.N. (formerly {A}{N}) where A is the integer stripped from trailing zeroes and N is the number of removed zeroes. Only for positive strict integers!

Completely rewritten at 1.3a to replace a double \xintReverseOrder by a direct \numexpr governed expansion to the end and back, à la 1.2. I should comment more... and perhaps improve again in future.

Testing shows significant gain at 100 digits or more.

```
\def\XINT_factortens #1{\expandafter\XINT_factortens_z
                        \romannumeral0\XINT_factortens_a#1%
                        \XINT_factortens_b123456789.}%
\def\XINT_factortens_z.\XINT_factortens_y{ }%
\def\XINT_factortens_a #1#2#3#4#5#6#7#8#9%
   {\expandafter\XINT_factortens_x
    \the\numexpr 1#1#2#3#4#5#6#7#8#9\XINT_factortens_a}%
\def\XINT_factortens_b#1\XINT_factortens_a#2#3.%
   {.\XINT_factortens_cc 000000000-#2.}%
\def\XINT_factortens_x1#1.#2{#2#1}%
\def\XINT_factortens_y{.\XINT_factortens_y}%
\def\XINT_factortens_cc #1#2#3#4#5#6#7#8#9%
   {\if#90\xint_dothis
     {\expandafter\XINT_factortens_d\the\numexpr #8#7#6#5#4#3#2#1\relax
      \xint_c_i 2345678.}\fi
    \xint_orthat{\XINT_factortens_yy{#1#2#3#4#5#6#7#8#9}}}%
\def\XINT_factortens_yy #1#2.{.\XINT_factortens_y#1.0.}%
\def\XINT_factortens_c #1#2#3#4#5#6#7#8#9%
   {\if#90\xint_dothis
     {\expandafter\XINT_factortens_d\the\numexpr #8#7#6#5#4#3#2#1\relax
      \xint_c_i 2345678.}\fi
    \xint_orthat{.\XINT_factortens_y #1#2#3#4#5#6#7#8#9.}}%
\def\XINT_factortens_d #1#2#3#4#5#6#7#8#9%
   {\if#10\expandafter\XINT_factortens_e\fi
    \XINT_factortens_f #9#9#8#7#6#5#4#3#2#1.}%
\def\XINT_factortens_f #1#2\xint_c_i#3.#4.#5.%
   {\expandafter\XINT_factortens_g\the\numexpr#1+#5.#3.}%
\def\XINT_factortens_g #1.#2.{.\XINT_factortens_y#2.#1.}%
\def\XINT_factortens_e #1..#2.%
   {\expandafter.\expandafter\XINT_factortens_c
    \the\numexpr\xint_c_ix+#2.}%
```

8.9 \xintEq, \xintNotEq, \xintGt, \xintLt, \xintGtorEq, \xintLtorEq, \xintIsZero, \xintIsNotZero, \xintOdd, \xintEven, \xintifSgn, \xintifCmp, \xintifEq, \xintifGt, \xintifLt, \xintifZero, \xintifNotZero, \xintifOne, \xintifOdd

Moved here at 1.3. Formerly these macros were already defined in xint.sty or even xintcore.sty. They are slim wrappers of macros defined elsewhere in xintfrac.

```
\def\xintEq   {\romannumeral0\xinteq }%
\def\xinteq   #1#2{\xintifeq{#1}{#2}{1}{0}}%
\def\xintNotEq#1#2{\romannumeral0\xintifeq {#1}{#2}{0}{1}}%
\def\xintGt {\romannumeral0\xintgt }%
\def\xintgt #1#2{\xintifgt{#1}{#2}{1}{0}}%
\def\xintLt   {\romannumeral0\xintlt }%
\def\xintlt #1#2{\xintiflt{#1}{#2}{1}{0}}%
\def\xintGtorEq #1#2{\romannumeral0\xintiflt {#1}{#2}{0}{1}}%
\def\xintLtorEq #1#2{\romannumeral0\xintifgt {#1}{#2}{0}{1}}%
\def\xintIsZero   {\romannumeral0\xintiszero }%
\def\xintiszero #1{\if0\xintSgn{#1}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%
\def\xintIsNotZero{\romannumeral0\xintisnotzero }%
\def\xintisnotzero
          #1{\if0\xintSgn{#1}\xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi}%
\def\xintOdd      {\romannumeral0\xintodd }%
\def\xintodd #1%
{%
    \ifodd\xintLDg{\xintNum{#1}} %<- intentional space
        \xint_afterfi{ 1}%
    \else
        \xint_afterfi{ 0}%
    \fi
}%
\def\xintEven     {\romannumeral0\xinteven }%
\def\xinteven #1%
{%
    \ifodd\xintLDg{\xintNum{#1}} %<- intentional space
        \xint_afterfi{ 0}%
    \else
        \xint_afterfi{ 1}%
    \fi
}%
\def\xintifSgn{\romannumeral0\xintifsgn }%
\def\xintifsgn #1%
{%
    \ifcase \xintSgn{#1}
               \expandafter\xint_secondofthree_thenstop
            \or\expandafter\xint_thirdofthree_thenstop
          \else\expandafter\xint_firstofthree_thenstop
    \fi
}%
\def\xintifCmp{\romannumeral0\xintifcmp }%
\def\xintifcmp #1#2%
{%
    \ifcase\xintCmp {#1}{#2}
               \expandafter\xint_secondofthree_thenstop
            \or\expandafter\xint_thirdofthree_thenstop
          \else\expandafter\xint_firstofthree_thenstop
    \fi
}%
\def\xintifEq {\romannumeral0\xintifeq }%
\def\xintifeq #1#2%
{%
    \if0\xintCmp{#1}{#2}%
               \expandafter\xint_firstoftwo_thenstop
          \else\expandafter\xint_secondoftwo_thenstop
    \fi
}%
\def\xintifGt {\romannumeral0\xintifgt }%
\def\xintifgt #1#2%
{%
    \if1\xintCmp{#1}{#2}%
               \expandafter\xint_firstoftwo_thenstop
          \else\expandafter\xint_secondoftwo_thenstop
    \fi
}%
\def\xintifLt {\romannumeral0\xintiflt }%
\def\xintiflt #1#2%
{%
    \ifnum\xintCmp{#1}{#2}<\xint_c_
          \expandafter\xint_firstoftwo_thenstop
    \else \expandafter\xint_secondoftwo_thenstop
    \fi
}%
\def\xintifZero   {\romannumeral0\xintifzero }%
\def\xintifzero #1%
{%
    \if0\xintSgn{#1}%
       \expandafter\xint_firstoftwo_thenstop
    \else
       \expandafter\xint_secondoftwo_thenstop
    \fi
}%
\def\xintifNotZero{\romannumeral0\xintifnotzero }%
\def\xintifnotzero #1%
{%
    \if0\xintSgn{#1}%
       \expandafter\xint_secondoftwo_thenstop
    \else
       \expandafter\xint_firstoftwo_thenstop
    \fi
}%
\def\xintifOne {\romannumeral0\xintifone }%
\def\xintifone #1%
{%
    \if1\xintIsOne{#1}%
       \expandafter\xint_firstoftwo_thenstop
    \else
```

```
        \expandafter\xint_secondoftwo_thenstop
     \fi
}%
\def\xintifOdd {\romannumeral0\xintifodd }%
\def\xintifodd #1%
{%
     \if\xintOdd{#1}1%
        \expandafter\xint_firstoftwo_thenstop
     \else
        \expandafter\xint_secondoftwo_thenstop
     \fi
}%
```

8.10 \xintRaw

1.07: this macro simply prints in a user readable form the fraction after its initial scanning. Useful when put inside braces in an \xintexpr, when the input is not yet in the A/B[n] form.

```
\def\xintRaw {\romannumeral0\xintraw }%
\def\xintraw
{%
     \expandafter\XINT_raw\romannumeral0\XINT_infrac
}%
\def\XINT_raw #1#2#3{ #2/#3[#1]}%
```

8.11 \xintPRaw

1.09b

```
\def\xintPRaw {\romannumeral0\xintpraw }%
\def\xintpraw
{%
     \expandafter\XINT_praw\romannumeral0\XINT_infrac
}%
\def\XINT_praw #1%
{%
     \ifnum #1=\xint_c_ \expandafter\XINT_praw_a\fi \XINT_praw_A {#1}%
}%
\def\XINT_praw_A #1#2#3%
{%
     \if\XINT_isOne{#3}1\expandafter\xint_firstoftwo
                  \else\expandafter\xint_secondoftwo
     \fi { #2[#1]}{ #2/#3[#1]}%
}%
\def\XINT_praw_a\XINT_praw_A #1#2#3%
{%
     \if\XINT_isOne{#3}1\expandafter\xint_firstoftwo
                  \else\expandafter\xint_secondoftwo
     \fi { #2}{ #2/#3}%
}%
```

8.12 \xintRawWithZeros

This was called \xintRaw in versions earlier than 1.07

```
\def\xintRawWithZeros {\romannumeral0\xintrawwithzeros }%
\def\xintrawwithzeros
{%
    \expandafter\XINT_rawz_fork\romannumeral0\XINT_infrac
}%
```

```
\def\XINT_rawz_fork #1%
{%
    \ifnum#1<\xint_c_
      \expandafter\XINT_rawz_Ba
    \else
      \expandafter\XINT_rawz_A
    \fi
    #1.%
}%
\def\XINT_rawz_A  #1.#2#3{\XINT_dsx_addzeros{#1}#2;/#3}%
\def\XINT_rawz_Ba -#1.#2#3{\expandafter\XINT_rawz_Bb
    \expandafter{\romannumeral0\XINT_dsx_addzeros{#1}#3;}{#2}}%
\def\XINT_rawz_Bb #1#2{ #2/#1}%
```

8.13 \xintDecToString

1.3. This is a backport from polexpr 0.4. It is definitely not in final form, consider it to be an unstable macro.

```
\def\xintDecToString{\romannumeral0\xintdectostring}%
\def\xintdectostring#1{\expandafter\XINT_dectostr\romannumeral0\xintraw{#1}}%
\def\XINT_dectostr #1/#2[#3]{\xintiiifZero {#1}%
      \XINT_dectostr_z
      {\if1\XINT_isOne{#2}\expandafter\XINT_dectostr_a
                     \else\expandafter\XINT_dectostr_b
       \fi}%
  #1/#2[#3]%
}%
\def\XINT_dectostr_z#1[#2]{ 0}%
\def\XINT_dectostr_a#1/#2[#3]{%
    \ifnum#3<\xint_c_\xint_dothis{\xinttrunc{-#3}{#1[#3]}}\fi
    \xint_orthat{\xintiie{#1}{#3}}%
}%
\def\XINT_dectostr_b#1/#2[#3]{% just to handle this somehow
    \ifnum#3<\xint_c_\xint_dothis{\xinttrunc{-#3}{#1[#3]}/#2}\fi
    \xint_orthat{\xintiie{#1}{#3}/#2}%
}%
```

8.14 \xintFloor, \xintiFloor

1.09a, 1.1 for \xintiFloor/\xintFloor. Not efficient if big negative decimal exponent. Also sub-efficient if big positive decimal exponent.

```
\def\xintFloor {\romannumeral0\xintfloor }%
\def\xintfloor #1% devrais-je faire \xintREZ?
    {\expandafter\XINT_ifloor \romannumeral0\xintrawwithzeros {#1}./1[0]}%
\def\xintiFloor {\romannumeral0\xintifloor }%
\def\xintifloor #1%
    {\expandafter\XINT_ifloor \romannumeral0\xintrawwithzeros {#1}.}%
\def\XINT_ifloor #1/#2.{\xintiiquo {#1}{#2}}%
```

8.15 \xintCeil, \xintiCeil

1.09a

```
\def\xintCeil {\romannumeral0\xintceil }%
\def\xintceil #1{\xintiiopp {\xintFloor {\xintOpp{#1}}}}%
\def\xintiCeil {\romannumeral0\xinticeil }%
\def\xinticeil #1{\xintiiopp {\xintiFloor {\xintOpp{#1}}}}%
```

8.16 \xintNumerator

```
\def\xintNumerator {\romannumeral0\xintnumerator }%
\def\xintnumerator
{%
    \expandafter\XINT_numer\romannumeral0\XINT_infrac
}%
\def\XINT_numer #1%
{%
    \ifcase\XINT_cntSgn #1\xint:
      \expandafter\XINT_numer_B
    \or
      \expandafter\XINT_numer_A
    \else
      \expandafter\XINT_numer_B
    \fi
    {#1}%
}%
\def\XINT_numer_A #1#2#3{\XINT_dsx_addzeros{#1}#2;}%
\def\XINT_numer_B #1#2#3{ #2}%
```

8.17 \xintDenominator

```
\def\xintDenominator {\romannumeral0\xintdenominator }%
\def\xintdenominator
{%
    \expandafter\XINT_denom_fork\romannumeral0\XINT_infrac
}%
\def\XINT_denom_fork #1%
{%

    \ifnum#1<\xint_c_
      \expandafter\XINT_denom_B
    \else
      \expandafter\XINT_denom_A
    \fi
    #1.%
}%
\def\XINT_denom_A #1.#2#3{ #3}%
\def\XINT_denom_B -#1.#2#3{\XINT_dsx_addzeros{#1}#3;}%
```

8.18 \xintFrac

Useless typesetting macro.

```
\def\xintFrac {\romannumeral0\xintfrac }%
\def\xintfrac #1%
{%
    \expandafter\XINT_fracfrac_A\romannumeral0\XINT_infrac {#1}%
}%
\def\XINT_fracfrac_A #1{\XINT_fracfrac_B #1\Z }%
\catcode`^=7
\def\XINT_fracfrac_B #1#2\Z
{%
    \xint_gob_til_zero #1\XINT_fracfrac_C 0\XINT_fracfrac_D {10^{#1#2}}%
}%
\def\XINT_fracfrac_C 0\XINT_fracfrac_D #1#2#3%
{%
    \if1\XINT_isOne {#3}%
        \xint_afterfi {\expandafter\xint_firstoftwo_thenstop\xint_gobble_ii }%
    \fi
    \space
    \frac {#2}{#3}%
}%
\def\XINT_fracfrac_D #1#2#3%
{%
    \if1\XINT_isOne {#3}\XINT_fracfrac_E\fi
    \space
    \frac {#2}{#3}#1%
}%
\def\XINT_fracfrac_E \fi\space\frac #1#2{\fi \space #1\cdot }%
```

8.19 \xintSignedFrac

```
\def\xintSignedFrac {\romannumeral0\xintsignedfrac }%
\def\xintsignedfrac #1%
{%
    \expandafter\XINT_sgnfrac_a\romannumeral0\XINT_infrac {#1}%
}%
\def\XINT_sgnfrac_a #1#2%
{%
    \XINT_sgnfrac_b #2\Z {#1}%
}%
\def\XINT_sgnfrac_b #1%
{%
    \xint_UDsignfork
      #1\XINT_sgnfrac_N
       -{\XINT_sgnfrac_P #1}%
    \krof
}%
\def\XINT_sgnfrac_P #1\Z #2%
```

```
{%
    \XINT_fracfrac_A {#2}{#1}%
}%
\def\XINT_sgnfrac_N
{%
    \expandafter-\romannumeral0\XINT_sgnfrac_P
}%
```

8.20 \xintFwOver

```
\def\xintFwOver {\romannumeral0\xintfwover }%
\def\xintfwover #1%
{%
    \expandafter\XINT_fwover_A\romannumeral0\XINT_infrac {#1}%
}%
\def\XINT_fwover_A #1{\XINT_fwover_B #1\Z }%
\def\XINT_fwover_B #1#2\Z
{%
    \xint_gob_til_zero #1\XINT_fwover_C 0\XINT_fwover_D {10^{#1#2}}%
}%
\catcode`^=11
\def\XINT_fwover_C #1#2#3#4#5%
{%
    \if0\XINT_isOne {#5}\xint_afterfi { {#4\over #5}}%
                   \else\xint_afterfi { #4}%
    \fi
}%
\def\XINT_fwover_D #1#2#3%
{%
    \if0\XINT_isOne {#3}\xint_afterfi { {#2\over #3}}%
                   \else\xint_afterfi { #2\cdot }%
    \fi
    #1%
}%
```

8.21 \xintSignedFwOver

```
\def\xintSignedFwOver {\romannumeral0\xintsignedfwover }%
\def\xintsignedfwover #1%
{%
    \expandafter\XINT_sgnfwover_a\romannumeral0\XINT_infrac {#1}%
}%
\def\XINT_sgnfwover_a #1#2%
{%
    \XINT_sgnfwover_b #2\Z {#1}%
}%
\def\XINT_sgnfwover_b #1%
{%
    \xint_UDsignfork
      #1\XINT_sgnfwover_N
       -{\XINT_sgnfwover_P #1}%
    \krof
}%
\def\XINT_sgnfwover_P #1\Z #2%
{%
```

```
    \XINT_fwover_A {#2}{#1}%
}%
\def\XINT_sgnfwover_N
{%
    \expandafter-\romannumeral0\XINT_sgnfwover_P
}%
```

8.22 \xintREZ

Removes trailing zeros from A and B and adjust the N in A/B[N].

The macro really doing the job \XINT_factortens was redone at 1.3a. But speed gain really noticeable only beyond about 100 digits.

```
\def\xintREZ {\romannumeral0\xintrez }%
\def\xintrez
{%
    \expandafter\XINT_rez_A\romannumeral0\XINT_infrac
}%
\def\XINT_rez_A #1#2%
{%
    \XINT_rez_AB #2\Z {#1}%
}%
\def\XINT_rez_AB #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_rez_zero
      0#1\XINT_rez_neg
       0-{\XINT_rez_B #1}%
    \krof
}%
\def\XINT_rez_zero #1\Z #2#3{ 0/1[0]}%
\def\XINT_rez_neg {\expandafter-\romannumeral0\XINT_rez_B }%
\def\XINT_rez_B #1\Z
{%
    \expandafter\XINT_rez_C\romannumeral0\XINT_factortens {#1}%
}%
\def\XINT_rez_C #1.#2.#3#4%
{%
    \expandafter\XINT_rez_D\romannumeral0\XINT_factortens {#4}#3+#2.#1.%
}%
\def\XINT_rez_D #1.#2.#3.%
{%
    \expandafter\XINT_rez_E\the\numexpr #3-#2.#1.%
}%
\def\XINT_rez_E #1.#2.#3.{ #3/#2[#1]}%
```

8.23 \xintE

1.07: The fraction is the first argument contrarily to \xintTrunc and \xintRound.

1.1 modifies and moves \xintiiE to xint.sty.

```
\def\xintE {\romannumeral0\xinte }%
\def\xinte #1%
```

```
{%
    \expandafter\XINT_e \romannumeral0\XINT_infrac {#1}%
}%
\def\XINT_e #1#2#3#4%
{%
    \expandafter\XINT_e_end\the\numexpr #1+#4.{#2}{#3}%
}%
\def\XINT_e_end #1.#2#3{ #2/#3[#1]}%
```

8.24 \xintIrr, \xintPIrr

\xintPIrr (partial Irr, which ignores the decimal part) added at 1.3.

```
\def\xintIrr {\romannumeral0\xintirr }%
\def\xintPIrr{\romannumeral0\xintpirr }%
\def\xintirr #1%
{%
    \expandafter\XINT_irr_start\romannumeral0\xintrawwithzeros {#1}\Z
}%
\def\xintpirr #1%
{%
    \expandafter\XINT_pirr_start\romannumeral0\xintraw{#1}%
}%
\def\XINT_irr_start #1#2/#3\Z
{%
    \if0\XINT_isOne {#3}%
      \xint_afterfi
          {\xint_UDsignfork
               #1\XINT_irr_negative
                -{\XINT_irr_nonneg #1}%
           \krof}%
    \else
      \xint_afterfi{\XINT_irr_denomisone #1}%
    \fi
    #2\Z {#3}%
}%
\def\XINT_pirr_start #1#2/#3[%
{%
    \if0\XINT_isOne {#3}%
      \xint_afterfi
          {\xint_UDsignfork
               #1\XINT_irr_negative
                -{\XINT_irr_nonneg #1}%
           \krof}%
    \else
      \xint_afterfi{\XINT_irr_denomisone #1}%
    \fi
    #2\Z {#3}[%
}%
\def\XINT_irr_denomisone #1\Z #2{ #1/1}% changed in 1.08
\def\XINT_irr_negative   #1\Z #2{\XINT_irr_D #1\Z #2\Z -}%
\def\XINT_irr_nonneg     #1\Z #2{\XINT_irr_D #1\Z #2\Z \space}%
\def\XINT_irr_D #1#2\Z #3#4\Z
```

```
{%
    \xint_UDzerosfork
       #3#1\XINT_irr_indeterminate
       #30\XINT_irr_divisionbyzero
       #10\XINT_irr_zero
        00\XINT_irr_loop_a
    \krof
    {#3#4}{#1#2}{#3#4}{#1#2}%
}%
\def\XINT_irr_indeterminate #1#2#3#4#5%
{%
    \XINT_signalcondition{DivisionUndefined}{indeterminate: 0/0}{}{0/1}%
}%
\def\XINT_irr_divisionbyzero #1#2#3#4#5%
{%
    \XINT_signalcondition{DivisionByZero}{vanishing denominator: #5#2/0}{}{0/1}%
}%
\def\XINT_irr_zero #1#2#3#4#5{ 0/1}% changed in 1.08
\def\XINT_irr_loop_a #1#2%
{%
    \expandafter\XINT_irr_loop_d
    \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
}%
\def\XINT_irr_loop_d #1#2%
{%
    \XINT_irr_loop_e #2\Z
}%
\def\XINT_irr_loop_e #1#2\Z
{%
    \xint_gob_til_zero #1\XINT_irr_loop_exit0\XINT_irr_loop_a {#1#2}%
}%
\def\XINT_irr_loop_exit0\XINT_irr_loop_a #1#2#3#4%
{%
    \expandafter\XINT_irr_loop_exitb\expandafter
    {\romannumeral0\xintiiquo {#3}{#2}}%
    {\romannumeral0\xintiiquo {#4}{#2}}%
}%
\def\XINT_irr_loop_exitb #1#2%
{%
   \expandafter\XINT_irr_finish\expandafter {#2}{#1}%
}%
\def\XINT_irr_finish #1#2#3{#3#1/#2}% changed in 1.08
```

8.25 \xintifInt

```
\def\xintifInt   {\romannumeral0\xintifint }%
\def\xintifint #1{\expandafter\XINT_ifint\romannumeral0\xintrawwithzeros {#1}.}%
\def\XINT_ifint #1/#2.%
{%
    \if 0\xintiiRem {#1}{#2}%
     \expandafter\xint_firstoftwo_thenstop
    \else
     \expandafter\xint_secondoftwo_thenstop
    \fi
}%
```

8.26 \xintJrr

```
\def\xintJrr {\romannumeral0\xintjrr }%
\def\xintjrr #1%
{%
    \expandafter\XINT_jrr_start\romannumeral0\xintrawwithzeros {#1}\Z
}%
\def\XINT_jrr_start #1#2/#3\Z
{%
    \if0\XINT_isOne {#3}\xint_afterfi
          {\xint_UDsignfork
               #1\XINT_jrr_negative
                -{\XINT_jrr_nonneg #1}%
           \krof}%
    \else
      \xint_afterfi{\XINT_jrr_denomisone #1}%
    \fi
    #2\Z {#3}%
}%
\def\XINT_jrr_denomisone #1\Z #2{ #1/1}% changed in 1.08
\def\XINT_jrr_negative   #1\Z #2{\XINT_jrr_D #1\Z #2\Z -}%
\def\XINT_jrr_nonneg     #1\Z #2{\XINT_jrr_D #1\Z #2\Z \space}%
\def\XINT_jrr_D #1#2\Z #3#4\Z
{%
    \xint_UDzerosfork
       #3#1\XINT_jrr_indeterminate
       #30\XINT_jrr_divisionbyzero
       #10\XINT_jrr_zero
        00\XINT_jrr_loop_a
    \krof
    {#3#4}{#1#2}1001%
}%
\def\XINT_jrr_indeterminate #1#2#3#4#5#6#7%
{%
    \XINT_signalcondition{DivisionUndefined}{indeterminate: 0/0}{}{0/1}%
}%
\def\XINT_jrr_divisionbyzero #1#2#3#4#5#6#7%
{%
    \XINT_signalcondition{DivisionByZero}{Vanishing denominator: #7#2/0}{}{0/1}%
}%
\def\XINT_jrr_zero #1#2#3#4#5#6#7{ 0/1}% changed in 1.08
\def\XINT_jrr_loop_a #1#2%
{%
    \expandafter\XINT_jrr_loop_b
    \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
}%
\def\XINT_jrr_loop_b #1#2#3#4#5#6#7%
{%
    \expandafter \XINT_jrr_loop_c \expandafter
        {\romannumeral0\xintiiadd{\XINT_mul_fork #4\xint:#1\xint:}{#6}}%
        {\romannumeral0\xintiiadd{\XINT_mul_fork #5\xint:#1\xint:}{#7}}%
    {#2}{#3}{#4}{#5}%
}%
\def\XINT_jrr_loop_c #1#2%
{%
    \expandafter \XINT_jrr_loop_d \expandafter{#2}{#1}%
}%
\def\XINT_jrr_loop_d #1#2#3#4%
{%
    \XINT_jrr_loop_e #3\Z {#4}{#2}{#1}%
}%
\def\XINT_jrr_loop_e #1#2\Z
{%
    \xint_gob_til_zero #1\XINT_jrr_loop_exit0\XINT_jrr_loop_a {#1#2}%
}%
\def\XINT_jrr_loop_exit0\XINT_jrr_loop_a #1#2#3#4#5#6%
{%
    \XINT_irr_finish {#3}{#4}%
}%
```

8.27 \xintTFrac

1.09i, for frac in \xintexpr. And \xintFrac is already assigned. T for truncation. However, potentially not very efficient with numbers in scientific notations, with big exponents. Will have to think it again some day. I hesitated how to call the macro. Same convention as in maple, but some people reserve fractional part to x - floor(x). Also, not clear if I had to make it negative (or zero) if x < 0, or rather always positive. There should be in fact such a thing for each rounding function, trunc, round, floor, ceil.

```
\def\xintTFrac {\romannumeral0\xinttfrac }%
\def\xinttfrac #1{\expandafter\XINT_tfrac_fork\romannumeral0\xintrawwithzeros {#1}\Z }%
\def\XINT_tfrac_fork #1%
{%
    \xint_UDzerominusfork
        #1-\XINT_tfrac_zero
        0#1{\xintiiopp\XINT_tfrac_P }%
         0-{\XINT_tfrac_P #1}%
    \krof
}%
\def\XINT_tfrac_zero #1\Z { 0/1[0]}%
\def\XINT_tfrac_P #1/#2\Z {\expandafter\XINT_rez_AB
                          \romannumeral0\xintiirem{#1}{#2}\Z {0}{#2}}%
```

8.28 \xintTrunc, \xintiTrunc

1.2i release notes: ever since its inception this macro was stupid for a decimal input: it did not handle it separately from the general fraction case A/B[N] with B>1, hence ended up doing divisions by powers of ten. But this meant that nesting \xintTrunc with itself was very inefficient.

1.2i version is better. However it still handles B>1, N<0 via adding zeros to B and dividing with this extended B. A possibly more efficient approach is implemented in \xintXTrunc, but its logic is more complicated, the code is quite longer and making it f-expandable would not shorten it... I decided for the time being to not complicate things here.

```
\def\xintTrunc  {\romannumeral0\xinttrunc }%
```

```
\def\xintiTrunc {\romannumeral0\xintitrunc}%
\def\xinttrunc #1{\expandafter\XINT_trunc\the\numexpr#1.\XINT_trunc_G}%
\def\xintitrunc #1{\expandafter\XINT_trunc\the\numexpr#1.\XINT_itrunc_G}%
\def\XINT_trunc #1.#2#3%
{%
    \expandafter\XINT_trunc_a\romannumeral0\XINT_infrac{#3}#1.#2%
}%
\def\XINT_trunc_a #1#2#3#4.#5%
{%
    \if0\XINT_Sgn#2\xint:\xint_dothis\XINT_trunc_zero\fi
    \if1\XINT_is_One#3XY\xint_dothis\XINT_trunc_sp_b\fi
    \xint_orthat\XINT_trunc_b #1+#4.{#2}{#3}#5#4.%
}%
\def\XINT_trunc_zero #1.#2.{ 0}%

\def\XINT_trunc_b     {\expandafter\XINT_trunc_B\the\numexpr}%
\def\XINT_trunc_sp_b  {\expandafter\XINT_trunc_sp_B\the\numexpr}%

\def\XINT_trunc_B #1%
{%
    \xint_UDsignfork
      #1\XINT_trunc_C
       -\XINT_trunc_D
    \krof #1%
}%

\def\XINT_trunc_sp_B #1%
{%
    \xint_UDsignfork
      #1\XINT_trunc_sp_C
       -\XINT_trunc_sp_D
    \krof #1%
}%

\def\XINT_trunc_C -#1.#2#3%
{%
    \expandafter\XINT_trunc_CE
    \romannumeral0\XINT_dsx_addzeros{#1}#3;.{#2}%
}%
\def\XINT_trunc_CE #1.#2{\XINT_trunc_E #2.{#1}}%

\def\XINT_trunc_sp_C -#1.#2#3{\XINT_trunc_sp_Ca #2.#1.}%
\def\XINT_trunc_sp_Ca #1%
{%
    \xint_UDsignfork
       #1{\XINT_trunc_sp_Cb -}%
        -{\XINT_trunc_sp_Cb \space#1}%
    \krof
}%
\def\XINT_trunc_sp_Cb #1#2.#3.%
{%
    \expandafter\XINT_trunc_sp_Cc
    \romannumeral0\expandafter\XINT_split_fromright_a
    \the\numexpr#3-\numexpr\XINT_length_loop
    #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
      \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
      \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
    .#2\xint_bye2345678\xint_bye..#1%
}%
```

```
\def\XINT_trunc_sp_Cc #1%
{%
    \if.#1\xint_dothis{\XINT_trunc_sp_Cd 0.}\fi
    \xint_orthat {\XINT_trunc_sp_Cd #1}%
}%
\def\XINT_trunc_sp_Cd #1.#2.#3%
{%
    \XINT_trunc_sp_F #3#1.%
}%
\def\XINT_trunc_D #1.#2%
{%
    \expandafter\XINT_trunc_E
    \romannumeral0\XINT_dsx_addzeros {#1}#2;.%
}%
\def\XINT_trunc_sp_D #1.#2#3%
{%
    \expandafter\XINT_trunc_sp_E
    \romannumeral0\XINT_dsx_addzeros {#1}#2;.%
}%
\def\XINT_trunc_E #1%
{%
    \xint_UDsignfork
       #1{\XINT_trunc_F -}%
        -{\XINT_trunc_F \space#1}%
    \krof
}%
\def\XINT_trunc_sp_E #1%
{%
    \xint_UDsignfork
       #1{\XINT_trunc_sp_F -}%
        -{\XINT_trunc_sp_F\space#1}%
    \krof
}%
\def\XINT_trunc_F #1#2.#3#4%
   {\expandafter#4\romannumeral`&&@\expandafter\xint_firstoftwo
                  \romannumeral0\XINT_div_prepare {#3}{#2}.#1}%
\def\XINT_trunc_sp_F #1#2.#3{#3#2.#1}%
\def\XINT_itrunc_G #1#2.#3#4.{\if#10\xint_dothis{ 0}\fi\xint_orthat{#3#1}#2}%
\def\XINT_trunc_G  #1.#2#3.%
{%
    \expandafter\XINT_trunc_H
    \the\numexpr\romannumeral0\xintlength {#1}-#3.#3.{#1}#2%
}%
\def\XINT_trunc_H #1.#2.%
```

```
{%
    \ifnum #1 > \xint_c_
        \xint_afterfi {\XINT_trunc_Ha {#2}}%
    \else
        \xint_afterfi {\XINT_trunc_Hb {-#1}}% -0,--1,--2, ....
    \fi
}%
\def\XINT_trunc_Ha{\expandafter\XINT_trunc_Haa\romannumeral0\xintdecsplit}%
\def\XINT_trunc_Haa #1#2#3{#3#1.#2}%
\def\XINT_trunc_Hb #1#2#3%
{%
    \expandafter #3\expandafter0\expandafter.%
    \romannumeral\xintreplicate{#1}0#2%
}%
```

8.29 \xintTTrunc

1.1. Modified in 1.2i, it does simply \xintiTrunc0 with no shortcut (the latter having been modified)

```
\def\xintTTrunc {\romannumeral0\xintttrunc }%
\def\xintttrunc {\xintitrunc\xint_c_}%
```

8.30 \xintNum

```
\let\xintnum \xintttrunc
```

8.31 \xintRound, \xintiRound

Modified in 1.2i.

It benefits first of all from the faster \xintTrunc, particularly when the input is already a decimal number (denominator B=1).

And the rounding is now done in 1.2 style (with much delay, sorry), like of the rewritten \xintInc and \xintDec.

```
\def\xintRound  {\romannumeral0\xintround }%
\def\xintiRound {\romannumeral0\xintiround }%
\def\xintround  #1{\expandafter\XINT_round\the\numexpr #1.\XINT_round_A}%
\def\xintiround #1{\expandafter\XINT_round\the\numexpr #1.\XINT_iround_A}%
\def\XINT_round #1.{\expandafter\XINT_round_aa\the\numexpr #1+\xint_c_i.#1.}%
\def\XINT_round_aa #1.#2.#3#4%
{%
    \expandafter\XINT_round_a\romannumeral0\XINT_infrac{#4}#1.#3#2.%
}%
\def\XINT_round_a #1#2#3#4.%
{%
    \if0\XINT_Sgn#2\xint:\xint_dothis\XINT_trunc_zero\fi
    \if1\XINT_is_One#3XY\xint_dothis\XINT_trunc_sp_b\fi
    \xint_orthat\XINT_trunc_b #1+#4.{#2}{#3}%
}%
\def\XINT_round_A{\expandafter\XINT_trunc_G\romannumeral0\XINT_round_B}%
\def\XINT_iround_A{\expandafter\XINT_itrunc_G\romannumeral0\XINT_round_B}%
```

```
\def\XINT_round_B #1.%
    {\XINT_dsrr #1\xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax.}%
```

8.32 \xintXTrunc

1.09j [2014/01/06] This is completely expandable but not f-expandable. Rewritten for 1.2i (2016/12/04):
  - no more use of \xintiloop from xinttools.sty (replaced by \xintreplicate... from xintkernel.sty),
  - no more use in 0>N>-D case of a dummy control sequence name via \csname...\endcsname
  - handles better the case of an input already a decimal number
  Need to transfer code comments into public dtx.

```
\def\xintXTrunc #1%#2%
{%
    \expandafter\XINT_xtrunc_a
    \the\numexpr #1\expandafter.\romannumeral0\xintraw
}%
\def\XINT_xtrunc_a #1.% ?? faire autre chose
{%
    \expandafter\XINT_xtrunc_b\the\numexpr\ifnum#1<\xint_c_i \xint_c_i-\fi #1.%
}%

\def\XINT_xtrunc_b #1.#2{\XINT_xtrunc_c #2{#1}}%

\def\XINT_xtrunc_c #1%
{%
    \xint_UDzerominusfork
        #1-\XINT_xtrunc_zero
        0#1{-\XINT_xtrunc_d {}}%
         0-{\XINT_xtrunc_d #1}%
    \krof
}%[
\def\XINT_xtrunc_zero #1#2]{0.\romannumeral\xintreplicate{#1}0}%

\def\XINT_xtrunc_d #1#2#3/#4[#5]%
{%
    \XINT_xtrunc_prepare_a#4\R\R\R\R\R\R\R\R {10}0000001\W
    !{#4};{#5}{#2}{#1#3}%
}%
\def\XINT_xtrunc_prepare_a #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_R #9\XINT_xtrunc_prepare_small\R
    \XINT_xtrunc_prepare_b #9%
}%
\def\XINT_xtrunc_prepare_small\R #1!#2;%
{%
    \ifcase #2
    \or\expandafter\XINT_xtrunc_BisOne
    \or\expandafter\XINT_xtrunc_BisTwo
    \or
    \or\expandafter\XINT_xtrunc_BisFour
    \or\expandafter\XINT_xtrunc_BisFive
    \or
    \or
    \or\expandafter\XINT_xtrunc_BisEight
    \fi\XINT_xtrunc_BisSmall {#2}%
}%
```

```
\def\XINT_xtrunc_BisOne\XINT_xtrunc_BisSmall #1#2#3#4%
   {\XINT_xtrunc_sp_e {#2}{#4}{#3}}%
\def\XINT_xtrunc_BisTwo\XINT_xtrunc_BisSmall #1#2#3#4%
{%
    \expandafter\XINT_xtrunc_sp_e\expandafter
    {\the\numexpr #2-\xint_c_i\expandafter}\expandafter
    {\romannumeral0\xintiimul 5{#4}}{#3}%
}%
\def\XINT_xtrunc_BisFour\XINT_xtrunc_BisSmall #1#2#3#4%
{%
    \expandafter\XINT_xtrunc_sp_e\expandafter
    {\the\numexpr #2-\xint_c_ii\expandafter}\expandafter
    {\romannumeral0\xintiimul {25}{#4}}{#3}%
}%
\def\XINT_xtrunc_BisFive\XINT_xtrunc_BisSmall #1#2#3#4%
{%
    \expandafter\XINT_xtrunc_sp_e\expandafter
    {\the\numexpr #2-\xint_c_i\expandafter}\expandafter
    {\romannumeral0\xintdouble {#4}}{#3}%
}%
\def\XINT_xtrunc_BisEight\XINT_xtrunc_BisSmall #1#2#3#4%
{%
    \expandafter\XINT_xtrunc_sp_e\expandafter
    {\the\numexpr #2-\xint_c_iii\expandafter}\expandafter
    {\romannumeral0\xintiimul {125}{#4}}{#3}%
}%
\def\XINT_xtrunc_BisSmall #1%
{%
    \expandafter\XINT_xtrunc_e\expandafter
    {\expandafter\XINT_xtrunc_small_a
    \the\numexpr #1/\xint_c_ii\expandafter
    .\the\numexpr \xint_c_x^viii+#1!}%
}%
```

```
\def\XINT_xtrunc_small_a #1.#2!#3%
{%
    \expandafter\XINT_div_small_b\the\numexpr #1\expandafter
    \xint:\the\numexpr #2\expandafter!%
    \romannumeral0\XINT_div_small_ba #3\R\R\R\R\R\R\R\R{10}0000001\W
       #3\XINT_sepbyviii_Z_end 2345678\relax
}%
```

```
\def\XINT_xtrunc_prepare_b
```

```
   {\expandafter\XINT_xtrunc_prepare_c\romannumeral0\XINT_zeroes_forviii }%
\def\XINT_xtrunc_prepare_c #1!%
{%
     \XINT_xtrunc_prepare_d  #1.00000000!{#1}%
}%
\def\XINT_xtrunc_prepare_d #1#2#3#4#5#6#7#8#9%
{%
    \expandafter\XINT_xtrunc_prepare_e
    \xint_gob_til_dot #1#2#3#4#5#6#7#8#9!%
}%
\def\XINT_xtrunc_prepare_e #1!#2!#3#4%
{%
    \XINT_xtrunc_prepare_f #4#3\X {#1}{#3}%
}%
\def\XINT_xtrunc_prepare_f #1#2#3#4#5#6#7#8#9\X
{%
    \expandafter\XINT_xtrunc_prepare_g\expandafter
    \XINT_div_prepare_g
     \the\numexpr  #1#2#3#4#5#6#7#8+\xint_c_i\expandafter
    \xint:\the\numexpr (#1#2#3#4#5#6#7#8+\xint_c_i)/\xint_c_ii\expandafter
    \xint:\the\numexpr #1#2#3#4#5#6#7#8\expandafter
    \xint:\romannumeral0\XINT_sepandrev_andcount
    #1#2#3#4#5#6#7#8#9\XINT_rsepbyviii_end_A 2345678%
                      \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
              \R\xint:\xint_c_xii \R\xint:\xint_c_x  \R\xint:\xint_c_viii \R\xint:\xint_c_vi
              \R\xint:\xint_c_iv  \R\xint:\xint_c_ii \R\xint:\xint_c_\W
    \X
}%

\def\XINT_xtrunc_prepare_g #1;{\XINT_xtrunc_e {#1}}%

\def\XINT_xtrunc_e #1#2%
{%
    \ifnum #2<\xint_c_
        \expandafter\XINT_xtrunc_I
    \else
        \expandafter\XINT_xtrunc_II
    \fi  #2\xint:{#1}%
}%

\def\XINT_xtrunc_I -#1\xint:#2#3#4%
{%
    \expandafter\XINT_xtrunc_I_a\romannumeral0#2{#4}{#2}{#1}{#3}%
}%

\def\XINT_xtrunc_I_a #1#2#3#4#5%
{%
    \expandafter\XINT_xtrunc_I_b\the\numexpr #4-#5\xint:#4\xint:{#5}{#2}{#3}{#1}%
}%
```

```
\def\XINT_xtrunc_I_b #1%
{%
    \xint_UDsignfork
      #1\XINT_xtrunc_IA_c
       -\XINT_xtrunc_IB_c
    \krof #1%
}%
```

```
\def\XINT_xtrunc_IA_c -#1\xint:#2\xint:#3#4#5#6%
{%
   \expandafter\XINT_xtrunc_IA_d
   \the\numexpr#2-\xintLength{#6}\xint:{#6}%
   \expandafter\XINT_xtrunc_IA_xd
   \the\numexpr (#1+\xint_c_ii^v)/\xint_c_ii^vi-\xint_c_i\xint:#1\xint:{#5}{#4}%
}%
```

```
\def\XINT_xtrunc_IA_d #1%
{%
    \xint_UDsignfork
      #1\XINT_xtrunc_IAA_e
       -\XINT_xtrunc_IAB_e
    \krof #1%
}%
```

```
\def\XINT_xtrunc_IAA_e -#1\xint:#2%
{%
    \romannumeral0\XINT_split_fromleft
    #1.#2\xint_gobble_i\xint_bye2345678\xint_bye..%
}%
```

```
\def\XINT_xtrunc_IAB_e #1\xint:#2%
{%
    0.\romannumeral\XINT_rep#1\endcsname0#2%
}%
```

```
\def\XINT_xtrunc_IA_xd #1\xint:#2\xint:%
{%
    \expandafter\XINT_xtrunc_IA_xe\the\numexpr #2-\xint_c_ii^vi*#1\xint:#1\xint:%
}%
```

```
\def\XINT_xtrunc_IA_xe #1\xint:#2\xint:#3#4%
{%
    \XINT_xtrunc_loop {#2}{#4}{#3}{#1}%
}%
```

```
\def\XINT_xtrunc_IB_c #1\xint:#2\xint:#3#4#5#6%
{%
    \expandafter\XINT_xtrunc_IB_d
    \romannumeral0\XINT_split_xfork #1.#6\xint_bye2345678\xint_bye..{#3}%
}%
```

```
\def\XINT_xtrunc_IB_d #1.#2.#3%
{%
    \expandafter\XINT_xtrunc_IA_d\the\numexpr#3-\xintLength {#1}\xint:{#1}%
}%

\def\XINT_xtrunc_II #1\xint:%
{%
    \expandafter\XINT_xtrunc_II_a\romannumeral\xintreplicate{#1}0\xint:%
}%
\def\XINT_xtrunc_II_a #1\xint:#2#3#4%
{%
    \expandafter\XINT_xtrunc_II_b
    \the\numexpr (#3+\xint_c_ii^v)/\xint_c_ii^vi-\xint_c_i\expandafter\xint:%
    \the\numexpr #3\expandafter\xint:\romannumeral0#2{#4#1}{#2}%
}%
\def\XINT_xtrunc_II_b #1\xint:#2\xint:%
{%
    \expandafter\XINT_xtrunc_II_c\the\numexpr #2-\xint_c_ii^vi*#1\xint:#1\xint:%
}%

\def\XINT_xtrunc_II_c #1\xint:#2\xint:#3#4#5%
{%
    #3.\XINT_xtrunc_loop {#2}{#4}{#5}{#1}%
}%

\def\XINT_xtrunc_loop #1%
{%
    \ifnum #1=\xint_c_ \expandafter\XINT_xtrunc_transition\fi
    \expandafter\XINT_xtrunc_loop_a\the\numexpr #1-\xint_c_i\xint:%
}%
\def\XINT_xtrunc_loop_a #1\xint:#2#3%
{%
    \expandafter\XINT_xtrunc_loop_b\romannumeral0#3%
    {#20000000000000000000000000000000000000000000000000000000000000000}%
    {#1}{#3}%
}%
\def\XINT_xtrunc_loop_b #1#2#3%
{%
    \romannumeral\xintreplicate{\xint_c_ii^vi-\xintLength{#1}}0#1%
    \XINT_xtrunc_loop {#3}{#2}%
}%

\def\XINT_xtrunc_transition
    \expandafter\XINT_xtrunc_loop_a\the\numexpr #1\xint:#2#3#4%
{%
    \ifnum #4=\xint_c_ \expandafter\xint_gobble_vi\fi
    \expandafter\XINT_xtrunc_finish\expandafter
    {\romannumeral0\XINT_dsx_addzeros{#4}#2;}{#3}{#4}%
}%
\def\XINT_xtrunc_finish #1#2%
{%
```

```
    \expandafter\XINT_xtrunc_finish_a\romannumeral0#2{#1}%
}%
\def\XINT_xtrunc_finish_a #1#2#3%
{%
    \romannumeral\xintreplicate{#3-\xintLength{#1}}0#1%
}%

\def\XINT_xtrunc_sp_e #1%
{%
    \ifnum #1<\xint_c_
        \expandafter\XINT_xtrunc_sp_I
    \else
        \expandafter\XINT_xtrunc_sp_II
    \fi  #1\xint:%
}%

\def\XINT_xtrunc_sp_I -#1\xint:#2#3%
{%
    \expandafter\XINT_xtrunc_sp_I_a\the\numexpr #1-#3\xint:#1\xint:{#3}{#2}%
}%

\def\XINT_xtrunc_sp_I_a #1%
{%
    \xint_UDsignfork
      #1\XINT_xtrunc_sp_IA_b
       -\XINT_xtrunc_sp_IB_b
    \krof #1%
}%

\def\XINT_xtrunc_sp_IA_b -#1\xint:#2\xint:#3#4%
{%
   \expandafter\XINT_xtrunc_sp_IA_c
   \the\numexpr#2-\xintLength{#4}\xint:{#4}\romannumeral\XINT_rep#1\endcsname0%
}%

\def\XINT_xtrunc_sp_IA_c #1%
{%
    \xint_UDsignfork
      #1\XINT_xtrunc_sp_IAA
       -\XINT_xtrunc_sp_IAB
    \krof #1%
}%

\def\XINT_xtrunc_sp_IAA -#1\xint:#2%
{%
    \romannumeral0\XINT_split_fromleft
    #1.#2\xint_gobble_i\xint_bye2345678\xint_bye..%
}%
```

```
\def\XINT_xtrunc_sp_IAB #1\xint:#2%
{%
    0.\romannumeral\XINT_rep#1\endcsname0#2%
}%
```

```
\def\XINT_xtrunc_sp_IB_b #1\xint:#2\xint:#3#4%
{%
    \expandafter\XINT_xtrunc_sp_IB_c
    \romannumeral0\XINT_split_xfork #1.#4\xint_bye2345678\xint_bye..{#3}%
}%
```

```
\def\XINT_xtrunc_sp_IB_c #1.#2.#3%
{%
    \expandafter\XINT_xtrunc_sp_IA_c\the\numexpr#3-\xintLength {#1}\xint:{#1}%
}%
```

```
\def\XINT_xtrunc_sp_II #1\xint:#2#3%
{%
    #2\romannumeral\XINT_rep#1\endcsname0.\romannumeral\XINT_rep#3\endcsname0%
}%
```

8.33 \xintDigits

The mathchardef used to be called \XINT_digits, but for reasons originating in \xintNewExpr (and now obsolete), release 1.09a uses \XINTdigits without underscore.

```
\mathchardef\XINTdigits 16
\def\xintDigits #1#2%
   {\afterassignment \xint_gobble_i \mathchardef\XINTdigits=}%
\def\xinttheDigits {\number\XINTdigits }%
```

8.34 \xintAdd

Big change at 1.3: a/b+c/d uses lcm(b,d) as denominator.

```
\def\xintAdd {\romannumeral0\xintadd }%
\def\xintadd #1{\expandafter\XINT_fadd\romannumeral0\xintraw {#1}}%
\def\XINT_fadd #1{\xint_gob_til_zero #1\XINT_fadd_Azero 0\XINT_fadd_a #1}%
\def\XINT_fadd_Azero #1]{\xintraw }%
\def\XINT_fadd_a #1/#2[#3]#4%
   {\expandafter\XINT_fadd_b\romannumeral0\xintraw {#4}{#3}{#1}{#2}}%
\def\XINT_fadd_b #1{\xint_gob_til_zero #1\XINT_fadd_Bzero 0\XINT_fadd_c #1}%
\def\XINT_fadd_Bzero #1]#2#3#4{ #3/#4[#2]}%
\def\XINT_fadd_c #1/#2[#3]#4%
{%
    \expandafter\XINT_fadd_Aa\the\numexpr #4-#3.{#3}{#4}{#1}{#2}%
}%
\def\XINT_fadd_Aa #1%
{%
    \xint_UDzerominusfork
       #1-\XINT_fadd_B
```

```
        0#1\XINT_fadd_Bb
        0-\XINT_fadd_Ba
    \krof #1%
}%
\def\XINT_fadd_B   #1.#2#3#4#5#6#7{\XINT_fadd_C {#4}{#5}{#7}{#6}[#3]}%
\def\XINT_fadd_Ba  #1.#2#3#4#5#6#7%
{%
    \expandafter\XINT_fadd_C\expandafter
        {\romannumeral0\XINT_dsx_addzeros {#1}#6;}%
    {#7}{#5}{#4}[#2]%
}%
\def\XINT_fadd_Bb -#1.#2#3#4#5#6#7%
{%
    \expandafter\XINT_fadd_C\expandafter
        {\romannumeral0\XINT_dsx_addzeros {#1}#4;}%
    {#5}{#7}{#6}[#3]%
}%
\def\XINT_fadd_iszero #1[#2]{ 0/1[0]}% ou [#2] originel?
\def\XINT_fadd_C #1#2#3%
{%
    \expandafter\XINT_fadd_D_b
    \romannumeral0\XINT_div_prepare{#2}{#3}{#2}{#2}{#3}{#1}%
}%
```

Basically a clone of the \XINT_irr_loop_a loop. I should modify the output of \XINT_div_prepare perhaps to be optimized for checking if remainder vanishes.

```
\def\XINT_fadd_D_a #1#2%
{%
    \expandafter\XINT_fadd_D_b
    \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
}%
\def\XINT_fadd_D_b #1#2{\XINT_fadd_D_c #2\Z}%
\def\XINT_fadd_D_c #1#2\Z
{%
    \xint_gob_til_zero #1\XINT_fadd_D_exit0\XINT_fadd_D_a {#1#2}%
}%
\def\XINT_fadd_D_exit0\XINT_fadd_D_a #1#2#3%
{%
    \expandafter\XINT_fadd_E
    \romannumeral0\xintiiquo {#3}{#2}.{#2}%
}%
\def\XINT_fadd_E #1.#2#3%
{%
    \expandafter\XINT_fadd_F
    \romannumeral0\xintiimul{#1}{#3}.{\xintiiQuo{#3}{#2}}{#1}%
}%
\def\XINT_fadd_F #1.#2#3#4#5%
{%
    \expandafter\XINT_fadd_G
    \romannumeral0\xintiiadd{\xintiiMul{#2}{#4}}{\xintiiMul{#3}{#5}}/#1%
}%
\def\XINT_fadd_G #1{%
\def\XINT_fadd_G ##1{\if0##1\expandafter\XINT_fadd_iszero\fi#1##1}%
}\XINT_fadd_G{ }%
```

8.35 \xintSub

Since 1.3 will use least common multiple of denominators.

```
\def\xintSub   {\romannumeral0\xintsub }%
\def\xintsub #1{\expandafter\XINT_fsub\romannumeral0\xintraw {#1}}%
\def\XINT_fsub #1{\xint_gob_til_zero #1\XINT_fsub_Azero 0\XINT_fsub_a #1}%
\def\XINT_fsub_Azero #1]{\xintopp }%
\def\XINT_fsub_a #1/#2[#3]#4%
   {\expandafter\XINT_fsub_b\romannumeral0\xintraw {#4}{#3}{#1}{#2}}%
\def\XINT_fsub_b #1{\xint_UDzerominusfork
                      #1-\XINT_fadd_Bzero
                       0#1\XINT_fadd_c
                       0-{\XINT_fadd_c -#1}%
                     \krof }%
```

8.36 \xintSum

There was (not documented anymore since 1.09d, 2013/10/22) a macro \xintSumExpr, but it has been deleted at 1.21.

Empty items are not accepted by this macro.

```
\def\xintSum {\romannumeral0\xintsum }%
\def\xintsum #1{\expandafter\XINT_fsumexpr\romannumeral`&&@#1\xint:}%
\def\XINT_fsumexpr {\XINT_fsum_loop_a {0/1[0]}}%
\def\XINT_fsum_loop_a #1#2%
{%
    \expandafter\XINT_fsum_loop_b \romannumeral`&&@#2\xint:{#1}%
}%
\def\XINT_fsum_loop_b #1%
{%
    \xint_gob_til_xint: #1\XINT_fsum_finished\xint:\XINT_fsum_loop_c #1%
}%
\def\XINT_fsum_loop_c #1\xint:#2%
{%
    \expandafter\XINT_fsum_loop_a\expandafter{\romannumeral0\xintadd {#2}{#1}}%
}%
\def\XINT_fsum_finished #1\xint:\xint:#2{ #2}%
```

8.37 \xintMul

```
\def\xintMul {\romannumeral0\xintmul }%
\def\xintmul #1{\expandafter\XINT_fmul\romannumeral0\xintraw {#1}.}%
\def\XINT_fmul #1{\xint_gob_til_zero #1\XINT_fmul_zero 0\XINT_fmul_a #1}%
\def\XINT_fmul_a #1[#2].#3%
   {\expandafter\XINT_fmul_b\romannumeral0\xintraw {#3}#1[#2.]}%
\def\XINT_fmul_b #1{\xint_gob_til_zero #1\XINT_fmul_zero 0\XINT_fmul_c #1}%
\def\XINT_fmul_c #1/#2[#3]#4/#5[#6.]%
{%
    \expandafter\XINT_fmul_d
    \expandafter{\the\numexpr #3+#6\expandafter}%
    \expandafter{\romannumeral0\xintiimul {#5}{#2}}%
    {\romannumeral0\xintiimul {#4}{#1}}%
}%
\def\XINT_fmul_d #1#2#3%
{%
    \expandafter \XINT_fmul_e \expandafter{#3}{#1}{#2}%
}%
\def\XINT_fmul_e #1#2{\XINT_outfrac {#2}{#1}}%
\def\XINT_fmul_zero #1.#2{ 0/1[0]}%
```

8.38 \xintSqr

1.1 modifs comme xintMul.

```
\def\xintSqr {\romannumeral0\xintsqr }%
\def\xintsqr #1{\expandafter\XINT_fsqr\romannumeral0\xintraw {#1}}%
\def\XINT_fsqr #1{\xint_gob_til_zero #1\XINT_fsqr_zero 0\XINT_fsqr_a #1}%
\def\XINT_fsqr_a #1/#2[#3]%
{%
    \expandafter\XINT_fsqr_b
    \expandafter{\the\numexpr #3+#3\expandafter}%
    \expandafter{\romannumeral0\xintiisqr {#2}}%
    {\romannumeral0\xintiisqr {#1}}%
}%
\def\XINT_fsqr_b #1#2#3{\expandafter \XINT_fmul_e \expandafter{#3}{#1}{#2}}%
\def\XINT_fsqr_zero #1]{ 0/1[0]}%
```

8.39 \xintPow

1.2f: to be coherent with the "i" convention \xintiPow should parse also its exponent via \xintNum when xintfrac.sty is loaded. This was not the case so far. Cependant le problème est que le fait d'appliquer \xintNum rend impossible certains inputs qui auraient pu être gérès par \numexpr. Le \numexpr externe est ici pour intercepter trop grand input.

```
\def\xintipow #1#2%
{%
    \expandafter\xint_pow\the\numexpr \xintNum{#2}\expandafter
    .\romannumeral0\xintnum{#1}\xint:
}%
\def\xintPow {\romannumeral0\xintpow }%
\def\xintpow #1%
{%
    \expandafter\XINT_fpow\expandafter {\romannumeral0\XINT_infrac {#1}}%
}%
\def\XINT_fpow #1#2%
{%
    \expandafter\XINT_fpow_fork\the\numexpr \xintNum{#2}\relax\Z #1%
}%
\def\XINT_fpow_fork #1#2\Z
{%
    \xint_UDzerominusfork
      #1-\XINT_fpow_zero
      0#1\XINT_fpow_neg
       0-{\XINT_fpow_pos #1}%
    \krof
    {#2}%
}%
\def\XINT_fpow_zero #1#2#3#4{ 1/1[0]}%
\def\XINT_fpow_pos #1#2#3#4#5%
{%
    \expandafter\XINT_fpow_pos_A\expandafter
    {\the\numexpr #1#2*#3\expandafter}\expandafter
    {\romannumeral0\xintiipow {#5}{#1#2}}%
    {\romannumeral0\xintiipow {#4}{#1#2}}%
}%
\def\XINT_fpow_neg #1#2#3#4%
{%
    \expandafter\XINT_fpow_pos_A\expandafter
    {\the\numexpr -#1*#2\expandafter}\expandafter
    {\romannumeral0\xintiipow {#3}{#1}}%
    {\romannumeral0\xintiipow {#4}{#1}}%
}%
\def\XINT_fpow_pos_A #1#2#3%
{%
    \expandafter\XINT_fpow_pos_B\expandafter {#3}{#1}{#2}%
}%
\def\XINT_fpow_pos_B #1#2{\XINT_outfrac {#2}{#1}}%
```

8.40 \xintFac

Factorial coefficients: variant which can be chained with other xintfrac macros. \xintiFac deprecated at 1.2o and removed at 1.3; \xintFac used by xintexpr.sty.

```
\def\xintFac  {\romannumeral0\xintfac}%
\def\xintfac  #1{\expandafter\XINT_fac_fork\the\numexpr\xintNum{#1}.[0]}%
```

8.41 \xintBinomial

1.2f. Binomial coefficients. \xintiBinomial deprecated at 1.2o and removed at 1.3; \xintBinomial needed by xintexpr.sty.

```
\def\xintBinomial {\romannumeral0\xintbinomial}%
\def\xintbinomial #1#2%
{%
    \expandafter\XINT_binom_pre
    \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.[0]%
}%
```

8.42 \xintPFactorial

1.2f. Partial factorial. For needs of xintexpr.sty.

```
\def\xintipfactorial #1#2%
{%
    \expandafter\XINT_pfac_fork
    \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.%
}%
\def\xintPFactorial {\romannumeral0\xintpfactorial}%
\def\xintpfactorial #1#2%
{%
    \expandafter\XINT_pfac_fork
    \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.[0]%
}%
```

8.43 \xintPrd

There was (not documented anymore since 1.09d, 2013/10/22) a macro \xintPrdExpr, but it has been deleted at 1.21

```
\def\xintPrd {\romannumeral0\xintprd }%
\def\xintprd #1{\expandafter\XINT_fprdexpr \romannumeral`&&@#1\xint:}%
\def\XINT_fprdexpr {\XINT_fprod_loop_a {1/1[0]}}%
\def\XINT_fprod_loop_a #1#2%
{%
    \expandafter\XINT_fprod_loop_b \romannumeral`&&@#2\xint:{#1}%
}%
\def\XINT_fprod_loop_b #1%
{%
    \xint_gob_til_xint: #1\XINT_fprod_finished\xint:\XINT_fprod_loop_c #1%
}%
\def\XINT_fprod_loop_c #1\xint:#2%
{%
  \expandafter\XINT_fprod_loop_a\expandafter{\romannumeral0\xintmul {#1}{#2}}%
}%
\def\XINT_fprod_finished#1\xint:\xint:#2{ #2}%
```

8.44 \xintDiv

```
\def\xintDiv {\romannumeral0\xintdiv }%
\def\xintdiv #1%
{%
    \expandafter\XINT_fdiv\expandafter {\romannumeral0\XINT_infrac {#1}}%
}%
\def\XINT_fdiv #1#2%
   {\expandafter\XINT_fdiv_A\romannumeral0\XINT_infrac {#2}#1}%
\def\XINT_fdiv_A #1#2#3#4#5#6%
{%
    \expandafter\XINT_fdiv_B
    \expandafter{\the\numexpr #4-#1\expandafter}%
    \expandafter{\romannumeral0\xintiimul {#2}{#6}}%
    {\romannumeral0\xintiimul {#3}{#5}}%
}%
\def\XINT_fdiv_B #1#2#3%
{%
    \expandafter\XINT_fdiv_C
    \expandafter{#3}{#1}{#2}%
}%
\def\XINT_fdiv_C #1#2{\XINT_outfrac {#2}{#1}}%
```

8.45 \xintDivFloor

1.1. Changed at 1.2p to not append /1[0] ending but rather output a big integer in strict format, like \xintDivTrunc and \xintDivRound.

```
\def\xintDivFloor     {\romannumeral0\xintdivfloor }%
\def\xintdivfloor #1#2{\xintifloor{\xintDiv {#1}{#2}}}%
```

8.46 \xintDivTrunc

1.1. \xintttrunc rather than \xintitrunc0 in 1.1a

```
\def\xintDivTrunc     {\romannumeral0\xintdivtrunc }%
\def\xintdivtrunc #1#2{\xintttrunc {\xintDiv {#1}{#2}}}%
```

8.47 \xintDivRound

1.1

```
\def\xintDivRound     {\romannumeral0\xintdivround }%
\def\xintdivround #1#2{\xintiround 0{\xintDiv {#1}{#2}}}%
```

8.48 \xintModTrunc

1.1. \xintModTrunc {q1}{q2} computes q1 - q2*t(q1/q2) with t(q1/q2) equal to the truncated division of two fractions q1 and q2.

Its former name, prior to 1.2p, was \xintMod.

At 1.3, uses least common multiple denominator, like \xintMod (next).

```
\def\xintModTrunc {\romannumeral0\xintmodtrunc }%
\def\xintmodtrunc #1{\expandafter\XINT_modtrunc_a\romannumeral0\xintraw{#1}.}%
\def\XINT_modtrunc_a #1#2.#3%
   {\expandafter\XINT_modtrunc_b\expandafter #1\romannumeral0\xintraw{#3}#2.}%
\def\XINT_modtrunc_b #1#2% #1 de A, #2 de B.
{%
    \if0#2\xint_dothis{\XINT_modtrunc_divbyzero #1#2}\fi
    \if0#1\xint_dothis\XINT_modtrunc_aiszero\fi
    \if-#2\xint_dothis{\XINT_modtrunc_bneg #1}\fi
          \xint_orthat{\XINT_modtrunc_bpos #1#2}%
}%
\def\XINT_modtrunc_divbyzero #1#2[#3]#4.%
{%
    \XINT_signalcondition{DivisionByZero}{Division by #2[#3] of #1#4}{}{0/1[0]}%
}%
\def\XINT_modtrunc_aiszero #1.{ 0/1[0]}%
\def\XINT_modtrunc_bneg #1%
{%
    \xint_UDsignfork
         #1{\xintiiopp\XINT_modtrunc_pos {}}%
          -{\XINT_modtrunc_pos #1}%
    \krof
}%
\def\XINT_modtrunc_bpos #1%
```

```
{%
    \xint_UDsignfork
            #1{\xintiiopp\XINT_modtrunc_pos {}}%
             -{\XINT_modtrunc_pos #1}%
    \krof
}%
```

Attention. This crucially uses that xint's \xintiiE{x}{e} is defined to return x unchanged if e is negative (and x extended by e zeroes if e >= 0).

```
\def\XINT_modtrunc_pos #1#2/#3[#4]#5/#6[#7].%
{%
    \expandafter\XINT_modtrunc_pos_a
    \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.%
    \romannumeral0\expandafter\XINT_mod_D_b
    \romannumeral0\XINT_div_prepare{#3}{#6}{#3}{#3}{#6}%
    {#1#5}{#7-#4}{#2}{#4-#7}%
}%
\def\XINT_modtrunc_pos_a #1.#2#3#4{\xintiirem {#3}{#4}/#2[#1]}%
```

8.49 \xintDivMod

1.2p. \xintDivMod{q1}{q2} outputs {floor(q1/q2)}{q1 - q2*floor(q1/q2)}. Attention that it relies on \xintiiE{x}{e} returning x if e < 0.

Modified (like \xintAdd and \xintSub) at 1.3 to use a l.c.m for final denominator of the "mod" part.

```
\def\xintDivMod {\romannumeral0\xintdivmod }%
\def\xintdivmod #1{\expandafter\XINT_divmod_a\romannumeral0\xintraw{#1}.}%
\def\XINT_divmod_a #1#2.#3%
   {\expandafter\XINT_divmod_b\expandafter #1\romannumeral0\xintraw{#3}#2.}%
\def\XINT_divmod_b #1#2% #1 de A, #2 de B.
{%
    \if0#2\xint_dothis{\XINT_divmod_divbyzero #1#2}\fi
    \if0#1\xint_dothis\XINT_divmod_aiszero\fi
    \if-#2\xint_dothis{\XINT_divmod_bneg #1}\fi
          \xint_orthat{\XINT_divmod_bpos #1#2}%
}%
\def\XINT_divmod_divbyzero #1#2[#3]#4.%
{%
    \XINT_signalcondition{DivisionByZero}{Division by #2[#3] of #1#4}{}%
    {{0}{0/1[0]}}% à revoir...
}%
\def\XINT_divmod_aiszero #1.{{0}{0/1[0]}}%
\def\XINT_divmod_bneg #1% f // -g = (-f) // g, f % -g = - ((-f) % g)
{%
    \expandafter\XINT_divmod_bneg_finish
    \romannumeral0\xint_UDsignfork
        #1{\XINT_divmod_bpos {}}%
         -{\XINT_divmod_bpos {-#1}}%
    \krof
}%
\def\XINT_divmod_bneg_finish#1#2%
```

```
{%
    \expandafter\xint_exchangetwo_keepbraces\expandafter
    {\romannumeral0\xintiiopp#2}{#1}%
}%
\def\XINT_divmod_bpos #1#2/#3[#4]#5/#6[#7].%
{%
    \expandafter\XINT_divmod_bpos_a
    \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.%
    \romannumeral0\expandafter\XINT_mod_D_b
    \romannumeral0\XINT_div_prepare{#3}{#6}{#3}{#3}{#6}%
    {#1#5}{#7-#4}{#2}{#4-#7}%
}%
\def\XINT_divmod_bpos_a #1.#2#3#4%
{%
    \expandafter\XINT_divmod_bpos_finish
    \romannumeral0\xintiidivision{#3}{#4}{/#2[#1]}%
}%
\def\XINT_divmod_bpos_finish #1#2#3{{#1}{#2#3}}%
```

8.50 \xintMod

1.2p. \xintMod{q1}{q2} computes q1 - q2*floor(q1/q2). Attention that it relies on \xintiiE{x}{e} returning x if e < 0.

Prior to 1.2p, that macro had the meaning now attributed to \xintModTrunc.

Modified (like \xintAdd and \xintSub) at 1.3 to use a l.c.m for final denominator.

```
\def\xintMod {\romannumeral0\xintmod }%
\def\xintmod #1{\expandafter\XINT_mod_a\romannumeral0\xintraw{#1}.}%
\def\XINT_mod_a #1#2.#3%
   {\expandafter\XINT_mod_b\expandafter #1\romannumeral0\xintraw{#3}#2.}%
\def\XINT_mod_b #1#2% #1 de A, #2 de B.
{%
    \if0#2\xint_dothis{\XINT_mod_divbyzero #1#2}\fi
    \if0#1\xint_dothis\XINT_mod_aiszero\fi
    \if-#2\xint_dothis{\XINT_mod_bneg #1}\fi
          \xint_orthat{\XINT_mod_bpos #1#2}%
}%
```

Attention to not move ModTrunc code beyond that point.

```
\let\XINT_mod_divbyzero\XINT_modtrunc_divbyzero
\let\XINT_mod_aiszero  \XINT_modtrunc_aiszero
\def\XINT_mod_bneg #1% f % -g = - ((-f) % g), for g > 0
{%
    \xintiiopp\xint_UDsignfork
        #1{\XINT_mod_bpos {}}%
         -{\XINT_mod_bpos {-#1}}%
    \krof
}%
\def\XINT_mod_bpos #1#2/#3[#4]#5/#6[#7].%
{%
    \expandafter\XINT_mod_bpos_a
    \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.%
    \romannumeral0\expandafter\XINT_mod_D_b
    \romannumeral0\XINT_div_prepare{#3}{#6}{#3}{#3}{#6}%
    {#1#5}{#7-#4}{#2}{#4-#7}%
}%
\def\XINT_mod_D_a #1#2%
{%
    \expandafter\XINT_mod_D_b
    \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
}%
\def\XINT_mod_D_b #1#2{\XINT_mod_D_c #2\Z}%
\def\XINT_mod_D_c #1#2\Z
{%
    \xint_gob_til_zero #1\XINT_mod_D_exit0\XINT_mod_D_a {#1#2}%
}%
\def\XINT_mod_D_exit0\XINT_mod_D_a #1#2#3%
{%
    \expandafter\XINT_mod_E
    \romannumeral0\xintiiquo {#3}{#2}.{#2}%
}%
\def\XINT_mod_E #1.#2#3%
{%
    \expandafter\XINT_mod_F
    \romannumeral0\xintiimul{#1}{#3}.{\xintiiQuo{#3}{#2}}{#1}%
}%
\def\XINT_mod_F #1.#2#3#4#5#6#7%
{%
    {#1}{\xintiiE{\xintiiMul{#4}{#3}}{#5}}%
        {\xintiiE{\xintiiMul{#6}{#2}}{#7}}%
}%
\def\XINT_mod_bpos_a #1.#2#3#4{\xintiirem {#3}{#4}/#2[#1]}%
```

8.51 \xintIsOne

New with 1.09a. Could be more efficient. For fractions with big powers of tens, it is better to use \xintCmp{f}{1}. Restyled in 1.09i.

```
\def\xintIsOne   {\romannumeral0\xintisone }%
\def\xintisone #1{\expandafter\XINT_fracisone
                 \romannumeral0\xintrawwithzeros{#1}\Z }%
\def\XINT_fracisone #1/#2\Z
    {\if0\xintiiCmp {#1}{#2}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%
```

8.52 \xintGeq

```
\def\xintGeq {\romannumeral0\xintgeq }%
\def\xintgeq #1%
{%
    \expandafter\XINT_fgeq\expandafter {\romannumeral0\xintabs {#1}}%
}%
\def\XINT_fgeq #1#2%
{%
    \expandafter\XINT_fgeq_A \romannumeral0\xintabs {#2}#1%
}%
```

```
\def\XINT_fgeq_A #1%
{%
    \xint_gob_til_zero #1\XINT_fgeq_Zii 0%
    \XINT_fgeq_B #1%
}%
\def\XINT_fgeq_Zii 0\XINT_fgeq_B #1[#2]#3[#4]{ 1}%
\def\XINT_fgeq_B #1/#2[#3]#4#5/#6[#7]%
{%
    \xint_gob_til_zero #4\XINT_fgeq_Zi 0%
    \expandafter\XINT_fgeq_C\expandafter
    {\the\numexpr #7-#3\expandafter}\expandafter
    {\romannumeral0\xintiimul {#4#5}{#2}}%
    {\romannumeral0\xintiimul {#6}{#1}}%
}%
\def\XINT_fgeq_Zi 0#1#2#3#4#5#6#7{ 0}%
\def\XINT_fgeq_C #1#2#3%
{%
    \expandafter\XINT_fgeq_D\expandafter
    {#3}{#1}{#2}%
}%
\def\XINT_fgeq_D #1#2#3%
{%
    \expandafter\XINT_cntSgnFork\romannumeral`&&@\expandafter\XINT_cntSgn
     \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\xint:
    { 0}{\XINT_fgeq_E #2\Z {#3}{#1}}{ 1}%
}%
\def\XINT_fgeq_E #1%
{%
    \xint_UDsignfork
        #1\XINT_fgeq_Fd
         -{\XINT_fgeq_Fn #1}%
    \krof
}%

\def\XINT_fgeq_Fd #1\Z #2#3%
{%
    \expandafter\XINT_fgeq_Fe
    \romannumeral0\XINT_dsx_addzeros {#1}#3;\xint:#2\xint:
}%
\def\XINT_fgeq_Fe #1\xint:#2#3\xint:{\XINT_geq_plusplus #2#1\xint:#3\xint:}%
\def\XINT_fgeq_Fn #1\Z #2#3%
{%
    \expandafter\XINT_fgeq_Fo
    \romannumeral0\XINT_dsx_addzeros {#1}#2;\xint:#3\xint:
}%
\def\XINT_fgeq_Fo #1#2\xint:#3\xint:{\XINT_geq_plusplus #1#3\xint:#2\xint:}%
```

8.53 \xintMax

```
\def\xintMax {\romannumeral0\xintmax }%
\def\xintmax #1%
{%
    \expandafter\XINT_fmax\expandafter {\romannumeral0\xintraw {#1}}%
}%
```

```
\def\XINT_fmax #1#2%
{%
    \expandafter\XINT_fmax_A\romannumeral0\xintraw {#2}#1%
}%
\def\XINT_fmax_A #1#2/#3[#4]#5#6/#7[#8]%
{%
    \xint_UDsignsfork
      #1#5\XINT_fmax_minusminus
       -#5\XINT_fmax_firstneg
       #1-\XINT_fmax_secondneg
        --\XINT_fmax_nonneg_a
    \krof
    #1#5{#2/#3[#4]}{#6/#7[#8]}%
}%
\def\XINT_fmax_minusminus --%
   {\expandafter-\romannumeral0\XINT_fmin_nonneg_b }%
\def\XINT_fmax_firstneg #1-#2#3{ #1#2}%
\def\XINT_fmax_secondneg -#1#2#3{ #1#3}%
\def\XINT_fmax_nonneg_a #1#2#3#4%
{%
    \XINT_fmax_nonneg_b {#1#3}{#2#4}%
}%
\def\XINT_fmax_nonneg_b #1#2%
{%
    \if0\romannumeral0\XINT_fgeq_A #1#2%
          \xint_afterfi{ #1}%
    \else \xint_afterfi{ #2}%
    \fi
}%
```

8.54 \xintMaxof

1.21 protects \xintMaxof against items with non terminated \the\numexpr expressions.
The macro is not compatible with an empty list.

```
\def\xintMaxof      {\romannumeral0\xintmaxof }%
\def\xintmaxof    #1{\expandafter\XINT_maxof_a\romannumeral`&&@#1\xint:}%
\def\XINT_maxof_a #1{\expandafter\XINT_maxof_b\romannumeral0\xintraw{#1}!}%
\def\XINT_maxof_b #1!#2%
           {\expandafter\XINT_maxof_c\romannumeral`&&@#2!{#1}!}%
\def\XINT_maxof_c #1%
           {\xint_gob_til_xint: #1\XINT_maxof_e\xint:\XINT_maxof_d #1}%
\def\XINT_maxof_d #1!%
           {\expandafter\XINT_maxof_b\romannumeral0\xintmax {#1}}%
\def\XINT_maxof_e #1!#2!{ #2}%
```

8.55 \xintMin

```
\def\xintMin {\romannumeral0\xintmin }%
\def\xintmin #1%
{%
    \expandafter\XINT_fmin\expandafter {\romannumeral0\xintraw {#1}}%
}%
\def\XINT_fmin #1#2%
```

```
{%
    \expandafter\XINT_fmin_A\romannumeral0\xintraw {#2}#1%
}%
\def\XINT_fmin_A #1#2/#3[#4]#5#6/#7[#8]%
{%
    \xint_UDsignsfork
      #1#5\XINT_fmin_minusminus
       -#5\XINT_fmin_firstneg
       #1-\XINT_fmin_secondneg
        --\XINT_fmin_nonneg_a
    \krof
    #1#5{#2/#3[#4]}{#6/#7[#8]}%
}%
\def\XINT_fmin_minusminus --%
   {\expandafter-\romannumeral0\XINT_fmax_nonneg_b }%
\def\XINT_fmin_firstneg #1-#2#3{ -#3}%
\def\XINT_fmin_secondneg -#1#2#3{ -#2}%
\def\XINT_fmin_nonneg_a #1#2#3#4%
{%
    \XINT_fmin_nonneg_b {#1#3}{#2#4}%
}%
\def\XINT_fmin_nonneg_b #1#2%
{%
    \if0\romannumeral0\XINT_fgeq_A #1#2%
          \xint_afterfi{ #2}%
    \else \xint_afterfi{ #1}%
    \fi
}%
```

8.56 \xintMinof

1.21 protects \xintMinof against items with non terminated \the\numexpr expressions.
The macro is not compatible with an empty list.

```
\def\xintMinof      {\romannumeral0\xintminof }%
\def\xintminof    #1{\expandafter\XINT_minof_a\romannumeral`&&@#1\xint:}%
\def\XINT_minof_a #1{\expandafter\XINT_minof_b\romannumeral0\xintraw{#1}!}%
\def\XINT_minof_b #1!#2%
           {\expandafter\XINT_minof_c\romannumeral`&&@#2!{#1}!}%
\def\XINT_minof_c #1%
           {\xint_gob_til_xint: #1\XINT_minof_e\xint:\XINT_minof_d #1}%
\def\XINT_minof_d #1!%
           {\expandafter\XINT_minof_b\romannumeral0\xintmin {#1}}%
\def\XINT_minof_e #1!#2!{ #2}%
```

8.57 \xintCmp

```
\def\xintCmp {\romannumeral0\xintcmp }%
\def\xintcmp #1%
{%
    \expandafter\XINT_fcmp\expandafter {\romannumeral0\xintraw {#1}}%
}%
\def\XINT_fcmp #1#2%
{%
```

```
    \expandafter\XINT_fcmp_A\romannumeral0\xintraw {#2}#1%
}%
\def\XINT_fcmp_A #1#2/#3[#4]#5#6/#7[#8]%
{%
    \xint_UDsignsfork
      #1#5\XINT_fcmp_minusminus
       -#5\XINT_fcmp_firstneg
       #1-\XINT_fcmp_secondneg
        --\XINT_fcmp_nonneg_a
    \krof
    #1#5{#2/#3[#4]}{#6/#7[#8]}%
}%
\def\XINT_fcmp_minusminus --#1#2{\XINT_fcmp_B #2#1}%
\def\XINT_fcmp_firstneg #1-#2#3{ -1}%
\def\XINT_fcmp_secondneg -#1#2#3{ 1}%
\def\XINT_fcmp_nonneg_a #1#2%
{%
    \xint_UDzerosfork
      #1#2\XINT_fcmp_zerozero
       0#2\XINT_fcmp_firstzero
       #10\XINT_fcmp_secondzero
        00\XINT_fcmp_pos
    \krof
    #1#2%
}%
\def\XINT_fcmp_zerozero   #1#2#3#4{ 0}%
\def\XINT_fcmp_firstzero  #1#2#3#4{ -1}%
\def\XINT_fcmp_secondzero #1#2#3#4{ 1}%
\def\XINT_fcmp_pos #1#2#3#4%
{%
    \XINT_fcmp_B #1#3#2#4%
}%
\def\XINT_fcmp_B #1/#2[#3]#4/#5[#6]%
{%
    \expandafter\XINT_fcmp_C\expandafter
    {\the\numexpr #6-#3\expandafter}\expandafter
    {\romannumeral0\xintiimul {#4}{#2}}%
    {\romannumeral0\xintiimul {#5}{#1}}%
}%
\def\XINT_fcmp_C #1#2#3%
{%
    \expandafter\XINT_fcmp_D\expandafter
    {#3}{#1}{#2}%
}%
\def\XINT_fcmp_D #1#2#3%
{%
    \expandafter\XINT_cntSgnFork\romannumeral`&&@\expandafter\XINT_cntSgn
    \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\xint:
    { -1}{\XINT_fcmp_E #2\Z {#3}{#1}}{ 1}%
}%
\def\XINT_fcmp_E #1%
{%
```

```
    \xint_UDsignfork
        #1\XINT_fcmp_Fd
         -{\XINT_fcmp_Fn #1}%
    \krof
}%
```

```
\def\XINT_fcmp_Fd #1\Z #2#3%
{%
    \expandafter\XINT_fcmp_Fe
    \romannumeral0\XINT_dsx_addzeros {#1}#3;\xint:#2\xint:
}%
\def\XINT_fcmp_Fe #1\xint:#2#3\xint:{\XINT_cmp_plusplus #2#1\xint:#3\xint:}%
\def\XINT_fcmp_Fn #1\Z #2#3%
{%
    \expandafter\XINT_fcmp_Fo
    \romannumeral0\XINT_dsx_addzeros {#1}#2;\xint:#3\xint:
}%
\def\XINT_fcmp_Fo #1#2\xint:#3\xint:{\XINT_cmp_plusplus #1#3\xint:#2\xint:}%
```

8.58 `\xintAbs`

```
\def\xintAbs   {\romannumeral0\xintabs }%
\def\xintabs #1{\expandafter\XINT_abs\romannumeral0\xintraw {#1}}%
```

8.59 `\xintOpp`

```
\def\xintOpp   {\romannumeral0\xintopp }%
\def\xintopp #1{\expandafter\XINT_opp\romannumeral0\xintraw {#1}}%
```

8.60 `\xintSgn`

```
\def\xintSgn   {\romannumeral0\xintsgn }%
\def\xintsgn #1{\expandafter\XINT_sgn\romannumeral0\xintraw {#1}\xint:}%
```

8.61 Floating point macros

For a long time the float routines dating back to releases 1.07/1.08a (May-June 2013) were not modified.

Since 1.2f (March 2016) the four operations first round their arguments to \xinttheDigits-floats (or P-floats), not (\xinttheDigits+2)-floats or (P+2)-floats as was the case with earlier releases.

The four operations addition, subtraction, multiplication, division have always produced the correct rounding of the theoretical exact value to P or \xinttheDigits digits when the inputs are decimal numbers with at most P digits, and arbitrary decimal exponent part.

From 1.08a to 1.2j, \xintFloat (and \XINTinFloat which is used to parse inputs to other float macros) handled a fractional input A/B via an initial replacement to A'/B' where A' and B' were A and B truncated to Q+2 digits (where asked-for precision is Q), and then they correctly rounded A'/B' to Q digits. But this meant that this rounding of the input could differ (by up to one unit in the last place) from the correct rounding of the original A/B to the asked-for number of digits (which until 1.2f in uses as auxiliary to the macros for the basic operations was 2 more than the prevailing precision).

Since 1.2k all inputs are correctly rounded to the asked-for number of digits (this was, I think, the case in the 1.07 release -- there are no code comments -- but was, afaicr, not very efficiently done, and this is why the 1.08a release opeted for truncation of the numerator and denominator.)

Notice that in float expressions, the / is treated as operator, hence the above discussion makes a difference only for the special input form qfloat(A/B) or for an \xintexpr A/B\relax embedded in the float expression, with A or B having more digits than the prevailing float precision.

Internally there is no inner representation of P-floats as such !!!!!

The input parser will again compute the length of the mantissa on each use !!! This is obviously something that must be improved upon before implementation of higher functions.

Currently, special tricks are used to quickly recognize inputs having no denominators, or fractions whose numerators and denominators are not too long compared to the target precision P, and in particular P-floats or quotients of two such.

Another long-standing issue is that float multiplication will first compute the 2P or 2P-1 digits of the exact product, and then round it to P digits. This is sub-optimal for large P particularly as the multiplication algorithm is basically the schoolbook one, hence *worse* than quadratic in the TEX implementation which has extra cost of fetching long sequences of tokens.

8.62 `\xintFloat`

1.2f and 1.2g brought some refactoring which resulted in faster treatment of decimal inputs. 1.2i dropped use of some old routines dating back to pre 1.2 era in favor of more modern \xintDSRr for rounding. Then 1.2k improves again the handling of denominators B with few digits.

But the main change with 1.2k is a complete rewrite of the B>1 case in order to achieve again correct rounding in all cases.

The original version from 1.07 (May 2013) computed the exact rounding to P digits for all inputs. But from 1.08 on (June 2013), the macro handled A/B input by first truncating both A and B to at most P+2 digits. This meant that decimal input (arbitrarily long, with scientific part) was correctly rounded, but in case of fractional input there could be up to 0.6 unit in the last place difference of the produced rounding to the input, hence the output could differ from the correct rounding.

Example with 16 digits (the default): \xintFloat {1/17597472569900621233}
with xintfrac 1.07: 5.682634230727187e-20
with xintfrac 1.08b--1.2j: 5.682634230727188e-20
with xintfrac 1.2k: 5.682634230727187e-20
The exact value is 5.682634230727187499924124...e-20, showing that 1.07 and 1.2k produce the correct rounding.

Currently the code ends in a more costly branch in about 1 case among 500, where it does some extra operations (a multiplication in particular). There is a free parameter delta (here set at 4), I have yet to make some numerical explorations, to see if it could be favorable to set it to a higher value (with delta=5, there is only 1 exceptional case in 5000, etc...).

I have always hesitated about the policy of printing 10.00...0 in case of rounding upwards to the next power of ten. Already since 1.2f \XINTinFloat always produced a mantissa with exactly P digits (except for the zero value). Starting with 1.2k, \xintFloat drops this habit of printing 10.00..0 in such cases. Side note: the rounding-up detection worked when the input A/B was with numerator A and denominator B having each less than P+2 digits, or with B=1, else, it could happen that the output was a power of ten but not detected to be a rounding up of the original fraction. The value was ok, but printed 1.0...0eN with P-1 zeroes, not 10.0...0e(N-1).

I decided it was not worth the effort to enhance the algorithm to detect with 100% fiability all cases of rounding up to next power of ten, hence 1.2k dropped this.

To avoid duplication of code, and any extra burden on \XINTinFloat, which is the macro used internally by the float macros for parsing their inputs, we simply make now \xintFloat a wrapper of \XINTinFloat.

```
\def\xintFloat   {\romannumeral0\xintfloat }%
\def\xintfloat #1{\XINT_float_chkopt #1\xint:}%
\def\XINT_float_chkopt #1%
{%
```

```
    \ifx [#1\expandafter\XINT_float_opt
       \else\expandafter\XINT_float_noopt
    \fi  #1%
}%
\def\XINT_float_noopt #1\xint:%
{%
    \expandafter\XINT_float_post
    \romannumeral0\XINTinfloat[\XINTdigits]{#1}\XINTdigits.%
}%
\def\XINT_float_opt [\xint:#1]%
{%
    \expandafter\XINT_float_opt_a\the\numexpr #1.%
}%
\def\XINT_float_opt_a #1.#2%
{%
    \expandafter\XINT_float_post
    \romannumeral0\XINTinfloat[#1]{#2}#1.%
}%
\def\XINT_float_post #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_float_zero
       0#1\XINT_float_neg
       0-\XINT_float_pos
    \krof #1%
}%[
\def\XINT_float_zero #1]#2.{ 0.e0}%
\def\XINT_float_neg-{\expandafter-\romannumeral0\XINT_float_pos}%
\def\XINT_float_pos #1#2[#3]#4.%
{%
    \expandafter\XINT_float_pos_done\the\numexpr#3+#4-\xint_c_i.#1.#2;%
}%
\def\XINT_float_pos_done #1.#2;{ #2e#1}%
```

8.63 \XINTinFloat, \XINTinFloatS

This routine is like \xintFloat but produces an output of the shape A[N] which is then parsed faster as input to other float macros. Float operations in \xintfloatexpr...\relax use internally this format.

It must be used in form \XINTinFloat[P]{f}: the optional [P] is mandatory.

Since 1.2f, the mantissa always has exactly P digits even in case of rounding up to next power of ten. This simplifies other routines.

1.2g added a variant \XINTinFloatS which, in case of decimal input with less than the asked for precision P will not add extra zeros to the mantissa. For example it may output 2[0] even if P=500, rather than the canonical representation 200...000[-499]. This is how \xintFloatMul and \xintFloatDiv parse their inputs, which speeds-up follow-up processing. But \xintFloatAdd and \xintFloatSub still use \XINTinFloat for parsing their inputs; anyway this will have to be changed again when inner structure will carry upfront at least the length of mantissa as data.

Each time \XINTinFloat is called it at least computes a length. Naturally if we had some format for floats that would be dispensed of...
something like <letterP><length of mantissa>.mantissa.exponent, etc... not yet.

Since 1.2k, \XINTinFloat always correctly rounds its argument, even if it is a fraction with very big numerator and denominator. See the discussion of \xintFloat.

```
\def\XINTinFloat {\romannumeral0\XINTinfloat }%
\def\XINTinfloat
   {\expandafter\XINT_infloat_clean\romannumeral0\XINT_infloat}%
\def\XINT_infloat_clean #1%
   {\if #1!\xint_dothis\XINT_infloat_clean_a\fi\xint_orthat{ }#1}%
```

Ici on ajoute les zeros pour faire exactement avec P chiffres. Car le #1 = P - L avec L la longueur de #2, (ou de abs(#2), ici le #2 peut avoir un signe) qui est < P

```
\def\XINT_infloat_clean_a !#1.#2[#3]%
{%
    \expandafter\XINT_infloat_done
    \the\numexpr #3-#1\expandafter.%
    \romannumeral0\XINT_dsx_addzeros {#1}#2;;%
}%
\def\XINT_infloat_done #1.#2;{ #2[#1]}%
```

variant which allows output with shorter mantissas.

```
\def\XINTinFloatS {\romannumeral0\XINTinfloatS}%
\def\XINTinfloatS
   {\expandafter\XINT_infloatS_clean\romannumeral0\XINT_infloat}%
\def\XINT_infloatS_clean #1%
   {\if #1!\xint_dothis\XINT_infloatS_clean_a\fi\xint_orthat{ }#1}%
\def\XINT_infloatS_clean_a !#1.{ }%
```

début de la routine proprement dite, l'argument optionnel est obligatoire.

```
\def\XINT_infloat [#1]#2%
{%
    \expandafter\XINT_infloat_a\the\numexpr #1\expandafter.%
    \romannumeral0\XINT_infrac {#2}%
}%
```

#1=P, #2=n, #3=A, #4=B.

```
\def\XINT_infloat_a #1.#2#3#4%
{%
```

micro boost au lieu d'utiliser \XINT_isOne{#4}, mais pas bon style.

```
    \if1\XINT_is_One#4XY%
      \expandafter\XINT_infloat_sp
    \else\expandafter\XINT_infloat_fork
    \fi #3.{#1}{#2}{#4}%
}%
```

Special quick treatment of B=1 case (1.2f then again 1.2g.)
maintenant: A.{P}{N}{1} Il est possible que A soit nul.

```
\def\XINT_infloat_sp #1%
{%
    \xint_UDzerominusfork
     #1-\XINT_infloat_spzero
    0#1\XINT_infloat_spneg
     0-\XINT_infloat_sppos
    \krof #1%
}%
```

Attention surtout pas 0/1[0] ici.

```
\def\XINT_infloat_spzero 0.#1#2#3{ 0[0]}%
\def\XINT_infloat_spneg-%
    {\expandafter\XINT_infloat_spnegend\romannumeral0\XINT_infloat_sppos}%
\def\XINT_infloat_spnegend #1%
    {\if#1!\expandafter\XINT_infloat_spneg_needzeros\fi -#1}%
\def\XINT_infloat_spneg_needzeros -!#1.{!#1.-}%
```

in: A.{P}{N}{1}
out: P-L.A.P.N.

```
\def\XINT_infloat_sppos #1.#2#3#4%
{%
    \expandafter\XINT_infloat_sp_b\the\numexpr#2-\xintLength{#1}.#1.#2.#3.%
}%
```

#1= P-L. Si c'est positif ou nul il faut retrancher #1 à l'exposant, et ajouter autant de zéros. On regarde premier token. P-L.A.P.N.

```
\def\XINT_infloat_sp_b #1%
{%
    \xint_UDzerominusfork
     #1-\XINT_infloat_sp_quick
     0#1\XINT_infloat_sp_c
      0-\XINT_infloat_sp_needzeros
    \krof #1%
}%
```

Ici P=L. Le cas usuel dans \xintfloatexpr.

```
\def\XINT_infloat_sp_quick 0.#1.#2.#3.{ #1[#3]}%
```

Ici #1=P-L est >0. L'exposant sera N-(P-L). #2=A. #3=P. #4=N.
18 mars 2016. En fait dans certains contextes il est sous-optimal d'ajouter les zéros. Par exemple quand c'est appelé par la multiplication ou la division, c'est idiot de convertir 2 en 200000...00000[-499]. Donc je redéfinis addzeros en needzeroes. Si on appelle sous la forme \XINTinFloatS, on ne fait pas l'addition de zeros.

```
\def\XINT_infloat_sp_needzeros #1.#2.#3.#4.{!#1.#2[#4]}%
```

L-P=#1.A=#2#3.P=#4.N=#5.
Ici P<L. Il va falloir arrondir. Attention si on va à la puissance de 10 suivante. En #1 on a L-P qui est >0. L'exposant final sera N+L-P, sauf dans le cas spécial, il sera alors N+L-P+1. L'ajustement final est fait par \XINT_infloat_Y.

```
\def\XINT_infloat_sp_c -#1.#2#3.#4.#5.%
{%
    \expandafter\XINT_infloat_Y
    \the\numexpr #5+#1\expandafter.%
    \romannumeral0\expandafter\XINT_infloat_sp_round
    \romannumeral0\XINT_split_fromleft
    (\xint_c_i+#4).#2#3\xint_bye2345678\xint_bye..#2%
}%
\def\XINT_infloat_sp_round #1.#2.%
{%
    \XINT_dsrr#1\xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax.%
}%
```

General branch for A/B with B>1 inputs. It achieves correct rounding always since 1.2k (done January 2, 2017.) This branch is never taken for A=0 because \XINT_infrac will have returned B=1 then.

```
\def\XINT_infloat_fork #1%
{%
    \xint_UDsignfork
     #1\XINT_infloat_J
     -\XINT_infloat_K
    \krof #1%
}%
\def\XINT_infloat_J-{\expandafter-\romannumeral0\XINT_infloat_K }%
```

A.{P}{n}{B} avec B>1.

```
\def\XINT_infloat_K #1.#2%
{%
    \expandafter\XINT_infloat_L
    \the\numexpr\xintLength{#1}\expandafter.\the\numexpr #2+\xint_c_iv.{#1}{#2}%
}%
```

|A|.P+4.{A}{P}{n}{B}. We check if A already has length <= P+4.

```
\def\XINT_infloat_L #1.#2.%
{%
    \ifnum #1>#2
      \expandafter\XINT_infloat_Ma
    \else
      \expandafter\XINT_infloat_Mb
    \fi #1.#2.%
}%
```

|A|.P+4.{A}{P}{n}{B}. We will keep only the first P+4 digits of A, denoted A'' in what follows.
output: u=-0.A''.junk.P+4.|A|.{A}{P}{n}{B}

```
\def\XINT_infloat_Ma #1.#2.#3%
{%
    \expandafter\XINT_infloat_MtoN\expandafter-\expandafter0\expandafter.%
    \romannumeral0\XINT_split_fromleft#2.#3\xint_bye2345678\xint_bye..%
    #2.#1.{#3}%
}%
```

|A|.P+4.{A}{P}{n}{B}.
Here A is short. We set u = P+4-|A|, and A''=A (A' = 10^u A)
output: u.A''..P+4.|A|.{A}{P}{n}{B}

```
\def\XINT_infloat_Mb #1.#2.#3%
```

```
{%
    \expandafter\XINT_infloat_MtoN\the\numexpr#2-#1.%
    #3..#2.#1.{#3}%
}%
```

input u.A''.junk.P+4.|A|.{A}{P}{n}{B}
output |B|.P+4.{B}u.A''.P.|A|.n.{A}{B}

```
\def\XINT_infloat_MtoN #1.#2.#3.#4.#5.#6#7#8#9%
{%
   \expandafter\XINT_infloat_N
   \the\numexpr\xintLength{#9}.#4.{#9}#1.#2.#7.#5.#8.{#6}{#9}%
}%
\def\XINT_infloat_N #1.#2.%
{%
    \ifnum #1>#2
      \expandafter\XINT_infloat_Oa
    \else
      \expandafter\XINT_infloat_Ob
    \fi #1.#2.%
}%
```

input |B|.P+4.{B}u.A''.P.|A|.n.{A}{B}
output v=-0.B''.junk.|B|.u.A''.P.|A|.n.{A}{B}

```
\def\XINT_infloat_Oa #1.#2.#3%
{%
    \expandafter\XINT_infloat_P\expandafter-\expandafter0\expandafter.%
    \romannumeral0\XINT_split_fromleft#2.#3\xint_bye2345678\xint_bye..%
    #1.%
}%
```

output v=P+4-|B|>=0.B''.junk.|B|.u.A''.P.|A|.n.{A}{B}

```
\def\XINT_infloat_Ob #1.#2.#3%
{%
    \expandafter\XINT_infloat_P\the\numexpr#2-#1.#3..#1.%
}%
```

input v.B''.junk.|B|.u.A''.P.|A|.n.{A}{B}
output Q1.P.|B|.|A|.n.{A}{B}
Q1 = division euclidienne de A''.10^{u-v+P+3} par B''.

Special detection of cases with A and B both having length at most P+4: this will happen when called from \xintFloatDiv as A and B (produced then via \XINTinFloatS) will have at most P digits. We then only need integer division with P+1 extra zeros, not P+3.

```
\def\XINT_infloat_P #1#2.#3.#4.#5.#6#7.#8.#9.%
{%
   \csname XINT_infloat_Q\if-#1\else\if-#6\else q\fi\fi\expandafter\endcsname
   \romannumeral0\xintiiquo
   {\romannumeral0\XINT_dsx_addzerosnofuss
      {#6#7-#1#2+#9+\xint_c_iii\if-#1\else\if-#6\else-\xint_c_ii\fi\fi}#8;}%
   {#3}.#9.#5.%
}%
```

«quick» branch.

```
\def\XINT_infloat_Qq #1.#2.%
{%
    \expandafter\XINT_infloat_Rq
    \romannumeral0\XINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..#2.%
}%
\def\XINT_infloat_Rq #1.#2#3.%
{%
    \ifnum#2<\xint_c_v
         \expandafter\XINT_infloat_SEq
    \else\expandafter\XINT_infloat_SUp
    \fi
    {\if.#3.\xint_c_\else\xint_c_i\fi}#1.%
}%
```

standard branch which will have to handle undecided rounding, if too close to a mid-value.

```
\def\XINT_infloat_Q #1.#2.%
{%
    \expandafter\XINT_infloat_R
    \romannumeral0\XINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..#2.%
}%
\def\XINT_infloat_R #1.#2#3#4#5.%
{%
    \if.#5.\expandafter\XINT_infloat_Sa\else\expandafter\XINT_infloat_Sb\fi
    #2#3#4#5.#1.%
}%
```

trailing digits.Q.P.|B|.|A|.n.{A}{B}
#1=trailing digits (they may have leading zeros.)

```
\def\XINT_infloat_Sa #1.%
{%
    \ifnum#1>500 \xint_dothis\XINT_infloat_SUp\fi
    \ifnum#1<499 \xint_dothis\XINT_infloat_SEq\fi
    \xint_orthat\XINT_infloat_X\xint_c_
}%
\def\XINT_infloat_Sb #1.%
{%
    \ifnum#1>5009 \xint_dothis\XINT_infloat_SUp\fi
    \ifnum#1<4990 \xint_dothis\XINT_infloat_SEq\fi
    \xint_orthat\XINT_infloat_X\xint_c_i
}%
```

epsilon #2=Q.#3=P.#4=|B|.#5=|A|.#6=n.{A}{B}
exposant final est n+|A|-|B|-P+epsilon

```
\def\XINT_infloat_SEq #1#2.#3.#4.#5.#6.#7#8%
{%
    \expandafter\XINT_infloat_SY
    \the\numexpr #6+#5-#4-#3+#1.#2.%
}%
\def\XINT_infloat_SY #1.#2.{ #2[#1]}%
```

initial digit #2 put aside to check for case of rounding up to next power of ten, which will need adjustment of mantissa and exponent.

```
\def\XINT_infloat_SUp #1#2#3.#4.#5.#6.#7.#8#9%
{%
    \expandafter\XINT_infloat_Y
    \the\numexpr#7+#6-#5-#4+#1\expandafter.%
    \romannumeral0\xintinc{#2#3}.#2%
}%
```

epsilon Q.P.|B|.|A|.n.{A}{B}

\xintDSH{-x}{U} multiplies U by 10^x. When x is negative, this means it truncates (i.e. it drops the last -x digits).

We don't try to optimize too much macro calls here, the odds are 2 per 1000 for this branch to be taken. Perhaps in future I will use higher free parameter d, which currently is set at 4.

#1=epsilon, #2#3=Q, #4=P, #5=|B|, #6=|A|, #7=n, #8=A, #9=B

```
\def\XINT_infloat_X #1#2#3.#4.#5.#6.#7.#8#9%
{%
   \expandafter\XINT_infloat_Y
   \the\numexpr #7+#6-#5-#4+#1\expandafter.%
   \romannumeral`&&@\romannumeral0\xintiiiflt
     {\xintDSH{#6-#5-#4+#1}{\xintDouble{#8}}}%
     {\xintiiMul{\xintInc{\xintDouble{#2#3}}}{#9}}%
   \xint_firstofone
   \xintinc{#2#3}.#2%
}%
```

check for rounding up to next power of ten.

```
\def\XINT_infloat_Y #1{%
\def\XINT_infloat_Y ##1.##2##3.##4%
{%
   \if##49\if##21\expandafter\expandafter\expandafter\XINT_infloat_Z\fi\fi
   #1##2##3[##1]%
}}\XINT_infloat_Y{ }%
```

#1=1, #2=0.

```
\def\XINT_infloat_Z #1#2#3[#4]%
{%
    \expandafter\XINT_infloat_ZZ\the\numexpr#4+\xint_c_i.#3.%
}%
\def\XINT_infloat_ZZ #1.#2.{ 1#2[#1]}%
```

8.64 \xintPFloat

1.1. This is a prettifying printing macro for floats.

The macro applies one simple rule: x.yz...eN will drop scientific notation in favor of pure decimal notation if -5<=N<=5. This is the default behaviour of Maple. The N here is as produced on output by \xintFloat.

Special case: the zero value is printed 0. (with a dot)

The coding got simpler with 1.2k as its \xintFloat always produces a mantissa with exactly P digits (no more 10.0...0eN annoying exception).

```
\def\xintPFloat   {\romannumeral0\xintpfloat }%
\def\xintpfloat #1{\XINT_pfloat_chkopt #1\xint:}%
\def\XINT_pfloat_chkopt #1%
{%
    \ifx [#1\expandafter\XINT_pfloat_opt
       \else\expandafter\XINT_pfloat_noopt
    \fi  #1%
}%
\def\XINT_pfloat_noopt #1\xint:%
{%
    \expandafter\XINT_pfloat_a
    \romannumeral0\xintfloat [\XINTdigits]{#1};\XINTdigits.%
}%

\def\XINT_pfloat_opt [\xint:#1]%
{%
    \expandafter\XINT_pfloat_opt_a \the\numexpr #1.%
}%
\def\XINT_pfloat_opt_a #1.#2%
{%
    \expandafter\XINT_pfloat_a\romannumeral0\xintfloat [#1]{#2};#1.%
}%
\def\XINT_pfloat_a #1%
{%
    \xint_UDzerominusfork
        #1-\XINT_pfloat_zero
        0#1\XINT_pfloat_neg
        0-\XINT_pfloat_pos
    \krof #1%
}%

\def\XINT_pfloat_zero #1;#2.{ 0.}%
\def\XINT_pfloat_neg-{\expandafter-\romannumeral0\XINT_pfloat_pos }%

\def\XINT_pfloat_pos #1.#2e#3;#4.%
{%
    \ifnum #3>\xint_c_v  \xint_dothis\XINT_pfloat_no\fi
    \ifnum #3<-\xint_c_v \xint_dothis\XINT_pfloat_no\fi
    \ifnum #3<\xint_c_   \xint_dothis\XINT_pfloat_N\fi
    \ifnum #3>\numexpr #4-\xint_c_i\relax \xint_dothis\XINT_pfloat_Ps\fi
    \xint_orthat\XINT_pfloat_P #1#2e#3;%
}%
\def\XINT_pfloat_no #1#2;{ #1.#2}%
```

This is all simpler coded, now that 1.2k's \xintFloat always outputs a mantissa with exactly one digits before decimal mark always.

```
\def\XINT_pfloat_N #1e-#2;%
{%
```

```
    \csname XINT_pfloat_N_\romannumeral#2\endcsname #1%
}%
\def\XINT_pfloat_N_i  { 0.}%
\def\XINT_pfloat_N_ii { 0.0}%
\def\XINT_pfloat_N_iii{ 0.00}%
\def\XINT_pfloat_N_iv { 0.000}%
\def\XINT_pfloat_N_v  { 0.0000}%
```

```
\def\XINT_pfloat_P #1e#2;%
{%
    \csname XINT_pfloat_P_\romannumeral#2\endcsname #1%
}%
\def\XINT_pfloat_P_   #1{ #1.}%
\def\XINT_pfloat_P_i  #1#2{ #1#2.}%
\def\XINT_pfloat_P_ii #1#2#3{ #1#2#3.}%
\def\XINT_pfloat_P_iii#1#2#3#4{ #1#2#3#4.}%
\def\XINT_pfloat_P_iv #1#2#3#4#5{ #1#2#3#4#5.}%
\def\XINT_pfloat_P_v  #1#2#3#4#5#6{ #1#2#3#4#5#6.}%
```

```
\def\XINT_pfloat_Ps #1e#2;%
{%
    \csname XINT_pfloat_Ps\romannumeral#2\endcsname #100000;%
}%
\def\XINT_pfloat_Psi  #1#2#3;{ #1#2.}%
\def\XINT_pfloat_Psii #1#2#3#4;{ #1#2#3.}%
\def\XINT_pfloat_Psiii#1#2#3#4#5;{ #1#2#3#4.}%
\def\XINT_pfloat_Psiv #1#2#3#4#5#6;{ #1#2#3#4#5.}%
\def\XINT_pfloat_Psv  #1#2#3#4#5#6#7;{ #1#2#3#4#5#6.}%
```

8.65 \XINTinFloatFracdigits

1.09i, for frac function in \xintfloatexpr. This version computes exactly from the input the fractional part and then only converts it into a float with the asked-for number of digits. I will have to think it again some day, certainly.

1.1 removes optional argument for which there was anyhow no interface, for technical reasons having to do with \xintNewExpr.

1.1a renames the macro as \XINTinFloatFracdigits (from \XINTinFloatFrac) to be synchronous with the \XINTinFloatSqrt and \XINTinFloat habits related to \xintNewExpr problems.

Note to myself: I still have to rethink the whole thing about what is the best to do, the initial way of going through \xinttfrac was just a first implementation.

```
\def\XINTinFloatFracdigits {\romannumeral0\XINTinfloatfracdigits }%
\def\XINTinfloatfracdigits #1%
{%
    \expandafter\XINT_infloatfracdg_a\expandafter {\romannumeral0\xinttfrac{#1}}%
}%
\def\XINT_infloatfracdg_a {\XINTinfloat [\XINTdigits]}%
```

8.66 \xintFloatAdd, \XINTinFloatAdd

First included in release 1.07.

1.09ka improved a bit the efficiency. However the add, sub, mul, div routines were provisory and supposed to be revised soon.

Which didn't happen until 1.2f. Now, the inputs are first rounded to P digits, not P+2 as earlier.

```
\def\xintFloatAdd      {\romannumeral0\xintfloatadd }%
\def\xintfloatadd    #1{\XINT_fladd_chkopt \xintfloat #1\xint:}%
\def\XINTinFloatAdd    {\romannumeral0\XINTinfloatadd }%
\def\XINTinfloatadd  #1{\XINT_fladd_chkopt \XINTinfloatS #1\xint:}%
\def\XINT_fladd_chkopt #1#2%
{%
    \ifx [#2\expandafter\XINT_fladd_opt
       \else\expandafter\XINT_fladd_noopt
    \fi  #1#2%
}%
\def\XINT_fladd_noopt #1#2\xint:#3%
{%
    #1[\XINTdigits]%
    {\expandafter\XINT_FL_add_a
     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{#3}}%
}%
\def\XINT_fladd_opt #1[\xint:#2]%#3#4%
{%
    \expandafter\XINT_fladd_opt_a\the\numexpr #2.#1%
}%
\def\XINT_fladd_opt_a #1.#2#3#4%
{%
    #2[#1]{\expandafter\XINT_FL_add_a\romannumeral0\XINTinfloat[#1]{#3}#1.{#4}}%
}%
\def\XINT_FL_add_a #1%
{%
    \xint_gob_til_zero #1\XINT_FL_add_zero 0\XINT_FL_add_b #1%
}%

\def\XINT_FL_add_zero #1.#2{#2}%[[
\def\XINT_FL_add_b #1]#2.#3%
{%
    \expandafter\XINT_FL_add_c\romannumeral0\XINTinfloat[#2]{#3}#2.#1]%
}%

\def\XINT_FL_add_c #1%
{%
    \xint_gob_til_zero #1\XINT_FL_add_zero 0\XINT_FL_add_d #1%
}%

\def\XINT_FL_add_d #1[#2]#3.#4[#5]%
{%
    \ifnum\numexpr #2-#3-#5>\xint_c_\xint_dothis\xint_firstoftwo\fi
    \ifnum\numexpr #5-#3-#2>\xint_c_\xint_dothis\xint_secondoftwo\fi
    \xint_orthat\xintAdd {#1[#2]}{#4[#5]}%
}%
```

8.67 \xintFloatSub, \XINTinFloatSub

First done 1.07.
Starting with 1.2f the arguments undergo an intial rounding to the target precision P not P+2.

```
\def\xintFloatSub      {\romannumeral0\xintfloatsub }%
\def\xintfloatsub    #1{\XINT_flsub_chkopt \xintfloat #1\xint:}%
\def\XINTinFloatSub    {\romannumeral0\XINTinfloatsub }%
\def\XINTinfloatsub  #1{\XINT_flsub_chkopt \XINTinfloatS #1\xint:}%
\def\XINT_flsub_chkopt #1#2%
{%
    \ifx [#2\expandafter\XINT_flsub_opt
       \else\expandafter\XINT_flsub_noopt
    \fi  #1#2%
}%
\def\XINT_flsub_noopt #1#2\xint:#3%
{%
    #1[\XINTdigits]%
    {\expandafter\XINT_FL_add_a
     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{\xintOpp{#3}}}%
}%
\def\XINT_flsub_opt #1[\xint:#2]%#3#4%
{%
    \expandafter\XINT_flsub_opt_a\the\numexpr #2.#1%
}%
\def\XINT_flsub_opt_a #1.#2#3#4%
{%
    #2[#1]{\expandafter\XINT_FL_add_a\romannumeral0\XINTinfloat[#1]{#3}#1.{\xintOpp{#4}}}%
}%
```

8.68 \xintFloatMul, \XINTinFloatMul

1.07.
Starting with 1.2f the arguments are rounded to the target precision P not P+2.
1.2g handles the inputs via \XINTinFloatS which will be more efficient when the precision is large and the input is for example a small constant like 2.
1.2k does a micro improvement to the way the macro passes over control to its output routine (former version used a higher level \xintE causing some extra un-needed processing with two calls to \XINT_infrac where one was amply enough).

```
\def\xintFloatMul   {\romannumeral0\xintfloatmul   }%
\def\xintfloatmul   #1{\XINT_flmul_chkopt \xintfloat #1\xint:}%
\def\XINTinFloatMul {\romannumeral0\XINTinfloatmul }%
\def\XINTinfloatmul #1{\XINT_flmul_chkopt \XINTinfloatS #1\xint:}%
\def\XINT_flmul_chkopt #1#2%
{%
    \ifx [#2\expandafter\XINT_flmul_opt
       \else\expandafter\XINT_flmul_noopt
    \fi  #1#2%
}%
\def\XINT_flmul_noopt #1#2\xint:#3%
{%
    #1[\XINTdigits]%
    {\expandafter\XINT_FL_mul_a
     \romannumeral0\XINTinfloatS[\XINTdigits]{#2}\XINTdigits.{#3}}%
}%
\def\XINT_flmul_opt #1[\xint:#2]%#3#4%
{%
    \expandafter\XINT_flmul_opt_a\the\numexpr #2.#1%
}%
\def\XINT_flmul_opt_a #1.#2#3#4%
{%
    #2[#1]{\expandafter\XINT_FL_mul_a\romannumeral0\XINTinfloatS[#1]{#3}#1.{#4}}%
}%
\def\XINT_FL_mul_a #1[#2]#3.#4%
{%
    \expandafter\XINT_FL_mul_b\romannumeral0\XINTinfloatS[#3]{#4}#1[#2]%
}%

\def\XINT_FL_mul_b #1[#2]#3[#4]{\xintiiMul{#3}{#1}/1[#4+#2]}%
```

8.69 \xintFloatDiv, \XINTinFloatDiv

1.07.

Starting with 1.2f the arguments are rounded to the target precision P not P+2.

1.2g handles the inputs via \XINTinFloatS which will be more efficient when the precision is large and the input is for example a small constant like 2.

The actual rounding of the quotient is handled via \xintfloat (or \XINTinfloatS).

1.2k does the same kind of improvement in \XINT_FL_div_b as for multiplication: earlier code was unnecessarily high level.

```
\def\xintFloatDiv   {\romannumeral0\xintfloatdiv   }%
\def\xintfloatdiv   #1{\XINT_fldiv_chkopt \xintfloat #1\xint:}%
\def\XINTinFloatDiv {\romannumeral0\XINTinfloatdiv }%
\def\XINTinfloatdiv #1{\XINT_fldiv_chkopt \XINTinfloatS #1\xint:}%
\def\XINT_fldiv_chkopt #1#2%
{%
    \ifx [#2\expandafter\XINT_fldiv_opt
       \else\expandafter\XINT_fldiv_noopt
    \fi  #1#2%
}%

\def\XINT_fldiv_noopt #1#2\xint:#3%
{%
    #1[\XINTdigits]%
    {\expandafter\XINT_FL_div_a
     \romannumeral0\XINTinfloatS[\XINTdigits]{#3}\XINTdigits.{#2}}%
}%
\def\XINT_fldiv_opt #1[\xint:#2]%#3#4%
{%
    \expandafter\XINT_fldiv_opt_a\the\numexpr #2.#1%
}%

\def\XINT_fldiv_opt_a #1.#2#3#4%
```

```
{%
    #2[#1]{\expandafter\XINT_FL_div_a\romannumeral0\XINTinfloatS[#1]{#4}#1.{#3}}%
}%
\def\XINT_FL_div_a #1[#2]#3.#4%
{%
    \expandafter\XINT_FL_div_b\romannumeral0\XINTinfloatS[#3]{#4}/#1e#2%
}%

\def\XINT_FL_div_b #1[#2]{#1e#2}%
```

8.70 `\xintFloatPow`, `\XINTinFloatPow`

1.07: initial version. 1.09j has re-organized the core loop.

2015/12/07. I have hesitated to map ^ in expressions to \xintFloatPow rather than \xintFloat-Power. But for 1.234567890123456 to the power 2145678912 with P=16, using Pow rather than Power seems to bring only about 5% gain.

This routine requires the exponent x to be compatible with \numexpr parsing.

1.2f has rewritten the code for better efficiency. Also, now the argument A for A^x is first rounded to P digits before switching to the increased working precision (which depends upon x).

```
\def\xintFloatPow   {\romannumeral0\xintfloatpow}%
\def\xintfloatpow #1{\XINT_flpow_chkopt \xintfloat #1\xint:}%
\def\XINTinFloatPow {\romannumeral0\XINTinfloatpow }%
\def\XINTinfloatpow #1{\XINT_flpow_chkopt \XINTinfloatS #1\xint:}%
\def\XINT_flpow_chkopt #1#2%
{%
    \ifx [#2\expandafter\XINT_flpow_opt
       \else\expandafter\XINT_flpow_noopt
    \fi
    #1#2%
}%
\def\XINT_flpow_noopt  #1#2\xint:#3%
{%
   \expandafter\XINT_flpow_checkB_a
   \the\numexpr #3.\XINTdigits.{#2}{#1[\XINTdigits]}%
}%
\def\XINT_flpow_opt #1[\xint:#2]%
{%
   \expandafter\XINT_flpow_opt_a\the\numexpr #2.#1%
}%
\def\XINT_flpow_opt_a #1.#2#3#4%
{%
   \expandafter\XINT_flpow_checkB_a\the\numexpr #4.#1.{#3}{#2[#1]}%
}%
\def\XINT_flpow_checkB_a #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_flpow_BisZero
      0#1{\XINT_flpow_checkB_b -}%
       0-{\XINT_flpow_checkB_b {}#1}%
    \krof
}%
\def\XINT_flpow_BisZero .#1.#2#3{#3{1[0]}}%
```

```
\def\XINT_flpow_checkB_b #1#2.#3.%
{%
    \expandafter\XINT_flpow_checkB_c
    \the\numexpr\xintLength{#2}+\xint_c_iii.#3.#2.{#1}%
}%
\def\XINT_flpow_checkB_c #1.#2.%
{%
    \expandafter\XINT_flpow_checkB_d\the\numexpr#1+#2.#1.#2.%
}%
```

1.2f rounds input to P digits, first.

```
\def\XINT_flpow_checkB_d #1.#2.#3.#4.#5#6%
{%
    \expandafter \XINT_flpow_aa
    \romannumeral0\XINTinfloat [#3]{#6}{#2}{#1}{#4}{#5}%
}%

\def\XINT_flpow_aa #1[#2]#3%
{%
    \expandafter\XINT_flpow_ab\the\numexpr #2-#3\expandafter.%
    \romannumeral\XINT_rep #3\endcsname0.#1.%
}%

\def\XINT_flpow_ab #1.#2.#3.{\XINT_flpow_a #3#2[#1]}%

\def\XINT_flpow_a #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_flpow_zero
      0#1{\XINT_flpow_b \iftrue}%
       0-{\XINT_flpow_b \iffalse#1}%
    \krof
}%
\def\XINT_flpow_zero #1[#2]#3#4#5#6%
{%
    #6{\if 1#51\xint_dothis {0[0]}\fi
       \xint_orthat
       {\XINT_signalcondition{DivisionByZero}{0 to the power #4}{}{0[0]}}%
      }%
}%

\def\XINT_flpow_b #1#2[#3]#4#5%
{%
    \XINT_flpow_loopI #5.#3.#2.#4.{#1\ifodd #5 \xint_c_i\fi\fi}%
}%

\def\XINT_flpow_truncate #1.#2.#3.%
{%
    \expandafter\XINT_flpow_truncate_a
    \romannumeral0\XINT_split_fromleft
    #3.#2\xint_bye2345678\xint_bye..#1.#3.%
}%
```

```
\def\XINT_flpow_truncate_a #1.#2.#3.{#3+\xintLength{#2}.#1.}%
\def\XINT_flpow_loopI #1.%
{%
    \ifnum #1=\xint_c_i\expandafter\XINT_flpow_ItoIII\fi
    \ifodd #1
       \expandafter\XINT_flpow_loopI_odd
    \else
       \expandafter\XINT_flpow_loopI_even
    \fi
    #1.%
}%

\def\XINT_flpow_ItoIII\ifodd #1\fi #2.#3.#4.#5.#6%
{%
    \expandafter\XINT_flpow_III\the\numexpr #6+\xint_c_.#3.#4.#5.%
}%

\def\XINT_flpow_loopI_even #1.#2.#3.%#4.%
{%
    \expandafter\XINT_flpow_loopI
    \the\numexpr #1/\xint_c_ii\expandafter.%
    \the\numexpr\expandafter\XINT_flpow_truncate
    \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.%
}%
\def\XINT_flpow_loopI_odd #1.#2.#3.#4.%
{%
    \expandafter\XINT_flpow_loopII
    \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter.%
    \the\numexpr\expandafter\XINT_flpow_truncate
    \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#2.#3.%
}%
\def\XINT_flpow_loopII #1.%
{%
    \ifnum #1 = \xint_c_i\expandafter\XINT_flpow_IItoIII\fi
    \ifodd #1
       \expandafter\XINT_flpow_loopII_odd
    \else
       \expandafter\XINT_flpow_loopII_even
    \fi
    #1.%
}%
\def\XINT_flpow_loopII_even #1.#2.#3.%#4.%
{%
    \expandafter\XINT_flpow_loopII
    \the\numexpr #1/\xint_c_ii\expandafter.%
    \the\numexpr\expandafter\XINT_flpow_truncate
    \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.%
}%
\def\XINT_flpow_loopII_odd #1.#2.#3.#4.#5.#6.%
{%
    \expandafter\XINT_flpow_loopII_odda
    \the\numexpr\expandafter\XINT_flpow_truncate
    \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
    #1.#2.#3.%
}%
\def\XINT_flpow_loopII_odda #1.#2.#3.#4.#5.#6.%
{%
    \expandafter\XINT_flpow_loopII
    \the\numexpr #4/\xint_c_ii-\xint_c_i\expandafter.%
    \the\numexpr\expandafter\XINT_flpow_truncate
    \the\numexpr\xint_c_ii*#5\expandafter.\romannumeral0\xintiisqr{#6}.#3.%
    #1.#2.%
}%

\def\XINT_flpow_IItoIII\ifodd #1\fi #2.#3.#4.#5.#6.#7.#8%
{%
    \expandafter\XINT_flpow_III\the\numexpr #8+\xint_c_\expandafter.%
    \the\numexpr\expandafter\XINT_flpow_truncate
    \the\numexpr#3+#6\expandafter.\romannumeral0\xintiimul{#4}{#7}.#5.%
}%
```

This ending is common with \xintFloatPower.

In the case of negative exponent we need to inverse the Q-digits mantissa. This requires no special attention now as 1.2k's \xintFloat does correct rounding of fractions hence it is easy to bound the total error. It can be checked that the algorithm after final rounding to the target precision computes a value Z whose distance to the exact theoretical will be less than 0.52 ulp(Z) (and worst cases can only be slightly worse than 0.51 ulp(Z)).

In the case of the half-integer exponent (only via the expression interface,) the computation (which proceeds via \XINTinFloatPowerH) ends with a square root. This square root extraction is done with 3 guard digits (the power operations were done with more.) Then the value is rounded to the target precision. There is thus this rounding to 3 guard digits (in the case of negative exponent the reciprocal is computed before the square-root), then the square root is (computed with exact rounding for these 3 guard digits), and then there is the final rounding of this to the target precision. The total error (for positive as well as negative exponent) has been estimated to at worst possibly exceed slightly 0.5125 ulp(Z), and at any rate it is less than 0.52 ulp(Z).

```
\def\XINT_flpow_III #1.#2.#3.#4.#5%
{%
    \expandafter\XINT_flpow_IIIend
    \xint_UDsignfork
         #5{{1/#3[-#2]}}%
          -{{#3[#2]}}%
    \krof #1%
}%

\def\XINT_flpow_IIIend #1#2#3%
    {#3{\if#21\xint_afterfi{\expandafter-\romannumeral`&&@}\fi#1}}%
```

8.71 \xintFloatPower, \XINTinFloatPower

1.07. The core loop has been re-organized in 1.09j for some slight efficiency gain. The exponent B is given to \xintNum. The ^ in expressions is mapped to this routine.

Same modifications as in \xintFloatPow for 1.2f.

1.2f adds a special private macro for allowing half-integral exponents for use with ^ within \xintfloatexpr. The exponent will be first truncated to either an integer or an half-integer. The macro is not for general use.

1.2k does anew this 1.2f handling of half-integer exponents for the \xintfloatexpr parser: with 1.2f's code the final square-root extraction was applied to a value already rounded to the target precision, unneedlessly losing precision.

```
\def\xintFloatPower   {\romannumeral0\xintfloatpower}%
\def\xintfloatpower #1{\XINT_flpower_chkopt \xintfloat #1\xint:}%
\def\XINTinFloatPower {\romannumeral0\XINTinfloatpower }%
\def\XINTinfloatpower #1{\XINT_flpower_chkopt \XINTinfloatS #1\xint:}%
```

First the special macro for use by the expression parser which checks if one raises to an half-integer exponent. This is always with \XINTdigits precision. Rewritten for 1.2k in order for the final square root to keep three guard digits.

We have to be careful that exponent #2 is not constrained by TeX bound. And we must allow fractions. The 1.2k variant does a rounding to nearest integer of half-integer, 1.2f did a truncation rather (this is done after truncation of #2 to fixed point with one digit after mark.) We try to recognize quickly the case of integer exponent, for speed, but there is overhead of going through \xintiTrunc1.

```
\def\XINTinFloatPowerH {\romannumeral0\XINTinfloatpowerh }%


\def\XINTinfloatpowerh #1#2%
{%
    \expandafter\XINT_flpowerh_a\romannumeral0\xintitrunc1{#2};%
    \XINTdigits.{#1}{\XINTinfloatS[\XINTdigits]}%
}%


\def\XINT_flpowerh_a #1;%
{%
    \if0\xintLDg{#1}\expandafter\XINT_flpowerh_int
        \else\expandafter\XINT_flpowerh_b
    \fi #1.%
}%
\def\XINT_flpowerh_int #1%
{%
    \if0#1\expandafter\XINT_flpower_BisZero
     \else\expandafter\XINT_flpowerh_i
    \fi #1%
}%
\def\XINT_flpowerh_i #10.{\expandafter\XINT_flpower_checkB_a#1.}%
\def\XINT_flpowerh_b #1.%
{%
   \expandafter\XINT_flpowerh_c\romannumeral0\xintdsrr{\xintDouble{#1}}.%
}%
\def\XINT_flpowerh_c #1.%
{%
    \ifodd\xintLDg{#1} %<- intentional space
        \expandafter\XINT_flpowerh_d\else\expandafter\XINT_flpowerh_e
    \fi #1.%
}%
```

```
\def\XINT_flpowerh_d #1.\XINTdigits.#2#3%
{%
   \XINT_flpower_checkB_a #1.\XINTdigits.{#2}\XINT_flpowerh_finish
}%
\def\XINT_flpowerh_finish #1%
   {\XINTinfloatS[\XINTdigits]{\XINTinFloatSqrt[\XINTdigits+\xint_c_iii]{#1}}}%
\def\XINT_flpowerh_e #1.%
   {\expandafter\XINT_flpower_checkB_a\romannumeral0\xinthalf{#1}.}%
```

Start of macro. Check for optional argument.

```
\def\XINT_flpower_chkopt #1#2%
{%
    \ifx [#2\expandafter\XINT_flpower_opt
       \else\expandafter\XINT_flpower_noopt
    \fi
    #1#2%
}%
\def\XINT_flpower_noopt  #1#2\xint:#3%
{%
   \expandafter\XINT_flpower_checkB_a
   \romannumeral0\xintnum{#3}.\XINTdigits.{#2}{#1[\XINTdigits]}%
}%
\def\XINT_flpower_opt #1[\xint:#2]%
{%
   \expandafter\XINT_flpower_opt_a\the\numexpr #2.#1%
}%
\def\XINT_flpower_opt_a #1.#2#3#4%
{%
   \expandafter\XINT_flpower_checkB_a
   \romannumeral0\xintnum{#4}.#1.{#3}{#2[#1]}%
}%
\def\XINT_flpower_checkB_a #1%
{%
    \xint_UDzerominusfork
      #1-{\XINT_flpower_BisZero 0}%
      0#1{\XINT_flpower_checkB_b -}%
       0-{\XINT_flpower_checkB_b {}#1}%
    \krof
}%
\def\XINT_flpower_BisZero 0.#1.#2#3{#3{1[0]}}%
\def\XINT_flpower_checkB_b #1#2.#3.%
{%
    \expandafter\XINT_flpower_checkB_c
    \the\numexpr\xintLength{#2}+\xint_c_iii.#3.#2.{#1}%
}%


\def\XINT_flpower_checkB_c #1.#2.%
{%
    \expandafter\XINT_flpower_checkB_d\the\numexpr#1+#2.#1.#2.%
}%
```

```
\def\XINT_flpower_checkB_d #1.#2.#3.#4.#5#6%
{%
    \expandafter \XINT_flpower_aa
    \romannumeral0\XINTinfloat [#3]{#6}{#2}{#1}{#4}{#5}%
}%

\def\XINT_flpower_aa #1[#2]#3%
{%
    \expandafter\XINT_flpower_ab\the\numexpr #2-#3\expandafter.%
    \romannumeral\XINT_rep #3\endcsname0.#1.%
}%
\def\XINT_flpower_ab #1.#2.#3.{\XINT_flpower_a #3#2[#1]}%
\def\XINT_flpower_a #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_flpow_zero
      0#1{\XINT_flpower_b \iftrue}%
       0-{\XINT_flpower_b \iffalse#1}%
    \krof
}%
\def\XINT_flpower_b #1#2[#3]#4#5%
{%
    \XINT_flpower_loopI #5.#3.#2.#4.{#1\xintiiOdd{#5}\fi}%
}%
\def\XINT_flpower_loopI #1.%
{%
    \if1\XINT_isOne {#1}\xint_dothis\XINT_flpower_ItoIII\fi
    \ifodd\xintLDg{#1} %<- intentional space
       \xint_dothis{\expandafter\XINT_flpower_loopI_odd}\fi
    \xint_orthat{\expandafter\XINT_flpower_loopI_even}%

    \romannumeral0\XINT_half
    #1\xint_bye\xint_Bye345678\xint_bye
    *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax.%
}%
\def\XINT_flpower_ItoIII #1.#2.#3.#4.#5%
{%
    \expandafter\XINT_flpow_III\the\numexpr #5+\xint_c_.#2.#3.#4.%
}%
\def\XINT_flpower_loopI_even #1.#2.#3.#4.%
{%
    \expandafter\XINT_flpower_toloopI
    \the\numexpr\expandafter\XINT_flpow_truncate
    \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#1.%
}%
\def\XINT_flpower_toloopI #1.#2.#3.#4.{\XINT_flpower_loopI #4.#1.#2.#3.}%
\def\XINT_flpower_loopI_odd #1.#2.#3.#4.%
{%
    \expandafter\XINT_flpower_toloopII
    \the\numexpr\expandafter\XINT_flpow_truncate
    \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.%
    #1.#2.#3.%
```

```
}%
\def\XINT_flpower_toloopII #1.#2.#3.#4.{\XINT_flpower_loopII #4.#1.#2.#3.}%
\def\XINT_flpower_loopII #1.%
{%
    \if1\XINT_isOne{#1}\xint_dothis\XINT_flpower_IItoIII\fi
    \ifodd\xintLDg{#1} %<- intentional space
        \xint_dothis{\expandafter\XINT_flpower_loopII_odd}\fi
    \xint_orthat{\expandafter\XINT_flpower_loopII_even}%

    \romannumeral0\XINT_half#1\xint_bye\xint_Bye345678\xint_bye
    *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax.%
}%
\def\XINT_flpower_loopII_even #1.#2.#3.#4.%
{%
    \expandafter\XINT_flpower_toloopII
    \the\numexpr\expandafter\XINT_flpow_truncate
    \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#1.%
}%
\def\XINT_flpower_loopII_odd #1.#2.#3.#4.#5.#6.%
{%
    \expandafter\XINT_flpower_loopII_odda
    \the\numexpr\expandafter\XINT_flpow_truncate
    \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
    #1.#2.#3.%
}%
\def\XINT_flpower_loopII_odda #1.#2.#3.#4.#5.#6.%
{%
    \expandafter\XINT_flpower_toloopII
    \the\numexpr\expandafter\XINT_flpow_truncate
    \the\numexpr\xint_c_ii*#5\expandafter.\romannumeral0\xintiisqr{#6}.#3.%
    #4.#1.#2.%
}%
\def\XINT_flpower_IItoIII #1.#2.#3.#4.#5.#6.#7%
{%
    \expandafter\XINT_flpow_III\the\numexpr #7+\xint_c_\expandafter.%
    \the\numexpr\expandafter\XINT_flpow_truncate
    \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
}%
```

8.72 \xintFloatFac, \XINTFloatFac

```
\def\xintFloatFac     {\romannumeral0\xintfloatfac}%
\def\xintfloatfac   #1{\XINT_flfac_chkopt \xintfloat #1\xint:}%
\def\XINTinFloatFac   {\romannumeral0\XINTinfloatfac }%
\def\XINTinfloatfac #1{\XINT_flfac_chkopt \XINTinfloat #1\xint:}%
\def\XINT_flfac_chkopt #1#2%
{%
    \ifx [#2\expandafter\XINT_flfac_opt
       \else\expandafter\XINT_flfac_noopt
    \fi
     #1#2%
}%
```

```
\def\XINT_flfac_noopt  #1#2\xint:
{%
   \expandafter\XINT_FL_fac_fork_a
   \the\numexpr \xintNum{#2}.\xint_c_i \XINTdigits\XINT_FL_fac_out{#1[\XINTdigits]}%
}%
\def\XINT_flfac_opt #1[\xint:#2]%
{%
   \expandafter\XINT_flfac_opt_a\the\numexpr #2.#1%
}%
\def\XINT_flfac_opt_a #1.#2#3%
{%
   \expandafter\XINT_FL_fac_fork_a\the\numexpr \xintNum{#3}.\xint_c_i {#1}\XINT_FL_fac_out{#2[#1]}%
}%
\def\XINT_FL_fac_fork_a #1%
{%
     \xint_UDzerominusfork
     #1-\XINT_FL_fac_iszero
     0#1\XINT_FL_fac_isneg
      0-{\XINT_FL_fac_fork_b #1}%
    \krof
}%
\def\XINT_FL_fac_iszero #1.#2#3#4#5{#5{1[0]}}%
```

1.2f XINT_FL_fac_isneg returns 0, earlier versions used 1 here.

```
\def\XINT_FL_fac_isneg  #1.#2#3#4#5%
{%
    #5{\XINT_signalcondition{InvalidOperation}
                     {Factorial of negative: (-#1)!}{}{0[0]}}%
}%
\def\XINT_FL_fac_fork_b #1.%
{%
    \ifnum #1>\xint_c_x^viii_mone\xint_dothis\XINT_FL_fac_toobig\fi
    \ifnum #1>\xint_c_x^iv\xint_dothis\XINT_FL_fac_vbig \fi
    \ifnum #1>465  \xint_dothis\XINT_FL_fac_big\fi
    \ifnum #1>101  \xint_dothis\XINT_FL_fac_med\fi
                   \xint_orthat\XINT_FL_fac_small
    #1.%
}%
\def\XINT_FL_fac_toobig #1.#2#3#4#5%
{%
    #5{\XINT_signalcondition{InvalidOperation}
                     {Factorial of too big: (#1)!}{}{0[0]}}%
}%
```

Computations are done with Q blocks of eight digits. When a multiplication has a carry, hence creates Q+1 blocks, the least significant one is dropped. The goal is to compute an approximate value X' to the exact value X, such that the final relative error (X-X')/X will be at most 10^{-P-1} with P the desired precision. Then, when we round X' to X'' with P significant digits, we can prove that the absolute error |X-X''| is bounded (strictly) by 0.6 ulp(X''). (ulp= unit in the last (significant) place). Let N be the number of such operations, the formula for Q deduces from the previous explanations is that 8Q should be at least P+9+k, with k the number of digits of N (in base 10). Note that 1.2 version used P+10+k, for 1.2f I reduced to P+9+k. Also, k should be the number of digits of the number N of multiplications done, hence for n<=10000 we can take N=n/2, or

N/3, or N/4. This is rounded above by numexpr and always an overestimate of the actual number of approximate multiplications done (the first ones are exact). (vérifier ce que je raconte, j'ai la flemme là).

We then want ceil((P+k+n)/8). Using \numexpr rounding division (ARRRRRGGGHHHH), if m is a positive integer, ceil(m/8) can be computed as (m+3)/8. Thus with m=P+10+k, this gives Q<-(P+13+k)/8. The routine actually computes 8(Q-1) for use in \XINT_FL_fac_addzeros.

With 1.2f the formula is m=P+9+k, Q<-(P+12+k)/8, and we use now 4=12-8 rather than the earlier 5=13-8. Whatever happens, the value computed in \XINT_FL_fac_increaseP is at least 8. There will always be an extra block.

Note: with Digits:=32; Maple gives for 200!:

> factorial(200.);

375

0.78865786736479050355236321393218 10

My 1.2f routine (and also 1.2) outputs:

7.8865786736479050355236321393219e374

and this is the correct rounding because for 40 digits it computes

7.886578673647905035523632139321850622951e374

Maple's result (contrarily to xint) is thus not the correct rounding but still it is less than 0.6 ulp wrong.

```
\def\XINT_FL_fac_vbig
   {\expandafter\XINT_FL_fac_vbigloop_a
    \the\numexpr \XINT_FL_fac_increaseP \xint_c_i   }%
\def\XINT_FL_fac_big
   {\expandafter\XINT_FL_fac_bigloop_a
    \the\numexpr \XINT_FL_fac_increaseP \xint_c_ii  }%
\def\XINT_FL_fac_med
   {\expandafter\XINT_FL_fac_medloop_a
    \the\numexpr \XINT_FL_fac_increaseP \xint_c_iii }%
\def\XINT_FL_fac_small
   {\expandafter\XINT_FL_fac_smallloop_a
    \the\numexpr \XINT_FL_fac_increaseP \xint_c_iv  }%
\def\XINT_FL_fac_increaseP #1#2.#3#4%
{%
    #2\expandafter.\the\numexpr\xint_c_viii*%
    ((\xint_c_iv+#4+\expandafter\XINT_FL_fac_countdigits
                    \the\numexpr #2/(#1*#3)\relax 87654321\Z)/\xint_c_viii).%
}%
\def\XINT_FL_fac_countdigits #1#2#3#4#5#6#7#8{\XINT_FL_fac_countdone }%
\def\XINT_FL_fac_countdone   #1#2\Z {#1}%
\def\XINT_FL_fac_out #1;![#2]#3%
    {#3{\romannumeral0\XINT_mul_out
         #1;!1\R!1\R!1\R!1\R!%
                    1\R!1\R!1\R!1\R!\W [#2]}}%
\def\XINT_FL_fac_vbigloop_a #1.#2.%
{%
    \XINT_FL_fac_bigloop_a \xint_c_x^iv.#2.%
    {\expandafter\XINT_FL_fac_vbigloop_loop\the\numexpr 100010001\expandafter.%
     \the\numexpr \xint_c_x^viii+#1.}%
}%
\def\XINT_FL_fac_vbigloop_loop #1.#2.%
{%
    \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
    \expandafter\XINT_FL_fac_vbigloop_loop
    \the\numexpr #1+\xint_c_i\expandafter.%
    \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_mul #1!%
}%
\def\XINT_FL_fac_bigloop_a #1.%
{%
    \expandafter\XINT_FL_fac_bigloop_b \the\numexpr
    #1+\xint_c_i-\xint_c_ii*((#1-464)/\xint_c_ii).#1.%
}%
\def\XINT_FL_fac_bigloop_b #1.#2.#3.%
{%
    \expandafter\XINT_FL_fac_medloop_a
        \the\numexpr #1-\xint_c_i.#3.{\XINT_FL_fac_bigloop_loop #1.#2.}%
}%
\def\XINT_FL_fac_bigloop_loop #1.#2.%
{%
    \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
    \expandafter\XINT_FL_fac_bigloop_loop
    \the\numexpr #1+\xint_c_ii\expandafter.%
    \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_bigloop_mul #1!%
}%
\def\XINT_FL_fac_bigloop_mul #1!%
{%
    \expandafter\XINT_FL_fac_mul
        \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
}%
\def\XINT_FL_fac_medloop_a #1.%
{%
    \expandafter\XINT_FL_fac_medloop_b
        \the\numexpr #1+\xint_c_i-\xint_c_iii*((#1-100)/\xint_c_iii).#1.%
}%
\def\XINT_FL_fac_medloop_b #1.#2.#3.%
{%
    \expandafter\XINT_FL_fac_smallloop_a
        \the\numexpr #1-\xint_c_i.#3.{\XINT_FL_fac_medloop_loop #1.#2.}%
}%
\def\XINT_FL_fac_medloop_loop #1.#2.%
{%
    \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
    \expandafter\XINT_FL_fac_medloop_loop
    \the\numexpr #1+\xint_c_iii\expandafter.%
    \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_medloop_mul #1!%
}%
\def\XINT_FL_fac_medloop_mul #1!%
{%
    \expandafter\XINT_FL_fac_mul
    \the\numexpr
        \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
}%
\def\XINT_FL_fac_smallloop_a #1.%
{%
    \csname
```

```
      XINT_FL_fac_smallloop_\the\numexpr #1-\xint_c_iv*(#1/\xint_c_iv)\relax
    \endcsname #1.%
}%
\expandafter\def\csname XINT_FL_fac_smallloop_1\endcsname #1.#2.%
{%
    \XINT_FL_fac_addzeros #2.100000001!.{2.#1.}{#2}%
}%
\expandafter\def\csname XINT_FL_fac_smallloop_-2\endcsname #1.#2.%
{%
    \XINT_FL_fac_addzeros #2.100000002!.{3.#1.}{#2}%
}%
\expandafter\def\csname XINT_FL_fac_smallloop_-1\endcsname #1.#2.%
{%
    \XINT_FL_fac_addzeros #2.100000006!.{4.#1.}{#2}%
}%
\expandafter\def\csname XINT_FL_fac_smallloop_0\endcsname #1.#2.%
{%
    \XINT_FL_fac_addzeros #2.100000024!.{5.#1.}{#2}%
}%
\def\XINT_FL_fac_addzeros #1.%
{%
    \ifnum #1=\xint_c_viii \expandafter\XINT_FL_fac_addzeros_exit\fi
    \expandafter\XINT_FL_fac_addzeros
    \the\numexpr #1-\xint_c_viii.100000000!%
}%
```

We will manipulate by successive *small* multiplications Q blocks 1<8d>!, terminated by 1;!. We need a custom small multiplication which tells us when it has create a new block, and the least significant one should be dropped.

```
\def\XINT_FL_fac_addzeros_exit #1.#2.#3#4{\XINT_FL_fac_smallloop_loop #3#21;![-#4]}%
\def\XINT_FL_fac_smallloop_loop #1.#2.%
{%
    \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
    \expandafter\XINT_FL_fac_smallloop_loop
    \the\numexpr #1+\xint_c_iv\expandafter.%
    \the\numexpr #2\expandafter.\romannumeral0\XINT_FL_fac_smallloop_mul #1!%
}%
\def\XINT_FL_fac_smallloop_mul #1!%
{%
    \expandafter\XINT_FL_fac_mul
    \the\numexpr
        \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
}%[[
\def\XINT_FL_fac_loop_exit #1!#2]#3{#3#2]}%
\def\XINT_FL_fac_mul 1#1!%
   {\expandafter\XINT_FL_fac_mul_a\the\numexpr\XINT_FL_fac_smallmul 10!{#1}}%
\def\XINT_FL_fac_mul_a #1-#2%
{%
    \if#21\xint_afterfi{\expandafter\space\xint_gob_til_exclam}\else
    \expandafter\space\fi #11;!%
}%
\def\XINT_FL_fac_minimulwc_a #1#2#3#4#5!#6#7#8#9%
{%
```

```
    \XINT_FL_fac_minimulwc_b {#1#2#3#4}{#5}{#6#7#8#9}%
}%
\def\XINT_FL_fac_minimulwc_b #1#2#3#4!#5%
{%
    \expandafter\XINT_FL_fac_minimulwc_c
    \the\numexpr \xint_c_x^ix+#5+#2*#4!{{#1}{#2}{#3}{#4}}%
}%
\def\XINT_FL_fac_minimulwc_c 1#1#2#3#4#5#6!#7%
{%
    \expandafter\XINT_FL_fac_minimulwc_d {#1#2#3#4#5}#7{#6}%
}%
\def\XINT_FL_fac_minimulwc_d #1#2#3#4#5%
{%
    \expandafter\XINT_FL_fac_minimulwc_e
    \the\numexpr \xint_c_x^ix+#1+#2*#5+#3*#4!{#2}{#4}%
}%
\def\XINT_FL_fac_minimulwc_e 1#1#2#3#4#5#6!#7#8#9%
{%
    1#6#9\expandafter!%
    \the\numexpr\expandafter\XINT_FL_fac_smallmul
    \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#7*#8!%
}%
\def\XINT_FL_fac_smallmul 1#1!#21#3!%
{%
    \xint_gob_til_sc #3\XINT_FL_fac_smallmul_end;%
    \XINT_FL_fac_minimulwc_a #2!#3!{#1}{#2}%
}%
```

This is the crucial ending. I note that I used here an \ifnum test rather than the gob_til_eightzeroes thing. Actually for eight digits there is much less difference than for only four.

The "carry" situation is marked by a final !-1 rather than !-2 for no-carry. (a \numexpr muste be stopped, and leaving a - as delimiter is good as it will not arise earlier.)

```
\def\XINT_FL_fac_smallmul_end;\XINT_FL_fac_minimulwc_a #1!;!#2#3[#4]%
{%
   \ifnum #2=\xint_c_
       \expandafter\xint_firstoftwo\else
       \expandafter\xint_secondoftwo
   \fi
   {-2\relax[#4]}%
   {1#2\expandafter!\expandafter-\expandafter1\expandafter
                  [\the\numexpr #4+\xint_c_viii]}%
}%
```

8.73 \xintFloatPFactorial, \XINTinFloatPFactorial

2015/11/29 for 1.2f. Partial factorial pfactorial(a,b)=(a+1)...b, only for non-negative integers with a<=b<10^8.

1.2h (2016/11/20) now avoids raising \xintError:OutOfRangePFac if the condition 0<=a<=b<10^8 is violated. Same as for \xintiiPFactorial.

```
\def\xintFloatPFactorial {\romannumeral0\xintfloatpfactorial}%
\def\xintfloatpfactorial #1{\XINT_flpfac_chkopt \xintfloat #1\xint:}%
\def\XINTinFloatPFactorial {\romannumeral0\XINTinfloatpfactorial }%
```

```
\def\XINTinfloatpfactorial #1{\XINT_flpfac_chkopt \XINTinfloat #1\xint:}%
\def\XINT_flpfac_chkopt #1#2%
{%
    \ifx [#2\expandafter\XINT_flpfac_opt
       \else\expandafter\XINT_flpfac_noopt
    \fi
     #1#2%
}%
\def\XINT_flpfac_noopt  #1#2\xint:#3%
{%
   \expandafter\XINT_FL_pfac_fork
   \the\numexpr \xintNum{#2}\expandafter.%
   \the\numexpr \xintNum{#3}.\xint_c_i{\XINTdigits}{#1[\XINTdigits]}%
}%
\def\XINT_flpfac_opt #1[\xint:#2]%
{%
   \expandafter\XINT_flpfac_opt_b\the\numexpr #2.#1%
}%
\def\XINT_flpfac_opt_b #1.#2#3#4%
{%
   \expandafter\XINT_FL_pfac_fork
   \the\numexpr \xintNum{#3}\expandafter.%
   \the\numexpr \xintNum{#4}.\xint_c_i{#1}{#2[#1]}%
}%
\def\XINT_FL_pfac_fork #1#2.#3#4.%
{%
    \unless\ifnum #1#2<#3#4 \xint_dothis\XINT_FL_pfac_one\fi
    \if-#3\xint_dothis\XINT_FL_pfac_neg \fi
    \if-#1\xint_dothis\XINT_FL_pfac_zero\fi
    \ifnum #3#4>\xint_c_x^viii_mone\xint_dothis\XINT_FL_pfac_outofrange\fi
    \xint_orthat \XINT_FL_pfac_increaseP #1#2.#3#4.%
}%
\def\XINT_FL_pfac_outofrange #1.#2.#3#4#5%
{%
    #5{\XINT_signalcondition{InvalidOperation}
                     {pfactorial second arg too big: 99999999 < #2}{}{0[0]}}%
}%
\def\XINT_FL_pfac_one  #1.#2.#3#4#5{#5{1[0]}}%
\def\XINT_FL_pfac_zero #1.#2.#3#4#5{#5{0[0]}}%
\def\XINT_FL_pfac_neg -#1.-#2.%
{%
    \ifnum #1>\xint_c_x^viii\xint_dothis\XINT_FL_pfac_outofrange\fi
    \xint_orthat {%
    \ifodd\numexpr#2-#1\relax\xint_afterfi{\expandafter-\romannumeral`&&@}\fi
    \expandafter\XINT_FL_pfac_increaseP}%
    \the\numexpr #2-\xint_c_i\expandafter.\the\numexpr#1-\xint_c_i.%
}%
```

See the comments for \XINT_FL_pfac_increaseP. Case of b=a+1 should be filtered out perhaps. We only needed here to copy the \xintPFactorial macros and re-use \XINT_FL_fac_mul/\XINT_FL_fac_out. Had to modify a bit \XINT_FL_pfac_addzeroes. We can enter here directly with #3 equal to specify the precision (the calculated value before final rounding has a relative error less than #3.10^{-#4-1}), and #5 would hold the macro doing the final rounding (or truncating, if I make a FloatTrunc available) to a given number of digits, possibly not #4. By default the #3 is 1, but FloatBinomial calls it with #3=4.

```
\def\XINT_FL_pfac_increaseP #1.#2.#3#4%
{%
    \expandafter\XINT_FL_pfac_a
    \the\numexpr \xint_c_viii*((\xint_c_iv+#4+\expandafter
                 \XINT_FL_fac_countdigits\the\numexpr (#2-#1-\xint_c_i)%
                    /\ifnum #2>\xint_c_x^iv #3\else(#3*\xint_c_ii)\fi\relax
                 87654321\Z)/\xint_c_viii).#1.#2.%
}%
\def\XINT_FL_pfac_a #1.#2.#3.%
{%
    \expandafter\XINT_FL_pfac_b\the\numexpr \xint_c_i+#2\expandafter.%
    \the\numexpr#3\expandafter.%
    \romannumeral0\XINT_FL_pfac_addzeroes #1.100000001!1;![-#1]%
}%
\def\XINT_FL_pfac_addzeroes #1.%
{%
    \ifnum #1=\xint_c_viii \expandafter\XINT_FL_pfac_addzeroes_exit\fi
    \expandafter\XINT_FL_pfac_addzeroes\the\numexpr #1-\xint_c_viii.100000000!%
}%
\def\XINT_FL_pfac_addzeroes_exit #1.{ }%
\def\XINT_FL_pfac_b #1.%
{%
    \ifnum #1>9999 \xint_dothis\XINT_FL_pfac_vbigloop \fi
    \ifnum #1>463  \xint_dothis\XINT_FL_pfac_bigloop   \fi
    \ifnum #1>98   \xint_dothis\XINT_FL_pfac_medloop   \fi
                   \xint_orthat\XINT_FL_pfac_smallloop #1.%
}%
\def\XINT_FL_pfac_smallloop #1.#2.%
{%
    \ifcase\numexpr #2-#1\relax
        \expandafter\XINT_FL_pfac_end_
    \or \expandafter\XINT_FL_pfac_end_i
    \or \expandafter\XINT_FL_pfac_end_ii
    \or \expandafter\XINT_FL_pfac_end_iii
    \else\expandafter\XINT_FL_pfac_smallloop_a
    \fi #1.#2.%
}%
\def\XINT_FL_pfac_smallloop_a #1.#2.%
{%
    \expandafter\XINT_FL_pfac_smallloop_b
    \the\numexpr #1+\xint_c_iv\expandafter.%
    \the\numexpr #2\expandafter.%
    \romannumeral0\expandafter\XINT_FL_fac_mul
    \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
}%
\def\XINT_FL_pfac_smallloop_b #1.%
{%
    \ifnum #1>98  \expandafter\XINT_FL_pfac_medloop   \else
                  \expandafter\XINT_FL_pfac_smallloop \fi #1.%
}%
```

```
\def\XINT_FL_pfac_medloop #1.#2.%
{%
    \ifcase\numexpr #2-#1\relax
        \expandafter\XINT_FL_pfac_end_
    \or \expandafter\XINT_FL_pfac_end_i
    \or \expandafter\XINT_FL_pfac_end_ii
    \else\expandafter\XINT_FL_pfac_medloop_a
    \fi #1.#2.%
}%
\def\XINT_FL_pfac_medloop_a #1.#2.%
{%
    \expandafter\XINT_FL_pfac_medloop_b
    \the\numexpr #1+\xint_c_iii\expandafter.%
    \the\numexpr #2\expandafter.%
    \romannumeral0\expandafter\XINT_FL_fac_mul
    \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
}%
\def\XINT_FL_pfac_medloop_b #1.%
{%
    \ifnum #1>463 \expandafter\XINT_FL_pfac_bigloop   \else
                  \expandafter\XINT_FL_pfac_medloop   \fi #1.%
}%
\def\XINT_FL_pfac_bigloop #1.#2.%
{%
    \ifcase\numexpr #2-#1\relax
        \expandafter\XINT_FL_pfac_end_
    \or \expandafter\XINT_FL_pfac_end_i
    \else\expandafter\XINT_FL_pfac_bigloop_a
    \fi #1.#2.%
}%
\def\XINT_FL_pfac_bigloop_a #1.#2.%
{%
    \expandafter\XINT_FL_pfac_bigloop_b
    \the\numexpr #1+\xint_c_ii\expandafter.%
    \the\numexpr #2\expandafter.%
    \romannumeral0\expandafter\XINT_FL_fac_mul
    \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
}%
\def\XINT_FL_pfac_bigloop_b #1.%
{%
    \ifnum #1>9999 \expandafter\XINT_FL_pfac_vbigloop  \else
                   \expandafter\XINT_FL_pfac_bigloop   \fi #1.%
}%
\def\XINT_FL_pfac_vbigloop #1.#2.%
{%
    \ifnum #2=#1
         \expandafter\XINT_FL_pfac_end_
    \else\expandafter\XINT_FL_pfac_vbigloop_a
    \fi #1.#2.%
}%
\def\XINT_FL_pfac_vbigloop_a #1.#2.%
{%
```

```
    \expandafter\XINT_FL_pfac_vbigloop
    \the\numexpr #1+\xint_c_i\expandafter.%
    \the\numexpr #2\expandafter.%
    \romannumeral0\expandafter\XINT_FL_fac_mul
    \the\numexpr\xint_c_x^viii+#1!%
}%
\def\XINT_FL_pfac_end_iii #1.#2.%
{%
    \expandafter\XINT_FL_fac_out
    \romannumeral0\expandafter\XINT_FL_fac_mul
    \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
}%
\def\XINT_FL_pfac_end_ii #1.#2.%
{%
    \expandafter\XINT_FL_fac_out
    \romannumeral0\expandafter\XINT_FL_fac_mul
    \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
}%
\def\XINT_FL_pfac_end_i #1.#2.%
{%
    \expandafter\XINT_FL_fac_out
    \romannumeral0\expandafter\XINT_FL_fac_mul
    \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
}%
\def\XINT_FL_pfac_end_ #1.#2.%
{%
    \expandafter\XINT_FL_fac_out
    \romannumeral0\expandafter\XINT_FL_fac_mul
    \the\numexpr \xint_c_x^viii+#1!%
}%
```

8.74 \xintFloatBinomial, \XINTinFloatBinomial

1.2f. We compute binomial(x,y) as pfac(x-y,x)/y!, where the numerator and denominator are computed with a relative error at most 4.10^{-P-2}, then rounded (once I have a float truncation, I will use truncation rather) to P+3 digits, and finally the quotient is correctly rounded to P digits. This will guarantee that the exact value X differs from the computed one Y by at most 0.6 ulp(Y). (2015/12/01).

2016/11/19 for 1.2h. As for \xintiiBinomial, hard to understand why last year I coded this to raise an error if y<0 or y>x ! The question of the Gamma function is for another occasion, here x and y must be (small) integers.

```
\def\xintFloatBinomial    {\romannumeral0\xintfloatbinomial}%
\def\xintfloatbinomial   #1{\XINT_flbinom_chkopt \xintfloat #1\xint:}%
\def\XINTinFloatBinomial   {\romannumeral0\XINTinfloatbinomial }%
\def\XINTinfloatbinomial #1{\XINT_flbinom_chkopt \XINTinfloat #1\xint:}%
\def\XINT_flbinom_chkopt #1#2%
{%
    \ifx [#2\expandafter\XINT_flbinom_opt
       \else\expandafter\XINT_flbinom_noopt
    \fi  #1#2%
}%
\def\XINT_flbinom_noopt #1#2\xint:#3%
```

```
{%
    \expandafter\XINT_FL_binom_a
    \the\numexpr\xintNum{#2}\expandafter.\the\numexpr\xintNum{#3}.\XINTdigits.#1%
}%
\def\XINT_flbinom_opt #1[\xint:#2]#3#4%
{%
    \expandafter\XINT_FL_binom_a
    \the\numexpr\xintNum{#3}\expandafter.\the\numexpr\xintNum{#4}\expandafter.%
    \the\numexpr #2.#1%
}%
\def\XINT_FL_binom_a #1.#2.%
{%
    \expandafter\XINT_FL_binom_fork \the\numexpr #1-#2.#2.#1.%
}%
\def\XINT_FL_binom_fork #1#2.#3#4.#5#6.%
{%
    \if-#5\xint_dothis \XINT_FL_binom_neg\fi
    \if-#1\xint_dothis \XINT_FL_binom_zero\fi
    \if-#3\xint_dothis \XINT_FL_binom_zero\fi
    \if0#1\xint_dothis \XINT_FL_binom_one\fi
    \if0#3\xint_dothis \XINT_FL_binom_one\fi
    \ifnum #5#6>\xint_c_x^viii_mone \xint_dothis\XINT_FL_binom_toobig\fi
    \ifnum #1#2>#3#4  \xint_dothis\XINT_FL_binom_ab \fi
                      \xint_orthat\XINT_FL_binom_aa
    #1#2.#3#4.#5#6.%
}%
\def\XINT_FL_binom_neg #1.#2.#3.#4.#5%
{%
    #5[#4]{\XINT_signalcondition{InvalidOperation}
                         {binomial with first arg negative: #3}{}{0[0]}}%
}%
\def\XINT_FL_binom_toobig #1.#2.#3.#4.#5%
{%
    #5[#4]{\XINT_signalcondition{InvalidOperation}
                         {binomial with first arg too big: 99999999 < #3}{}{0[0]}}%
}%
\def\XINT_FL_binom_one  #1.#2.#3.#4.#5{#5[#4]{1[0]}}%
\def\XINT_FL_binom_zero #1.#2.#3.#4.#5{#5[#4]{0[0]}}%

\def\XINT_FL_binom_aa  #1.#2.#3.#4.#5%
{%
    #5[#4]{\xintDiv{\XINT_FL_pfac_increaseP
           #2.#3.\xint_c_iv{#4+\xint_c_i}{\XINTinfloat[#4+\xint_c_iii]}}%
           {\XINT_FL_fac_fork_b
           #1.\xint_c_iv{#4+\xint_c_i}\XINT_FL_fac_out{\XINTinfloat[#4+\xint_c_iii]}}}%
}%
\def\XINT_FL_binom_ab  #1.#2.#3.#4.#5%
{%
    #5[#4]{\xintDiv{\XINT_FL_pfac_increaseP
           #1.#3.\xint_c_iv{#4+\xint_c_i}{\XINTinfloat[#4+\xint_c_iii]}}%
           {\XINT_FL_fac_fork_b
           #2.\xint_c_iv{#4+\xint_c_i}\XINT_FL_fac_out{\XINTinfloat[#4+\xint_c_iii]}}}%
}%
```

8.75 \xintFloatSqrt, \XINTinFloatSqrt

First done for 1.08.

The float version was developed at the same time as the integer one and even a bit earlier. As a result the integer variant had some sub-optimal parts. Anyway, for 1.2f I have rewritten the integer variant, and the float variant delegates all preparatory wrok for it until the last step. In particular the very low precisions are not penalized anymore from doing computations for at least 17 or 18 digits. Both the large and small precisions give quite shorter computation times.

Also, after examining more closely the achieved precision I decided to extend the float version in order for it to obtain the correct rounding (for inputs already of at most P digits with P the precision) of the theoretical exact value.

Beyond about 500 digits of precision the efficiency decreases swiftly, as is the case generally speaking with xintcore/xint/xintfrac arithmetic macros.

Final note: with 1.2f the input is always first rounded to P significant places.

```
\def\xintFloatSqrt     {\romannumeral0\xintfloatsqrt }%
\def\xintfloatsqrt   #1{\XINT_flsqrt_chkopt \xintfloat #1\xint:}%
\def\XINTinFloatSqrt   {\romannumeral0\XINTinfloatsqrt }%
\def\XINTinfloatsqrt #1{\XINT_flsqrt_chkopt \XINTinfloat #1\xint:}%
\def\XINT_flsqrt_chkopt #1#2%
{%
    \ifx [#2\expandafter\XINT_flsqrt_opt
       \else\expandafter\XINT_flsqrt_noopt
    \fi  #1#2%
}%
\def\XINT_flsqrt_noopt #1#2\xint:%
{%
    \expandafter\XINT_FL_sqrt_a
                \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.#1%
}%
\def\XINT_flsqrt_opt #1[\xint:#2]%#3%
{%
    \expandafter\XINT_flsqrt_opt_a\the\numexpr #2.#1%
}%
\def\XINT_flsqrt_opt_a #1.#2#3%
{%
    \expandafter\XINT_FL_sqrt_a\romannumeral0\XINTinfloat[#1]{#3}#1.#2%
}%
\def\XINT_FL_sqrt_a #1%
{%
    \xint_UDzerominusfork
     #1-\XINT_FL_sqrt_iszero
     0#1\XINT_FL_sqrt_isneg
      0-{\XINT_FL_sqrt_pos #1}%
    \krof
}%[
\def\XINT_FL_sqrt_iszero #1]#2.#3{#3[#2]{0[0]}}%
\def\XINT_FL_sqrt_isneg #1]#2.#3%
{%
  #3[#2]{\XINT_signalcondition{InvalidOperation}
                          {Square root of negative: -#1]}{}{0[0]}}%
}%
```

```
\def\XINT_FL_sqrt_pos #1[#2]#3.%
{%
    \expandafter\XINT_flsqrt
    \the\numexpr #3\ifodd #2 \xint_dothis {+\xint_c_iii.(#2+\xint_c_i).0}\fi
    \xint_orthat {+\xint_c_ii.#2.{}}#100.#3.%
}%

\def\XINT_flsqrt #1.#2.%
{%
    \expandafter\XINT_flsqrt_a
    \the\numexpr #2/\xint_c_ii-(#1-\xint_c_i)/\xint_c_ii.#1.%
}%

\def\XINT_flsqrt_a #1.#2.#3#4.#5.%
{%
    \expandafter\XINT_flsqrt_b
    \the\numexpr (#2-\xint_c_i)/\xint_c_ii\expandafter.%
    \romannumeral0\XINT_sqrt_start #2.#4#3.#5.#2.#4#3.#5.#1.%
}%

\def\XINT_flsqrt_b #1.#2#3%
{%
   \expandafter\XINT_flsqrt_c
   \romannumeral0\xintiisub
    {\XINT_dsx_addzeros {#1}#2;}%
    {\xintiiDivRound{\XINT_dsx_addzeros {#1}#3;}%
                    {\XINT_dbl#2\xint_bye2345678\xint_bye*\xint_c_ii\relax}}.%
}%

\def\XINT_flsqrt_c #1.#2.%
{%
    \expandafter\XINT_flsqrt_d
    \romannumeral0\XINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..%
}%

\def\XINT_flsqrt_d #1.#2#3.%
{%
    \ifnum #2=\xint_c_v
    \expandafter\XINT_flsqrt_f\else\expandafter\XINT_flsqrt_finish\fi
    #2#3.#1.%
}%

\def\XINT_flsqrt_finish #1#2.#3.#4.#5.#6.#7.#8{#8[#6]{#3#1[#7]}}%

\def\XINT_flsqrt_f 5#1.%
   {\expandafter\XINT_flsqrt_g\romannumeral0\xintinum{#1}\relax.}%
\def\XINT_flsqrt_g #1#2#3.{\if\relax#2\xint_dothis{\XINT_flsqrt_h #1}\fi
                           \xint_orthat{\XINT_flsqrt_finish 5.}}%
\def\XINT_flsqrt_h #1{\ifnum #1<\xint_c_iii\xint_dothis{\XINT_flsqrt_again}\fi
                      \xint_orthat{\XINT_flsqrt_finish 5.}}%
```

```
\def\XINT_flsqrt_again #1.#2.%
{%
    \expandafter\XINT_flsqrt_again_a\the\numexpr #2+\xint_c_viii.%
}%

\def\XINT_flsqrt_again_a #1.#2.#3.%
{%
    \expandafter\XINT_flsqrt_b
    \the\numexpr (#1-\xint_c_i)/\xint_c_ii\expandafter.%
    \romannumeral0\XINT_sqrt_start #1.#200000000.#3.%
                                  #1.#200000000.#3.%
}%
```

8.76 \xintFloatE, \XINTinFloatE

1.07: The fraction is the first argument contrarily to \xintTrunc and \xintRound.

1.2k had to rewrite this since there is no more a \XINT_float_a macro. Attention about \XINTinFloatE: it is for use by xintexpr.sty, contrarily to other \XINTinFloat<foo> macros it inserts itself the [\XINTdigits] thing, and with value 0 it produces on output 0[N], not 0[0].

```
\def\xintFloatE   {\romannumeral0\xintfloate }%
\def\xintfloate #1{\XINT_floate_chkopt #1\xint:}%
\def\XINT_floate_chkopt #1%
{%
    \ifx [#1\expandafter\XINT_floate_opt
       \else\expandafter\XINT_floate_noopt
    \fi  #1%
}%
\def\XINT_floate_noopt #1\xint:%
{%
    \expandafter\XINT_floate_post
    \romannumeral0\XINTinfloat[\XINTdigits]{#1}\XINTdigits.%
}%
\def\XINT_floate_opt [\xint:#1]%
{%
    \expandafter\XINT_floate_opt_a\the\numexpr #1.%
}%
\def\XINT_floate_opt_a #1.#2%
{%
    \expandafter\XINT_floate_post
    \romannumeral0\XINTinfloat[#1]{#2}#1.%
}%
\def\XINT_floate_post #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_floate_zero
       0#1\XINT_floate_neg
       0-\XINT_floate_pos
    \krof #1%
}%[
\def\XINT_floate_zero #1]#2.#3{ 0.e0}%
\def\XINT_floate_neg-{\expandafter-\romannumeral0\XINT_floate_pos}%
```

```
\def\XINT_floate_pos #1#2[#3]#4.#5%
{%
    \expandafter\XINT_float_pos_done\the\numexpr#3+#4+#5-\xint_c_i.#1.#2;%
}%
\def\XINTinFloatE {\romannumeral0\XINTinfloate }%
\def\XINTinfloate
   {\expandafter\XINT_infloate\romannumeral0\XINTinfloat[\XINTdigits]}%
\def\XINT_infloate #1[#2]#3%
   {\expandafter\XINT_infloate_end\the\numexpr #3+#2.{#1}}%
\def\XINT_infloate_end #1.#2{ #2[#1]}%
```

8.77 \XINTinFloatMod

1.1. Pour emploi dans xintexpr. Code shortened at 1.2p.

```
\def\XINTinFloatMod {\romannumeral0\XINTinfloatmod [\XINTdigits]}%
\def\XINTinfloatmod [#1]#2#3%
{%
    \XINTinfloat[#1]{\xintMod
        {\romannumeral0\XINTinfloat[#1]{#2}}%
        {\romannumeral0\XINTinfloat[#1]{#3}}}%
}%
```

8.78 \XINTinFloatDivFloor

1.2p. Formerly // and /: in \xintfloatexpr used \xintDivFloor and \xintMod, hence did not round their operands to float precision beforehand.

```
\def\XINTinFloatDivFloor {\romannumeral0\XINTinfloatdivfloor [\XINTdigits]}%
\def\XINTinfloatdivfloor [#1]#2#3%
{%
    \xintdivfloor
          {\romannumeral0\XINTinfloat[#1]{#2}}%
          {\romannumeral0\XINTinfloat[#1]{#3}}%
}%
```

8.79 \XINTinFloatDivMod

1.2p. Pour emploi dans xintexpr, donc je ne prends pas la peine de faire l'expansion du modulo, qui se produira dans le \csname.
  Hésitation sur le quotient, faut-il l'arrondir immédiatement ? Finalement non, le produire comme un integer.

```
\def\XINTinFloatDivMod {\romannumeral0\XINTinfloatdivmod [\XINTdigits]}%
\def\XINTinfloatdivmod [#1]#2#3%
{%
    \expandafter\XINT_infloatdivmod
    \romannumeral0\xintdivmod
          {\romannumeral0\XINTinfloat[#1]{#2}}%
          {\romannumeral0\XINTinfloat[#1]{#3}}%
    {#1}%
}%
\def\XINT_infloatdivmod #1#2#3{ #1,\XINTinFloat[#3]{#2}}%
```

8.80 `\xintifFloatInt`

```
\def\xintifFloatInt {\romannumeral0\xintiffloatint}%
\def\xintiffloatint #1{\expandafter\XINT_iffloatint
                       \romannumeral0\xintrez{\XINTinFloat[\XINTdigits]{#1}}}%
\def\XINT_iffloatint #1#2/1[#3]%
{%
  \if 0#1\xint_dothis\xint_firstoftwo_thenstop\fi
  \ifnum#3<\xint_c_\xint_dothis\xint_secondoftwo_thenstop\fi
  \xint_orthat\xint_firstoftwo_thenstop
}%
```

8.81 (WIP) `\XINTinRandomFloatS`, `\XINTinRandomFloatSdigits`

1.3b. Support for random() function.

Thus as it is a priori only for xintexpr usage, it expands inside \csname context, but as we need to get rid of initial zeros we use \xintRandomDigits not \xintXRandomDigits (\expanded would have a use case here).

And anyway as we want to be able to use random() in \xintdeffunc/\xintNewExpr, it is good to have f-expandable macros, so we add the small overhead to make it f-expandable.

We don't have to be very efficient in removing leading zeroes, as there is only 10% chance for each successive one. Besides we use (current) internal storage format of the type A[N], where A is not required to be with \xintDigits digits, so N will simply be -\xintDigits and needs no adjustment.

In case we use in future with #1 something else than \xintDigits we do the 0-(#1) construct.

I had some qualms about doing a random float like this which means that when there are leading zeros in the random digits the (virtual) mantissa ends up with trailing zeros. That did not feel right but I checked random() in Python (which of course uses radix 2), and indeed this is what happens there.

```
\def\XINTinRandomFloatS{\romannumeral0\XINTinrandomfloatS}%
\def\XINTinRandomFloatSdigits{\XINTinRandomFloatS[\XINTdigits]}%
\def\XINTinrandomfloatS[#1]%
{%
    \expandafter\XINT_inrandomfloatS\the\numexpr\xint_c_-(#1)\xint:
}%
\def\XINT_inrandomfloatS-#1\xint:
{%
    \expandafter\XINT_inrandomfloatS_a
    \romannumeral0\xintrandomdigits{#1}[-#1]%
}%
```

We add one macro to handle a tiny bit faster 90of cases, after all we also use one extra macro for the completely improbable all 0 case.

```
\def\XINT_inrandomfloatS_a#1%
{%
    \if#10\xint_dothis{\XINT_inrandomfloatS_b}\fi
    \xint_orthat{ #1}%
}%[
\def\XINT_inrandomfloatS_b#1%
{%
    \if#1[\xint_dothis{\XINT_inrandomfloatS_zero}\fi% ]
    \if#10\xint_dothis{\XINT_inrandomfloatS_b}\fi
    \xint_orthat{ #1}%
}%[
\def\XINT_inrandomfloatS_zero#1]{ 0[0]}%
```

8.82 (WIP) \XINTinRandomFloatSixteen

1.3b. Support for qrand() function.

```
\def\XINTinRandomFloatSixteen%
{%
    \romannumeral0\expandafter\XINT_inrandomfloatS_a
    \romannumeral`&&@\expandafter\XINT_eightrandomdigits
                \romannumeral`&&@\XINT_eightrandomdigits[-16]%
}%
\XINT_restorecatcodes_endinput%
```

9 Package xintseries implementation

The commenting is currently (2018/06/17) very sparse.

9.1 Catcodes, ε-TEX and reload detection

The code for reload detection was initially copied from Heiko Oberdiek's packages, then modified.

The method for catcodes was also initially directly inspired by these packages.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode64=11   % @
  \catcode35=6    % #
  \catcode44=12   % ,
  \catcode45=12   % -
  \catcode46=12   % .
  \catcode58=12   % :
  \let\z\endgroup
  \expandafter\let\expandafter\x\csname ver@xintseries.sty\endcsname
  \expandafter\let\expandafter\w\csname ver@xintfrac.sty\endcsname
  \expandafter
    \ifx\csname PackageInfo\endcsname\relax
      \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
    \else
      \def\y#1#2{\PackageInfo{#1}{#2}}%
    \fi
  \expandafter
  \ifx\csname numexpr\endcsname\relax
     \y{xintseries}{\numexpr not available, aborting input}%
     \aftergroup\endinput
  \else
    \ifx\x\relax   % plain-TeX, first loading of xintseries.sty
      \ifx\w\relax % but xintfrac.sty not yet loaded.
         \def\z{\endgroup\input xintfrac.sty\relax}%
      \fi
    \else
      \def\empty {}%
      \ifx\x\empty % LaTeX, first loading,
      % variable is initialized, but \ProvidesPackage not yet seen
          \ifx\w\relax % xintfrac.sty not yet loaded.
            \def\z{\endgroup\RequirePackage{xintfrac}}%
          \fi
      \else
```

```
        \aftergroup\endinput % xintseries already loaded.
      \fi
    \fi
  \fi
\z%
\XINTsetupcatcodes% defined in xintkernel.sty
```

9.2 Package identification

```
\XINT_providespackage
\ProvidesPackage{xintseries}%
  [2018/06/17 1.3c Expandable partial sums with xint package (JFB)]%
```

9.3 `\xintSeries`

```
\def\xintSeries {\romannumeral0\xintseries }%
\def\xintseries #1#2%
{%
    \expandafter\XINT_series\expandafter
    {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
}%
\def\XINT_series #1#2#3%
{%
   \ifnum #2<#1
      \xint_afterfi { 0/1[0]}%
   \else
      \xint_afterfi {\XINT_series_loop {#1}{0}{#2}{#3}}%
   \fi
}%
\def\XINT_series_loop #1#2#3#4%
{%
    \ifnum #3>#1 \else \XINT_series_exit \fi
    \expandafter\XINT_series_loop\expandafter
    {\the\numexpr #1+1\expandafter }\expandafter
    {\romannumeral0\xintadd {#2}{#4{#1}}}%
    {#3}{#4}%
}%
\def\XINT_series_exit \fi #1#2#3#4#5#6#7#8%
{%
    \fi\xint_gobble_ii #6%
}%
```

9.4 `\xintiSeries`

```
\def\xintiSeries {\romannumeral0\xintiseries }%
\def\xintiseries #1#2%
{%
    \expandafter\XINT_iseries\expandafter
    {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
}%
\def\XINT_iseries #1#2#3%
{%
   \ifnum #2<#1
      \xint_afterfi { 0}%
   \else
```

```
      \xint_afterfi {\XINT_iseries_loop {#1}{0}{#2}{#3}}%
   \fi
}%
\def\XINT_iseries_loop #1#2#3#4%
{%
    \ifnum #3>#1 \else \XINT_iseries_exit \fi
    \expandafter\XINT_iseries_loop\expandafter
    {\the\numexpr #1+1\expandafter }\expandafter
    {\romannumeral0\xintiiadd {#2}{#4{#1}}}%
    {#3}{#4}%
}%
\def\XINT_iseries_exit \fi #1#2#3#4#5#6#7#8%
{%
    \fi\xint_gobble_ii #6%
}%
```

9.5 \xintPowerSeries

The 1.03 version was very lame and created a build-up of denominators. (this was at a time \xintAdd always multiplied denominators, by the way) The Horner scheme for polynomial evaluation is used in 1.04, this cures the denominator problem and drastically improves the efficiency of the macro. Modified in 1.06 to give the indices first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that. 1.08a adds the forgotten optimization following that previous change.

```
\def\xintPowerSeries {\romannumeral0\xintpowerseries }%
\def\xintpowerseries #1#2%
{%
    \expandafter\XINT_powseries\expandafter
    {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
}%
\def\XINT_powseries #1#2#3#4%
{%
   \ifnum #2<#1
      \xint_afterfi { 0/1[0]}%
   \else
      \xint_afterfi
      {\XINT_powseries_loop_i {#3{#2}}{#1}{#2}{#3}{#4}}%
   \fi
}%
\def\XINT_powseries_loop_i #1#2#3#4#5%
{%
    \ifnum #3>#2 \else\XINT_powseries_exit_i\fi
    \expandafter\XINT_powseries_loop_ii\expandafter
    {\the\numexpr #3-1\expandafter}\expandafter
    {\romannumeral0\xintmul {#1}{#5}}{#2}{#4}{#5}%
}%
\def\XINT_powseries_loop_ii #1#2#3#4%
{%
   \expandafter\XINT_powseries_loop_i\expandafter
   {\romannumeral0\xintadd {#4{#1}}{#2}}{#3}{#1}{#4}%
}%
\def\XINT_powseries_exit_i\fi #1#2#3#4#5#6#7#8#9%
{%
```

```
    \fi \XINT_powseries_exit_ii  #6{#7}%
}%
\def\XINT_powseries_exit_ii #1#2#3#4#5#6%
{%
    \xintmul{\xintPow {#5}{#6}}{#4}%
}%
```

9.6 `\xintPowerSeriesX`

Same as \xintPowerSeries except for the initial expansion of the x parameter. Modified in 1.06 to give the indices first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that. 1.08a adds the forgotten optimization following that previous change.

```
\def\xintPowerSeriesX {\romannumeral0\xintpowerseriesx }%
\def\xintpowerseriesx #1#2%
{%
    \expandafter\XINT_powseriesx\expandafter
    {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
}%
\def\XINT_powseriesx #1#2#3#4%
{%
   \ifnum #2<#1
      \xint_afterfi { 0/1[0]}%
   \else
      \xint_afterfi
      {\expandafter\XINT_powseriesx_pre\expandafter
                  {\romannumeral`&&@#4}{#1}{#2}{#3}%
      }%
   \fi
}%
\def\XINT_powseriesx_pre #1#2#3#4%
{%
    \XINT_powseries_loop_i {#4{#3}}{#2}{#3}{#4}{#1}%
}%
```

9.7 `\xintRationalSeries`

This computes F(a)+...+F(b) on the basis of the value of F(a) and the ratios F(n)/F(n-1). As in \xintPowerSeries we use an iterative scheme which has the great advantage to avoid denominator build-up. This makes exact computations possible with exponential type series, which would be completely inaccessible to \xintSeries. #1=a, #2=b, #3=F(a), #4=ratio function Modified in 1.06 to give the indices first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that. 1.08a adds the forgotten optimization following that previous change.

```
\def\xintRationalSeries {\romannumeral0\xintratseries }%
\def\xintratseries #1#2%
{%
    \expandafter\XINT_ratseries\expandafter
    {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
}%
\def\XINT_ratseries #1#2#3#4%
```

```
{%
  \ifnum #2<#1
     \xint_afterfi { 0/1[0]}%
  \else
     \xint_afterfi
     {\XINT_ratseries_loop {#2}{1}{#1}{#4}{#3}}%
  \fi
}%
\def\XINT_ratseries_loop #1#2#3#4%
{%
    \ifnum #1>#3 \else\XINT_ratseries_exit_i\fi
    \expandafter\XINT_ratseries_loop\expandafter
    {\the\numexpr #1-1\expandafter}\expandafter
    {\romannumeral0\xintadd {1}{\xintMul {#2}{#4{#1}}}}{#3}{#4}%
}%
\def\XINT_ratseries_exit_i\fi #1#2#3#4#5#6#7#8%
{%
    \fi \XINT_ratseries_exit_ii  #6%
}%
\def\XINT_ratseries_exit_ii #1#2#3#4#5%
{%
    \XINT_ratseries_exit_iii #5%
}%
\def\XINT_ratseries_exit_iii #1#2#3#4%
{%
    \xintmul{#2}{#4}%
}%
```

9.8 \xintRationalSeriesX

a,b,initial,ratiofunction,x
This computes F(a,x)+...+F(b,x) on the basis of the value of F(a,x) and the ratios F(n,x)/F(n-1,x). The argument x is first expanded and it is the value resulting from this which is used then throughout. The initial term F(a,x) must be defined as one-parameter macro which will be given x. Modified in 1.06 to give the indices first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that. 1.08a adds the forgotten optimization following that previous change.

```
\def\xintRationalSeriesX {\romannumeral0\xintratseriesx }%
\def\xintratseriesx #1#2%
{%
    \expandafter\XINT_ratseriesx\expandafter
    {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
}%
\def\XINT_ratseriesx #1#2#3#4#5%
{%
  \ifnum #2<#1
     \xint_afterfi { 0/1[0]}%
  \else
     \xint_afterfi
     {\expandafter\XINT_ratseriesx_pre\expandafter
                  {\romannumeral`&&@#5}{#2}{#1}{#4}{#3}%
     }%
```

```
   \fi
}%
\def\XINT_ratseriesx_pre #1#2#3#4#5%
{%
    \XINT_ratseries_loop {#2}{1}{#3}{#4{#1}}{#5{#1}}%
}%
```

9.9 \xintFxPtPowerSeries

I am not two happy with this piece of code. Will make it more economical another day. Modified in 1.06 to give the indices first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that. 1.08a: forgot last time some optimization from the change to \numexpr.

```
\def\xintFxPtPowerSeries {\romannumeral0\xintfxptpowerseries }%
\def\xintfxptpowerseries #1#2%
{%
    \expandafter\XINT_fppowseries\expandafter
    {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
}%
\def\XINT_fppowseries #1#2#3#4#5%
{%
   \ifnum #2<#1
      \xint_afterfi { 0}%
   \else
      \xint_afterfi
        {\expandafter\XINT_fppowseries_loop_pre\expandafter
           {\romannumeral0\xinttrunc {#5}{\xintPow {#4}{#1}}}%
          {#1}{#4}{#2}{#3}{#5}%
        }%
   \fi
}%
\def\XINT_fppowseries_loop_pre #1#2#3#4#5#6%
{%
    \ifnum #4>#2 \else\XINT_fppowseries_dont_i \fi
    \expandafter\XINT_fppowseries_loop_i\expandafter
    {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
    {\romannumeral0\xintitrunc {#6}{\xintMul {#5{#2}}{#1}}}%
    {#1}{#3}{#4}{#5}{#6}%
}%
\def\XINT_fppowseries_dont_i \fi\expandafter\XINT_fppowseries_loop_i
    {\fi \expandafter\XINT_fppowseries_dont_ii }%
\def\XINT_fppowseries_dont_ii #1#2#3#4#5#6#7{\xinttrunc {#7}{#2[-#7]}}%
\def\XINT_fppowseries_loop_i #1#2#3#4#5#6#7%
{%
    \ifnum #5>#1 \else \XINT_fppowseries_exit_i \fi
    \expandafter\XINT_fppowseries_loop_ii\expandafter
    {\romannumeral0\xinttrunc {#7}{\xintMul {#3}{#4}}}%
    {#1}{#4}{#2}{#5}{#6}{#7}%
}%
\def\XINT_fppowseries_loop_ii #1#2#3#4#5#6#7%
{%
    \expandafter\XINT_fppowseries_loop_i\expandafter
    {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
    {\romannumeral0\xintiiadd {#4}{\xintiTrunc {#7}{\xintMul {#6{#2}}{#1}}}}%
    {#1}{#3}{#5}{#6}{#7}%
}%
\def\XINT_fppowseries_exit_i\fi\expandafter\XINT_fppowseries_loop_ii
    {\fi \expandafter\XINT_fppowseries_exit_ii }%
\def\XINT_fppowseries_exit_ii #1#2#3#4#5#6#7%
{%
    \xinttrunc {#7}
    {\xintiiadd {#4}{\xintiTrunc {#7}{\xintMul {#6{#2}}{#1}}}[-#7]}%
}%
```

9.10 \xintFxPtPowerSeriesX

a,b,coeff,x,D
Modified in 1.06 to give the indices first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that. 1.08a adds the forgotten optimization following that previous change.

```
\def\xintFxPtPowerSeriesX {\romannumeral0\xintfxptpowerseriesx }%
\def\xintfxptpowerseriesx #1#2%
{%
    \expandafter\XINT_fppowseriesx\expandafter
    {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
}%
\def\XINT_fppowseriesx #1#2#3#4#5%
{%
  \ifnum #2<#1
     \xint_afterfi { 0}%
  \else
     \xint_afterfi
       {\expandafter \XINT_fppowseriesx_pre \expandafter
        {\romannumeral`&&@#4}{#1}{#2}{#3}{#5}%
       }%
  \fi
}%
\def\XINT_fppowseriesx_pre #1#2#3#4#5%
{%
    \expandafter\XINT_fppowseries_loop_pre\expandafter
       {\romannumeral0\xinttrunc {#5}{\xintPow {#1}{#2}}}%
       {#2}{#1}{#3}{#4}{#5}%
}%
```

9.11 \xintFloatPowerSeries

1.08a. I still have to re-visit \xintFxPtPowerSeries; temporarily I just adapted the code to the case of floats.

```
\def\xintFloatPowerSeries {\romannumeral0\xintfloatpowerseries }%
\def\xintfloatpowerseries #1{\XINT_flpowseries_chkopt #1\xint:}%
\def\XINT_flpowseries_chkopt #1%
{%
    \ifx [#1\expandafter\XINT_flpowseries_opt
```

```
      \else\expandafter\XINT_flpowseries_noopt
    \fi
    #1%
}%
\def\XINT_flpowseries_noopt  #1\xint:#2%
{%
    \expandafter\XINT_flpowseries\expandafter
    {\the\numexpr #1\expandafter}\expandafter
    {\the\numexpr #2}\XINTdigits
}%
\def\XINT_flpowseries_opt [\xint:#1]#2#3%
{%
    \expandafter\XINT_flpowseries\expandafter
    {\the\numexpr #2\expandafter}\expandafter
    {\the\numexpr #3\expandafter}{\the\numexpr #1}%
}%
\def\XINT_flpowseries #1#2#3#4#5%
{%
   \ifnum #2<#1
      \xint_afterfi { 0.e0}%
   \else
      \xint_afterfi
        {\expandafter\XINT_flpowseries_loop_pre\expandafter
           {\romannumeral0\XINTinfloatpow [#3]{#5}{#1}}%
          {#1}{#5}{#2}{#4}{#3}%
        }%
   \fi
}%
\def\XINT_flpowseries_loop_pre #1#2#3#4#5#6%
{%
    \ifnum #4>#2 \else\XINT_flpowseries_dont_i \fi
    \expandafter\XINT_flpowseries_loop_i\expandafter
    {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
    {\romannumeral0\XINTinfloatmul [#6]{#5{#2}}{#1}}%
    {#1}{#3}{#4}{#5}{#6}%
}%
\def\XINT_flpowseries_dont_i \fi\expandafter\XINT_flpowseries_loop_i
    {\fi \expandafter\XINT_flpowseries_dont_ii }%
\def\XINT_flpowseries_dont_ii #1#2#3#4#5#6#7{\xintfloat [#7]{#2}}%
\def\XINT_flpowseries_loop_i #1#2#3#4#5#6#7%
{%
    \ifnum #5>#1 \else \XINT_flpowseries_exit_i \fi
    \expandafter\XINT_flpowseries_loop_ii\expandafter
    {\romannumeral0\XINTinfloatmul [#7]{#3}{#4}}%
    {#1}{#4}{#2}{#5}{#6}{#7}%
}%
\def\XINT_flpowseries_loop_ii #1#2#3#4#5#6#7%
{%
    \expandafter\XINT_flpowseries_loop_i\expandafter
    {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
    {\romannumeral0\XINTinfloatadd [#7]{#4}%
                        {\XINTinfloatmul [#7]{#6{#2}}{#1}}}%
    {#1}{#3}{#5}{#6}{#7}%
}%
\def\XINT_flpowseries_exit_i\fi\expandafter\XINT_flpowseries_loop_ii
    {\fi \expandafter\XINT_flpowseries_exit_ii }%
\def\XINT_flpowseries_exit_ii #1#2#3#4#5#6#7%
{%
    \xintfloatadd [#7]{#4}{\XINTinfloatmul [#7]{#6{#2}}{#1}}%
}%
```

9.12 \xintFloatPowerSeriesX

1.08a

```
\def\xintFloatPowerSeriesX {\romannumeral0\xintfloatpowerseriesx }%
\def\xintfloatpowerseriesx #1{\XINT_flpowseriesx_chkopt #1\xint:}%
\def\XINT_flpowseriesx_chkopt #1%
{%
    \ifx [#1\expandafter\XINT_flpowseriesx_opt
       \else\expandafter\XINT_flpowseriesx_noopt
    \fi
    #1%
}%
\def\XINT_flpowseriesx_noopt  #1\xint:#2%
{%
    \expandafter\XINT_flpowseriesx\expandafter
    {\the\numexpr #1\expandafter}\expandafter
    {\the\numexpr #2}\XINTdigits
}%
\def\XINT_flpowseriesx_opt [\xint:#1]#2#3%
{%
    \expandafter\XINT_flpowseriesx\expandafter
    {\the\numexpr #2\expandafter}\expandafter
    {\the\numexpr #3\expandafter}{\the\numexpr #1}%
}%
\def\XINT_flpowseriesx #1#2#3#4#5%
{%
   \ifnum #2<#1
      \xint_afterfi { 0.e0}%
   \else
      \xint_afterfi
        {\expandafter \XINT_flpowseriesx_pre \expandafter
         {\romannumeral`&&@#5}{#1}{#2}{#4}{#3}%
        }%
   \fi
}%
\def\XINT_flpowseriesx_pre #1#2#3#4#5%
{%
    \expandafter\XINT_flpowseries_loop_pre\expandafter
       {\romannumeral0\XINTinfloatpow [#5]{#1}{#2}}%
       {#2}{#1}{#3}{#4}{#5}%
}%
\XINT_restorecatcodes_endinput%
```

10 Package xintcfrac implementation

The commenting is currently (2018/06/17) very sparse. Release 1.09m (2014/02/26) has modified a few things: \xintFtoCs and \xintCntoCs insert spaces after the commas, \xintCstoF and \xintCstoCv authorize spaces in the input also before the commas, \xintCntoCs does not brace the produced coefficients, new macros \xintFtoC, \xintCtoF, \xintCtoCv, \xintFGtoC, and \xintGGCFrac.

There is partial dependency on xinttools due to \xintCstoF and \xintCsToCv.

10.1 Catcodes, ε-TEX and reload detection

The code for reload detection was initially copied from Heiko Oberdiek's packages, then modified.

The method for catcodes was also initially directly inspired by these packages.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode64=11   % @
  \catcode35=6    % #
  \catcode44=12   % ,
  \catcode45=12   % -
  \catcode46=12   % .
  \catcode58=12   % :
  \let\z\endgroup
  \expandafter\let\expandafter\x\csname ver@xintcfrac.sty\endcsname
  \expandafter\let\expandafter\w\csname ver@xintfrac.sty\endcsname
  \expandafter
    \ifx\csname PackageInfo\endcsname\relax
      \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
    \else
      \def\y#1#2{\PackageInfo{#1}{#2}}%
    \fi
  \expandafter
  \ifx\csname numexpr\endcsname\relax
     \y{xintcfrac}{\numexpr not available, aborting input}%
     \aftergroup\endinput
  \else
    \ifx\x\relax   % plain-TeX, first loading of xintcfrac.sty
      \ifx\w\relax % but xintfrac.sty not yet loaded.
         \def\z{\endgroup\input xintfrac.sty\relax}%
      \fi
    \else
      \def\empty {}%
      \ifx\x\empty % LaTeX, first loading,
      % variable is initialized, but \ProvidesPackage not yet seen
          \ifx\w\relax % xintfrac.sty not yet loaded.
            \def\z{\endgroup\RequirePackage{xintfrac}}%
          \fi
      \else
        \aftergroup\endinput % xintcfrac already loaded.
      \fi
    \fi
  \fi
\z%
\XINTsetupcatcodes% defined in xintkernel.sty
```

10.2 Package identification

```
\XINT_providespackage
\ProvidesPackage{xintcfrac}%
  [2018/06/17 1.3c Expandable continued fractions with xint package (JFB)]%
```

10.3 \xintCFrac

```
\def\xintCFrac {\romannumeral0\xintcfrac }%
\def\xintcfrac #1%
{%
    \XINT_cfrac_opt_a #1\xint:
}%
\def\XINT_cfrac_opt_a #1%
{%
    \ifx[#1\XINT_cfrac_opt_b\fi \XINT_cfrac_noopt #1%
}%
\def\XINT_cfrac_noopt #1\xint:
{%
    \expandafter\XINT_cfrac_A\romannumeral0\xintrawwithzeros {#1}\Z
    \relax\relax
}%
\def\XINT_cfrac_opt_b\fi\XINT_cfrac_noopt [\xint:#1]%
{%
    \fi\csname XINT_cfrac_opt#1\endcsname
}%
\def\XINT_cfrac_optl #1%
{%
    \expandafter\XINT_cfrac_A\romannumeral0\xintrawwithzeros {#1}\Z
    \relax\hfill
}%
\def\XINT_cfrac_optc #1%
{%
    \expandafter\XINT_cfrac_A\romannumeral0\xintrawwithzeros {#1}\Z
    \relax\relax
}%
\def\XINT_cfrac_optr #1%
{%
    \expandafter\XINT_cfrac_A\romannumeral0\xintrawwithzeros {#1}\Z
    \hfill\relax
}%
\def\XINT_cfrac_A #1/#2\Z
{%
    \expandafter\XINT_cfrac_B\romannumeral0\xintiidivision {#1}{#2}{#2}%
}%
\def\XINT_cfrac_B #1#2%
{%
    \XINT_cfrac_C #2\Z {#1}%
}%
\def\XINT_cfrac_C #1%
{%
    \xint_gob_til_zero #1\XINT_cfrac_integer 0\XINT_cfrac_D #1%
}%
\def\XINT_cfrac_integer 0\XINT_cfrac_D 0#1\Z #2#3#4#5{ #2}%
\def\XINT_cfrac_D #1\Z #2#3{\XINT_cfrac_loop_a {#1}{#3}{#1}{{#2}}}%
\def\XINT_cfrac_loop_a
{%
    \expandafter\XINT_cfrac_loop_d\romannumeral0\XINT_div_prepare
}%
\def\XINT_cfrac_loop_d #1#2%
{%
    \XINT_cfrac_loop_e #2.{#1}%
}%
\def\XINT_cfrac_loop_e #1%
{%
    \xint_gob_til_zero #1\xint_cfrac_loop_exit0\XINT_cfrac_loop_f #1%
}%
\def\XINT_cfrac_loop_f #1.#2#3#4%
{%
    \XINT_cfrac_loop_a {#1}{#3}{#1}{{#2}#4}%
}%
\def\xint_cfrac_loop_exit0\XINT_cfrac_loop_f #1.#2#3#4#5#6%
   {\XINT_cfrac_T #5#6{#2}#4\Z }%
\def\XINT_cfrac_T #1#2#3#4%
{%
  \xint_gob_til_Z #4\XINT_cfrac_end\Z\XINT_cfrac_T #1#2{#4+\cfrac{#11#2}{#3}}%
}%
\def\XINT_cfrac_end\Z\XINT_cfrac_T #1#2#3%
{%
    \XINT_cfrac_end_b #3%
}%
\def\XINT_cfrac_end_b \Z+\cfrac#1#2{ #2}%
```

10.4 \xintGCFrac

```
\def\xintGCFrac {\romannumeral0\xintgcfrac }%
\def\xintgcfrac #1{\XINT_gcfrac_opt_a #1\xint:}%
\def\XINT_gcfrac_opt_a #1%
```

```
{%
    \ifx[#1\XINT_gcfrac_opt_b\fi \XINT_gcfrac_noopt #1%
}%
\def\XINT_gcfrac_noopt #1\xint:%
{%
    \XINT_gcfrac #1+!/\relax\relax
}%
\def\XINT_gcfrac_opt_b\fi\XINT_gcfrac_noopt [\xint:#1]%
{%
    \fi\csname XINT_gcfrac_opt#1\endcsname
}%
\def\XINT_gcfrac_optl #1%
{%
    \XINT_gcfrac #1+!/\relax\hfill
}%
\def\XINT_gcfrac_optc #1%
{%
    \XINT_gcfrac #1+!/\relax\relax
}%
\def\XINT_gcfrac_optr #1%
{%
    \XINT_gcfrac #1+!/\hfill\relax
}%
\def\XINT_gcfrac
{%
    \expandafter\XINT_gcfrac_enter\romannumeral`&&@%
}%
\def\XINT_gcfrac_enter {\XINT_gcfrac_loop {}}%
\def\XINT_gcfrac_loop #1#2+#3/%
{%
    \xint_gob_til_exclam #3\XINT_gcfrac_endloop!%
    \XINT_gcfrac_loop {{#3}{#2}#1}%
}%
\def\XINT_gcfrac_endloop!\XINT_gcfrac_loop #1#2#3%
{%
    \XINT_gcfrac_T #2#3#1!!%
}%
\def\XINT_gcfrac_T #1#2#3#4{\XINT_gcfrac_U #1#2{\xintFrac{#4}}}%
\def\XINT_gcfrac_U #1#2#3#4#5%
{%
    \xint_gob_til_exclam #5\XINT_gcfrac_end!\XINT_gcfrac_U
              #1#2{\xintFrac{#5}%
               \ifcase\xintSgn{#4}
               +\or+\else-\fi
               \cfrac{#1\xintFrac{\xintAbs{#4}}#2}{#3}}%
}%
\def\XINT_gcfrac_end!\XINT_gcfrac_U #1#2#3%
{%
    \XINT_gcfrac_end_b #3%
}%
\def\XINT_gcfrac_end_b #1\cfrac#2#3{ #3}%
```

10.5 \xintGGCFrac

New with 1.09m

```
\def\xintGGCFrac {\romannumeral0\xintggcfrac }%
\def\xintggcfrac #1{\XINT_ggcfrac_opt_a #1\xint:}%
\def\XINT_ggcfrac_opt_a #1%
{%
    \ifx[#1\XINT_ggcfrac_opt_b\fi \XINT_ggcfrac_noopt #1%
}%
\def\XINT_ggcfrac_noopt #1\xint:
{%
    \XINT_ggcfrac #1+!/\relax\relax
}%
\def\XINT_ggcfrac_opt_b\fi\XINT_ggcfrac_noopt [\xint:#1]%
{%
    \fi\csname XINT_ggcfrac_opt#1\endcsname
}%
\def\XINT_ggcfrac_optl #1%
{%
    \XINT_ggcfrac #1+!/\relax\hfill
}%
\def\XINT_ggcfrac_optc #1%
{%
    \XINT_ggcfrac #1+!/\relax\relax
}%
\def\XINT_ggcfrac_optr #1%
{%
    \XINT_ggcfrac #1+!/\hfill\relax
}%
\def\XINT_ggcfrac
{%
    \expandafter\XINT_ggcfrac_enter\romannumeral`&&@%
}%
\def\XINT_ggcfrac_enter {\XINT_ggcfrac_loop {}}%
\def\XINT_ggcfrac_loop #1#2+#3/%
{%
    \xint_gob_til_exclam #3\XINT_ggcfrac_endloop!%
    \XINT_ggcfrac_loop {{#3}{#2}#1}%
}%
\def\XINT_ggcfrac_endloop!\XINT_ggcfrac_loop #1#2#3%
{%
    \XINT_ggcfrac_T #2#3#1!!%
}%
\def\XINT_ggcfrac_T #1#2#3#4{\XINT_ggcfrac_U #1#2{#4}}%
\def\XINT_ggcfrac_U #1#2#3#4#5%
{%
    \xint_gob_til_exclam #5\XINT_ggcfrac_end!\XINT_ggcfrac_U
              #1#2{#5+\cfrac{#1#4#2}{#3}}%
}%
\def\XINT_ggcfrac_end!\XINT_ggcfrac_U #1#2#3%
{%
    \XINT_ggcfrac_end_b #3%
}%
\def\XINT_ggcfrac_end_b #1\cfrac#2#3{ #3}%
```

10.6 \xintGCtoGCx

```
\def\xintGCtoGCx {\romannumeral0\xintgctogcx }%
\def\xintgctogcx #1#2#3%
{%
    \expandafter\XINT_gctgcx_start\expandafter {\romannumeral`&&@#3}{#1}{#2}%
}%
\def\XINT_gctgcx_start #1#2#3{\XINT_gctgcx_loop_a {}{#2}{#3}#1+!/}%
\def\XINT_gctgcx_loop_a #1#2#3#4+#5/%
{%
    \xint_gob_til_exclam #5\XINT_gctgcx_end!%
    \XINT_gctgcx_loop_b {#1{#4}}{#2{#5}#3}{#2}{#3}%
}%
\def\XINT_gctgcx_loop_b #1#2%
{%
    \XINT_gctgcx_loop_a {#1#2}%
}%
\def\XINT_gctgcx_end!\XINT_gctgcx_loop_b #1#2#3#4{ #1}%
```

10.7 \xintFtoCs

Modified in 1.09m: a space is added after the inserted commas.

```
\def\xintFtoCs {\romannumeral0\xintftocs }%
\def\xintftocs #1%
{%
    \expandafter\XINT_ftc_A\romannumeral0\xintrawwithzeros {#1}\Z
}%
\def\XINT_ftc_A #1/#2\Z
{%
    \expandafter\XINT_ftc_B\romannumeral0\xintiidivision {#1}{#2}{#2}%
}%
\def\XINT_ftc_B #1#2%
{%
    \XINT_ftc_C #2.{#1}%
}%
\def\XINT_ftc_C #1%
{%
    \xint_gob_til_zero #1\XINT_ftc_integer 0\XINT_ftc_D #1%
}%
\def\XINT_ftc_integer 0\XINT_ftc_D 0#1.#2#3{ #2}%
\def\XINT_ftc_D #1.#2#3{\XINT_ftc_loop_a {#1}{#3}{#1}{#2, }}% 1.09m adds a space
\def\XINT_ftc_loop_a
{%
    \expandafter\XINT_ftc_loop_d\romannumeral0\XINT_div_prepare
}%
\def\XINT_ftc_loop_d #1#2%
{%
    \XINT_ftc_loop_e #2.{#1}%
}%
\def\XINT_ftc_loop_e #1%
```

```
{%
    \xint_gob_til_zero #1\xint_ftc_loop_exit0\XINT_ftc_loop_f #1%
}%
\def\XINT_ftc_loop_f #1.#2#3#4%
{%
    \XINT_ftc_loop_a {#1}{#3}{#1}{#4#2, }% 1.09m has an added space here
}%
\def\xint_ftc_loop_exit0\XINT_ftc_loop_f #1.#2#3#4{ #4#2}%
```

10.8 \xintFtoCx

```
\def\xintFtoCx {\romannumeral0\xintftocx }%
\def\xintftocx #1#2%
{%
    \expandafter\XINT_ftcx_A\romannumeral0\xintrawwithzeros {#2}\Z {#1}%
}%
\def\XINT_ftcx_A #1/#2\Z
{%
    \expandafter\XINT_ftcx_B\romannumeral0\xintiidivision {#1}{#2}{#2}%
}%
\def\XINT_ftcx_B #1#2%
{%
    \XINT_ftcx_C #2.{#1}%
}%
\def\XINT_ftcx_C #1%
{%
    \xint_gob_til_zero #1\XINT_ftcx_integer 0\XINT_ftcx_D #1%
}%
\def\XINT_ftcx_integer 0\XINT_ftcx_D 0#1.#2#3#4{ #2}%
\def\XINT_ftcx_D #1.#2#3#4{\XINT_ftcx_loop_a {#1}{#3}{#1}{{#2}#4}{#4}}%
\def\XINT_ftcx_loop_a
{%
    \expandafter\XINT_ftcx_loop_d\romannumeral0\XINT_div_prepare
}%
\def\XINT_ftcx_loop_d #1#2%
{%
    \XINT_ftcx_loop_e #2.{#1}%
}%
\def\XINT_ftcx_loop_e #1%
{%
    \xint_gob_til_zero #1\xint_ftcx_loop_exit0\XINT_ftcx_loop_f #1%
}%
\def\XINT_ftcx_loop_f #1.#2#3#4#5%
{%
    \XINT_ftcx_loop_a {#1}{#3}{#1}{#4{#2}#5}{#5}%
}%
\def\xint_ftcx_loop_exit0\XINT_ftcx_loop_f #1.#2#3#4#5{ #4{#2}}%
```

10.9 \xintFtoC

New in 1.09m: this is the same as \xintFtoCx with empty separator. I had temporarily during preparation of 1.09m removed braces from \xintFtoCx, but I recalled later why that was useful (see doc), thus let's just here do \xintFtoCx {}

```
\def\xintFtoC {\romannumeral0\xintftoc }%
\def\xintftoc {\xintftocx {}}%
```

10.10 \xintFtoGC

```
\def\xintFtoGC {\romannumeral0\xintftogc }%
\def\xintftogc {\xintftocx {+1/}}%
```

10.11 \xintFGtoC

New with 1.09m of 2014/02/26. Computes the common initial coefficients for the two fractions f and g, and outputs them as a sequence of braced items.

```
\def\xintFGtoC {\romannumeral0\xintfgtoc}%
\def\xintfgtoc#1%
{%
    \expandafter\XINT_fgtc_a\romannumeral0\xintrawwithzeros {#1}\Z
}%
\def\XINT_fgtc_a #1/#2\Z #3%
{%
    \expandafter\XINT_fgtc_b\romannumeral0\xintrawwithzeros {#3}\Z #1/#2\Z { }%
}%
\def\XINT_fgtc_b #1/#2\Z
{%
    \expandafter\XINT_fgtc_c\romannumeral0\xintiidivision {#1}{#2}{#2}%
}%
\def\XINT_fgtc_c #1#2#3#4/#5\Z
{%
    \expandafter\XINT_fgtc_d\romannumeral0\xintiidivision
                                        {#4}{#5}{#5}{#1}{#2}{#3}%
}%
\def\XINT_fgtc_d #1#2#3#4%#5#6#7%
{%
    \xintifEq {#1}{#4}{\XINT_fgtc_da {#1}{#2}{#3}{#4}}%
                      {\xint_thirdofthree}%
}%
\def\XINT_fgtc_da #1#2#3#4#5#6#7%
{%
     \XINT_fgtc_e {#2}{#5}{#3}{#6}{#7{#1}}%
}%
\def\XINT_fgtc_e #1%
{%
    \xintiiifZero {#1}{\expandafter\xint_firstofone\xint_gobble_iii}%
                    {\XINT_fgtc_f {#1}}%
}%
\def\XINT_fgtc_f #1#2%
{%
   \xintiiifZero {#2}{\xint_thirdofthree}{\XINT_fgtc_g {#1}{#2}}%
}%
\def\XINT_fgtc_g #1#2#3%
{%
    \expandafter\XINT_fgtc_h\romannumeral0\XINT_div_prepare {#1}{#3}{#1}{#2}%
}%
\def\XINT_fgtc_h #1#2#3#4#5%
```

```
{%
    \expandafter\XINT_fgtc_d\romannumeral0\XINT_div_prepare
                       {#4}{#5}{#4}{#1}{#2}{#3}%
}%
```

10.12 \xintFtoCC

```
\def\xintFtoCC {\romannumeral0\xintftocc }%
\def\xintftocc #1%
{%
    \expandafter\XINT_ftcc_A\expandafter {\romannumeral0\xintrawwithzeros {#1}}%
}%
\def\XINT_ftcc_A #1%
{%
    \expandafter\XINT_ftcc_B
    \romannumeral0\xintrawwithzeros {\xintAdd {1/2[0]}{#1[0]}}\Z {#1[0]}%
}%
\def\XINT_ftcc_B #1/#2\Z
{%
    \expandafter\XINT_ftcc_C\expandafter {\romannumeral0\xintiiquo {#1}{#2}}%
}%
\def\XINT_ftcc_C #1#2%
{%
    \expandafter\XINT_ftcc_D\romannumeral0\xintsub {#2}{#1}\Z {#1}%
}%
\def\XINT_ftcc_D #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_ftcc_integer
      0#1\XINT_ftcc_En
       0-{\XINT_ftcc_Ep #1}%
    \krof
}%
\def\XINT_ftcc_Ep #1\Z #2%
{%
    \expandafter\XINT_ftcc_loop_a\expandafter
    {\romannumeral0\xintdiv {1[0]}{#1}}{#2+1/}%
}%
\def\XINT_ftcc_En #1\Z #2%
{%
    \expandafter\XINT_ftcc_loop_a\expandafter
    {\romannumeral0\xintdiv {1[0]}{#1}}{#2+-1/}%
}%
\def\XINT_ftcc_integer #1\Z #2{ #2}%
\def\XINT_ftcc_loop_a #1%
{%
    \expandafter\XINT_ftcc_loop_b
    \romannumeral0\xintrawwithzeros {\xintAdd {1/2[0]}{#1}}\Z {#1}%
}%
\def\XINT_ftcc_loop_b #1/#2\Z
{%
    \expandafter\XINT_ftcc_loop_c\expandafter
    {\romannumeral0\xintiiquo {#1}{#2}}%
}%
\def\XINT_ftcc_loop_c #1#2%
{%
    \expandafter\XINT_ftcc_loop_d
    \romannumeral0\xintsub {#2}{#1[0]}\Z {#1}%
}%
\def\XINT_ftcc_loop_d #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_ftcc_end
      0#1\XINT_ftcc_loop_N
       0-{\XINT_ftcc_loop_P #1}%
    \krof
}%
\def\XINT_ftcc_end #1\Z #2#3{ #3#2}%
\def\XINT_ftcc_loop_P #1\Z #2#3%
{%
    \expandafter\XINT_ftcc_loop_a\expandafter
    {\romannumeral0\xintdiv {1[0]}{#1}}{#3#2+1/}%
}%
\def\XINT_ftcc_loop_N #1\Z #2#3%
{%
    \expandafter\XINT_ftcc_loop_a\expandafter
    {\romannumeral0\xintdiv {1[0]}{#1}}{#3#2+-1/}%
}%
```

10.13 \xintCtoF, \xintCstoF

1.09m uses \xintCSVtoList on the argument of \xintCstoF to allow spaces also before the commas. And the original \xintCstoF code became the one of the new \xintCtoF dealing with a braced rather than comma separated list.

```
\def\xintCstoF {\romannumeral0\xintcstof }%
\def\xintcstof #1%
{%
    \expandafter\XINT_ctf_prep \romannumeral0\xintcsvtolist{#1}!%
}%
\def\xintCtoF {\romannumeral0\xintctof }%
\def\xintctof #1%
{%
    \expandafter\XINT_ctf_prep \romannumeral`&&@#1!%
}%
\def\XINT_ctf_prep
{%
    \XINT_ctf_loop_a 1001%
}%
\def\XINT_ctf_loop_a #1#2#3#4#5%
{%
    \xint_gob_til_exclam #5\XINT_ctf_end!%
    \expandafter\XINT_ctf_loop_b
    \romannumeral0\xintrawwithzeros {#5}.{#1}{#2}{#3}{#4}%
}%
\def\XINT_ctf_loop_b #1/#2.#3#4#5#6%
{%
```

```
    \expandafter\XINT_ctf_loop_c\expandafter
    {\romannumeral0\XINT_mul_fork #2\xint:#4\xint:}%
    {\romannumeral0\XINT_mul_fork #2\xint:#3\xint:}%
    {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#6\xint:}%
                              {\XINT_mul_fork #1\xint:#4\xint:}}%
    {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#5\xint:}%
                              {\XINT_mul_fork #1\xint:#3\xint:}}%
}%
\def\XINT_ctf_loop_c #1#2%
{%
    \expandafter\XINT_ctf_loop_d\expandafter {\expandafter{#2}{#1}}%
}%
\def\XINT_ctf_loop_d #1#2%
{%
    \expandafter\XINT_ctf_loop_e\expandafter {\expandafter{#2}#1}%
}%
\def\XINT_ctf_loop_e #1#2%
{%
    \expandafter\XINT_ctf_loop_a\expandafter{#2}#1%
}%
\def\XINT_ctf_end #1.#2#3#4#5{\xintrawwithzeros {#2/#3}}% 1.09b removes [0]
```

10.14 \xintiCstoF

```
\def\xintiCstoF {\romannumeral0\xinticstof }%
\def\xinticstof #1%
{%
    \expandafter\XINT_icstf_prep \romannumeral`&&@#1,!,%
}%
\def\XINT_icstf_prep
{%
    \XINT_icstf_loop_a 1001%
}%
\def\XINT_icstf_loop_a #1#2#3#4#5,%
{%
    \xint_gob_til_exclam #5\XINT_icstf_end!%
    \expandafter
    \XINT_icstf_loop_b \romannumeral`&&@#5.{#1}{#2}{#3}{#4}%
}%
\def\XINT_icstf_loop_b #1.#2#3#4#5%
{%
    \expandafter\XINT_icstf_loop_c\expandafter
    {\romannumeral0\xintiiadd {#5}{\XINT_mul_fork #1\xint:#3\xint:}}%
    {\romannumeral0\xintiiadd {#4}{\XINT_mul_fork #1\xint:#2\xint:}}%
    {#2}{#3}%
}%
\def\XINT_icstf_loop_c #1#2%
{%
    \expandafter\XINT_icstf_loop_a\expandafter {#2}{#1}%
}%
\def\XINT_icstf_end#1.#2#3#4#5{\xintrawwithzeros {#2/#3}}% 1.09b removes [0]
```

10.15 \xintGCtoF

```
\def\xintGCtoF {\romannumeral0\xintgctof }%
\def\xintgctof #1%
{%
    \expandafter\XINT_gctf_prep \romannumeral`&&@#1+!/%
}%
\def\XINT_gctf_prep
{%
    \XINT_gctf_loop_a 1001%
}%
\def\XINT_gctf_loop_a #1#2#3#4#5+%
{%
    \expandafter\XINT_gctf_loop_b
    \romannumeral0\xintrawwithzeros {#5}.{#1}{#2}{#3}{#4}%
}%
\def\XINT_gctf_loop_b #1/#2.#3#4#5#6%
{%
    \expandafter\XINT_gctf_loop_c\expandafter
    {\romannumeral0\XINT_mul_fork #2\xint:#4\xint:}%
    {\romannumeral0\XINT_mul_fork #2\xint:#3\xint:}%
    {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#6\xint:}%
                              {\XINT_mul_fork #1\xint:#4\xint:}}%
    {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#5\xint:}%
                              {\XINT_mul_fork #1\xint:#3\xint:}}%
}%
\def\XINT_gctf_loop_c #1#2%
{%
    \expandafter\XINT_gctf_loop_d\expandafter {\expandafter{#2}{#1}}%
}%
\def\XINT_gctf_loop_d #1#2%
{%
    \expandafter\XINT_gctf_loop_e\expandafter {\expandafter{#2}#1}%
}%
\def\XINT_gctf_loop_e #1#2%
{%
    \expandafter\XINT_gctf_loop_f\expandafter {\expandafter{#2}#1}%
}%
\def\XINT_gctf_loop_f #1#2/%
{%
    \xint_gob_til_exclam #2\XINT_gctf_end!%
    \expandafter\XINT_gctf_loop_g
    \romannumeral0\xintrawwithzeros {#2}.#1%
}%
\def\XINT_gctf_loop_g #1/#2.#3#4#5#6%
{%
    \expandafter\XINT_gctf_loop_h\expandafter
    {\romannumeral0\XINT_mul_fork #1\xint:#6\xint:}%
    {\romannumeral0\XINT_mul_fork #1\xint:#5\xint:}%
    {\romannumeral0\XINT_mul_fork #2\xint:#4\xint:}%
    {\romannumeral0\XINT_mul_fork #2\xint:#3\xint:}%
}%
\def\XINT_gctf_loop_h #1#2%
{%
```

```
    \expandafter\XINT_gctf_loop_i\expandafter {\expandafter{#2}{#1}}%
}%
\def\XINT_gctf_loop_i #1#2%
{%
    \expandafter\XINT_gctf_loop_j\expandafter {\expandafter{#2}#1}%
}%
\def\XINT_gctf_loop_j #1#2%
{%
    \expandafter\XINT_gctf_loop_a\expandafter {#2}#1%
}%
\def\XINT_gctf_end #1.#2#3#4#5{\xintrawwithzeros {#2/#3}}% 1.09b removes [0]
```

10.16 \xintiGCtoF

```
\def\xintiGCtoF {\romannumeral0\xintigctof }%
\def\xintigctof #1%
{%
    \expandafter\XINT_igctf_prep \romannumeral`&&@#1+!/%
}%
\def\XINT_igctf_prep
{%
    \XINT_igctf_loop_a 1001%
}%
\def\XINT_igctf_loop_a #1#2#3#4#5+%
{%
    \expandafter\XINT_igctf_loop_b
    \romannumeral`&&@#5.{#1}{#2}{#3}{#4}%
}%
\def\XINT_igctf_loop_b #1.#2#3#4#5%
{%
    \expandafter\XINT_igctf_loop_c\expandafter
    {\romannumeral0\xintiiadd {#5}{\XINT_mul_fork #1\xint:#3\xint:}}%
    {\romannumeral0\xintiiadd {#4}{\XINT_mul_fork #1\xint:#2\xint:}}%
    {#2}{#3}%
}%
\def\XINT_igctf_loop_c #1#2%
{%
    \expandafter\XINT_igctf_loop_f\expandafter {\expandafter{#2}{#1}}%
}%
\def\XINT_igctf_loop_f #1#2#3#4/%
{%
    \xint_gob_til_exclam #4\XINT_igctf_end!%
    \expandafter\XINT_igctf_loop_g
    \romannumeral`&&@#4.{#2}{#3}#1%
}%
\def\XINT_igctf_loop_g #1.#2#3%
{%
    \expandafter\XINT_igctf_loop_h\expandafter
    {\romannumeral0\XINT_mul_fork #1\xint:#3\xint:}%
    {\romannumeral0\XINT_mul_fork #1\xint:#2\xint:}%
}%
\def\XINT_igctf_loop_h #1#2%
{%
    \expandafter\XINT_igctf_loop_i\expandafter {#2}{#1}%
}%
\def\XINT_igctf_loop_i #1#2#3#4%
{%
    \XINT_igctf_loop_a {#3}{#4}{#1}{#2}%
}%
\def\XINT_igctf_end #1.#2#3#4#5{\xintrawwithzeros {#4/#5}}% 1.09b removes [0]
```

10.17 \xintCtoCv, \xintCstoCv

1.09m uses \xintCSVtoList on the argument of \xintCstoCv to allow spaces also before the commas. The original \xintCstoCv code became the one of the new \xintCtoF dealing with a braced rather than comma separated list.

```
\def\xintCstoCv {\romannumeral0\xintcstocv }%
\def\xintcstocv #1%
{%
    \expandafter\XINT_ctcv_prep\romannumeral0\xintcsvtolist{#1}!%
}%
\def\xintCtoCv {\romannumeral0\xintctocv }%
\def\xintctocv #1%
{%
    \expandafter\XINT_ctcv_prep\romannumeral`&&@#1!%
}%
\def\XINT_ctcv_prep
{%
    \XINT_ctcv_loop_a {}1001%
}%
\def\XINT_ctcv_loop_a #1#2#3#4#5#6%
{%
    \xint_gob_til_exclam #6\XINT_ctcv_end!%
    \expandafter\XINT_ctcv_loop_b
    \romannumeral0\xintrawwithzeros {#6}.{#2}{#3}{#4}{#5}{#1}%
}%
\def\XINT_ctcv_loop_b #1/#2.#3#4#5#6%
{%
    \expandafter\XINT_ctcv_loop_c\expandafter
    {\romannumeral0\XINT_mul_fork #2\xint:#4\xint:}%
    {\romannumeral0\XINT_mul_fork #2\xint:#3\xint:}%
    {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#6\xint:}%
                              {\XINT_mul_fork #1\xint:#4\xint:}}%
    {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#5\xint:}%
                              {\XINT_mul_fork #1\xint:#3\xint:}}%
}%
\def\XINT_ctcv_loop_c #1#2%
{%
    \expandafter\XINT_ctcv_loop_d\expandafter {\expandafter{#2}{#1}}%
}%
\def\XINT_ctcv_loop_d #1#2%
{%
    \expandafter\XINT_ctcv_loop_e\expandafter {\expandafter{#2}#1}%
}%
\def\XINT_ctcv_loop_e #1#2%
{%
    \expandafter\XINT_ctcv_loop_f\expandafter{#2}#1%
}%
\def\XINT_ctcv_loop_f #1#2#3#4#5%
{%
    \expandafter\XINT_ctcv_loop_g\expandafter
    {\romannumeral0\xintrawwithzeros {#1/#2}}{#5}{#1}{#2}{#3}{#4}%
}%
\def\XINT_ctcv_loop_g #1#2{\XINT_ctcv_loop_a {#2{#1}}}% 1.09b removes [0]
\def\XINT_ctcv_end #1.#2#3#4#5#6{ #6}%
```

10.18 \xintiCstoCv

```
\def\xintiCstoCv {\romannumeral0\xinticstocv }%
\def\xinticstocv #1%
{%
    \expandafter\XINT_icstcv_prep \romannumeral`&&@#1,!,%
}%
\def\XINT_icstcv_prep
{%
    \XINT_icstcv_loop_a {}1001%
}%
\def\XINT_icstcv_loop_a #1#2#3#4#5#6,%
{%
    \xint_gob_til_exclam #6\XINT_icstcv_end!%
    \expandafter
    \XINT_icstcv_loop_b \romannumeral`&&@#6.{#2}{#3}{#4}{#5}{#1}%
}%
\def\XINT_icstcv_loop_b #1.#2#3#4#5%
{%
    \expandafter\XINT_icstcv_loop_c\expandafter
    {\romannumeral0\xintiiadd {#5}{\XINT_mul_fork #1\xint:#3\xint:}}%
    {\romannumeral0\xintiiadd {#4}{\XINT_mul_fork #1\xint:#2\xint:}}%
    {{#2}{#3}}%
}%
\def\XINT_icstcv_loop_c #1#2%
{%
    \expandafter\XINT_icstcv_loop_d\expandafter {#2}{#1}%
}%
\def\XINT_icstcv_loop_d #1#2%
{%
    \expandafter\XINT_icstcv_loop_e\expandafter
    {\romannumeral0\xintrawwithzeros {#1/#2}}{{#1}{#2}}%
}%
\def\XINT_icstcv_loop_e #1#2#3#4{\XINT_icstcv_loop_a {#4{#1}}#2#3}%
\def\XINT_icstcv_end #1.#2#3#4#5#6{ #6}%  1.09b removes [0]
```

10.19 \xintGCtoCv

```
\def\xintGCtoCv {\romannumeral0\xintgctocv }%
\def\xintgctocv #1%
{%
    \expandafter\XINT_gctcv_prep \romannumeral`&&@#1+!/%
}%
\def\XINT_gctcv_prep
{%
```

```
    \XINT_gctcv_loop_a {}1001%
}%
\def\XINT_gctcv_loop_a #1#2#3#4#5#6+%
{%
    \expandafter\XINT_gctcv_loop_b
    \romannumeral0\xintrawwithzeros {#6}.{#2}{#3}{#4}{#5}{#1}%
}%
\def\XINT_gctcv_loop_b #1/#2.#3#4#5#6%
{%
    \expandafter\XINT_gctcv_loop_c\expandafter
    {\romannumeral0\XINT_mul_fork #2\xint:#4\xint:}%
    {\romannumeral0\XINT_mul_fork #2\xint:#3\xint:}%
    {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#6\xint:}%
                              {\XINT_mul_fork #1\xint:#4\xint:}}%
    {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#5\xint:}%
                              {\XINT_mul_fork #1\xint:#3\xint:}}%
}%
\def\XINT_gctcv_loop_c #1#2%
{%
    \expandafter\XINT_gctcv_loop_d\expandafter {\expandafter{#2}{#1}}%
}%
\def\XINT_gctcv_loop_d #1#2%
{%
    \expandafter\XINT_gctcv_loop_e\expandafter {\expandafter{#2}{#1}}%
}%
\def\XINT_gctcv_loop_e #1#2%
{%
    \expandafter\XINT_gctcv_loop_f\expandafter {#2}#1%
}%
\def\XINT_gctcv_loop_f #1#2%
{%
    \expandafter\XINT_gctcv_loop_g\expandafter
    {\romannumeral0\xintrawwithzeros {#1/#2}}{{#1}{#2}}%
}%
\def\XINT_gctcv_loop_g #1#2#3#4%
{%
    \XINT_gctcv_loop_h {#4{#1}}{#2#3}% 1.09b removes [0]
}%
\def\XINT_gctcv_loop_h #1#2#3/%
{%
    \xint_gob_til_exclam #3\XINT_gctcv_end!%
    \expandafter\XINT_gctcv_loop_i
    \romannumeral0\xintrawwithzeros {#3}.#2{#1}%
}%
\def\XINT_gctcv_loop_i #1/#2.#3#4#5#6%
{%
    \expandafter\XINT_gctcv_loop_j\expandafter
    {\romannumeral0\XINT_mul_fork #1\xint:#6\xint:}%
    {\romannumeral0\XINT_mul_fork #1\xint:#5\xint:}%
    {\romannumeral0\XINT_mul_fork #2\xint:#4\xint:}%
    {\romannumeral0\XINT_mul_fork #2\xint:#3\xint:}%
}%
```

```
\def\XINT_gctcv_loop_j #1#2%
{%
    \expandafter\XINT_gctcv_loop_k\expandafter {\expandafter{#2}{#1}}%
}%
\def\XINT_gctcv_loop_k #1#2%
{%
    \expandafter\XINT_gctcv_loop_l\expandafter {\expandafter{#2}#1}%
}%
\def\XINT_gctcv_loop_l #1#2%
{%
    \expandafter\XINT_gctcv_loop_m\expandafter {\expandafter{#2}#1}%
}%
\def\XINT_gctcv_loop_m #1#2{\XINT_gctcv_loop_a {#2}#1}%
\def\XINT_gctcv_end #1.#2#3#4#5#6{ #6}%
```

10.20 \xintiGCtoCv

```
\def\xintiGCtoCv {\romannumeral0\xintigctocv }%
\def\xintigctocv #1%
{%
    \expandafter\XINT_igctcv_prep \romannumeral`&&@#1+!/%
}%
\def\XINT_igctcv_prep
{%
    \XINT_igctcv_loop_a {}1001%
}%
\def\XINT_igctcv_loop_a #1#2#3#4#5#6+%
{%
    \expandafter\XINT_igctcv_loop_b
    \romannumeral`&&@#6.{#2}{#3}{#4}{#5}{#1}%
}%
\def\XINT_igctcv_loop_b #1.#2#3#4#5%
{%
    \expandafter\XINT_igctcv_loop_c\expandafter
    {\romannumeral0\xintiiadd {#5}{\XINT_mul_fork #1\xint:#3\xint:}}%
    {\romannumeral0\xintiiadd {#4}{\XINT_mul_fork #1\xint:#2\xint:}}%
    {{#2}{#3}}%
}%
\def\XINT_igctcv_loop_c #1#2%
{%
    \expandafter\XINT_igctcv_loop_f\expandafter {\expandafter{#2}{#1}}%
}%
\def\XINT_igctcv_loop_f #1#2#3#4/%
{%
    \xint_gob_til_exclam #4\XINT_igctcv_end_a!%
    \expandafter\XINT_igctcv_loop_g
    \romannumeral`&&@#4.#1#2{#3}%
}%
\def\XINT_igctcv_loop_g #1.#2#3#4#5%
{%
    \expandafter\XINT_igctcv_loop_h\expandafter
    {\romannumeral0\XINT_mul_fork #1\xint:#5\xint:}%
    {\romannumeral0\XINT_mul_fork #1\xint:#4\xint:}%
    {{#2}{#3}}%
}%
\def\XINT_igctcv_loop_h #1#2%
{%
    \expandafter\XINT_igctcv_loop_i\expandafter {\expandafter{#2}{#1}}%
}%
\def\XINT_igctcv_loop_i #1#2{\XINT_igctcv_loop_k #2{#2#1}}%
\def\XINT_igctcv_loop_k #1#2%
{%
    \expandafter\XINT_igctcv_loop_l\expandafter
    {\romannumeral0\xintrawwithzeros {#1/#2}}%
}%
\def\XINT_igctcv_loop_l #1#2#3{\XINT_igctcv_loop_a {#3{#1}}#2}%1.09i removes [0]
\def\XINT_igctcv_end_a #1.#2#3#4#5%
{%
    \expandafter\XINT_igctcv_end_b\expandafter
    {\romannumeral0\xintrawwithzeros {#2/#3}}%
}%
\def\XINT_igctcv_end_b #1#2{ #2{#1}}% 1.09b removes [0]
```

10.21 \xintFtoCv

Still uses \xinticstocv \xintFtoCs rather than \xintctocv \xintFtoC.

```
\def\xintFtoCv {\romannumeral0\xintftocv }%
\def\xintftocv #1%
{%
    \xinticstocv {\xintFtoCs {#1}}%
}%
```

10.22 \xintFtoCCv

```
\def\xintFtoCCv {\romannumeral0\xintftoccv }%
\def\xintftoccv #1%
{%
    \xintigctocv {\xintFtoCC {#1}}%
}%
```

10.23 \xintCntoF

Modified in 1.06 to give the N first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that.

```
\def\xintCntoF {\romannumeral0\xintcntof }%
\def\xintcntof #1%
{%
    \expandafter\XINT_cntf\expandafter {\the\numexpr #1}%
}%
\def\XINT_cntf #1#2%
{%
   \ifnum #1>\xint_c_
      \xint_afterfi {\expandafter\XINT_cntf_loop\expandafter
                     {\the\numexpr #1-1\expandafter}\expandafter
                     {\romannumeral`&&@#2{#1}}{#2}}%
   \else
```

```
      \xint_afterfi
         {\ifnum #1=\xint_c_
              \xint_afterfi {\expandafter\space \romannumeral`&&@#2{0}}%
          \else \xint_afterfi { }% 1.09m now returns nothing.
          \fi}%
   \fi
}%
\def\XINT_cntf_loop #1#2#3%
{%
    \ifnum #1>\xint_c_ \else \XINT_cntf_exit \fi
    \expandafter\XINT_cntf_loop\expandafter
    {\the\numexpr #1-1\expandafter }\expandafter
    {\romannumeral0\xintadd {\xintDiv {1[0]}{#2}}{#3{#1}}}%
    {#3}%
}%
\def\XINT_cntf_exit \fi
    \expandafter\XINT_cntf_loop\expandafter
    #1\expandafter #2#3%
{%
    \fi\xint_gobble_ii #2%
}%
```

10.24 \xintGCntoF

Modified in 1.06 to give the N argument first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that.

```
\def\xintGCntoF {\romannumeral0\xintgcntof }%
\def\xintgcntof #1%
{%
    \expandafter\XINT_gcntf\expandafter {\the\numexpr #1}%
}%
\def\XINT_gcntf #1#2#3%
{%
   \ifnum #1>\xint_c_
      \xint_afterfi {\expandafter\XINT_gcntf_loop\expandafter
                     {\the\numexpr #1-1\expandafter}\expandafter
                     {\romannumeral`&&@#2{#1}}{#2}{#3}}%
   \else
      \xint_afterfi
         {\ifnum #1=\xint_c_
              \xint_afterfi {\expandafter\space\romannumeral`&&@#2{0}}%
          \else \xint_afterfi { }% 1.09m now returns nothing rather than 0/1[0]
          \fi}%
   \fi
}%
\def\XINT_gcntf_loop #1#2#3#4%
{%
    \ifnum #1>\xint_c_ \else \XINT_gcntf_exit \fi
    \expandafter\XINT_gcntf_loop\expandafter
    {\the\numexpr #1-1\expandafter }\expandafter
    {\romannumeral0\xintadd {\xintDiv {#4{#1}}{#2}}{#3{#1}}}%
    {#3}{#4}%
}%
\def\XINT_gcntf_exit \fi
    \expandafter\XINT_gcntf_loop\expandafter
    #1\expandafter #2#3#4%
{%
    \fi\xint_gobble_ii #2%
}%
```

10.25 \xintCntoCs

Modified in 1.09m: added spaces after the commas in the produced list. Moreover the coefficients are not braced anymore. A slight induced limitation is that the macro argument should not contain some explicit comma (cf. \XINT_cntcs_exit_b), hence \xintCntoCs {\macro,} with \def\macro,#1{<stuff>} would crash. Not a very serious limitation, I believe.

```
\def\xintCntoCs {\romannumeral0\xintcntocs }%
\def\xintcntocs #1%
{%
    \expandafter\XINT_cntcs\expandafter {\the\numexpr #1}%
}%
\def\XINT_cntcs #1#2%
{%
   \ifnum #1<0
      \xint_afterfi { }% 1.09i: a 0/1[0] was here, now the macro returns nothing
   \else
      \xint_afterfi {\expandafter\XINT_cntcs_loop\expandafter
                     {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
                     {\romannumeral`&&@#2{#1}}{#2}}% produced coeff not braced
   \fi
}%
\def\XINT_cntcs_loop #1#2#3%
{%
    \ifnum #1>-\xint_c_i \else \XINT_cntcs_exit \fi
    \expandafter\XINT_cntcs_loop\expandafter
    {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
    {\romannumeral`&&@#3{#1}, #2}{#3}% space added, 1.09m
}%
\def\XINT_cntcs_exit \fi
    \expandafter\XINT_cntcs_loop\expandafter
    #1\expandafter #2#3%
{%
    \fi\XINT_cntcs_exit_b #2%
}%
\def\XINT_cntcs_exit_b #1,{}% romannumeral stopping space already there
```

10.26 \xintCntoGC

Modified in 1.06 to give the N first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that.

1.09m maintains the braces, as the coeff are allowed to be fraction and the slash can not be naked in the GC format, contrarily to what happens in \xintCntoCs. Also the separators given to \xintGCtoGCx may then fetch the coefficients as argument, as they are braced.

```
\def\xintCntoGC {\romannumeral0\xintcntogc }%
\def\xintcntogc #1%
{%
    \expandafter\XINT_cntgc\expandafter {\the\numexpr #1}%
}%
\def\XINT_cntgc #1#2%
{%
   \ifnum #1<0
      \xint_afterfi { }% 1.09i there was as strange 0/1[0] here, removed
   \else
      \xint_afterfi {\expandafter\XINT_cntgc_loop\expandafter
                     {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
                     {\expandafter{\romannumeral`&&@#2{#1}}}{#2}}%
   \fi
}%
\def\XINT_cntgc_loop #1#2#3%
{%
    \ifnum #1>-\xint_c_i \else \XINT_cntgc_exit \fi
    \expandafter\XINT_cntgc_loop\expandafter
    {\the\numexpr #1-\xint_c_i\expandafter }\expandafter
    {\expandafter{\romannumeral`&&@#3{#1}}+1/#2}{#3}%
}%
\def\XINT_cntgc_exit \fi
    \expandafter\XINT_cntgc_loop\expandafter
    #1\expandafter #2#3%
{%
    \fi\XINT_cntgc_exit_b #2%
}%
\def\XINT_cntgc_exit_b #1+1/{ }%
```

10.27 \xintGCntoGC

Modified in 1.06 to give the N first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that.

```
\def\xintGCntoGC {\romannumeral0\xintgcntogc }%
\def\xintgcntogc #1%
{%
    \expandafter\XINT_gcntgc\expandafter {\the\numexpr #1}%
}%
\def\XINT_gcntgc #1#2#3%
{%
   \ifnum #1<0
      \xint_afterfi { }% 1.09i now returns nothing
   \else
      \xint_afterfi {\expandafter\XINT_gcntgc_loop\expandafter
                     {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
                     {\expandafter{\romannumeral`&&@#2{#1}}}{#2}{#3}}%
   \fi
}%
\def\XINT_gcntgc_loop #1#2#3#4%
{%
    \ifnum #1>-\xint_c_i \else \XINT_gcntgc_exit \fi
    \expandafter\XINT_gcntgc_loop_b\expandafter
    {\expandafter{\romannumeral`&&@#4{#1}}/#2}{#3{#1}}{#1}{#3}{#4}%
}%
\def\XINT_gcntgc_loop_b #1#2#3%
{%
    \expandafter\XINT_gcntgc_loop\expandafter
    {\the\numexpr #3-\xint_c_i \expandafter}\expandafter
    {\expandafter{\romannumeral`&&@#2}+#1}%
}%
\def\XINT_gcntgc_exit \fi
    \expandafter\XINT_gcntgc_loop_b\expandafter #1#2#3#4#5%
{%
    \fi\XINT_gcntgc_exit_b #1%
}%
\def\XINT_gcntgc_exit_b #1/{ }%
```

10.28 \xintCstoGC

```
\def\xintCstoGC {\romannumeral0\xintcstogc }%
\def\xintcstogc #1%
{%
    \expandafter\XINT_cstc_prep \romannumeral`&&@#1,!,%
}%
\def\XINT_cstc_prep #1,{\XINT_cstc_loop_a {{#1}}}%
\def\XINT_cstc_loop_a #1#2,%
{%
    \xint_gob_til_exclam #2\XINT_cstc_end!%
    \XINT_cstc_loop_b {#1}{#2}%
}%
\def\XINT_cstc_loop_b #1#2{\XINT_cstc_loop_a {#1+1/{#2}}}%
\def\XINT_cstc_end!\XINT_cstc_loop_b #1#2{ #1}%
```

10.29 \xintGCtoGC

```
\def\xintGCtoGC {\romannumeral0\xintgctogc }%
\def\xintgctogc #1%
{%
    \expandafter\XINT_gctgc_start \romannumeral`&&@#1+!/%
}%
\def\XINT_gctgc_start {\XINT_gctgc_loop_a {}}%
\def\XINT_gctgc_loop_a #1#2+#3/%
{%
    \xint_gob_til_exclam #3\XINT_gctgc_end!%
    \expandafter\XINT_gctgc_loop_b\expandafter
    {\romannumeral`&&@#2}{#3}{#1}%
}%
\def\XINT_gctgc_loop_b #1#2%
{%
    \expandafter\XINT_gctgc_loop_c\expandafter
    {\romannumeral`&&@#2}{#1}%
}%
\def\XINT_gctgc_loop_c #1#2#3%
{%
    \XINT_gctgc_loop_a {#3{#2}+{#1}/}%
}%
\def\XINT_gctgc_end!\expandafter\XINT_gctgc_loop_b
{%
    \expandafter\XINT_gctgc_end_b
}%
\def\XINT_gctgc_end_b #1#2#3{ #3{#1}}%
\XINT_restorecatcodes_endinput%
```

11 Package xintexpr implementation

Contents

This is release 1.3c of [2018/06/17].

11.1 Old comments

These general comments were last updated at the end of the 1.09x series in 2014. The principles remain in place to this day but refer to CHANGES.html for some significant evolutions since.

The first version was released in June 2013. I was greatly helped in this task of writing an expandable parser of infix operations by the comments provided in l3fp-parse.dtx (in its version as available in April-May 2013). One will recognize in particular the idea of the \`until' macros; I have not looked into the actual l3fp code beyond the very useful comments provided in its documentation.

A main worry was that my data has no a priori bound on its size; to keep the code reasonably efficient, I experimented with a technique of storing and retrieving data expandably as *names* of control sequences. Intermediate computation results are stored as control sequences \.=a/b[n].

Roughly speaking, the parser mechanism is as follows: at any given time the last found \`\`operator'' has its associated until macro awaiting some news from the token flow; first getnext expands forward in the hope to construct some number, which may come from a parenthesized sub-expression, from some braced material, or from a digit by digit scan. After this number has been formed the next operator is looked for by the getop macro. Once getop has finished its job, until is presented with three tokens: the first one is the precedence level of the new found operator (which may be an end of expression marker), the second is the operator character token (earlier versions had here already some macro name, but in order to keep as much common code to expr and floatexpr common as possible, this was modified) of the new found operator, and the third one is the newly found number (which was encountered just before the new operator).

The until macro of the earlier operator examines the precedence level of the new found one, and either executes the earlier operator (in the case of a binary operation, with the found number and a previously stored one) or it delays execution, giving the hand to the until macro of the operator having been found of higher precedence.

A minus sign acting as prefix gets converted into a (unary) operator inheriting the precedence level of the previous operator.

Once the end of the expression is found (it has to be marked by a \relax) the final result is output as four tokens (five tokens since 1.09j) the first one a catcode 11 exclamation mark, the second one an error generating macro, the third one is a protection mechanism, the fourth one a printing macro and the fifth is \.=a/b[n]. The prefix \xintthe makes the output printable by killing the first three tokens.

11.2 Catcodes, ε-TEX and reload detection

The code for reload detection was initially copied from Heiko Oberdiek's packages, then modified.

The method for catcodes was also initially directly inspired by these packages.

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode64=11   % @
  \catcode35=6    % #
  \catcode44=12   % ,
  \catcode45=12   % -
  \catcode46=12   % .
  \catcode58=12   % :
  \def\z {\endgroup}%
  \expandafter\let\expandafter\x\csname ver@xintexpr.sty\endcsname
  \expandafter\let\expandafter\w\csname ver@xintfrac.sty\endcsname
  \expandafter\let\expandafter\t\csname ver@xinttools.sty\endcsname
  \expandafter
    \ifx\csname PackageInfo\endcsname\relax
      \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
    \else
      \def\y#1#2{\PackageInfo{#1}{#2}}%
    \fi
  \expandafter
  \ifx\csname numexpr\endcsname\relax
     \y{xintexpr}{\numexpr not available, aborting input}%
     \aftergroup\endinput
  \else
    \ifx\x\relax   % plain-TeX, first loading of xintexpr.sty
      \ifx\w\relax % but xintfrac.sty not yet loaded.
         \expandafter\def\expandafter\z\expandafter
                    {\z\input xintfrac.sty\relax}%
      \fi
      \ifx\t\relax % but xinttools.sty not yet loaded.
         \expandafter\def\expandafter\z\expandafter
                    {\z\input xinttools.sty\relax}%
      \fi
    \else
      \def\empty {}%
      \ifx\x\empty % LaTeX, first loading,
      % variable is initialized, but \ProvidesPackage not yet seen
          \ifx\w\relax % xintfrac.sty not yet loaded.
            \expandafter\def\expandafter\z\expandafter
                               {\z\RequirePackage{xintfrac}}%
          \fi
          \ifx\t\relax % xinttools.sty not yet loaded.
            \expandafter\def\expandafter\z\expandafter
                               {\z\RequirePackage{xinttools}}%
          \fi
      \else
        \aftergroup\endinput % xintexpr already loaded.
      \fi
    \fi
  \fi
\z%
\XINTsetupcatcodes%
```

11.3 Package identification

```
\XINT_providespackage
\ProvidesPackage{xintexpr}%
  [2018/06/17 1.3c Expandable expression parser (JFB)]%
\catcode`! 11
\let\XINT_Cmp \xintiiCmp
```

11.4 Locking and unlocking

Some renaming and modifications here with release 1.2 to switch from using chains of \romannumeral-`0 in order to gather numbers, possibly hexadecimals, to using a \csname governed expansion. In this way no more limit at 5000 digits, and besides this is a logical move because the \xintexpr parser is already based on \csname...\endcsname storage of numbers as one token.

The limitation at 5000 digits didn't worry me too much because it was not very realistic to launch computations with thousands of digits... such computations are still slow with 1.2 but less so now. Chains or \romannumeral are still used for the gathering of function names and other stuff which I have half-forgotten because the parser does many things.

In the earlier versions we used the lockscan macro after a chain of \romannumeral-`0 had ended gathering digits; this uses has been replaced by direct processing inside a \csname...\endcsname and the macro is kept only for matters of dummy variables.

Currently, the parsing of hexadecimal numbers needs two nested \csname...\endcsname, first to gather the letters (possibly with a hexadecimal fractional part), and in a second stage to apply \xintHexToDec to do the actual conversion. This should be faster than updating on the fly the number (which would be hard for the fraction part...).

```
\def\xint_gob_til_! #1!{}% ! with catcode 11
\def\XINT_expr_lockscan#1{% not used for decimal numbers in xintexpr 1.2
\def\XINT_expr_lockscan##1!{\expandafter#1\csname .=##1\endcsname}%
}\XINT_expr_lockscan{ }%
\def\XINT_expr_lockit#1{%
\def\XINT_expr_lockit##1{\expandafter#1\csname .=##1\endcsname}%
}\XINT_expr_lockit{ }%
\def\XINT_expr_unlock_hex_in #1%  expanded inside \csname..\endcsname
   {\expandafter\XINT_expr_inhex\romannumeral`&&@\XINT_expr_unlock#1;}%
\def\XINT_expr_inhex #1.#2#3;%    expanded inside \csname..\endcsname
{%
    \if#2>%
      \xintHexToDec{#1}%
    \else
      \xintiiMul{\xintiiPow{625}{\xintLength{#3}}}{\xintHexToDec{#1#3}}%
      [\the\numexpr-4*\xintLength{#3}]%
    \fi
}%
\def\XINT_expr_unlock  {\expandafter\XINT_expr_unlock_a\string }%
\def\XINT_expr_unlock_a #1.={}%
\def\XINT_expr_unexpectedtoken {\xintError:ignored }%
```

```
\let\XINT_expr_done\space
```

11.5 \XINT_expr_wrap, \XINT_iiexpr_wrap

```
\def\XINT_expr_wrap   { !\XINT_expr_usethe\XINT_protectii\XINT_expr_print }%
\def\XINT_iiexpr_wrap { !\XINT_expr_usethe\XINT_protectii\XINT_iiexpr_print }%
```

11.6 \XINT_protectii, \XINT_expr_usethe

```
\def\XINT_protectii #1{\noexpand\XINT_protectii\noexpand #1\noexpand }%
\protected\def\XINT_expr_usethe\XINT_protectii {\xintError:missing_xintthe!}%
```

11.7 \XINT_expr_print, \XINT_iiexpr_print, \XINT_boolexpr_print

See also the \XINT_flexpr_print which is special, below.

```
\def\XINT_expr_print     #1{\xintSPRaw::csv  {\XINT_expr_unlock #1}}%
\def\XINT_iiexpr_print   #1{\xintCSV::csv    {\XINT_expr_unlock #1}}%
\def\XINT_boolexpr_print #1{\xintIsTrue::csv {\XINT_expr_unlock #1}}%
```

11.8 \xintexpr, \xintiexpr, \xintfloatexpr, \xintiiexpr

```
\def\xintexpr       {\romannumeral0\xinteval      }%
\def\xintiexpr      {\romannumeral0\xintieval     }%
\def\xintfloatexpr  {\romannumeral0\xintfloateval }%
\def\xintiiexpr     {\romannumeral0\xintiieval    }%
```

11.9 \xinttheexpr, \xinttheiexpr, \xintthefloatexpr, \xinttheiiexpr

```
\def\xinttheexpr
   {\romannumeral`&&@\expandafter\XINT_expr_print\romannumeral0\xintbareeval  }%
\def\xinttheiexpr     {\romannumeral`&&@\xintthe\xintiexpr }%
\def\xintthefloatexpr {\romannumeral`&&@\xintthe\xintfloatexpr }%
\def\xinttheiiexpr
   {\romannumeral`&&@\expandafter\XINT_iiexpr_print\romannumeral0\xintbareiieval }%
```

11.10 \xintthe

```
\def\xintthe #1{\romannumeral`&&@\expandafter\xint_gobble_iii\romannumeral`&&@#1}%
```

11.11 \thexintexpr, \thexintiexpr, \thexintfloatexpr, \thexintiiexpr

New with 1.2h. I have been three years long very strict in terms of prefixing macros, but well.

```
\let\thexintexpr     \xinttheexpr
\let\thexintiexpr    \xinttheiexpr
\let\thexintfloatexpr\xintthefloatexpr
\let\thexintiiexpr   \xinttheiiexpr
```

11.12 \xintthecoords

1.1 Wraps up an even number of comma separated items into pairs of TikZ coordinates; for use in the following way:

  coordinates {\xintthecoords\xintfloatexpr ... \relax}

  The crazyness with the \csname and unlock is due to TikZ somewhat STRANGE control of the TOTAL number of expansions which should not exceed the very low value of 100 !! As we implemented \XINT_thecoords_b in an "inline" style for efficiency, we need to hide its expansions.

Not to be used as \xintthecoords\xintthefloatexpr, only as \xintthecoords\xintfloatexpr (or \xintiexpr etc...). Perhaps \xintthecoords could make an extra check, but one should not accustom users to too loose requirements!

```
\def\xintthecoords  #1{\romannumeral`&&@\expandafter\expandafter\expandafter
                     \XINT_thecoords_a
                     \expandafter\xint_gobble_iii\romannumeral0#1}%
\def\XINT_thecoords_a #1#2% #1=print macro, indispensible for scientific notation
   {\expandafter\XINT_expr_unlock\csname.=\expandafter\XINT_thecoords_b
                         \romannumeral`&&@#1#2,!,!,^\endcsname }%
\def\XINT_thecoords_b #1#2,#3#4,%
   {\xint_gob_til_! #3\XINT_thecoords_c ! (#1#2, #3#4)\XINT_thecoords_b }%
\def\XINT_thecoords_c #1^{}%
```

11.13 \xintbareeval, \xintbarefloateval, \xintbareiieval

```
\def\xintbareeval
   {\expandafter\XINT_expr_until_end_a\romannumeral`&&@\XINT_expr_getnext }%
\def\xintbarefloateval
   {\expandafter\XINT_flexpr_until_end_a\romannumeral`&&@\XINT_expr_getnext }%
\def\xintbareiieval
   {\expandafter\XINT_iiexpr_until_end_a\romannumeral`&&@\XINT_expr_getnext }%
```

11.14 \xintthebareeval, \xintthebarefloateval, \xintthebareiieval

```
\def\xintthebareeval      {\expandafter\XINT_expr_unlock\romannumeral0\xintbareeval}%
\def\xintthebarefloateval {\expandafter\XINT_expr_unlock\romannumeral0\xintbarefloateval}%
\def\xintthebareiieval    {\expandafter\XINT_expr_unlock\romannumeral0\xintbareiieval}%
```

11.15 \xinteval, \xintiieval

```
\def\xinteval   {\expandafter\XINT_expr_wrap\romannumeral0\xintbareeval }%
\def\xintiieval {\expandafter\XINT_iiexpr_wrap\romannumeral0\xintbareiieval }%
```

11.16 \xintieval, \XINT_iexpr_wrap

Optional argument since 1.1.

```
\def\xintieval #1%
   {\ifx [#1\expandafter\XINT_iexpr_withopt\else\expandafter\XINT_iexpr_noopt \fi #1}%
\def\XINT_iexpr_noopt
   {\expandafter\XINT_iexpr_wrap \expandafter 0\romannumeral0\xintbareeval }%
\def\XINT_iexpr_withopt [#1]%
{%
    \expandafter\XINT_iexpr_wrap\expandafter
    {\the\numexpr \xint_zapspaces #1 \xint_gobble_i\expandafter}%
    \romannumeral0\xintbareeval
}%
\def\XINT_iexpr_wrap #1#2%
{%
    \expandafter\XINT_expr_wrap
    \csname .=\xintRound::csv {#1}{\XINT_expr_unlock #2}\endcsname
}%
```

11.17 `\xintfloateval`, `\XINT_flexpr_wrap`, `\XINT_flexpr_print`

Optional argument since 1.1

```
\def\xintfloateval #1%
{%
    \ifx [#1\expandafter\XINT_flexpr_withopt_a\else\expandafter\XINT_flexpr_noopt
    \fi #1%
}%
\def\XINT_flexpr_noopt
{%
   \expandafter\XINT_flexpr_withopt_b\expandafter\xinttheDigits
   \romannumeral0\xintbarefloateval
}%
\def\XINT_flexpr_withopt_a [#1]%
{%
   \expandafter\XINT_flexpr_withopt_b\expandafter
    {\the\numexpr\xint_zapspaces #1 \xint_gobble_i\expandafter}%
    \romannumeral0\xintbarefloateval
}%
\def\XINT_flexpr_withopt_b #1#2%
{%
    \expandafter\XINT_flexpr_wrap\csname .;#1.=% ; and not : as before b'cause NewExpr
    \XINTinFloat::csv {#1}{\XINT_expr_unlock #2}\endcsname
}%
\def\XINT_flexpr_wrap { !\XINT_expr_usethe\XINT_protectii\XINT_flexpr_print }%
\def\XINT_flexpr_print #1%
{%
    \expandafter\xintPFloat::csv
    \romannumeral`&&@\expandafter\XINT_expr_unlock_sp\string #1!%
}%
\def\XINT_expr_unlock_sp #1.;#2.=#3!{{#2}{#3}}%
```

11.18 `\xintboolexpr`, `\xinttheboolexpr`, `\thexintboolexpr`

```
\def\xintboolexpr      {\romannumeral0\expandafter\expandafter\expandafter
    \XINT_boolexpr_done \expandafter\xint_gobble_iv\romannumeral0\xinteval }%
\def\xinttheboolexpr   {\romannumeral`&&@\expandafter\expandafter\expandafter
    \XINT_boolexpr_print\expandafter\xint_gobble_iv\romannumeral0\xinteval }%
\let\thexintboolexpr\xinttheboolexpr
\def\XINT_boolexpr_done { !\XINT_expr_usethe\XINT_protectii\XINT_boolexpr_print }%
```

11.19 `\xintifboolexpr`, `\xintifboolfloatexpr`, `\xintifbooliiexpr`

Do not work with comma separated expressions.

```
\def\xintifboolexpr      #1{\romannumeral0\xintifnotzero {\xinttheexpr #1\relax}}%
\def\xintifboolfloatexpr #1{\romannumeral0\xintifnotzero {\xintthefloatexpr #1\relax}}%
\def\xintifbooliiexpr    #1{\romannumeral0\xintifnotzero {\xinttheiiexpr #1\relax}}%
```

11.20 Hooks for the functioning of `\xintNewExpr` and `\xintdeffunc`

This is new with 1.3. See `\XINT_expr_redefinemacros`.

```
\let\XINT:NEhook:two\empty
\let\XINT:NEhook:one\empty
\let\XINT:NEhook:csv\empty
```

11.21 Macros handling csv lists on output (for \XINT_expr_print et al. routines)

Changed completely for 1.1, which adds the optional arguments to \xintiexpr and \xintfloatexpr.

11.21.1 \XINT_::_end

Le mécanisme est le suivant, #2 est dans des accolades et commence par ,<sp>. Donc le gobble se débarrasse du, et le <sp> après brace stripping arrête un \romannumeral0 ou \romannumeral-`0

```
\def\XINT_::_end #1,#2{\xint_gobble_i #2}%
```

11.21.2 \xintCSV::csv

```
\def\xintCSV::csv #1{\expandafter\XINT_csv::_a\romannumeral`&&@#1,^,}%
\def\XINT_csv::_a {\XINT_csv::_b {}}%
\def\XINT_csv::_b #1#2,{\expandafter\XINT_csv::_c \romannumeral`&&@#2,{#1}}%
\def\XINT_csv::_c #1{\if ^#1\expandafter\XINT_::_end\fi\XINT_csv::_d #1}%
\def\XINT_csv::_d #1,#2{\XINT_csv::_b {#2, #1}}% possibly, item #1 is empty.
```

11.21.3 \xintSPRaw, \xintSPRaw::csv

```
\def\xintSPRaw    {\romannumeral0\xintspraw }%
\def\xintspraw  #1{\expandafter\XINT_spraw\romannumeral`&&@#1[\W]}%
\def\XINT_spraw #1[#2#3]{\xint_gob_til_W #2\XINT_spraw_a\W\XINT_spraw_p #1[#2#3]}%
\def\XINT_spraw_a\W\XINT_spraw_p #1[\W]{ #1}%
\def\XINT_spraw_p #1[\W]{\xintpraw {#1}}%
\def\xintSPRaw::csv #1{\romannumeral0\expandafter\XINT_spraw::_a\romannumeral`&&@#1,^,}%
\def\XINT_spraw::_a {\XINT_spraw::_b {}}%
\def\XINT_spraw::_b #1#2,{\expandafter\XINT_spraw::_c \romannumeral`&&@#2,{#1}}%
\def\XINT_spraw::_c #1{\if ,#1\xint_dothis\XINT_spraw::_e\fi
                       \if ^#1\xint_dothis\XINT_::_end\fi
                       \xint_orthat\XINT_spraw::_d #1}%
\def\XINT_spraw::_d #1,{\expandafter\XINT_spraw::_e\romannumeral0\XINT_spraw #1[\W],}%
\def\XINT_spraw::_e #1,#2{\XINT_spraw::_b {#2, #1}}%
```

11.21.4 \xintIsTrue::csv

```
\def\xintIsTrue::csv #1{\romannumeral0\expandafter\XINT_istrue::_a\romannumeral`&&@#1,^,}%
\def\XINT_istrue::_a {\XINT_istrue::_b {}}%
\def\XINT_istrue::_b #1#2,{\expandafter\XINT_istrue::_c \romannumeral`&&@#2,{#1}}%
\def\XINT_istrue::_c #1{\if ,#1\xint_dothis\XINT_istrue::_e\fi
                        \if ^#1\xint_dothis\XINT_::_end\fi
                        \xint_orthat\XINT_istrue::_d #1}%
```

```
\def\XINT_istrue::_d #1,{\expandafter\XINT_istrue::_e\romannumeral0\xintisnotzero {#1},}%
\def\XINT_istrue::_e #1,#2{\XINT_istrue::_b {#2, #1}}%
```

11.21.5 \xintRound::csv

```
\def\XINT_:::_end #1,#2#3{\xint_gobble_i #3}%
\def\xintRound::csv #1#2{\romannumeral0\expandafter\XINT_round::_b\expandafter
    {\the\numexpr#1\expandafter}\expandafter{\expandafter}\romannumeral`&&@#2,^,}%
\def\XINT_round::_b #1#2#3,{\expandafter\XINT_round::_c \romannumeral`&&@#3,{#1}{#2}}%
\def\XINT_round::_c #1{\if ,#1\xint_dothis\XINT_round::_e\fi
                       \if ^#1\xint_dothis\XINT_:::_end\fi
                       \xint_orthat\XINT_round::_d #1}%
\def\XINT_round::_d #1,#2{%
      \expandafter\XINT_round::_e\romannumeral0\ifnum#2>\xint_c_
      \expandafter\xintround\else\expandafter\xintiround\fi {#2}{#1},{#2}}%
\def\XINT_round::_e #1,#2#3{\XINT_round::_b {#2}{#3, #1}}%
```

11.21.6 \XINTinFloat::csv

```
\def\XINTinFloat::csv #1#2{\romannumeral0\expandafter\XINT_infloat::_b\expandafter
   {\the\numexpr #1\expandafter}\expandafter{\expandafter}\romannumeral`&&@#2,^,}%
\def\XINT_infloat::_b #1#2#3,{\XINT_infloat::_c #3,{#1}{#2}}%
\def\XINT_infloat::_c #1{\if ,#1\xint_dothis\XINT_infloat::_e\fi
                       \if ^#1\xint_dothis\XINT_:::_end\fi
                       \xint_orthat\XINT_infloat::_d #1}%
\def\XINT_infloat::_d #1,#2%
        {\expandafter\XINT_infloat::_e\romannumeral0\XINTinfloat [#2]{#1},{#2}}%
\def\XINT_infloat::_e #1,#2#3{\XINT_infloat::_b {#2}{#3, #1}}%
```

11.21.7 \xintPFloat::csv

```
\def\xintPFloat::csv #1#2{\romannumeral0\expandafter\XINT_pfloat::_b\expandafter
   {\the\numexpr #1\expandafter}\expandafter{\expandafter}\romannumeral`&&@#2,^,}%
\def\XINT_pfloat::_b #1#2#3,{\expandafter\XINT_pfloat::_c \romannumeral`&&@#3,{#1}{#2}}%
\def\XINT_pfloat::_c #1{\if ,#1\xint_dothis\XINT_pfloat::_e\fi
                       \if ^#1\xint_dothis\XINT_:::_end\fi
                       \xint_orthat\XINT_pfloat::_d #1}%
\def\XINT_pfloat::_d #1,#2%
 {\expandafter\XINT_pfloat::_e\romannumeral0\XINT_pfloat_opt [\xint:#2]{#1},{#2}}%
\def\XINT_pfloat::_e #1,#2#3{\XINT_pfloat::_b {#2}{#3, #1}}%
```

11.22 \XINT_expr_getnext: fetching some number then an operator

Big change in 1.1, no attempt to detect braced stuff anymore as the [N] notation is implemented otherwise. Now, braces should not be used at all; one level removed, then \romannumeral-`0 expansion.

```
\def\XINT_expr_getnext #1%
{%
    \expandafter\XINT_expr_getnext_a\romannumeral`&&@#1%
}%
\def\XINT_expr_getnext_a #1%
{% screens out sub-expressions and \count or \dimen registers/variables
    \xint_gob_til_! #1\XINT_expr_subexpr !% recall this ! has catcode 11
    \ifcat\relax#1% \count or \numexpr etc... token or count, dimen, skip cs
       \expandafter\XINT_expr_countetc
    \else
```

```
        \expandafter\expandafter\expandafter\XINT_expr_getnextfork\expandafter\string
    \fi
    #1%
}%
\def\XINT_expr_subexpr !#1\fi !{\expandafter\XINT_expr_getop\xint_gobble_iii }%
```

1.2 adds \ht, \dp, \wd and the eTeX font things.

```
\def\XINT_expr_countetc #1%
{%
    \ifx\count#1\else\ifx\dimen#1\else\ifx\numexpr#1\else\ifx\dimexpr#1\else
    \ifx\skip#1\else\ifx\glueexpr#1\else\ifx\fontdimen#1\else\ifx\ht#1\else
    \ifx\dp#1\else\ifx\wd#1\else\ifx\fontcharht#1\else\ifx\fontcharwd#1\else
    \ifx\fontchardp#1\else\ifx\fontcharic#1\else
      \XINT_expr_unpackvar
    \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
    \expandafter\XINT_expr_getnext\number #1%
}%
\def\XINT_expr_unpackvar\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
    \expandafter\XINT_expr_getnext\number #1%
    {\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
     \expandafter\XINT_expr_getop\csname .=\number#1\endcsname }%
\begingroup
\lccode`*=`#
\lowercase{\endgroup
\def\XINT_expr_getnextfork #1{%
    \if#1*\xint_dothis {\XINT_expr_scan_macropar *}\fi
    \if#1[\xint_dothis {\xint_c_xviii ({}}\fi
    \if#1+\xint_dothis \XINT_expr_getnext \fi
    \if#1.\xint_dothis {\XINT_expr_startdec}\fi
    \if#1-\xint_dothis -\fi
    \if#1(\xint_dothis {\xint_c_xviii ({}}\fi
    \xint_orthat {\XINT_expr_scan_nbr_or_func #1}%
}}%
\def\XINT_expr_scan_macropar #1#2{\expandafter\XINT_expr_getop\csname .=#1#2\endcsname }%
```

11.23 `\XINT_expr_scan_nbr_or_func`: the integer or decimal number or hexa-decimal number or function name or variable name or special hacky things big parser

1.2 release has replaced chains of \romannumeral-`0 by \csname governed expansion. Thus there is no more the limit at about 5000 digits for parsed numbers.

In order to avoid having to lock and unlock in succession to handle the scientific part and adjust the exponent according to the number of digits of the decimal part, the parsing of this decimal part counts on the fly the number of digits it encounters.

There is some slight annoyance with \xintiiexpr which should never be given a [n] inside its \csname.=<digits>\endcsname storage of numbers (because its arithmetic uses the ii macros which know nothing about the [N] notation). Hence if the parser has only seen digits when hitting something else than the dot or e (or E), it will not insert a [0]. Thus we very slightly compromise the efficiency of \xintexpr and \xintfloatexpr in order to be able to share the same code with \xintiiexpr.

Indeed, the parser at this location is completely common to all, it does not know if it is working inside \xintexpr or \xintiiexpr. On the other hand if a dot or a e (or E) is met, then the (common) parser has no scrupules ending this number with a [n], this will provoke an error later if that was within an \xintiiexpr, as soon as an arithmetic macro is used.

As the gathered numbers have no spaces, no pluses, no minuses, the only remaining issue is with leading zeroes, which are discarded on the fly. The hexadecimal numbers leading zeroes are stripped in a second stage by the \xintHexToDec macro.

With 1.2, \xinttheexpr . \relax does not work anymore (it did in earlier releases). There must be digits either before or after the decimal mark. Thus both \xinttheexpr 1.\relax and \xinttheexpr .1\relax are legal.

The ` syntax is here used for special constructs like `+`(..), `*`(..) where + or * will be treated as functions. Current implementation pick only one token (could have been braced stuff), thus here it will be + or *, and via \XINT_expr_op_` this into becomes a suitable \XINT_{expr|iiexpr|flexpr}_func_+ (or *). Documentation of 1.1 said to use `+`(...), but `+(...) is also valid. The opening parenthesis must be there, it is not allowed to come from expansion.

```
\catcode96 11 % `
\def\XINT_expr_scan_nbr_or_func #1% this #1 has necessarily here catcode 12
{%(
    \if )#1\xint_dothis \XINT_expr_gotnil \fi
    \if "#1\xint_dothis \XINT_expr_scanhex_I\fi
    \if `#1\xint_dothis {\XINT_expr_onliteral_`}\fi
    \ifnum \xint_c_ix<1#1 \xint_dothis \XINT_expr_startint\fi
    \xint_orthat \XINT_expr_scanfunc #1%
}%
\def\XINT_expr_gotnil{\expandafter\XINT_expr_getop\csname.= \endcsname}%
\def\XINT_expr_onliteral_` #1#2#3({\xint_c_xviii `{#2}}%
\catcode96 12 % `
\def\XINT_expr_startint #1%
{%
    \if #10\expandafter\XINT_expr_gobz_a\else\XINT_expr_scanint_a\fi #1%
}%
\def\XINT_expr_scanint_a #1#2%
    {\expandafter\XINT_expr_getop\csname.=#1%
     \expandafter\XINT_expr_scanint_b\romannumeral`&&@#2}%
\def\XINT_expr_gobz_a #1%
    {\expandafter\XINT_expr_getop\csname.=%
     \expandafter\XINT_expr_gobz_scanint_b\romannumeral`&&@#1}%
\def\XINT_expr_startdec #1%
    {\expandafter\XINT_expr_getop\csname.=%
     \expandafter\XINT_expr_scandec_a\romannumeral`&&@#1}%
```

11.23.1 Integral part (skipping zeroes)

1.2 has modified the code to give highest priority to digits, the accelerating impact is non-negligeable. I don't think the doubled \string is a serious penalty.

```
\def\XINT_expr_scanint_b #1%
{%
```

```
    \ifcat \relax #1\expandafter\XINT_expr_scanint_endbycs\expandafter #1\fi
    \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanint_c\fi
    \string#1\XINT_expr_scanint_d
}%
\def\XINT_expr_scanint_d #1%
{%
    \expandafter\XINT_expr_scanint_b\romannumeral`&&@#1%
}%
\def\XINT_expr_scanint_endbycs#1#2\XINT_expr_scanint_d{\endcsname #1}%
```

With 1.2d the tacit multiplication in front of a variable name or function name is now done with a higher precedence, intermediate between the common one of * and / and the one of ^. Thus x/2y is like x/(2y), but x^2y is like x^2*y and 2y! is not (2y)! but 2*y!.

Finally, 1.2d has moved away from the _scan macros all the business of the tacit multiplication in one unique place via \XINT_expr_getop. For this, the ending token is not first given to \string as was done earlier before handing over back control to \XINT_expr_getop. Earlier we had to identify the catcode 11 ! signaling a sub-expression here. With no \string applied we can do it in \XINT_expr_getop. As a corollary of this displacement, parsing of big numbers should be a tiny bit faster now.

Extended for 1.2l to ignore underscore character _ if encountered within digits; so it can serve as separator for better readability.

```
\def\XINT_expr_scanint_c\string #1\XINT_expr_scanint_d
{%
    \if    _#1\xint_dothis\XINT_expr_scanint_d\fi
    \if    e#1\xint_dothis{[\the\numexpr0\XINT_expr_scanexp_a +}\fi
    \if    E#1\xint_dothis{[\the\numexpr0\XINT_expr_scanexp_a +}\fi
    \if    .#1\xint_dothis{\XINT_expr_startdec_a .}\fi
    \xint_orthat {\endcsname #1}%
}%
\def\XINT_expr_startdec_a .#1%
{%
    \expandafter\XINT_expr_scandec_a\romannumeral`&&@#1%
}%
\def\XINT_expr_scandec_a #1%
{%
    \if .#1\xint_dothis{\endcsname..}\fi
    \xint_orthat {\XINT_expr_scandec_b 0.#1}%
}%
\def\XINT_expr_gobz_scanint_b #1%
{%
    \ifcat \relax #1\expandafter\XINT_expr_gobz_scanint_endbycs\expandafter #1\fi
    \ifnum\xint_c_x<1\string#1 \else\expandafter\XINT_expr_gobz_scanint_c\fi
    \string#1\XINT_expr_scanint_d
}%
\def\XINT_expr_gobz_scanint_endbycs#1#2\XINT_expr_scanint_d{0\endcsname #1}%
\def\XINT_expr_gobz_scanint_c\string #1\XINT_expr_scanint_d
{%
    \if    _#1\xint_dothis\XINT_expr_gobz_scanint_d\fi
    \if    e#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
    \if    E#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
    \if    .#1\xint_dothis{\XINT_expr_gobz_startdec_a .}\fi
    \if    0#1\xint_dothis\XINT_expr_gobz_scanint_d\fi
    \xint_orthat {0\endcsname #1}%
}%
\def\XINT_expr_gobz_scanint_d #1%
{%
    \expandafter\XINT_expr_gobz_scanint_b\romannumeral`&&@#1%
}%
\def\XINT_expr_gobz_startdec_a .#1%
{%
    \expandafter\XINT_expr_gobz_scandec_a\romannumeral`&&@#1%
}%
\def\XINT_expr_gobz_scandec_a #1%
{%
    \if .#1\xint_dothis{0\endcsname..}\fi
    \xint_orthat {\XINT_expr_gobz_scandec_b 0.#1}%
}%
```

11.23.2 Fractional part

Annoying duplication of code to allow 0. as input.

1.2a corrects a very bad bug in 1.2 \XINT_expr_gobz_scandec_b which should have stripped leading zeroes in the fractional part but didn't; as a result \xinttheexpr 0.01\relax returned 0 =:-(((Thanks to Kroum Tzanev who reported the issue. Does it improve things if I say the bug was introduced in 1.2, it wasn't present before ?

```
\def\XINT_expr_scandec_b #1.#2%
{%
    \ifcat \relax #2\expandafter\XINT_expr_scandec_endbycs\expandafter#2\fi
    \ifnum\xint_c_ix<1\string#2 \else\expandafter\XINT_expr_scandec_c\fi
    \string#2\expandafter\XINT_expr_scandec_d\the\numexpr #1-\xint_c_i.%
}%
\def\XINT_expr_scandec_endbycs #1#2\XINT_expr_scandec_d
    \the\numexpr#3-\xint_c_i.{[#3]\endcsname #1}%
\def\XINT_expr_scandec_d #1.#2%
{%
    \expandafter\XINT_expr_scandec_b
    \the\numexpr #1\expandafter.\romannumeral`&&@#2%
}%
\def\XINT_expr_scandec_c\string #1#2\the\numexpr#3-\xint_c_i.%
{%
    \if    _#1\xint_dothis{\XINT_expr_scandec_d#3.}\fi
    \if    e#1\xint_dothis{[\the\numexpr#3\XINT_expr_scanexp_a +}\fi
    \if    E#1\xint_dothis{[\the\numexpr#3\XINT_expr_scanexp_a +}\fi
    \xint_orthat {[#3]\endcsname #1}%
}%
\def\XINT_expr_gobz_scandec_b #1.#2%
{%
    \ifcat \relax #2\expandafter\XINT_expr_gobz_scandec_endbycs\expandafter#2\fi
    \ifnum\xint_c_ix<1\string#2 \else\expandafter\XINT_expr_gobz_scandec_c\fi
    \if0#2\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
    {\expandafter\XINT_expr_gobz_scandec_b}%
    {\string#2\expandafter\XINT_expr_scandec_d}\the\numexpr#1-\xint_c_i.%
}%
```

```
\def\XINT_expr_gobz_scandec_endbycs #1#2\xint_c_i.{0[0]\endcsname #1}%
\def\XINT_expr_gobz_scandec_c\if0#1#2\fi #3\numexpr#4-\xint_c_i.%
{%
    \if    _#1\xint_dothis{\XINT_expr_gobz_scandec_b #4.}\fi
    \if    e#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
    \if    E#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
    \xint_orthat {0[0]\endcsname #1}%
}%
```

11.23.3 Scientific notation

Some pluses and minuses are allowed at the start of the scientific part, however not later, and no parenthesis.

```
\def\XINT_expr_scanexp_a #1#2%
{%
    #1\expandafter\XINT_expr_scanexp_b\romannumeral`&&@#2%
}%
\def\XINT_expr_scanexp_b #1%
{%
    \ifcat \relax #1\expandafter\XINT_expr_scanexp_endbycs\expandafter #1\fi
    \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanexp_c\fi
    \string#1\XINT_expr_scanexp_d
}%
\def\XINT_expr_scanexpr_endbycs#1#2\XINT_expr_scanexp_d {]\endcsname #1}%
\def\XINT_expr_scanexp_d #1%
{%
    \expandafter\XINT_expr_scanexp_bb\romannumeral`&&@#1%
}%
\def\XINT_expr_scanexp_c\string #1\XINT_expr_scanexp_d
{%
    \if    _#1\xint_dothis  \XINT_expr_scanexp_d   \fi
    \if    +#1\xint_dothis {\XINT_expr_scanexp_a +}\fi
    \if    -#1\xint_dothis {\XINT_expr_scanexp_a -}\fi
    \xint_orthat {]\endcsname #1}%
}%
\def\XINT_expr_scanexp_bb #1%
{%
    \ifcat \relax #1\expandafter\XINT_expr_scanexp_endbycs_b\expandafter #1\fi
    \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanexp_cb\fi
    \string#1\XINT_expr_scanexp_db
}%
\def\XINT_expr_scanexp_endbycs_b#1#2\XINT_expr_scanexp_db {]\endcsname #1}%
\def\XINT_expr_scanexp_db #1%
{%
    \expandafter\XINT_expr_scanexp_bb\romannumeral`&&@#1%
}%
\def\XINT_expr_scanexp_cb\string #1\XINT_expr_scanexp_db
{%
    \if _#1\xint_dothis\XINT_expr_scanexp_d\fi
    \xint_orthat{]\endcsname #1}%
}%
```

11.23.4 Hexadecimal numbers

1.2d has moved most of the handling of tacit multiplication to \XINT_expr_getop, but we have to do some of it here, because we apply \string before calling \XINT_expr_scanhexI_aa. I do not insert the * in \XINT_expr_scanhexI_a, because it is its higher precedence variant which will is expected, to do the same as when a non-hexadecimal number prefixes a sub-expression. Tacit multiplication in front of variable or function names will not work (because of this \string).

Extended for 1.2l to ignore underscore character _ if encountered within digits.

```
\def\XINT_expr_scanhex_I #1% #1="
{%
    \expandafter\XINT_expr_getop\csname.=\expandafter
    \XINT_expr_unlock_hex_in\csname.=\XINT_expr_scanhexI_a
}%
\def\XINT_expr_scanhexI_a #1%
{%
    \ifcat #1\relax\xint_dothis{.>\endcsname\endcsname #1}\fi
    \ifx   !#1\xint_dothis{.>\endcsname\endcsname !}\fi
    \xint_orthat {\expandafter\XINT_expr_scanhexI_aa\string #1}%
}%
\def\XINT_expr_scanhexI_aa #1%
{%
    \if\ifnum`#1>`/
       \ifnum`#1>`9
       \ifnum`#1>`@
       \ifnum`#1>`F
       0\else1\fi\else0\fi\else1\fi\else0\fi 1%
       \expandafter\XINT_expr_scanhexI_b
    \else
       \if _#1\xint_dothis{\expandafter\XINT_expr_scanhexI_bgob}\fi
       \if .#1\xint_dothis{\expandafter\XINT_expr_scanhex_transition}\fi
       \xint_orthat % gather what we got so far, leave catcode 12 #1 in stream
       {\xint_afterfi {.>\endcsname\endcsname}}%
    \fi
    #1%
}%
\def\XINT_expr_scanhexI_b #1#2%
{%
    #1\expandafter\XINT_expr_scanhexI_a\romannumeral`&&@#2%
}%
\def\XINT_expr_scanhexI_bgob #1#2%
{%
    \expandafter\XINT_expr_scanhexI_a\romannumeral`&&@#2%
}%
\def\XINT_expr_scanhex_transition .#1%
{%
    \expandafter.\expandafter.\expandafter
    \XINT_expr_scanhexII_a\romannumeral`&&@#1%
}%
\def\XINT_expr_scanhexII_a #1%
{%
    \ifcat #1\relax\xint_dothis{\endcsname\endcsname#1}\fi
    \ifx   !#1\xint_dothis{\endcsname\endcsname !}\fi
    \xint_orthat {\expandafter\XINT_expr_scanhexII_aa\string #1}%
}%
\def\XINT_expr_scanhexII_aa #1%
{%
    \if\ifnum`#1>`/
       \ifnum`#1>`9
       \ifnum`#1>`@
       \ifnum`#1>`F
       0\else1\fi\else0\fi\else1\fi\else0\fi 1%
       \expandafter\XINT_expr_scanhexII_b
    \else
       \if _#1\xint_dothis{\expandafter\XINT_expr_scanhexII_bgob}\fi
       \xint_orthat{\xint_afterfi {\endcsname\endcsname}}%
    \fi
    #1%
}%
\def\XINT_expr_scanhexII_b #1#2%
{%
    #1\expandafter\XINT_expr_scanhexII_a\romannumeral`&&@#2%
}%
\def\XINT_expr_scanhexII_bgob #1#2%
{%
    \expandafter\XINT_expr_scanhexII_a\romannumeral`&&@#2%
}%
```

11.23.5 \XINT_expr_scanfunc: parsing names of functions and variables

```
\def\XINT_expr_scanfunc
{%
    \expandafter\XINT_expr_func\romannumeral`&&@\XINT_expr_scanfunc_a
}%
\def\XINT_expr_scanfunc_a #1#2%
{%
    \expandafter #1\romannumeral`&&@\expandafter\XINT_expr_scanfunc_b\romannumeral`&&@#2%
}%
```

This handles: 1) (indirectly) tacit multiplication by a variable in front a of sub-expression, 2) (indirectly) tacit multiplication in front of a \count etc..., 3) functions which are recognized via an encountered opening parenthesis (but later this must be disambiguated from variables with tacit multiplication) 4) 5) 6) 7) acceptable components of a variable or function names: @, underscore, digits, letters (or chars of category code letter.)

The short lived 1.2d which followed the even shorter lived 1.2c managed to introduce a bug here as it removed the check for catcode 11 !, which must be recognized if ! is not to be taken as part of a variable name. Don't know what I was thinking, it was the time when I was moving the handling of tacit mutliplication entirely to the \XINT_expr_getop side. Fixed in 1.2e.

I almost decided to remove the \ifcat\relax test whose rôle is to avoid the \string#1 to do something bad is the escape char is a digit! Perhaps I will remove it at some point ! I truly almost did it, but also the case of no escape char is a problem (\string\0, if \0 is a count ...)

The (indirectly) above means that via \XINT_expr_func then \XINT_expr_op__ one goes back to \XINT_expr_getop then \XINT_expr_getop_b which is the location where tacit multiplication is now centralized. This makes the treatment of tacit multiplication for situations such as <variable>\count or <variable>\xintexpr..\relax, perhaps a bit sub-optimal, but first the variable name must be gathered, second the variable must expand to its value.

```
\def\XINT_expr_scanfunc_b #1%
{%
  \ifx !#1\xint_dothis{(_}\fi
  \ifcat \relax#1\xint_dothis{(_}\fi
  \if (#1\xint_dothis{\xint_firstoftwo{(`}}\fi
  \if @#1\xint_dothis \XINT_expr_scanfunc_a \fi
  \if _#1\xint_dothis \XINT_expr_scanfunc_a \fi
  \ifnum \xint_c_ix<1\string#1 \xint_dothis \XINT_expr_scanfunc_a \fi
  \ifcat a#1\xint_dothis \XINT_expr_scanfunc_a \fi
  \xint_orthat {(_}%
    #1%
}%
```

Comments written 2015/11/12: earlier there was an \ifcsname test for checking if we had a variable in front of a (, for tacit multiplication for example in x(y+z(x+w)) to work. But after I had implemented functions (that was yesterday...), I had the problem if was impossible to re-declare a variable name such as "f" as a function name. The problem is that here we can not test if the function is available because we don't know if we are in expr, iiexpr or floatexpr. The \xint_c_xviii causes all fetching operations to stop and control is handed over to the routines which will be expr, iiexpr ou floatexpr specific, i.e. the \XINT_{expr|iiexpr|flexpr}_op_{`|_} which are invoked by the until_<op>_b macros earlier in the stream. Functions may exist for one but not the two other parsers. Variables are declared via one parser and usable in the others, but naturally \xintiiexpr has its restrictions.

Thinking about this again I decided to treat a priori cases such as x(...) as functions, after having assigned to each variable a low-weight macro which will convert this into _getop\.=<value of x>*(...). To activate that macro at the right time I could for this exploit the "onliteral" intercept, which is parser independent (1.2c).

This led to me necessarily to rewrite partially the seq, add, mul, subs, iter ... routines as now the variables fetch only one token. I think the thing is more efficient.

1.2c had \def\XINT_expr_func #1(#2{\xint_c_xviii #2{#1}}

In \XINT_expr_func the #2 is _ if #1 must be a variable name, or #2=` if #1 must be either a function name or possibly a variable name which will then have to be followed by tacit multiplication before the opening parenthesis.

The \xint_c_xviii is there because _op_` must know in which parser it works. Dispendious for _. Hence I modify for 1.2d.

```
\def\XINT_expr_func #1(#2{\if _#2\xint_dothis\XINT_expr_op__\fi
                          \xint_orthat{\xint_c_xviii #2}{#1}}%
```

11.24 \XINT_expr_getop: finding the next operator or closing parenthesis or end of expression

Release 1.1 implements multi-character operators.

1.2d adds tacit mutiplication also in front of variable or functions names starting with a letter, not only a @ or a _ as was already the case. This is for (x+y)z situations. It also applies higher precedence in cases like x/2y or x/2@, or x/2max(3,5), or x/2\xintexpr 3\relax.

In fact, finally I decide that all sorts of tacit multiplication will always use the higher precedence.

Indeed I hesitated somewhat: with the current code one does not know if \XINT_expr_getop as invoked after a closing parenthesis or because a number parsing ended, and I felt distinguishing the two was unneeded extra stuff. This means cases like (a+b)/(c+d)(e+f) will first multiply the last two parenthesized terms.

The ! starting a sub-expression must be distinguished from the post-fix ! for factorial, thus we must not do a too early \string. In versions < 1.2c, the catcode 11 ! had to be identified in all

branches of the number or function scans. Here it is simply treated as a special case of a letter.
1.2q adds tacit multiplication in cases such as (1+1)3 or 5!7!

```
\def\XINT_expr_getop #1#2% this #1 is the current locked computed value
{%
    \expandafter\XINT_expr_getop_a\expandafter #1\romannumeral`&&@#2%
}%
\catcode`* 11
\def\XINT_expr_getop_a #1#2%
{%
    \ifx   \relax #2\xint_dothis\xint_firstofthree\fi
    \ifcat \relax #2\xint_dothis\xint_secondofthree\fi
    \ifnum\xint_c_ix<1\string#2 \xint_dothis\xint_secondofthree\fi
    \if    _#2\xint_dothis      \xint_secondofthree\fi
    \if    @#2\xint_dothis      \xint_secondofthree\fi
    \if    (#2\xint_dothis      \xint_secondofthree\fi
    \ifcat a#2\xint_dothis      \xint_secondofthree\fi
    \xint_orthat \xint_thirdofthree
    {\XINT_expr_foundend #1}%
    {\XINT_expr_precedence_*** *#1#2}% tacit multiplication with higher precedence
    {\expandafter\XINT_expr_getop_b \string#2#1}%
}%
\catcode`* 12
\def\XINT_expr_foundend {\xint_c_ \relax }% \relax is a place holder here.
```

? is a very special operator with top precedence which will check if the next token is another ?, while avoiding removing a brace pair from token stream due to its syntax. Pre 1.1 releases used : rather than ??, but we need : for Python like slices of lists.

```
\def\XINT_expr_getop_b #1%
{%
     \if '#1\xint_dothis{\XINT_expr_binopwrd }\fi
     \if ?#1\xint_dothis{\XINT_expr_precedence_? ?}\fi
     \xint_orthat       {\XINT_expr_scanop_a #1}%
}%
\def\XINT_expr_binopwrd #1#2'{\expandafter\XINT_expr_foundop_a
    \csname XINT_expr_itself_\xint_zapspaces #2 \xint_gobble_i\endcsname #1}%
\def\XINT_expr_scanop_a #1#2#3%
    {\expandafter\XINT_expr_scanop_b\expandafter #1\expandafter #2\romannumeral`&&@#3}%
\def\XINT_expr_scanop_b #1#2#3%
{%
  \ifcat#3\relax\xint_dothis{\XINT_expr_foundop_a #1#2#3}\fi
  \ifcsname XINT_expr_itself_#1#3\endcsname
  \xint_dothis
        {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#3\endcsname #2}\fi
  \xint_orthat {\XINT_expr_foundop_a #1#2#3}%
}%
\def\XINT_expr_scanop_c #1#2#3%
{%
  \expandafter\XINT_expr_scanop_d\expandafter #1\expandafter #2\romannumeral`&&@#3%
}%
\def\XINT_expr_scanop_d #1#2#3%
{%
```

```
  \ifcat#3\relax \xint_dothis{\XINT_expr_foundop #1#2#3}\fi
  \ifcsname XINT_expr_itself_#1#3\endcsname
  \xint_dothis
        {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#3\endcsname #2}\fi
  \xint_orthat {\csname XINT_expr_precedence_#1\endcsname #1#2#3}%
}%
\def\XINT_expr_foundop_a #1%
{%
    \ifcsname XINT_expr_precedence_#1\endcsname
        \csname XINT_expr_precedence_#1\expandafter\endcsname
        \expandafter #1%
    \else
        \xint_afterfi{\XINT_expr_unknown_operator {#1}\XINT_expr_getop}%
    \fi
}%
\def\XINT_expr_unknown_operator #1{\xintError:removed \xint_gobble_i {#1}}%
\def\XINT_expr_foundop #1{\csname XINT_expr_precedence_#1\endcsname #1}%
```

11.25 Expansion spanning; opening and closing parentheses

Version 1.1 had a hack inside the until macros for handling the omit and abort in iterations over dummy variables. This has been removed by 1.2c, see the subsection where omit and abort are discussed.

```
\catcode`) 11
\def\XINT_tmpa #1#2#3#4%
{%
    \def#1##1%
    {%
        \xint_UDsignfork
                    ##1{\expandafter#1\romannumeral`&&@#3}%
                      -{#2##1}%
        \krof
    }%
    \def#2##1##2%
    {%
        \ifcase ##1\expandafter\XINT_expr_done
        \or\xint_afterfi{\XINT_expr_extra_)
                          \expandafter #1\romannumeral`&&@\XINT_expr_getop }%
        \else
        \xint_afterfi{\expandafter#1\romannumeral`&&@\csname XINT_#4_op_##2\endcsname }%
        \fi
    }%
}%
\def\XINT_expr_extra_) {\xintError:removed }%
\xintFor #1 in {expr,flexpr,iiexpr} \do {%
    \expandafter\XINT_tmpa
    \csname XINT_#1_until_end_a\expandafter\endcsname
    \csname XINT_#1_until_end_b\expandafter\endcsname
    \csname XINT_#1_op_-vi\endcsname
    {#1}%
}%
\def\XINT_tmpa #1#2#3#4#5#6%
```

```
{%
    \def #1##1{\expandafter #3\romannumeral`&&@\XINT_expr_getnext }%
    \def #2{\expandafter #3\romannumeral`&&@\XINT_expr_getnext }%
    \def #3##1{\xint_UDsignfork
                ##1{\expandafter #3\romannumeral`&&@#5}%
                  -{#4##1}%
               \krof }%
    \def #4##1##2{\ifcase ##1\expandafter\XINT_expr_missing_)
      \or   \csname XINT_#6_op_##2\expandafter\endcsname
      \else
      \xint_afterfi{\expandafter #3\romannumeral`&&@\csname XINT_#6_op_##2\endcsname }%
      \fi
    }%
}%
\def\XINT_expr_missing_) {\xintError:inserted \xint_c_ \XINT_expr_done }%
```

We should be using until_(notation to stay synchronous with until_+, until_* etc..., but I found that until_) was more telling.

```
\catcode`) 12
\xintFor #1 in {expr,flexpr,iiexpr} \do {%
    \expandafter\XINT_tmpa
    \csname XINT_#1_op_(\expandafter\endcsname
    \csname XINT_#1_oparen\expandafter\endcsname
    \csname XINT_#1_until_)_a\expandafter\endcsname
    \csname XINT_#1_until_)_b\expandafter\endcsname
    \csname XINT_#1_op_-vi\endcsname
    {#1}%
}%
\expandafter\let\csname XINT_expr_precedence_)\endcsname\xint_c_i
```

11.26 |, ||, &, &&, <, >, =, ==, <=, >=, !=, +, −, *, /, ^, **, //, /:, .., ..[,]..,][,][:, :], and ++ operators

11.26.1 Square brackets for lists, the !? for omit and abort, and the ++ postfix construct

This is all very clever and only need setting some suitable precedence levels, if only I could understand what I did in 2014... just joking. Notice that op_) macros are defined here in the \xintFor loop.

There is some clever business going on here with the letter a for handling constructs such as [3..5]*2 (I think...).

1.2c has replaced 1.1's private dealings with "^C" (which was done before dummy variables got implemented) by use of "!?". See discussion of omit and abort.

```
\expandafter\let\csname XINT_expr_precedence_]\endcsname\xint_c_i
\expandafter\let\csname XINT_expr_precedence_;\endcsname\xint_c_i
```

```
\let\XINT_expr_precedence_a \xint_c_xviii
\let\XINT_expr_precedence_!? \xint_c_ii
\expandafter\let\csname XINT_expr_precedence_++)\endcsname \xint_c_i
```

Comments added 2015/11/13 Here we have in particular the mechanism for post action on lists via op_] The precedence_] is the one of a closing parenthesis. We need the closing parenthesis to do its job, hence we can not define a op_]+ operator for example, as we want to assign it the precedence of addition not the one of closing parenthesis. The trick I used in 1.1 was to let the op_] insert the letter a, this letter exceptionnally also being a legitimate operator, launch the _getop and let it find a a*, a+, a/, a-, a^, a** operator standing for]*,]+,]/,]^,]** postfix item by item list operator. I thought I had in mind an example to show that having defined op_a and precedence_a for the letter a caused a reduction in syntax for this letter, but it seems I am lacking now an example.
2015/11/18: for 1.2d I accelerate \XINT_expr_op_] to jump over the \XINT_expr_getop_a which now does tacit multiplications also in front of letters, for reasons of things like, (x+y)z, hence it must not see the "a". I could have used a catcode12 a possibly, but anyhow jumping straight to \XINT_expr_scanop_a skips a few expansion steps (up to the potential price of less conceptual programming if I change things in the future.)

```
\catcode`. 11 \catcode`= 11 \catcode`+ 11
\xintFor #1 in {expr,flexpr,iiexpr} \do {%
    \expandafter\let\csname XINT_#1_op_)\endcsname \XINT_expr_getop
    \expandafter\let\csname XINT_#1_op_;\endcsname \space
    \expandafter\def\csname XINT_#1_op_]\endcsname ##1{\XINT_expr_scanop_a a##1}%
    \expandafter\let\csname XINT_#1_op_a\endcsname \XINT_expr_getop
```

1.1 2014/10/29 did \expandafter\.=+\xintiCeil which transformed it into \romannumeral0\xinticeil, which seems a bit weird. This exploited the fact that dummy variables macros could back then pick braced material (which in the case at hand here ended being {\romannumeral0\xinticeil...} and were submitted to two expansions. The result of this was to provide a not value which got expanded only in the first loop of the :_A and following macros of seq, iter, rseq, etc...
Anyhow with 1.2c I have changed the implementation of dummy variables which now need to fetch a single locked token, which they do not expand.
The \xintiCeil appears a bit dispendious, but I need the starting value in a \numexpr compatible form in the iteration loops.

```
    \expandafter\def\csname XINT_#1_op_++)\endcsname ##1##2\relax
  {\expandafter\XINT_expr_foundend \expandafter
      {\expandafter\.=+\csname .=\XINT:NEhook:one\xintiCeil{\XINT_expr_unlock ##1}\endcsname }}%
}%
\catcode`. 12 \catcode`= 12 \catcode`+ 12
```

1.2d adds the *** for tying via tacit multiplication, for example x/2y. Actually I don't need the _itself mechanism for ***, only a precedence.

```
\catcode`& 12
\xintFor* #1 in {{==}{<=}{>=}{!=}{&&}{||}{**}{//}{/:}{..}{..[}{].}{]..}%
                 {+[}{-[}{*[}{/[}{**[}{^[}{a+}{a-}{a*}{a/}{a**}{a^}%
                 {][}{][:}{:]}{!?}{++}{++)}}%{***}}
    \do {\expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}}%
\catcode`& 7
\expandafter\let\csname XINT_expr_precedence_***\endcsname \xint_c_viii
```

11.26.2 The |, &, xor, <, >, =, <=, >=, !=, //, /:, .., +, −, *, /, ^, ..[, and].. operators for expr, floatexpr and iiexpr operators

1.2d needed some room between /, * and ^. Hence precedence for ^ is now at 9

```
\def\XINT_expr_defbin_c #1#2#3#4#5#6#7#8#9%
{%
  \def #1##1% \XINT_expr_op_<op> ou flexpr ou iiexpr
  {% keep value, get next number and operator, then do until
    \expandafter #2\expandafter ##1%
    \romannumeral`&&@\expandafter\XINT_expr_getnext }%
  \def #2##1##2% \XINT_expr_until_<op>_a ou flexpr ou iiexpr
  {\xint_UDsignfork ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
    -{#3##1##2}%
    \krof }%
  \def #3##1##2##3##4% \XINT_expr_until_<op>_b ou flexpr ou iiexpr
  {% either execute next operation now, or first do next (possibly unary)
    \ifnum ##2>#7%
    \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
      \csname XINT_#8_op_##3\endcsname {##4}}%
    \else \xint_afterfi {\expandafter ##2\expandafter ##3%
      \csname .=#9#6{\XINT_expr_unlock ##1}{\XINT_expr_unlock ##4}\endcsname }%
    \fi }%
  \let #7#5%
}%
\def\XINT_expr_defbin_b #1#2#3#4#5%
{%
  \expandafter\XINT_expr_defbin_c
  \csname XINT_#1_op_#2\expandafter\endcsname
  \csname XINT_#1_until_#2_a\expandafter\endcsname
  \csname XINT_#1_until_#2_b\expandafter\endcsname
  \csname XINT_#1_op_-#4\expandafter\endcsname
  \csname xint_c_#3\expandafter\endcsname
  \csname #5\expandafter\endcsname
  \csname XINT_expr_precedence_#2\endcsname {#1}\XINT:NEhook:two
}%
\XINT_expr_defbin_b {expr}   |    {iii}{vi} {xintOR}%
\XINT_expr_defbin_b {flexpr} |    {iii}{vi} {xintOR}%
\XINT_expr_defbin_b {iiexpr} |    {iii}{vi} {xintOR}%
\XINT_expr_defbin_b {expr}   &    {iv}{vi}  {xintAND}%
\XINT_expr_defbin_b {flexpr} &    {iv}{vi}  {xintAND}%
\XINT_expr_defbin_b {iiexpr} &    {iv}{vi}  {xintAND}%
\XINT_expr_defbin_b {expr}  {xor}{iii}{vi} {xintXOR}%
\XINT_expr_defbin_b {flexpr}{xor}{iii}{vi} {xintXOR}%
\XINT_expr_defbin_b {iiexpr}{xor}{iii}{vi} {xintXOR}%
\XINT_expr_defbin_b {expr}   <    {v}{vi}   {xintLt}%
\XINT_expr_defbin_b {flexpr} <    {v}{vi}   {xintLt}%
\XINT_expr_defbin_b {iiexpr} <    {v}{vi}   {xintiiLt}%
\XINT_expr_defbin_b {expr}   >    {v}{vi}   {xintGt}%
\XINT_expr_defbin_b {flexpr} >    {v}{vi}   {xintGt}%
\XINT_expr_defbin_b {iiexpr} >    {v}{vi}   {xintiiGt}%
\XINT_expr_defbin_b {expr}   =    {v}{vi}   {xintEq}%
\XINT_expr_defbin_b {flexpr} =    {v}{vi}   {xintEq}%
```

```
\XINT_expr_defbin_b {iiexpr} =   {v}{vi}   {xintiiEq}%
\XINT_expr_defbin_b {expr}  {<=} {v}{vi}   {xintLtorEq}%
\XINT_expr_defbin_b {flexpr}{<=} {v}{vi}   {xintLtorEq}%
\XINT_expr_defbin_b {iiexpr}{<=} {v}{vi}   {xintiiLtorEq}%
\XINT_expr_defbin_b {expr}  {>=} {v}{vi}   {xintGtorEq}%
\XINT_expr_defbin_b {flexpr}{>=} {v}{vi}   {xintGtorEq}%
\XINT_expr_defbin_b {iiexpr}{>=} {v}{vi}   {xintiiGtorEq}%
\XINT_expr_defbin_b {expr}  {!=} {v}{vi}   {xintNotEq}%
\XINT_expr_defbin_b {flexpr}{!=} {v}{vi}   {xintNotEq}%
\XINT_expr_defbin_b {iiexpr}{!=} {v}{vi}   {xintiiNotEq}%
\XINT_expr_defbin_b {expr}  {//} {vii}{vii}{xintDivFloor}% CHANGED IN 1.2p!
\XINT_expr_defbin_b {flexpr}{//} {vii}{vii}{XINTinFloatDivFloor}%   "
\XINT_expr_defbin_b {iiexpr}{//} {vii}{vii}{xintiiDivFloor}% "
\XINT_expr_defbin_b {expr}  {/:} {vii}{vii}{xintMod}%        "
\XINT_expr_defbin_b {flexpr}{/:} {vii}{vii}{XINTinFloatMod}% "
\XINT_expr_defbin_b {iiexpr}{/:} {vii}{vii}{xintiiMod}%      "
\XINT_expr_defbin_b {expr}    +   {vi}{vi}  {xintAdd}%
\XINT_expr_defbin_b {flexpr} +    {vi}{vi}  {XINTinFloatAdd}%
\XINT_expr_defbin_b {iiexpr} +    {vi}{vi}  {xintiiAdd}%
\XINT_expr_defbin_b {expr}    -   {vi}{vi}  {xintSub}%
\XINT_expr_defbin_b {flexpr} -    {vi}{vi}  {XINTinFloatSub}%
\XINT_expr_defbin_b {iiexpr} -    {vi}{vi}  {xintiiSub}%
\XINT_expr_defbin_b {expr}    *   {vii}{vii}{xintMul}%
\XINT_expr_defbin_b {flexpr} *    {vii}{vii}{XINTinFloatMul}%
\XINT_expr_defbin_b {iiexpr} *    {vii}{vii}{xintiiMul}%
\XINT_expr_defbin_b {expr}    /   {vii}{vii}{xintDiv}%
\XINT_expr_defbin_b {flexpr} /    {vii}{vii}{XINTinFloatDiv}%
\XINT_expr_defbin_b {iiexpr} /    {vii}{vii}{xintiiDivRound}% CHANGED IN 1.1!
\XINT_expr_defbin_b {expr}    ^   {ix}{ix}  {xintPow}%
\XINT_expr_defbin_b {flexpr} ^    {ix}{ix}  {XINTinFloatPowerH}%
\XINT_expr_defbin_b {iiexpr} ^    {ix}{ix}  {xintiiPow}%
\XINT_expr_defbin_b {expr}  {..[}{iii}{vi} {xintSeqA::csv}%
\XINT_expr_defbin_b {flexpr}{..[}{iii}{vi} {XINTinFloatSeqA::csv}%
\XINT_expr_defbin_b {iiexpr}{..[}{iii}{vi} {xintiiSeqA::csv}%
\def\XINT_expr_defbin_b #1#2#3#4#5%
{%
  \expandafter\XINT_expr_defbin_c
  \csname XINT_#1_op_#2\expandafter\endcsname
  \csname XINT_#1_until_#2_a\expandafter\endcsname
  \csname XINT_#1_until_#2_b\expandafter\endcsname
  \csname XINT_#1_op_-#4\expandafter\endcsname
  \csname xint_c_#3\expandafter\endcsname
  \csname #5\expandafter\endcsname
  \csname XINT_expr_precedence_#2\endcsname {#1}{}%
}%
\XINT_expr_defbin_b {expr}  {..} {iii}{vi} {xintSeq::csv}%
\XINT_expr_defbin_b {flexpr}{..} {iii}{vi} {xintSeq::csv}%
\XINT_expr_defbin_b {iiexpr}{..} {iii}{vi} {xintiiSeq::csv}%
\XINT_expr_defbin_b {expr}  {]..}{iii}{vi} {xintSeqB::csv}%
\XINT_expr_defbin_b {flexpr}{]..}{iii}{vi} {XINTinFloatSeqB::csv}%
\XINT_expr_defbin_b {iiexpr}{]..}{iii}{vi} {xintiiSeqB::csv}%
```

11.26.3 The]+,]-,]*,]/,]^, +[, -[, *[, /[, and ^[list operators

\XINT_expr_binop_inline_b This handles acting on comma separated values (no need to bother about spaces in this context; expansion in a \csname...\endcsname.

```
\def\XINT_expr_binop_inline#1%
   {\XINT_expr_binop_inline_a{\expandafter\XINT:NEhook:two\expandafter#1}}%
\def\XINT_expr_binop_inline_a
   {\expandafter\xint_gobble_i\romannumeral`&&@\XINT_expr_binop_inline_b }%
\def\XINT_expr_binop_inline_b #1#2,{\XINT_expr_binop_inline_c #2,{#1}}%
\def\XINT_expr_binop_inline_c #1{%
   \if ,#1\xint_dothis\XINT_expr_binop_inline_e\fi
   \if ^#1\xint_dothis\XINT_expr_binop_inline_end\fi
   \xint_orthat\XINT_expr_binop_inline_d #1}%
\def\XINT_expr_binop_inline_d #1,#2{,#2{#1}\XINT_expr_binop_inline_b {#2}}%
\def\XINT_expr_binop_inline_e #1,#2{,\XINT_expr_binop_inline_b {#2}}%
\def\XINT_expr_binop_inline_end #1,#2{}%
\def\XINT_expr_deflistopr_c #1#2#3#4#5#6#7#8%
{%
  \def #1##1% \XINT_expr_op_<op> ou flexpr ou iiexpr
  {% keep value, get next number and operator, then do until
    \expandafter #2\expandafter ##1%
    \romannumeral`&&@\expandafter\XINT_expr_getnext }%
  \def #2##1##2% \XINT_expr_until_<op>_a ou flexpr ou iiexpr
  {\xint_UDsignfork ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
    -{#3##1##2}%
    \krof }%
  \def #3##1##2##3##4% \XINT_expr_until_<op>_b ou flexpr ou iiexpr
  {% either execute next operation now, or first do next (possibly unary)
    \ifnum ##2>#7%
    \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
      \csname XINT_#8_op_##3\endcsname {##4}}%
    \else \xint_afterfi {\expandafter ##2\expandafter ##3%
      \csname .=\expandafter\XINT_expr_binop_inline\expandafter
      {\expandafter#6\expandafter\xint_exchangetwo_keepbraces\expandafter
      {\expandafter\XINT_expr_unlock\expandafter ##4\expandafter}\expandafter}%
         \romannumeral`&&@\XINT_expr_unlock ##1,^,\endcsname }%
    \fi }%
  \let #7#5%
}%
\def\XINT_expr_deflistopr_b #1#2#3#4%
{%
  \expandafter\XINT_expr_deflistopr_c
  \csname XINT_#1_op_#2\expandafter\endcsname
  \csname XINT_#1_until_#2_a\expandafter\endcsname
  \csname XINT_#1_until_#2_b\expandafter\endcsname
  \csname XINT_#1_op_-#3\expandafter\endcsname
  \csname xint_c_#3\expandafter\endcsname
  \csname #4\expandafter\endcsname
  \csname XINT_expr_precedence_#2\endcsname {#1}%
}%
```

This is for [x..y]*z syntax etc.... Attention that with 1.2d, precedence level of ^ raised to ix to make room for ***.

```
\XINT_expr_deflistopr_b {expr}  {a+}{vi} {xintAdd}%
\XINT_expr_deflistopr_b {expr}  {a-}{vi} {xintSub}%
\XINT_expr_deflistopr_b {expr}  {a*}{vii}{xintMul}%
\XINT_expr_deflistopr_b {expr}  {a/}{vii}{xintDiv}%
\XINT_expr_deflistopr_b {expr}  {a^}{ix} {xintPow}%
\XINT_expr_deflistopr_b {iiexpr}{a+}{vi} {xintiiAdd}%
\XINT_expr_deflistopr_b {iiexpr}{a-}{vi} {xintiiSub}%
\XINT_expr_deflistopr_b {iiexpr}{a*}{vii}{xintiiMul}%
\XINT_expr_deflistopr_b {iiexpr}{a/}{vii}{xintiiDivRound}%
\XINT_expr_deflistopr_b {iiexpr}{a^}{ix} {xintiiPow}%
\XINT_expr_deflistopr_b {flexpr}{a+}{vi} {XINTinFloatAdd}%
\XINT_expr_deflistopr_b {flexpr}{a-}{vi} {XINTinFloatSub}%
\XINT_expr_deflistopr_b {flexpr}{a*}{vii}{XINTinFloatMul}%
\XINT_expr_deflistopr_b {flexpr}{a/}{vii}{XINTinFloatDiv}%
\XINT_expr_deflistopr_b {flexpr}{a^}{ix} {XINTinFloatPowerH}%
\def\XINT_expr_deflistopl_c #1#2#3#4#5#6#7%
{%
  \def #1##1{\expandafter#2\expandafter##1\romannumeral`&&@%
             \expandafter #3\romannumeral`&&@\XINT_expr_getnext }%
  \def #2##1##2##3##4%
  {% either execute next operation now, or first do next (possibly unary)
    \ifnum ##2>#6%
    \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
      \csname XINT_#7_op_##3\endcsname {##4}}%
    \else \xint_afterfi {\expandafter ##2\expandafter ##3%
      \csname .=\expandafter\XINT_expr_binop_inline\expandafter
      {\expandafter#5\expandafter
      {\expandafter\XINT_expr_unlock\expandafter ##1\expandafter}\expandafter}%
         \romannumeral`&&@\XINT_expr_unlock ##4,^,\endcsname }%
    \fi }%
  \let #6#4%
}%
\def\XINT_expr_deflistopl_b #1#2#3#4%
{%
  \expandafter\XINT_expr_deflistopl_c
  \csname XINT_#1_op_#2\expandafter\endcsname
  \csname XINT_#1_until_#2\expandafter\endcsname
  \csname XINT_#1_until_)_a\expandafter\endcsname
  \csname xint_c_#3\expandafter\endcsname
  \csname #4\expandafter\endcsname
  \csname XINT_expr_precedence_#2\endcsname {#1}%
}%
```

This is for z*[x..y] syntax etc...

```
\XINT_expr_deflistopl_b {expr}  {+[]{vi} {xintAdd}%
\XINT_expr_deflistopl_b {expr}  {-[]{vi} {xintSub}%
\XINT_expr_deflistopl_b {expr}  {*[]{vii}{xintMul}%
\XINT_expr_deflistopl_b {expr}  {/[]{vii}{xintDiv}%
\XINT_expr_deflistopl_b {expr}  {^[]{ix} {xintPow}%
\XINT_expr_deflistopl_b {iiexpr}{+[]{vi} {xintiiAdd}%
\XINT_expr_deflistopl_b {iiexpr}{-[]{vi} {xintiiSub}%
\XINT_expr_deflistopl_b {iiexpr}{*[]{vii}{xintiiMul}%
```

```
\XINT_expr_deflistopl_b {iiexpr}{/[]}{vii}{xintiiDivRound}%
\XINT_expr_deflistopl_b {iiexpr}{^[]}{ix} {xintiiPow}%
\XINT_expr_deflistopl_b {flexpr}{+[]}{vi} {XINTinFloatAdd}%
\XINT_expr_deflistopl_b {flexpr}{-[]}{vi} {XINTinFloatSub}%
\XINT_expr_deflistopl_b {flexpr}{*[]}{vii}{XINTinFloatMul}%
\XINT_expr_deflistopl_b {flexpr}{/[]}{vii}{XINTinFloatDiv}%
\XINT_expr_deflistopl_b {flexpr}{^[]}{ix} {XINTinFloatPowerH}%
```

11.26.4 The 'and', 'or', 'xor', and 'mod' as infix operator words

```
\xintFor #1 in {and,or,xor,mod} \do {%
   \expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}}%
\expandafter\let\csname XINT_expr_precedence_and\expandafter\endcsname
                \csname XINT_expr_precedence_&\endcsname
\expandafter\let\csname XINT_expr_precedence_or\expandafter\endcsname
                \csname XINT_expr_precedence_|\endcsname
\expandafter\let\csname XINT_expr_precedence_mod\expandafter\endcsname
                \csname XINT_expr_precedence_/:\endcsname
\xintFor #1 in {expr, flexpr, iiexpr} \do {%
   \expandafter\let\csname XINT_#1_op_and\expandafter\endcsname
                   \csname XINT_#1_op_&\endcsname
   \expandafter\let\csname XINT_#1_op_or\expandafter\endcsname
                   \csname XINT_#1_op_|\endcsname
   \expandafter\let\csname XINT_#1_op_mod\expandafter\endcsname
                   \csname XINT_#1_op_/:\endcsname
}%
```

11.26.5 The ||, &&, **, **[,]** operators as synonyms

```
\expandafter\let\csname XINT_expr_precedence_==\expandafter\endcsname
                \csname XINT_expr_precedence_=\endcsname
\expandafter\let\csname XINT_expr_precedence_&\string&\expandafter\endcsname
                \csname XINT_expr_precedence_&\endcsname
\expandafter\let\csname XINT_expr_precedence_||\expandafter\endcsname
                \csname XINT_expr_precedence_|\endcsname
\expandafter\let\csname XINT_expr_precedence_**\expandafter\endcsname
                \csname XINT_expr_precedence_^\endcsname
\expandafter\let\csname XINT_expr_precedence_a**\expandafter\endcsname
                \csname XINT_expr_precedence_a^\endcsname
\expandafter\let\csname XINT_expr_precedence_**[\expandafter\endcsname
                \csname XINT_expr_precedence_^[\endcsname
\xintFor #1 in {expr, flexpr, iiexpr} \do {%
   \expandafter\let\csname XINT_#1_op_==\expandafter\endcsname
                   \csname XINT_#1_op_=\endcsname
   \expandafter\let\csname XINT_#1_op_&\string&\expandafter\endcsname
                   \csname XINT_#1_op_&\endcsname
   \expandafter\let\csname XINT_#1_op_||\expandafter\endcsname
                   \csname XINT_#1_op_|\endcsname
   \expandafter\let\csname XINT_#1_op_**\expandafter\endcsname
                   \csname XINT_#1_op_^\endcsname
   \expandafter\let\csname XINT_#1_op_a**\expandafter\endcsname
                   \csname XINT_#1_op_a^\endcsname
   \expandafter\let\csname XINT_#1_op_**[\expandafter\endcsname
                   \csname XINT_#1_op_^[\endcsname
}%
```

11.27 Macros for list selectors: [list][N], [list][:b], [list][a:], [list][a:b]

Python slicing was first implemented for 1.1 (27 octobre 2014). But it used \xintCSVtoList and \xintListWithSep{,} to convert back and forth to token lists for use of \xintKeep, \xintTrim, \xintNthElt. Not very efficient! Also [list][a:b] was Python like but not [list][N] which counted items starting at one, and returned the length for N=0.

Release 1.2g changed this so [list][N] now counts starting at zero and len(list) computes the number of items. Also 1.2g had its own f-expandable macros handling directly the comma separated lists. They are located into xinttools.sty.

1.2j improved the xinttools.sty macros and furthermore it made the Python slicing in expressions a bit more efficient still by exploiting in some cases that expansion happens in \csname...\endcsname and does not have to be f-expandable. But the f-expandable variants must be kept for use by \xintNewExpr and \xintdeffunc.

```
\def\XINT_tmpa #1#2#3#4#5#6%
{%
    \def #1##1% \XINT_expr_op_][
    {%
        \expandafter #2\expandafter ##1\romannumeral`&&@\XINT_expr_getnext
    }%
    \def #2##1##2% \XINT_expr_until_][_a
    {\xint_UDsignfork
        ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
          -{#3##1##2}%
     \krof }%
    \def #3##1##2##3##4% \XINT_expr_until_][_b
    {%
      \ifnum ##2>#5%
        \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
                       \csname XINT_#6_op_##3\endcsname {##4}}%
      \else
        \xint_afterfi
        {\expandafter ##2\expandafter ##3\csname
           .=\expandafter\xintListSel:x:csv % will be \xintListSel:f:csv in \xintNewExpr output
             \romannumeral`&&@\XINT_expr_unlock ##4;% selector
             \XINT_expr_unlock ##1;\endcsname % unlock already pre-positioned for \xintNewExpr
        }%
      \fi
    }%
    \let #5\xint_c_ii
}%
\xintFor #1 in {expr,flexpr,iiexpr} \do {%
\expandafter\XINT_tmpa
    \csname XINT_#1_op_][\expandafter\endcsname
    \csname XINT_#1_until_][_a\expandafter\endcsname
    \csname XINT_#1_until_][_b\expandafter\endcsname
    \csname XINT_#1_op_-vi\expandafter\endcsname
    \csname XINT_expr_precedence_][\endcsname {#1}%
}%
\def\XINT_tmpa #1#2#3#4#5#6%
{%
    \def #1##1% \XINT_expr_op_:
    {%
        \expandafter #2\expandafter ##1\romannumeral`&&@\XINT_expr_getnext
    }%
    \def #2##1##2% \XINT_expr_until_:_a
    {\xint_UDsignfork
        ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
          -{#3##1##2}%
     \krof }%
    \def #3##1##2##3##4% \XINT_expr_until_:_b
    {%
      \ifnum ##2>#5%
        \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
                       \csname XINT_#6_op_##3\endcsname {##4}}%
      \else
        \xint_afterfi
        {\expandafter ##2\expandafter ##3\csname
         .=:\XINT:NEhook:one\xintNum{\XINT_expr_unlock ##1};%
            \XINT:NEhook:one\xintNum{\XINT_expr_unlock ##4}%
         \endcsname
        }%
      \fi
    }%
    \let #5\xint_c_iii
}%
\xintFor #1 in {expr,flexpr,iiexpr} \do {%
\expandafter\XINT_tmpa
    \csname XINT_#1_op_:\expandafter\endcsname
    \csname XINT_#1_until_:_a\expandafter\endcsname
    \csname XINT_#1_until_:_b\expandafter\endcsname
    \csname XINT_#1_op_-vi\expandafter\endcsname
    \csname XINT_expr_precedence_:\endcsname {#1}%
}%
\catcode`[ 11 \catcode`] 11
\let\XINT_expr_precedence_:] \xint_c_iii
\def\XINT_expr_op_:] #1%
{%
  \expandafter\xint_c_i\expandafter )%
  \csname .=]\XINT:NEhook:one\xintNum{\XINT_expr_unlock #1}\endcsname
}%
\let\XINT_flexpr_op_:] \XINT_expr_op_:]
\let\XINT_iiexpr_op_:] \XINT_expr_op_:]
\let\XINT_expr_precedence_][: \xint_c_iii
```

At the end of the replacement text of \XINT_expr_op_][:, the : after index 0 must be catcode 12, else will be mistaken for the start of variable by expression parser (as <digits><variable> is allowed by the syntax and does tacit multiplication).

```
\edef\XINT_expr_op_][: #1{\xint_c_ii\noexpand\XINT_expr_itself_][#10\string :}%
\let\XINT_flexpr_op_][: \XINT_expr_op_][:
\let\XINT_iiexpr_op_][: \XINT_expr_op_][:
\catcode`[ 12 \catcode`] 12
```

11.27.1 \xintListSel:x:csv

1.2j. Because there is \xintKeep:x:csv which is faster than \xintKeep:f:csv.

```
\def\xintListSel:x:csv #1%
{%
    \if ]\noexpand#1\xint_dothis\XINT_listsel:_s\fi
    \if :\noexpand#1\xint_dothis\XINT_listxsel:_:\fi
    \xint_orthat {\XINT_listsel:_nth #1}%
}%
\def\XINT_listsel:_s #1#2;#3;%
{%
   \if-#1\expandafter\xintKeep:f:csv\else\expandafter\xintTrim:f:csv\fi
   {#1#2}{#3}%
}%
\def\XINT_listsel:_nth #1;#2;{\xintNthEltPy:f:csv {\xintNum{#1}}{#2}}%
```

\XINT_listsel:_nth and \XINT_listsel:_s located in \xintListSel:f:csv.

```
\def\XINT_listxsel:_: #1#2;#3#4;%
{%
    \xint_UDsignsfork
       #1#3\XINT_listxsel:_N:N
        #1-\XINT_listxsel:_N:P
        -#3\XINT_listxsel:_P:N
         --\XINT_listxsel:_P:P
    \krof #1#2;#3#4;%
}%
\def\XINT_listxsel:_P:P #1;#2;#3;%
{%
    \unless\ifnum #1<#2 \expandafter\xint_gobble_iii\fi
    \xintKeep:x:csv{#2-#1}{\xintTrim:f:csv{#1}{#3}}%
}%
\def\XINT_listxsel:_N:N #1;#2;#3;%
{%
    \expandafter\XINT_listxsel:_N:N_a
    \the\numexpr #2-#1\expandafter;\the\numexpr#1+\xintLength:f:csv{#3};#3;%
}%
\def\XINT_listxsel:_N:N_a #1;#2;#3;%
{%
    \unless\ifnum #1>\xint_c_ \expandafter\xint_gobble_iii\fi
    \xintKeep:x:csv{#1}{\xintTrim:f:csv{\ifnum#2<\xint_c_\xint_c_\else#2\fi}{#3}}%
}%
\def\XINT_listxsel:_N:P #1;#2;#3;{\expandafter\XINT_listxsel:_N:P_a
```

```
                                  \the\numexpr #1+\xintLength:f:csv{#3};#2;#3;}%
\def\XINT_listxsel:_N:P_a #1#2;%
   {\if -#1\expandafter\XINT_listxsel:_O:P\fi\XINT_listxsel:_P:P #1#2;}%
\def\XINT_listxsel:_O:P\XINT_listxsel:_P:P #1;{\XINT_listxsel:_P:P 0;}%
\def\XINT_listxsel:_P:N #1;#2;#3;{\expandafter\XINT_listxsel:_P:N_a
                                  \the\numexpr #2+\xintLength:f:csv{#3};#1;#3;}%
\def\XINT_listxsel:_P:N_a #1#2;#3;%
   {\if -#1\expandafter\XINT_listxsel:_P:O\fi\XINT_listxsel:_P:P #3;#1#2;}%
\def\XINT_listxsel:_P:O\XINT_listxsel:_P:P #1;#2;{\XINT_listxsel:_P:P #1;0;}%
```

11.27.2 \xintListSel:f:csv

1.2g. Since 1.2j this is needed only for \xintNewExpr and user defined functions. Some extras compared to \xintListSel:x:csv because things may not yet have been expanded in the \xintNewExpr context.

```
\def\xintListSel:f:csv #1%
{%
    \if ]\noexpand#1\xint_dothis{\expandafter\XINT_listsel:_s\romannumeral`&&@}\fi
    \if :\noexpand#1\xint_dothis{\XINT_listsel:_:}\fi
    \xint_orthat {\XINT_listsel:_nth #1}%
}%
\def\XINT_listsel:_: #1;#2;%
{%
    \expandafter\XINT_listsel:_:a
    \the\numexpr #1\expandafter;\the\numexpr #2\expandafter;\romannumeral`&&@%
}%
\def\XINT_listsel:_:a #1#2;#3#4;%
{%
    \xint_UDsignsfork
       #1#3\XINT_listsel:_N:N
        #1-\XINT_listsel:_N:P
        -#3\XINT_listsel:_P:N
         --\XINT_listsel:_P:P
    \krof #1#2;#3#4;%
}%
\def\XINT_listsel:_P:P #1;#2;#3;%
{%
    \unless\ifnum #1<#2 \xint_afterfi{\expandafter\space\xint_gobble_iii}\fi
    \xintKeep:f:csv{#2-#1}{\xintTrim:f:csv{#1}{#3}}%
}%
\def\XINT_listsel:_N:N #1;#2;#3;%
{%
    \unless\ifnum #1<#2 \expandafter\XINT_listsel:_N:N_abort\fi
    \expandafter\XINT_listsel:_N:N_a
    \the\numexpr#1+\xintLength:f:csv{#3}\expandafter;\the\numexpr#2-#1;#3;%
}%
\def\XINT_listsel:_N:N_abort #1;#2;#3;{ }%
\def\XINT_listsel:_N:N_a #1;#2;#3;%
{%
    \xintKeep:f:csv{#2}{\xintTrim:f:csv{\ifnum#1<\xint_c_\xint_c_\else#1\fi}{#3}}%
}%
\def\XINT_listsel:_N:P #1;#2;#3;{\expandafter\XINT_listsel:_N:P_a
```

```
                                \the\numexpr #1+\xintLength:f:csv{#3};#2;#3;}%
\def\XINT_listsel:_N:P_a #1#2;%
   {\if -#1\expandafter\XINT_listsel:_O:P\fi\XINT_listsel:_P:P #1#2;}%
\def\XINT_listsel:_O:P\XINT_listsel:_P:P #1;{\XINT_listsel:_P:P 0;}%
\def\XINT_listsel:_P:N #1;#2;#3;{\expandafter\XINT_listsel:_P:N_a
                                \the\numexpr #2+\xintLength:f:csv{#3};#1;#3;}%
\def\XINT_listsel:_P:N_a #1#2;#3;%
   {\if -#1\expandafter\XINT_listsel:_P:O\fi\XINT_listsel:_P:P #3;#1#2;}%
\def\XINT_listsel:_P:O\XINT_listsel:_P:P #1;#2;{\XINT_listsel:_P:P #1;0;}%
```

11.27.3 \xintKeep:x:csv

1.2j. This macro is used only with positive first argument.

```
\def\xintKeep:x:csv #1#2%
{%
    \expandafter\xint_gobble_i
    \romannumeral0\expandafter\XINT_keep:x:csv_pos
    \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
}%
\def\XINT_keep:x:csv_pos #1.#2%
{%
    \expandafter\XINT_keep:x:csv_loop\the\numexpr#1-\xint_c_viii.%
    #2\xint_Bye,\xint_Bye,\xint_Bye,\xint_Bye,%
       \xint_Bye,\xint_Bye,\xint_Bye,\xint_Bye,\xint_bye
}%
\def\XINT_keep:x:csv_loop #1%
{%
    \xint_gob_til_minus#1\XINT_keep:x:csv_finish-%
    \XINT_keep:x:csv_loop_pickeight #1%
}%
\def\XINT_keep:x:csv_loop_pickeight #1.#2,#3,#4,#5,#6,#7,#8,#9,%
{%
    ,#2,#3,#4,#5,#6,#7,#8,#9%
    \expandafter\XINT_keep:x:csv_loop\the\numexpr#1-\xint_c_viii.%
}%
\def\XINT_keep:x:csv_finish-\XINT_keep:x:csv_loop_pickeight -#1.%
{%
    \csname XINT_keep:x:csv_finish#1\endcsname
}%
\expandafter\def\csname XINT_keep:x:csv_finish1\endcsname
  #1,#2,#3,#4,#5,#6,#7,{,#1,#2,#3,#4,#5,#6,#7\xint_Bye}%
\expandafter\def\csname XINT_keep:x:csv_finish2\endcsname
  #1,#2,#3,#4,#5,#6,{,#1,#2,#3,#4,#5,#6\xint_Bye}%
\expandafter\def\csname XINT_keep:x:csv_finish3\endcsname
  #1,#2,#3,#4,#5,{,#1,#2,#3,#4,#5\xint_Bye}%
\expandafter\def\csname XINT_keep:x:csv_finish4\endcsname
  #1,#2,#3,#4,{,#1,#2,#3,#4\xint_Bye}%
\expandafter\def\csname XINT_keep:x:csv_finish5\endcsname
  #1,#2,#3,{,#1,#2,#3\xint_Bye}%
\expandafter\def\csname XINT_keep:x:csv_finish6\endcsname
  #1,#2,{,#1,#2\xint_Bye}%
\expandafter\def\csname XINT_keep:x:csv_finish7\endcsname
```

```
  #1,{,#1\xint_Bye}%
\expandafter\let\csname XINT_keep:x:csv_finish8\endcsname\xint_Bye
```

11.27.4 \xintKeep:f:csv

1.2g. moved to xinttools.

11.27.5 \xintTrim:f:csv

1.2g. moved to xinttools.

11.27.6 \xintNthEltPy:f:csv

1.2g. moved to xinttools.

11.27.7 \xintLength:f:csv

1.2g. moved to xinttools.

11.27.8 \xintReverse:f:csv

1.2g. moved to xinttools.

11.28 Macros for a..b list generation

Ne produit que des listes d'entiers inférieurs à la borne de TeX ! mais sous la forme N/1[0] en ce qui concerne \xintSeq::csv.

11.28.1 \xintSeq::csv

Commence par remplacer a par ceil(a) et b par floor(b) et renvoie ensuite les entiers entre les deux, possiblement en décroissant, et extrémités comprises. Si a=b est non entier en obtient donc ceil(a) et floor(a). Ne renvoie jamais une liste vide.

Note: le a..b dans \xintfloatexpr utilise cette routine.

```
\def\xintSeq::csv {\romannumeral0\xintseq::csv }%
\def\xintseq::csv #1#2%
{%
    \expandafter\XINT_seq::csv\expandafter
       {\the\numexpr \xintiCeil{#1}\expandafter}\expandafter
       {\the\numexpr \xintiFloor{#2}}%
}%
\def\XINT_seq::csv #1#2%
{%
   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
      \expandafter\XINT_seq::csv_z
   \or
      \expandafter\XINT_seq::csv_p
   \else
```

```
      \expandafter\XINT_seq::csv_n
   \fi
   {#2}{#1}%
}%
\def\XINT_seq::csv_z #1#2{ #1/1[0]}%
\def\XINT_seq::csv_p #1#2%
{%
    \ifnum #1>#2
      \expandafter\expandafter\expandafter\XINT_seq::csv_p
    \else
      \expandafter\XINT_seq::csv_e
    \fi
    \expandafter{\the\numexpr #1-\xint_c_i}{#2},#1/1[0]%
}%
\def\XINT_seq::csv_n #1#2%
{%
    \ifnum #1<#2
      \expandafter\expandafter\expandafter\XINT_seq::csv_n
    \else
      \expandafter\XINT_seq::csv_e
    \fi
     \expandafter{\the\numexpr #1+\xint_c_i}{#2},#1/1[0]%
}%
\def\XINT_seq::csv_e #1,{ }%
```

11.28.2 \xintiiSeq::csv

```
\def\xintiiSeq::csv {\romannumeral0\xintiiseq::csv }%
\def\xintiiseq::csv #1#2%
{%
    \expandafter\XINT_iiseq::csv\expandafter
       {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
}%
\def\XINT_iiseq::csv #1#2%
{%
   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
      \expandafter\XINT_iiseq::csv_z
   \or
      \expandafter\XINT_iiseq::csv_p
   \else
      \expandafter\XINT_iiseq::csv_n
   \fi
   {#2}{#1}%
}%
\def\XINT_iiseq::csv_z #1#2{ #1}%
\def\XINT_iiseq::csv_p #1#2%
{%
    \ifnum #1>#2
      \expandafter\expandafter\expandafter\XINT_iiseq::csv_p
    \else
      \expandafter\XINT_seq::csv_e
    \fi
    \expandafter{\the\numexpr #1-\xint_c_i}{#2},#1%
}%
\def\XINT_iiseq::csv_n #1#2%
{%
    \ifnum #1<#2
      \expandafter\expandafter\expandafter\XINT_iiseq::csv_n
    \else
      \expandafter\XINT_seq::csv_e
    \fi
     \expandafter{\the\numexpr #1+\xint_c_i}{#2},#1%
}%
\def\XINT_seq::csv_e #1,{ }%
```

11.29 Macros for a..[d]..b list generation

Contrarily to a..b which is limited to small integers, this works with a, b, and d (big) fractions. It will produce a «nil» list, if a>b and d<0 or a<b and d>0.

11.29.1 \xintSeqA::csv, \xintiiSeqA::csv, \XINTinFloatSeqA::csv

```
\def\xintSeqA::csv #1%
   {\expandafter\XINT_seqa::csv\expandafter{\romannumeral0\xintraw {#1}}}%
\def\XINT_seqa::csv #1#2{\expandafter\XINT_seqa::csv_a \romannumeral0\xintraw {#2};#1;}%
\def\xintiiSeqA::csv #1{\expandafter\XINT_iiseqa::csv\expandafter{\romannumeral`&&@#1}}%
\def\XINT_iiseqa::csv #1#2{\expandafter\XINT_seqa::csv_a\romannumeral`&&@#2;#1;}%
\def\XINTinFloatSeqA::csv #1{\expandafter\XINT_flseqa::csv\expandafter
   {\romannumeral0\XINTinfloat [\XINTdigits]{#1}}}%
\def\XINT_flseqa::csv #1#2%
   {\expandafter\XINT_seqa::csv_a\romannumeral0\XINTinfloat [\XINTdigits]{#2};#1;}%
\def\XINT_seqa::csv_a #1{\xint_UDzerominusfork
                                  #1-{z}%
                                  0#1{n}%
                                  0-{p}%
                        \krof #1}%
```

11.29.2 \xintSeqB::csv

With one year late documentation, let's just say, the #1 is \XINT_expr_unlock\.=Ua;b; with U=z or n or p, a=step, b=start.

```
\def\xintSeqB::csv #1#2%
   {\expandafter\XINT_seqb::csv \expandafter{\romannumeral0\xintraw{#2}}{#1}}%
\def\XINT_seqb::csv #1#2{\expandafter\XINT_seqb::csv_a\romannumeral`&&@#2#1!}%
\def\XINT_seqb::csv_a #1#2;#3;#4!{\expandafter\XINT_expr_seq_empty?
      \romannumeral0\csname XINT_seqb::csv_#1\endcsname {#3}{#4}{#2}}%
\def\XINT_seqb::csv_p #1#2#3%
{%
   \xintifCmp {#1}{#2}{,#1\expandafter\XINT_seqb::csv_p\expandafter}%
   {,#1\xint_gobble_iii}{\xint_gobble_iii}%
```

\romannumeral0 stopped by \endcsname, XINT_expr_seq_empty? constructs "nil".

```
   {\romannumeral0\xintadd {#3}{#1}}{#2}{#3}%
}%
\def\XINT_seqb::csv_n #1#2#3%
{%
    \xintifCmp {#1}{#2}{\xint_gobble_iii}{,#1\xint_gobble_iii}%
    {,#1\expandafter\XINT_seqb::csv_n\expandafter}%
    {\romannumeral0\xintadd {#3}{#1}}{#2}{#3}%
}%
\def\XINT_seqb::csv_z #1#2#3{,#1}%
```

11.29.3 \xintiiSeqB::csv

```
\def\xintiiSeqB::csv #1#2{\XINT_iiseqb::csv #1#2}%
\def\XINT_iiseqb::csv #1#2#3#4%
   {\expandafter\XINT_iiseqb::csv_a
    \romannumeral`&&@\expandafter \XINT_expr_unlock\expandafter#2%
    \romannumeral`&&@\XINT_expr_unlock #4!}%
\def\XINT_iiseqb::csv_a #1#2;#3;#4!{\expandafter\XINT_expr_seq_empty?
      \romannumeral`&&@\csname XINT_iiseqb::csv_#1\endcsname {#3}{#4}{#2}}%
\def\XINT_iiseqb::csv_p #1#2#3%
{%
  \xintSgnFork{\XINT_Cmp {#1}{#2}}{,#1\expandafter\XINT_iiseqb::csv_p\expandafter}%
  {,#1\xint_gobble_iii}{\xint_gobble_iii}%
  {\romannumeral0\xintiiadd {#3}{#1}}{#2}{#3}%
}%
\def\XINT_iiseqb::csv_n #1#2#3%
{%
  \xintSgnFork{\XINT_Cmp {#1}{#2}}{\xint_gobble_iii}{,#1\xint_gobble_iii}%
  {,#1\expandafter\XINT_iiseqb::csv_n\expandafter}%
  {\romannumeral0\xintiiadd {#3}{#1}}{#2}{#3}%
}%
\def\XINT_iiseqb::csv_z #1#2#3{,#1}%
```

11.29.4 \XINTinFloatSeqB::csv

```
\def\XINTinFloatSeqB::csv #1#2{\expandafter\XINT_flseqb::csv \expandafter
    {\romannumeral0\XINTinfloat [\XINTdigits]{#2}}{#1}}%
\def\XINT_flseqb::csv #1#2{\expandafter\XINT_flseqb::csv_a\romannumeral`&&@#2#1!}%
\def\XINT_flseqb::csv_a #1#2;#3;#4!{\expandafter\XINT_expr_seq_empty?
      \romannumeral`&&@\csname XINT_flseqb::csv_#1\endcsname {#3}{#4}{#2}}%
\def\XINT_flseqb::csv_p #1#2#3%
{%
  \xintifCmp {#1}{#2}{,#1\expandafter\XINT_flseqb::csv_p\expandafter}%
  {,#1\xint_gobble_iii}{\xint_gobble_iii}%
  {\romannumeral0\XINTinfloatadd {#3}{#1}}{#2}{#3}%
}%
\def\XINT_flseqb::csv_n #1#2#3%
{%
  \xintifCmp {#1}{#2}{\xint_gobble_iii}{,#1\xint_gobble_iii}%
  {,#1\expandafter\XINT_flseqb::csv_n\expandafter}%
  {\romannumeral0\XINTinfloatadd {#3}{#1}}{#2}{#3}%
}%
```

```
\def\XINT_flseqb::csv_z #1#2#3{,#1}%
```

11.30 The comma as binary operator

New with 1.09a. Suffices to set its precedence level to two.

```
\def\XINT_tmpa #1#2#3#4#5#6%
{%
    \def #1##1% \XINT_expr_op_,
    {%
        \expandafter #2\expandafter ##1\romannumeral`&&@\XINT_expr_getnext
    }%
    \def #2##1##2% \XINT_expr_until_,_a
    {\xint_UDsignfork
        ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
          -{#3##1##2}%
     \krof }%
    \def #3##1##2##3##4% \XINT_expr_until_,_b
    {%
      \ifnum ##2>\xint_c_ii
        \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
                       \csname XINT_#6_op_##3\endcsname {##4}}%
      \else
        \xint_afterfi
        {\expandafter ##2\expandafter ##3%
         \csname .=\XINT_expr_unlock ##1,\XINT_expr_unlock ##4\endcsname }%
      \fi
    }%
    \let #5\xint_c_ii
}%
\xintFor #1 in {expr,flexpr,iiexpr} \do {%
\expandafter\XINT_tmpa
    \csname XINT_#1_op_,\expandafter\endcsname
    \csname XINT_#1_until_,_a\expandafter\endcsname
    \csname XINT_#1_until_,_b\expandafter\endcsname
    \csname XINT_#1_op_-vi\expandafter\endcsname
    \csname XINT_expr_precedence_,\endcsname {#1}%
}%
```

11.31 The minus as prefix operator of variable precedence level

Inherits the precedence level of the previous infix operator.

```
\def\XINT_tmpa #1#2#3%
{%
    \expandafter\XINT_tmpb
    \csname XINT_#1_op_-#3\expandafter\endcsname
    \csname XINT_#1_until_-#3_a\expandafter\endcsname
    \csname XINT_#1_until_-#3_b\expandafter\endcsname
    \csname xint_c_#3\endcsname {#1}#2%
}%
\def\XINT_tmpb #1#2#3#4#5#6%
{%
```

```
    \def #1% \XINT_expr_op_-<level>
    {%  get next number+operator then switch to _until macro
        \expandafter #2\romannumeral`&&@\XINT_expr_getnext
    }%
    \def #2##1% \XINT_expr_until_-<l>_a
    {\xint_UDsignfork
        ##1{\expandafter #2\romannumeral`&&@#1}%
          -{#3##1}%
     \krof }%
    \def #3##1##2##3% \XINT_expr_until_-<l>_b
    {%  _until tests precedence level with next op, executes now or postpones
        \ifnum ##1>#4%
         \xint_afterfi {\expandafter #2\romannumeral`&&@%
                        \csname XINT_#5_op_##2\endcsname {##3}}%
        \else
         \xint_afterfi {\expandafter ##1\expandafter ##2%
                        \csname .=%
                        \XINT:NEhook:one#6{\XINT_expr_unlock ##3}\endcsname }%
        \fi
    }%
}%
```

1.2d needs precedence 8 for *** and 9 for ^. Earlier, precedence level for ^ was only 8 but nevertheless the code did also "ix" here, which I think was unneeded back then.

```
\xintApplyInline{\XINT_tmpa {expr}\xintOpp}{{vi}{vii}{viii}{ix}}%
\xintApplyInline{\XINT_tmpa {flexpr}\xintOpp}{{vi}{vii}{viii}{ix}}%
\xintApplyInline{\XINT_tmpa {iiexpr}\xintiiOpp}{{vi}{vii}{viii}{ix}}%
```

11.32 ? as two-way and ?? as three-way conditionals with braced branches

In 1.1, I overload ? with ??, as : will be used for list extraction, problem with (stuff)?{?(1)}{0} for example, one should put a space (stuff)?{ ?(1)}{0} will work. Small idiosyncrasy. (which has been removed in 1.2h, there is no problem anymore with (test)?{?(1)}{0}, however (test)?{?}{!}(x) is not accepted; but (test)?{?(x)}{!(x)} is or even with {?(}{!(}x).)

syntax: ?{yes}{no} and ??{<0}{=0}{>0}.

The difficulty is to recognize the second ? without removing braces as would be the case with standard parsing of operators. Hence the ? operator is intercepted in \XINT_expr_getop_b.

1.2h corrects a bug in \XINT_expr_op_? which in context like (test)?{\foo}{bar} would provoke expansion of \foo, or also with (test)?{}{bar} would result in an error. The fix also solves the (test)?{?(1)}{0} issue mentioned above.

```
\let\XINT_expr_precedence_? \xint_c_x
\def\XINT_expr_op_? #1#2%
   {\XINT_expr_op_?checka #2!\xint_bye\XINT_expr_op_?a #1{#2}}%
\def\XINT_expr_op_?checka #1{\expandafter\XINT_expr_op_?checkb\detokenize{#1}}%
\def\XINT_expr_op_?checkb #1{\if ?#1\expandafter\XINT_expr_op_?checkc
                                     \else\expandafter\xint_bye\fi }%
\def\XINT_expr_op_?checkc #1{\xint_gob_til_! #1\XINT_expr_op_?? !\xint_bye}%
\def\XINT_expr_op_?a #1#2#3%
{%
    \xintiiifNotZero{\XINT_expr_unlock  #1}{\XINT_expr_getnext #2}{\XINT_expr_getnext #3}%
}%
```

```
\let\XINT_flexpr_op_?\XINT_expr_op_?
\let\XINT_iiexpr_op_?\XINT_expr_op_?
\def\XINT_expr_op_?? !\xint_bye\xint_bye\XINT_expr_op_?a #1#2#3#4#5%
{%
     \xintiiifSgn {\XINT_expr_unlock  #1}%
     {\XINT_expr_getnext #3}{\XINT_expr_getnext #4}{\XINT_expr_getnext #5}%
}%
```

11.33 ! as postfix factorial operator

```
\let\XINT_expr_precedence_! \xint_c_x
\def\XINT_expr_op_! #1{\expandafter\XINT_expr_getop
  \csname .=\XINT:NEhook:one\xintFac{\XINT_expr_unlock #1}\endcsname }%
\def\XINT_flexpr_op_! #1{\expandafter\XINT_expr_getop
  \csname .=\XINT:NEhook:one\XINTinFloatFac{\XINT_expr_unlock #1}\endcsname }%
\def\XINT_iiexpr_op_! #1{\expandafter\XINT_expr_getop
  \csname .=\XINT:NEhook:one\xintiiFac{\XINT_expr_unlock #1}\endcsname }%
```

11.34 The A/B[N] mechanism

Releases earlier than 1.1 required the use of braces around A/B[N] input. The [N] is now implemented directly. *BUT* this uses a delimited macro! thus N is not allowed to be itself an expression (I could add it...). \xintE, \xintiiE, and \XINTinFloatE all put #2 in a \numexpr. But attention to the fact that \numexpr stops at spaces separating digits: \the\numexpr 3 + 7 9\relax gives 109\relax !! Hence we have to be careful.

\numexpr will not handle catcode 11 digits, but adding a \detokenize will suddenly make illicit for N to rely on macro expansion.

```
\catcode`[ 11
\let\XINT_expr_precedence_[ \xint_c_vii
\def\XINT_expr_op_[ #1#2]{\expandafter\XINT_expr_getop
                \csname .=\xintE{\XINT_expr_unlock #1}%
                {\xint_zapspaces #2 \xint_gobble_i}\endcsname}%
\def\XINT_iiexpr_op_[ #1#2]{\expandafter\XINT_expr_getop
                \csname .=\xintiiE{\XINT_expr_unlock #1}%
                {\xint_zapspaces #2 \xint_gobble_i}\endcsname}%
\def\XINT_flexpr_op_[ #1#2]{\expandafter\XINT_expr_getop
                \csname .=\XINTinFloatE{\XINT_expr_unlock #1}%
                {\xint_zapspaces #2 \xint_gobble_i}\endcsname}%
\catcode`[ 12
```

11.35 **\XINT_expr_op_`** for recognizing functions

The "onliteral" intercepts is for bool, togl, protect, ... but also for add, mul, seq, etc... Genuine functions have expr, iiexpr and flexpr versions (or only one or two of the three).

With 1.2c "onliteral" is also used to disambiguate variables from functions. However as I use only a \ifcsname test, in order to be able to re-define a variable as function, I move the check for being a function first. Each variable name now has its onliteral_<name> associated macro which is the new way tacit multiplication in front of a parenthesis is implemented. This used to be decided much earlier at the time of \XINT_expr_func.

The advantage of our choices for 1.2c is that the same name can be used for a variable or a function, the parser will apply the correct interpretation which is decided by the presence or not of an opening parenthesis next.

```
\def\XINT_tmpa #1#2#3{%
    \def #1##1%
    {%
        \ifcsname XINT_#3_func_##1\endcsname
          \xint_dothis{\expandafter\expandafter
                      \csname XINT_#3_func_##1\endcsname\romannumeral`&&@#2}\fi
        \ifcsname XINT_expr_onliteral_##1\endcsname
          \xint_dothis{\csname XINT_expr_onliteral_##1\endcsname}\fi
        \xint_orthat{\XINT_expr_unknown_function {##1}%
           \expandafter\XINT_expr_func_unknown\romannumeral`&&@#2}%
   }%
}%
\def\XINT_expr_unknown_function #1{\xintError:removed \xint_gobble_i {#1}}%
\xintFor #1 in {expr,flexpr,iiexpr} \do {%
     \expandafter\XINT_tmpa
                \csname XINT_#1_op_`\expandafter\endcsname
                \csname XINT_#1_oparen\endcsname
                {#1}%
}%
\def\XINT_expr_func_unknown #1#2#3%
    {\expandafter #1\expandafter #2\csname .=0\endcsname }%
```

11.36 The \bool(), \togl(), \protect() pseudo "functions"

bool, togl and protect use delimited macros. They are not true functions, they turn off the parser to gather their "variable".

```
\def\XINT_expr_onliteral_bool #1)%
        {\expandafter\XINT_expr_getop\csname .=\xintBool{#1}\endcsname }%
\def\XINT_expr_onliteral_togl #1)%
        {\expandafter\XINT_expr_getop\csname .=\xintToggle{#1}\endcsname }%
\def\XINT_expr_onliteral_protect #1)%
        {\expandafter\XINT_expr_getop\csname .=\detokenize{#1}\endcsname }%
```

11.37 The \break() function

break is a true function, the parsing via expansion of the succeeding material proceeded via _oparen macros as with any other function.

```
\def\XINT_expr_func_break #1#2#3%
    {\expandafter #1\expandafter #2\csname.=?\romannumeral`&&@\XINT_expr_unlock #3\endcsname }%
\let\XINT_flexpr_func_break \XINT_expr_func_break
\let\XINT_iiexpr_func_break \XINT_expr_func_break
```

11.38 The \qraw(), \qint(), \qfrac(), and \qfloat() "functions"

1.2. adds qint(), qfrac(), qfloat().

1.3c. adds qraw(). Useful to limit impact on TEX memory from abuse of \csname's storage when generating many comma separated values from a loop.

They allow the user to hand over quickly a big number to the parser, spaces not immediately removed but should be harmless in general. The qraw() does no post-processing at all apart complete expansion, useful for comma-separated values, but must be obedient to (non really documented) expected format. Each uses a delimited macro, the closing parenthesis can not emerge from expansion.

```
\def\XINT_expr_onliteral_qint #1)%
        {\expandafter\XINT_expr_getop\csname .=\xintiNum{#1}\endcsname }%
\def\XINT_expr_onliteral_qfrac #1)%
        {\expandafter\XINT_expr_getop\csname .=\xintRaw{#1}\endcsname }%
\def\XINT_expr_onliteral_qfloat #1)%
        {\expandafter\XINT_expr_getop\csname .=\XINTinFloatdigits{#1}\endcsname }%
\def\XINT_expr_onliteral_qraw #1)%
        {\expandafter\XINT_expr_getop\csname .=#1\endcsname }%
```

11.39 The \random() and \qrand() "functions"

1.3b. Function-like syntax but with no argument currently, so let's use fast parsing which requires though the closing parenthesis to be explicit.

```
\def\XINT_expr_onliteral_random #1)%
   {\expandafter\XINT_expr_getop\csname .=\XINTinRandomFloatSdigits\endcsname }%
\def\XINT_expr_onliteral_qrand #1)%
   {\expandafter\XINT_expr_getop\csname .=\XINTinRandomFloatSixteen\endcsname }%
```

11.40 \XINT_expr_op__ for recognizing variables

The 1.1 mechanism for \XINT_expr_var_<varname> has been modified in 1.2c. The <varname> associated macro is now only expanded once, not twice. We arrive here via \XINT_expr_func.

```
\def\XINT_expr_op__  #1% op__ with two _'s
     {%
         \ifcsname XINT_expr_var_#1\endcsname
           \expandafter\xint_firstoftwo
         \else
           \expandafter\xint_secondoftwo
         \fi
         {\expandafter\expandafter\expandafter
          \XINT_expr_getop\csname XINT_expr_var_#1\endcsname}%
         {\XINT_expr_unknown_variable {#1}%
          \expandafter\XINT_expr_getop\csname .=0\endcsname}%
     }%
\def\XINT_expr_unknown_variable #1{\xintError:removed \xint_gobble_i {#1}}%
\let\XINT_flexpr_op__ \XINT_expr_op__
\let\XINT_iiexpr_op__ \XINT_expr_op__
```

11.41 User defined variables: \xintdefvar, \xintdefiivar, \xintdeffloatvar

1.1.

1.2p (2017/12/01). extends \xintdefvar et. al. to accept simultaneous assignments to multiple variables.

1.3c (2018/06/17). Use \xintexprSafeCatcodes (to palliate issue with active semi-colon from Babel+French if in body of a LaTeX document).
And allow usage with both syntaxes name:=expr; or name=expr;. Also the colon may have catcode 11, 12, or 13 with no issue.

Variable names may contain letters, digits, underscores, and must not start with a digit. Names starting with @ or an underscore are reserved.

```
\catcode`* 11
\def\XINT_expr_defvar_one #1#2%
{%
    \XINT_global
    \expandafter\edef\csname XINT_expr_var_#1\endcsname
              {\expandafter\noexpand#2}%
    \XINT_global
    \expandafter\edef\csname XINT_expr_onliteral_#1\endcsname
              {\XINT_expr_precedence_*** *\expandafter\noexpand#2(}%
    \ifxintverbose\xintMessage{xintexpr}{Info}
       {Variable "#1" \ifxintglobaldefs globally \fi
        defined with value \expandafter\XINT_expr_unlock#2.}%
    \fi
}%
\catcode`* 12
\catcode`~\active
\catcode`: 12
\def\XINT_expr_defvar_getname #1:#2~{\endgroup
    \def\XINT_expr_tmpa{#1}\edef\XINT_expr_tmpc{\xintCSVLength{#1}}}%
\def\XINT_expr_defvar #1#2#3;%
{%
    \xintexprRestoreCatcodes
```

Maybe SafeCatcodes was without effect because the colon and the rest are from some earlier macro definition. Give a safe definition to active colon (even if in math mode with a math active colon..).

```
    \begingroup\lccode`~`: \lowercase{\let~}\empty
    \edef\XINT_expr_tmpa{#2}%
    \edef\XINT_expr_tmpa{\xint_zapspaces_o\XINT_expr_tmpa}%
    \expandafter\XINT_expr_defvar_getname
                \detokenize\expandafter{\XINT_expr_tmpa}:~%
    \ifcase\XINT_expr_tmpc
      \xintMessage {xintexpr}{Warning}
      {Aborting: impossible to declare variable with empty name.}%
    \or
     \edef\XINT_expr_tmpb{\romannumeral0#1#3\relax}%
     \XINT_expr_defvar_one\XINT_expr_tmpa\XINT_expr_tmpb
    \else
     \edef\XINT_expr_tmpb
        {\expandafter\XINT_expr_unlock\romannumeral0#1#3\relax}%
     \edef\XINT_expr_tmpd{\xintCSVLength{\XINT_expr_tmpb}}%
     \ifnum\XINT_expr_tmpc=\XINT_expr_tmpd\space
       \xintAssignArray\xintCSVtoList\XINT_expr_tmpa\to\XINT_expr_tmpvar
       \xintAssignArray
         \xintApply\XINT_expr_lockit{\xintCSVtoList\XINT_expr_tmpb}%
       \to\XINT_expr_tmpval
       \def\XINT_expr_tmpd{1}%
       \xintloop
           \expandafter\XINT_expr_defvar_one
```

```
            \csname XINT_expr_tmpvar\XINT_expr_tmpd\expandafter\endcsname
            \csname XINT_expr_tmpval\XINT_expr_tmpd\endcsname
        \ifnum\XINT_expr_tmpd<\XINT_expr_tmpc\space
            \edef\XINT_expr_tmpd{\the\numexpr\XINT_expr_tmpd+1}%
        \repeat
        \xintRelaxArray\XINT_expr_tmpvar
        \xintRelaxArray\XINT_expr_tmpval
      \else
        \xintMessage {xintexpr}{Warning}
         {Aborting: mismatch between number of variables (\XINT_expr_tmpc)
          and number of values (\XINT_expr_tmpd).}%
      \fi
     \fi
}%
\catcode`~ 3
\catcode`: 11
```

This SafeCatcodes is mainly in the hope that semi-colon ending the expression can still be sanitized.

```
\def\xintdefvar      {\xintexprSafeCatcodes\xintdefvar_a}%
\def\xintdefiivar    {\xintexprSafeCatcodes\xintdefiivar_a}%
\def\xintdeffloatvar {\xintexprSafeCatcodes\xintdeffloatvar_a}%
\def\xintdefvar_a      #1={\XINT_expr_defvar\xintbareeval      {#1}}%
\def\xintdefiivar_a    #1={\XINT_expr_defvar\xintbareiieval    {#1}}%
\def\xintdeffloatvar_a #1={\XINT_expr_defvar\xintbarefloateval {#1}}%
```

11.42 \xintunassignvar

1.2e. Currently not possible to genuinely ``undefine'' a variable, all we can do is to let it stand for zero and generate an error. The reason is that I chose to use \ifcsname tests in \XINT_expr_op__ and \XINT_expr_op_`.

```
\def\xintunassignvar #1{%
   \edef\XINT_expr_tmpa{#1}%
   \edef\XINT_expr_tmpa {\xint_zapspaces_o\XINT_expr_tmpa}%
   \ifcsname XINT_expr_var_\XINT_expr_tmpa\endcsname
       \ifnum\expandafter\xintLength\expandafter{\XINT_expr_tmpa}=\@ne
         \expandafter\xintnewdummy\XINT_expr_tmpa
       \else
       \XINT_global
       \expandafter\edef\csname XINT_expr_var_\XINT_expr_tmpa\endcsname
           {\csname .=0\endcsname\noexpand\XINT_expr_undefined {\XINT_expr_tmpa}}%
       \XINT_global
       \expandafter\edef\csname XINT_expr_onliteral_\XINT_expr_tmpa\endcsname
           {\csname .=0\endcsname\noexpand\XINT_expr_undefined {\XINT_expr_tmpa}*}%
         \ifxintverbose\xintMessage {xintexpr}{Info}
           {Variable \XINT_expr_tmpa\space has been
            \ifxintglobaldefs globally \fi ``unassigned''.}%
         \fi
       \fi
   \else
       \xintMessage {xintexpr}{Warning}
```

```
            {Error: there was no such variable \XINT_expr_tmpa\space to unassign.}%
    \fi
}%
\def\XINT_expr_undefined #1{\xintError:replaced_by_zero\xint_gobble_i {#1}}%
```

11.43 seq and the implementation of dummy variables

All of seq, add, mul, rseq, etc... (actually all of the extensive changes from xintexpr 1.09n to 1.1) was done around June 15-25th 2014, but the problem is that I did not document the code enough, and I had a hard time understanding in October what I had done in June. Despite the lesson, again being short on time, I do not document enough my current understanding of the innards of the beast...

I added subs, and iter in October (also the [:n], [n:] list extractors), proving I did at least understand a bit (or rather could imitate) my earlier code (but don't ask me to explain \xintNewExpr !)

The \XINT_expr_onliteral_seq_a parses: "expression, variable=list)" (when it is called the opening (has been swallowed, and it looks for the ending one.) Both expression and list may themselves contain parentheses and commas, we allow nesting. For example "x^2,x=1..10)", at the end of seq_a we have {variable{expression}}{list}, in this example {x{x^2}}{1..10}, or more complicated "seq(add(y,y=1..x),x=1..10)" will work too. The variable is a single lowercase Latin letter.

The complications with \xint_c_xviii in seq_f is for the recurrent thing that we don't know in what type of expressions we are, hence we must move back up, with some loss of efficiency (superfluous check for minus sign, etc...). But the code manages simultaneously expr, flexpr and iiexpr.

11.43.1 All letters usable as dummy variables

The nil variable was introduced in 1.1 but isn't used under that name. However macros handling a..[d]..b, or for seq with dummy variable where omit has omitted everyting may in practice inject a nil value as current number.

1.2c has changed the way variables are disambiguated from functions and for this it has added here the definitions of \XINT_expr_onliteral_<name>.

In 1.1 a letter variable say X was acting as a delimited macro looking for !X{stuff} and then would expand the stuff inside a \csname.=...\endcsname. I don't think I used the possibilities this opened and the 1.2c version has stuff _already_ encapsulated thus a single token. Only one expansion, not two is then needed in \XINT_expr_op__.

I had to accordingly modify seq, add, mul and subs, but fortunately realized that the @, @1, etc... variables for rseq, rrseq and iter already had been defined in the way now also followed by the Latin letters as dummy variables.

The 1.2e \XINT_expr_makedummy was adjoined \xintnewdummy by 1.2k for a public interface. It should not be used with multi-letter argument. The add, mul, seq, etc... can only work with one-letter long dummy variable. And this will almost certainly not change.

Also 1.2e does the tacit multiplication x(stuff)->x*(stuff) in its higher precedence form. Things are easy now that variables always fetch a single already locked value \.=<number>.

The tacit multiplication in case of the ``nil'' variable doesn't make much sense but we do it anyhow.

```
\catcode`* 11
\def\XINT_expr_makedummy #1%
{%
   \XINT_global
   \expandafter\def\csname XINT_expr_var_#1\endcsname ##1\relax !#1##2%
      {##2##1\relax !#1##2}%
   \XINT_global
   \expandafter\def\csname XINT_expr_onliteral_#1\endcsname ##1\relax !#1##2%
      {\XINT_expr_precedence_*** *##2(##1\relax !#1##2}%
}%
\xintApplyUnbraced \XINT_expr_makedummy {abcdefghijklmnopqrstuvwxyz}%
\xintApplyUnbraced \XINT_expr_makedummy {ABCDEFGHIJKLMNOPQRSTUVWXYZ}%
\def\xintnewdummy #1{%
    \XINT_expr_makedummy{#1}%
    \ifxintverbose\xintMessage {xintexpr}{Info}%
       {#1 (with letter catcode) now
        \ifxintglobaldefs globally \fi usable as dummy variable.}%
    \fi
}%
\edef\XINT_expr_var_nil  {\expandafter\noexpand\csname .= \endcsname}%
\edef\XINT_expr_onliteral_nil
      {\XINT_expr_precedence_*** *\expandafter\noexpand\csname .= \endcsname (}%
\catcode`* 12
```

11.43.2 \omit() and \abort()

June 24 and 25, 2014.

Added comments 2015/11/13:

Et la documentation ? on n'y comprend plus rien. Trop rusé.

\def\XINT_expr_var_omit #1\relax !{1^C!{}{}{}{}\.=!\relax !}

\def\XINT_expr_var_abort #1\relax !{1^C!{}{}{}{}\.=^\relax !}

C'était accompagné de \XINT_expr_precedence_^C=0 et d'un hack au sein même des macros until de plus bas niveau.

Le mécanisme sioux était le suivant: ^C est déclaré comme un opérateur de précédence nulle. Lorsque le parseur trouve un "omit" dans un seq ou autre, il va insérer dans le stream \XINT_expr_getop suivi du texte de remplacement. Donc ici on avait un 1 comme place holder, puis l'opérateur ^C. Celui-ci étant de précédence zéro provoque la finalisation de tous les calculs antérieurs dans le sous-bareeval. Mais j'ai dû hacker le until_end_b (et le until_)_b) qui confronté à ^C, va se relancer à zéro, le getnext va trouver le !{}{}{}{}\.=! et ensuite il y aura \relax, et le résultat sera \.=! pour omit ou \.=^ pour abort. Les routines des boucles seq, iter, etc... peuvent alors repérer le ! ou ^ et agir en conséquence (un long paragraphe pour ne décrire que partiellement une ou deux lignes de codes...).

Mais ^C a été fait alors que je n'avais pas encore les variables muettes. Je dois trouver autre chose, car seq(2^C, C=1..5) est alors impossible. De toute façon ce ^C était à usage interne uniquement.

Il me faut un symbole d'opérateur qui ne rentre pas en conflit. Bon je vais prendre !?. Ensuite au lieu de hacker until_end, il vaut mieux lui donner précédence 2 (mais ça ne pourra pas marcher à l'intérieur de parenthèses il faut d'abord les fermer manuellement) et lui associer un simplement

un op spécial. Je n'avais pas fait cela peut-être pour éviter d'avoir à définir plusieurs macros. Le #1 dans la définition de \XINT_expr_op_!? est le résultat de l'évaluation forcée précédente.
  Attention que les premier ! doiventt être de catcode 12 sinon ils signalent une sous-expression qui déclenche une multiplication tacite.

```
\edef\XINT_expr_var_omit  #1\relax !{1\string !?!\relax !}%
\edef\XINT_expr_var_abort #1\relax !{1\string !?^\relax !}%
\def\XINT_expr_op_!? #1#2\relax {\expandafter\XINT_expr_foundend\csname .=#2\endcsname}%
\let\XINT_iiexpr_op_!? \XINT_expr_op_!?
\let\XINT_flexpr_op_!? \XINT_expr_op_!?
```

11.43.3 The special variables @, @1, @2, @3, @4, @@, @@(1), ..., @@@, @@@(1), ... for recursion

October 2014: I had completely forgotten what the @@@ etc... stuff were supposed to do: this is for nesting recursions! (I was mad back in June). @@(N) gives the Nth back, @@@(N) gives the Nth back of the higher recursion!
  1.2c adds the needed "onliteral" now that tacit multiplication between a variable and a (has a new mechanism. 1.2e does this tacit multiplication with higher precedence.
  For the record, the ~ has catcode 3 in this code.

```
\catcode`? 3 \catcode`* 11
\def\XINT_expr_var_@ #1~#2{#2#1~#2}%
\expandafter\let\csname XINT_expr_var_@1\endcsname \XINT_expr_var_@
\expandafter\def\csname XINT_expr_var_@2\endcsname #1~#2#3{#3#1~#2#3}%
\expandafter\def\csname XINT_expr_var_@3\endcsname #1~#2#3#4{#4#1~#2#3#4}%
\expandafter\def\csname XINT_expr_var_@4\endcsname #1~#2#3#4#5{#5#1~#2#3#4#5}%
\def\XINT_expr_onliteral_@ #1~#2{\XINT_expr_precedence_*** *#2(#1~#2}%
\expandafter\let\csname XINT_expr_onliteral_@1\endcsname \XINT_expr_onliteral_@
\expandafter\def\csname XINT_expr_onliteral_@2\endcsname #1~#2#3%
           {\XINT_expr_precedence_*** *#3(#1~#2#3}%
\expandafter\def\csname XINT_expr_onliteral_@3\endcsname #1~#2#3#4%
           {\XINT_expr_precedence_*** *#4(#1~#2#3#4}%
\expandafter\def\csname XINT_expr_onliteral_@4\endcsname #1~#2#3#4#5%
           {\XINT_expr_precedence_*** *#5(#1~#2#3#4#5}%
\catcode`* 12
\def\XINT_expr_func_@@ #1#2#3#4~#5?%
{%
   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
                              {\xintNum{\XINT_expr_unlock#3}}{#5}#4~#5?%
}%
\def\XINT_expr_func_@@@ #1#2#3#4~#5~#6?%
{%
   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
   {\xintNum{\XINT_expr_unlock#3}}{#6}#4~#5~#6?%
}%
\def\XINT_expr_func_@@@@ #1#2#3#4~#5~#6~#7?%
{%
   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
   {\xintNum{\XINT_expr_unlock#3}}{#7}#4~#5~#6~#7?%
}%
\let\XINT_flexpr_func_@@\XINT_expr_func_@@
\let\XINT_flexpr_func_@@@\XINT_expr_func_@@@
\let\XINT_flexpr_func_@@@@\XINT_expr_func_@@@@
```

```
\def\XINT_iiexpr_func_@@ #1#2#3#4~#5?%
{%
    \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
    {\XINT_expr_unlock#3}{#5}#4~#5?%
}%
\def\XINT_iiexpr_func_@@@ #1#2#3#4~#5~#6?%
{%
    \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
    {\XINT_expr_unlock#3}{#6}#4~#5~#6?%
}%
\def\XINT_iiexpr_func_@@@@ #1#2#3#4~#5~#6~#7?%
{%
    \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
    {\XINT_expr_unlock#3}{#7}#4~#5~#6~#7?%
}%
\catcode`? 11
```

11.43.4 \XINT_expr_onliteral_seq

```
\def\XINT_expr_onliteral_seq
 {\expandafter\XINT_expr_onliteral_seq_f\romannumeral`&&@\XINT_expr_onliteral_seq_a {}}%
\def\XINT_expr_onliteral_seq_f #1#2{\xint_c_xviii `{seqx}#2)\relax #1}%
```

11.43.5 \XINT_expr_onliteral_seq_a

```
\def\XINT_expr_onliteral_seq_a #1#2,%
{%
    \ifcase\XINT_isbalanced_a \relax #1#2(\xint_bye)\xint_bye
           \expandafter\XINT_expr_onliteral_seq_c
        \or\expandafter\XINT_expr_onliteral_seq_b
      \else\expandafter\xintError:we_are_doomed
    \fi {#1#2},%
}%
\def\XINT_expr_onliteral_seq_b #1,{\XINT_expr_onliteral_seq_a {#1,}}%
\def\XINT_expr_onliteral_seq_c #1,#2#3% #3 pour absorber le =
{%
    \XINT_expr_onliteral_seq_d {#2{#1}}{}%
}%
\def\XINT_expr_onliteral_seq_d #1#2#3)%
{%
    \ifcase\XINT_isbalanced_a \relax #2#3(\xint_bye)\xint_bye
        \or\expandafter\XINT_expr_onliteral_seq_e
       \else\expandafter\xintError:we_are_doomed
    \fi
    {#1}{#2#3}%
}%
\def\XINT_expr_onliteral_seq_e #1#2{\XINT_expr_onliteral_seq_d {#1}{#2)}}%
```

11.43.6 \XINT_isbalanced_a for \XINT_expr_onliteral_seq_a

Expands to \xint_c_mone in case a closing) had no opening (matching it, to \@ne if opening) had no closing) matching it, to \z@ if expression was balanced.

```
% use as \XINT_isbalanced_a \relax #1(\xint_bye)\xint_bye
\def\XINT_isbalanced_a #1({\XINT_isbalanced_b #1)\xint_bye }%
```

```
\def\XINT_isbalanced_b #1)#2%
   {\xint_bye #2\XINT_isbalanced_c\xint_bye\XINT_isbalanced_error }%
```

if #2 is not \xint_bye, a) was found, but there was no (. Hence error -> -1

```
\def\XINT_isbalanced_error #1)\xint_bye {\xint_c_mone}%
```

#2 was \xint_bye, was there a) in original #1?

```
\def\XINT_isbalanced_c\xint_bye\XINT_isbalanced_error #1%
    {\xint_bye #1\XINT_isbalanced_yes\xint_bye\XINT_isbalanced_d #1}%
```

#1 is \xint_bye, there was never (nor) in original #1, hence OK.

```
\def\XINT_isbalanced_yes\xint_bye\XINT_isbalanced_d\xint_bye )\xint_bye {\xint_c_ }%
```

#1 is not \xint_bye, there was indeed a (in original #1. We check if we see a). If we do, we then loop until no (nor) is to be found.

```
\def\XINT_isbalanced_d #1)#2%
   {\xint_bye #2\XINT_isbalanced_no\xint_bye\XINT_isbalanced_a #1#2}%
```

#2 was \xint_bye, we did not find a closing) in original #1. Error.

```
\def\XINT_isbalanced_no\xint_bye #1\xint_bye\xint_bye {\xint_c_i }%
```

11.43.7 \XINT_allexpr_func_seqx

1.2c uses \xintthebareval, ... which strangely were not available at 1.1 time. This spares some tokens from \XINT_expr_seq:_d and cousins. Also now variables have changed their mode of operation they pick only one token which must be an already encapsulated value.

In \XINT_allexp_seqx, #2 is the list, evaluated and encapsulated, #3 is the dummy variable, #4 is the expression to evaluate repeatedly.

A special case is a list generated by <variable>++: then #2 is {\.=+\.=<start>}.

```
\def\XINT_expr_func_seqx   #1#2{\XINT_allexpr_seqx \xintthebareeval }%
\def\XINT_flexpr_func_seqx #1#2{\XINT_allexpr_seqx \xintthebarefloateval}%
\def\XINT_iiexpr_func_seqx #1#2{\XINT_allexpr_seqx \xintthebareiieval }%
\def\XINT_allexpr_seqx #1#2#3#4%
{%
    \expandafter \XINT_expr_getop
    \csname .=\expandafter\XINT_expr_seq:_aa
           \romannumeral`&&@\XINT_expr_unlock #2!{#1#4\relax !#3}\endcsname
}%
\def\XINT_expr_seq:_aa #1{\if +#1\expandafter\XINT_expr_seq:_A\else
                              \expandafter\XINT_expr_seq:_a\fi #1}%
```

11.43.8 Evaluation over list, \XINT_expr_seq:_a with break, abort, omit

The #2 here is \...bareeval <expression>\relax !<variable name>. The #1 is a comma separated list of values to assign to the dummy variable. The \XINT_expr_seq_empty? intervenes immediately after handling of firstvalue.

1.2c has rewritten to a large extent this and other similar loops because the dummy variables now fetch a single encapsulated token (apart from a good means to lose a few hours needlessly -- as I have had to rewrite and review most everything, this change could make the thing more efficient if the same variable is used many times in an expression, but we are talking micro-seconds here anyhow.)

```
\def\XINT_expr_seq:_a #1!#2{\expandafter\XINT_expr_seq_empty?
                             \romannumeral0\XINT_expr_seq:_b {#2}#1,^,}%
\def\XINT_expr_seq:_b #1#2#3,{%
    \if  ,#2\xint_dothis\XINT_expr_seq:_noop\fi
    \if  ^#2\xint_dothis\XINT_expr_seq:_end\fi
    \xint_orthat{\expandafter\XINT_expr_seq:_c}\csname.=#2#3\endcsname {#1}%
}%
\def\XINT_expr_seq:_noop\csname.=,#1\endcsname #2{\XINT_expr_seq:_b {#2}#1,}%
\def\XINT_expr_seq:_end \csname.=^\endcsname #1{}%
\def\XINT_expr_seq:_c #1#2{\expandafter\XINT_expr_seq:_d\romannumeral`&&@#2#1{#2}}%
\def\XINT_expr_seq:_d #1{\if #1^\xint_dothis\XINT_expr_seq:_abort\fi
                          \if #1?\xint_dothis\XINT_expr_seq:_break\fi
                          \if #1!\xint_dothis\XINT_expr_seq:_omit\fi
                          \xint_orthat{\XINT_expr_seq:_goon #1}}%
\def\XINT_expr_seq:_abort #1!#2#3#4#5^,{}%
\def\XINT_expr_seq:_break #1!#2#3#4#5^,{,#1}%
\def\XINT_expr_seq:_omit  #1!#2#3#4{\XINT_expr_seq:_b {#4}}%
\def\XINT_expr_seq:_goon  #1!#2#3#4{,#1\XINT_expr_seq:_b {#4}}%
```

If all is omitted or list is empty, _empty? will fetch within the ##1 a \endcsname token and construct "nil" via <space>\endcsname, if not ##1 will be a comma and the gobble will swallow the space token and the extra \endcsname.

```
\def\XINT_expr_seq_empty? #1{%
\def\XINT_expr_seq_empty? ##1{\if ,##1\expandafter\xint_gobble_i\fi #1\endcsname }}%
\XINT_expr_seq_empty? { }%
```

11.43.9 Evaluation over ++ generated lists with \XINT_expr_seq:_A

This is for index lists generated by n++. The starting point will have been replaced by its ceil (added: in fact with version 1.1. the ceil was not yet evaluated, but _var_<letter> did an expansion of what they fetch). We use \numexpr rather than \xintInc, hence the indexing is limited to small integers.

The 1.2c version of n++ produces a #1 here which is already a single \.=<value> token.

```
\def\XINT_expr_seq:_A +#1!%
   {\expandafter\XINT_expr_seq_empty?\romannumeral0\XINT_expr_seq:_D #1}%
\def\XINT_expr_seq:_D #1#2{\expandafter\XINT_expr_seq:_E\romannumeral`&&@#2#1{#2}}%
\def\XINT_expr_seq:_E #1{\if #1^\xint_dothis\XINT_expr_seq:_Abort\fi
                          \if #1?\xint_dothis\XINT_expr_seq:_Break\fi
                          \if #1!\xint_dothis\XINT_expr_seq:_Omit\fi
                          \xint_orthat{\XINT_expr_seq:_Goon #1}}%
\def\XINT_expr_seq:_Abort #1!#2#3#4{}%
\def\XINT_expr_seq:_Break #1!#2#3#4{,#1}%
\def\XINT_expr_seq:_Omit  #1!#2#3%
    {\expandafter\XINT_expr_seq:_D
        \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
\def\XINT_expr_seq:_Goon  #1!#2#3%
    {,#1\expandafter\XINT_expr_seq:_D
        \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
```

11.44 \add(), \mul()

1.2c uses more directly the \xintiiAdd etc... macros and has opxadd/opxmul rather than a single opx. This is less conceptual as I use explicitely the associated macro names for +, * but this makes other things more efficient, and the code more readable.

```
\def\XINT_expr_onliteral_add
 {\expandafter\XINT_expr_onliteral_add_f\romannumeral`&&@\XINT_expr_onliteral_seq_a {}}%
\def\XINT_expr_onliteral_add_f #1#2{\xint_c_xviii `{opxadd}#2)\relax #1}%
\def\XINT_expr_onliteral_mul
 {\expandafter\XINT_expr_onliteral_mul_f\romannumeral`&&@\XINT_expr_onliteral_seq_a {}}%
\def\XINT_expr_onliteral_mul_f #1#2{\xint_c_xviii `{opxmul}#2)\relax #1}%
```

11.44.1 \XINT_expr_func_opxadd, \XINT_flexpr_func_opxadd, \XINT_iiexpr_func_opxadd and same for mul

modified 1.2c.

```
\def\XINT_expr_func_opxadd   #1#2{\XINT_allexpr_opx \xintbareeval      {\xintAdd 0}}%
\def\XINT_flexpr_func_opxadd #1#2{\XINT_allexpr_opx \xintbarefloateval {\XINTinFloatAdd 0}}%
\def\XINT_iiexpr_func_opxadd #1#2{\XINT_allexpr_opx \xintbareiieval    {\xintiiAdd 0}}%
\def\XINT_expr_func_opxmul   #1#2{\XINT_allexpr_opx \xintbareeval      {\xintMul 1}}%
\def\XINT_flexpr_func_opxmul #1#2{\XINT_allexpr_opx \xintbarefloateval {\XINTinFloatMul 1}}%
\def\XINT_iiexpr_func_opxmul #1#2{\XINT_allexpr_opx \xintbareiieval    {\xintiiMul 1}}%
```

#1=bareval etc, #2={Add0} ou {Mul1}, #3=liste encapsulée, #4=la variable, #5=expression

```
\def\XINT_allexpr_opx #1#2#3#4#5%
{%
    \expandafter\XINT_expr_getop
    \csname.=\romannumeral`&&@\expandafter\XINT_expr_op:_a
             \romannumeral`&&@\XINT_expr_unlock #3!{#1#5\relax !#4}{#2}\endcsname
}%
\def\XINT_expr_op:_a #1!#2#3{\XINT_expr_op:_b #3{#2}#1,^,}%
```

#2 in \XINT_expr_op:_b is the partial result of computation so far, not locked. A noop with have #4=, and #5 the next item which we need to recover. No need to be very efficient for that in op:_noop. In op:_d, #4 is \xintAdd or similar.

```
\def\XINT_expr_op:_b #1#2#3#4#5,{%
    \if  ,#4\xint_dothis\XINT_expr_op:_noop\fi
    \if  ^#4\xint_dothis\XINT_expr_op:_end\fi
    \xint_orthat{\expandafter\XINT_expr_op:_c}\csname.=#4#5\endcsname {#3}#1{#2}%
}%
\def\XINT_expr_op:_c #1#2#3#4%
   {\expandafter\XINT_expr_op:_d\romannumeral0#2#1#3{#4}{#2}}%
\def\XINT_expr_op:_d #1!#2#3#4#5%
    {\expandafter\XINT_expr_op:_b\expandafter #4\expandafter
                {\romannumeral`&&@\XINT:NEhook:two#4{\XINT_expr_unlock#1}{#5}}}%
```

The replacement text had expr_seq:_b rather than expr_op:_b due to a left-over from copy-paste. This made add and mul fail with an empty range for the variable (or "nil" in the list of values). Fixed in 1.2h.

```
\def\XINT_expr_op:_noop\csname.=,#1\endcsname #2#3#4{\XINT_expr_op:_b #3{#4}{#2}#1,}%
\def\XINT_expr_op:_end \csname.=^\endcsname #1#2#3{#3}%
```

11.45 \subs()

Got simpler with 1.2c as now the dummy variable fetches an already encapsulated value, which is anyhow the form in which we get it.

```
\def\XINT_expr_onliteral_subs
 {\expandafter\XINT_expr_onliteral_subs_f\romannumeral`&&@\XINT_expr_onliteral_seq_a {}}%
\def\XINT_expr_onliteral_subs_f #1#2{\xint_c_xviii `{subx}#2)\relax #1}%
\def\XINT_expr_func_subx   #1#2{\XINT_allexpr_subx \xintbareeval }%
\def\XINT_flexpr_func_subx #1#2{\XINT_allexpr_subx \xintbarefloateval}%
\def\XINT_iiexpr_func_subx #1#2{\XINT_allexpr_subx \xintbareiieval }%
\def\XINT_allexpr_subx #1#2#3#4% #2 is the value to assign to the dummy variable
{% #3 is the dummy variable, #4 is the expression to evaluate
    \expandafter\expandafter\expandafter\XINT_expr_getop
    \expandafter\XINT_expr_subx:_end\romannumeral0#1#4\relax !#3#2%
}%
\def\XINT_expr_subx:_end #1!#2#3{#1}%
```

11.46 \rseq()

When func_rseq has its turn, initial segment has been scanned by oparen, the ; mimicking the rôle of a closing parenthesis, and stopping further expansion. Notice that the ; is discovered during standard parsing mode, it may be for example {;} or arise from expansion as rseq does not use a delimited macro to locate it.

Here and in rrseq and iter, 1.2c adds also use of \xintthebareeval, etc...

```
\def\XINT_expr_func_rseq   {\XINT_allexpr_rseq \xintbareeval      \xintthebareeval      }%
\def\XINT_flexpr_func_rseq {\XINT_allexpr_rseq \xintbarefloateval \xintthebarefloateval }%
\def\XINT_iiexpr_func_rseq {\XINT_allexpr_rseq \xintbareiieval    \xintthebareiieval    }%
\def\XINT_allexpr_rseq #1#2#3%
{%
    \expandafter\XINT_expr_rseqx\expandafter #1\expandafter#2\expandafter
    #3\romannumeral`&&@\XINT_expr_onliteral_seq_a {}%
}%
```

11.46.1 \XINT_expr_rseqx

The (#5) is for ++ mechanism which must have its closing parenthesis.

```
\def\XINT_expr_rseqx #1#2#3#4#5%
{%
    \expandafter\XINT_expr_rseqy\romannumeral0#1(#5)\relax #3#4#2%
}%
```

11.46.2 \XINT_expr_rseqy

#1=valeurs pour variable (locked), #2=toutes les valeurs initiales (csv,locked), #3=variable, #4=expr, #5=\xintthebareeval ou \xintthebarefloateval ou \xintthebareiieval

```
\def\XINT_expr_rseqy #1#2#3#4#5%
{%
    \expandafter \XINT_expr_getop
    \csname .=\XINT_expr_unlock #2%
    \expandafter\XINT_expr_rseq:_aa
                \romannumeral`&&@\XINT_expr_unlock #1!{#5#4\relax !#3}#2\endcsname
}%
\def\XINT_expr_rseq:_aa #1{\if +#1\expandafter\XINT_expr_rseq:_A\else
                                  \expandafter\XINT_expr_rseq:_a\fi #1}%
```

11.46.3 \XINT_expr_rseq:_a etc...

```
\def\XINT_expr_rseq:_a #1!#2#3{\XINT_expr_rseq:_b {#3}{#2}#1,^,}%
\def\XINT_expr_rseq:_b #1#2#3#4,{%
     \if ,#3\xint_dothis\XINT_expr_rseq:_noop\fi
     \if ^#3\xint_dothis\XINT_expr_rseq:_end\fi
     \xint_orthat{\expandafter\XINT_expr_rseq:_c}\csname.=#3#4\endcsname
     {#1}{#2}%
}%
\def\XINT_expr_rseq:_noop\csname.=,#1\endcsname #2#3{\XINT_expr_rseq:_b {#2}{#3}#1,}%
\def\XINT_expr_rseq:_end \csname.=^\endcsname #1#2{}%
\def\XINT_expr_rseq:_c #1#2#3%
   {\expandafter\XINT_expr_rseq:_d\romannumeral`&&@#3#1~#2{#3}}%
\def\XINT_expr_rseq:_d #1{%
    \if ^#1\xint_dothis\XINT_expr_rseq:_abort\fi
    \if ?#1\xint_dothis\XINT_expr_rseq:_break\fi
    \if !#1\xint_dothis\XINT_expr_rseq:_omit\fi
    \xint_orthat{\XINT_expr_rseq:_goon #1}}%
\def\XINT_expr_rseq:_goon  #1!#2#3~#4#5{,#1\expandafter\XINT_expr_rseq:_b
        \romannumeral0\XINT_expr_lockit {#1}{#5}}%
\def\XINT_expr_rseq:_omit  #1!#2#3~{\XINT_expr_rseq:_b }%
\def\XINT_expr_rseq:_abort #1!#2#3~#4#5#6^,{}%
\def\XINT_expr_rseq:_break #1!#2#3~#4#5#6^,{,#1}%
```

11.46.4 \XINT_expr_rseq:_A etc...

n++ for rseq. With 1.2c dummy variables pick a single token.

```
\def\XINT_expr_rseq:_A +#1!#2#3{\XINT_expr_rseq:_D #1#3{#2}}%
\def\XINT_expr_rseq:_D #1#2#3%
   {\expandafter\XINT_expr_rseq:_E\romannumeral`&&@#3#1~#2{#3}}%
\def\XINT_expr_rseq:_E #1{\if #1^\xint_dothis\XINT_expr_rseq:_Abort\fi
                         \if #1?\xint_dothis\XINT_expr_rseq:_Break\fi
                         \if #1!\xint_dothis\XINT_expr_rseq:_Omit\fi
                         \xint_orthat{\XINT_expr_rseq:_Goon #1}}%
\def\XINT_expr_rseq:_Goon  #1!#2#3~#4#5%
    {,#1\expandafter\XINT_expr_rseq:_D
        \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
     \romannumeral0\XINT_expr_lockit{#1}{#5}}%
```

```
\def\XINT_expr_rseq:_Omit  #1!#2#3~%#4#5%
    {\expandafter\XINT_expr_rseq:_D
       \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname }%
\def\XINT_expr_rseq:_Abort #1!#2#3~#4#5{}%
\def\XINT_expr_rseq:_Break #1!#2#3~#4#5{,#1}%
```

11.47 \iter()

Prior to 1.2g, the iter keyword was what is now called iterr, analogous with rrseq. Somehow I forgot an iter functioning like rseq with the sole difference of printing only the last iteration. Both rseq and iter work well with list selectors, as @ refers to the whole comma separated sequence of the initial values. I have thus deliberately done the backwards incompatible renaming of iter to iterr, and the new iter.

```
\def\XINT_expr_func_iter   {\XINT_allexpr_iter \xintbareeval      \xintthebareeval      }%
\def\XINT_flexpr_func_iter {\XINT_allexpr_iter \xintbarefloateval \xintthebarefloateval }%
\def\XINT_iiexpr_func_iter {\XINT_allexpr_iter \xintbareiieval    \xintthebareiieval    }%
\def\XINT_allexpr_iter #1#2#3%
{%
    \expandafter\XINT_expr_iterx\expandafter #1\expandafter#2\expandafter
    #3\romannumeral`&&@\XINT_expr_onliteral_seq_a {}%
}%
```

11.47.1 \XINT_expr_iterx

The (#5) is for ++ mechanism which must have its closing parenthesis.

```
\def\XINT_expr_iterx #1#2#3#4#5%
{%
    \expandafter\XINT_expr_itery\romannumeral0#1(#5)\relax #3#4#2%
}%
```

11.47.2 \XINT_expr_itery

#1=valeurs pour variable (locked), #2=toutes les valeurs initiales (csv,locked), #3=variable, #4=expr, #5=\xintthebareeval ou \xintthebarefloateval ou \xintthebareiieval

```
\def\XINT_expr_itery #1#2#3#4#5%
{%
    \expandafter \XINT_expr_getop
    \csname .=%
    \expandafter\XINT_expr_iter:_aa
    \romannumeral`&&@\XINT_expr_unlock #1!{#5#4\relax !#3}#2\endcsname
}%
\def\XINT_expr_iter:_aa #1{\if +#1\expandafter\XINT_expr_iter:_A\else
                                   \expandafter\XINT_expr_iter:_a\fi #1}%
```

11.47.3 \XINT_expr_iter:_a etc...

```
\def\XINT_expr_iter:_a #1!#2#3{\XINT_expr_iter:_b {#3}{#2}#1,^,}%
\def\XINT_expr_iter:_b #1#2#3#4,{%
     \if ,#3\xint_dothis\XINT_expr_iter:_noop\fi
     \if ^#3\xint_dothis\XINT_expr_iter:_end\fi
     \xint_orthat{\expandafter\XINT_expr_iter:_c}%
     \csname.=#3#4\endcsname {#1}{#2}%
}%
\def\XINT_expr_iter:_noop\csname.=,#1\endcsname #2#3{\XINT_expr_iter:_b {#2}{#3}#1,}%
\def\XINT_expr_iter:_end \csname.=^\endcsname #1#2{\XINT_expr:_unlock #1}%
\def\XINT_expr_iter:_c #1#2#3%
   {\expandafter\XINT_expr_iter:_d\romannumeral`&&@#3#1~#2{#3}}%
\def\XINT_expr_iter:_d #1{%
    \if ^#1\xint_dothis\XINT_expr_iter:_abort\fi
    \if ?#1\xint_dothis\XINT_expr_iter:_break\fi
    \if !#1\xint_dothis\XINT_expr_iter:_omit\fi
    \xint_orthat{\XINT_expr_iter:_goon #1}}%
\def\XINT_expr_iter:_goon  #1!#2#3~#4#5%
   {\expandafter\XINT_expr_iter:_b\romannumeral0\XINT_expr_lockit {#1}{#5}}%
\def\XINT_expr_iter:_omit  #1!#2#3~{\XINT_expr_iter:_b }%
\def\XINT_expr_iter:_abort #1!#2#3~#4#5#6^,{\XINT_expr_unlock #4}%
\def\XINT_expr_iter:_break #1!#2#3~#4#5#6^,{#1}%
```

11.47.4 \XINT_expr_iter:_A etc...

n++ for iter. With 1.2c dummy variables pick a single token.

```
\def\XINT_expr_iter:_A +#1!#2#3{\XINT_expr_iter:_D #1#3{#2}}%
\def\XINT_expr_iter:_D #1#2#3%
   {\expandafter\XINT_expr_iter:_E\romannumeral`&&@#3#1~#2{#3}}%
\def\XINT_expr_iter:_E #1{\if #1^\xint_dothis\XINT_expr_iter:_Abort\fi
                          \if #1?\xint_dothis\XINT_expr_iter:_Break\fi
                          \if #1!\xint_dothis\XINT_expr_iter:_Omit\fi
                          \xint_orthat{\XINT_expr_iter:_Goon #1}}%
\def\XINT_expr_iter:_Goon  #1!#2#3~#4#5%
    {\expandafter\XINT_expr_iter:_D
     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
     \romannumeral0\XINT_expr_lockit{#1}{#5}}%
\def\XINT_expr_iter:_Omit  #1!#2#3~%#4#5%
    {\expandafter\XINT_expr_iter:_D
     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname }%
\def\XINT_expr_iter:_Abort #1!#2#3~#4#5{\XINT_expr:_unlock #4}%
\def\XINT_expr_iter:_Break #1!#2#3~#4#5{#1}%
```

11.48 \rrseq()

When func_rrseq has its turn, initial segment has been scanned by oparen, the ; mimicking the rôle of a closing parenthesis, and stopping further expansion.

```
\def\XINT_expr_func_rrseq   {\XINT_allexpr_rrseq \xintbareeval      \xintthebareeval      }%
\def\XINT_flexpr_func_rrseq {\XINT_allexpr_rrseq \xintbarefloateval \xintthebarefloateval }%
\def\XINT_iiexpr_func_rrseq {\XINT_allexpr_rrseq \xintbareiieval    \xintthebareiieval    }%
\def\XINT_allexpr_rrseq #1#2#3%
{%
    \expandafter\XINT_expr_rrseqx\expandafter #1\expandafter#2\expandafter
    #3\romannumeral`&&@\XINT_expr_onliteral_seq_a {}%
}%
```

11.48.1 \XINT_expr_rrseqx

The (#5) is for ++ mechanism which must have its closing parenthesis.

```
\def\XINT_expr_rrseqx #1#2#3#4#5%
{%
    \expandafter\XINT_expr_rrseqy\romannumeral0#1(#5)\expandafter\relax
    \expandafter{\romannumeral0\xintapply \XINT_expr_lockit
       {\xintRevWithBraces{\xintCSVtoListNonStripped{\XINT_expr_unlock #3}}}}%
    #3#4#2%
}%
```

11.48.2 \XINT_expr_rrseqy

#1=valeurs pour variable (locked), #2=initial values (reversed, one (braced) token each) #3=toutes les valeurs initiales (csv,locked), #4=variable, #5=expr, #6=\xintthebareeval ou \xintthebare-floateval ou \xintthebareiieval

```
\def\XINT_expr_rrseqy #1#2#3#4#5#6%
{%
    \expandafter \XINT_expr_getop
    \csname .=\XINT_expr_unlock #3%
    \expandafter\XINT_expr_rrseq:_aa
                \romannumeral`&&@\XINT_expr_unlock #1!{#6#5\relax !#4}{#2}\endcsname
}%
\def\XINT_expr_rrseq:_aa #1{\if +#1\expandafter\XINT_expr_rrseq:_A\else
                                  \expandafter\XINT_expr_rrseq:_a\fi #1}%
```

11.48.3 \XINT_expr_rrseq:_a etc...

Attention que ? a catcode 3 ici et dans iter.

```
\catcode`? 3
\def\XINT_expr_rrseq:_a #1!#2#3{\XINT_expr_rrseq:_b {#3}{#2}#1,^,}%
\def\XINT_expr_rrseq:_b #1#2#3#4,{%
     \if ,#3\xint_dothis\XINT_expr_rrseq:_noop\fi
     \if ^#3\xint_dothis\XINT_expr_rrseq:_end\fi
     \xint_orthat{\expandafter\XINT_expr_rrseq:_c}\csname.=#3#4\endcsname
     {#1}{#2}%
}%
\def\XINT_expr_rrseq:_noop\csname.=,#1\endcsname #2#3{\XINT_expr_rrseq:_b {#2}{#3}#1,}%
\def\XINT_expr_rrseq:_end \csname.=^\endcsname #1#2{}%
\def\XINT_expr_rrseq:_c #1#2#3%
   {\expandafter\XINT_expr_rrseq:_d\romannumeral`&&@#3#1~#2?{#3}}%
```

```
\def\XINT_expr_rrseq:_d #1{%
    \if ^#1\xint_dothis\XINT_expr_rrseq:_abort\fi
    \if ?#1\xint_dothis\XINT_expr_rrseq:_break\fi
    \if !#1\xint_dothis\XINT_expr_rrseq:_omit\fi
    \xint_orthat{\XINT_expr_rrseq:_goon #1}%
}%
\def\XINT_expr_rrseq:_goon  #1!#2#3~#4?#5{,#1\expandafter\XINT_expr_rrseq:_b\expandafter
        {\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
\def\XINT_expr_rrseq:_omit  #1!#2#3~{\XINT_expr_rrseq:_b }%
\def\XINT_expr_rrseq:_abort #1!#2#3~#4?#5#6^,{}%
\def\XINT_expr_rrseq:_break #1!#2#3~#4?#5#6^,{,#1}%
```

11.48.4 \XINT_expr_rrseq:_A etc...

n++ for rrseq. With 1.2C, the #1 in \XINT_expr_rrseq:_A is a single token.

```
\def\XINT_expr_rrseq:_A +#1!#2#3{\XINT_expr_rrseq:_D #1{#3}{#2}}%
\def\XINT_expr_rrseq:_D #1#2#3%
   {\expandafter\XINT_expr_rrseq:_E\romannumeral`&&@#3#1~#2?{#3}}%
\def\XINT_expr_rrseq:_Goon  #1!#2#3~#4?#5%
   {,#1\expandafter\XINT_expr_rrseq:_D
       \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
    \expandafter{\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
\def\XINT_expr_rrseq:_Omit  #1!#2#3~%#4?#5%
    {\expandafter\XINT_expr_rrseq:_D
            \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
\def\XINT_expr_rrseq:_Abort #1!#2#3~#4?#5{}%
\def\XINT_expr_rrseq:_Break #1!#2#3~#4?#5{,#1}%
\def\XINT_expr_rrseq:_E #1{\if #1^\xint_dothis\XINT_expr_rrseq:_Abort\fi
                           \if #1?\xint_dothis\XINT_expr_rrseq:_Break\fi
                           \if #1!\xint_dothis\XINT_expr_rrseq:_Omit\fi
                           \xint_orthat{\XINT_expr_rrseq:_Goon #1}}%
```

11.49 \iterr()

```
\def\XINT_expr_func_iterr   {\XINT_allexpr_iterr \xintbareeval      \xintthebareeval      }%
\def\XINT_flexpr_func_iterr {\XINT_allexpr_iterr \xintbarefloateval \xintthebarefloateval }%
\def\XINT_iiexpr_func_iterr {\XINT_allexpr_iterr \xintbareiieval    \xintthebareiieval    }%
\def\XINT_allexpr_iterr #1#2#3%
{%
    \expandafter\XINT_expr_iterrx\expandafter #1\expandafter #2\expandafter
    #3\romannumeral`&&@\XINT_expr_onliteral_seq_a {}%
}%
```

11.49.1 \XINT_expr_iterrx

The (#5) is for ++ mechanism which must have its closing parenthesis.

```
\def\XINT_expr_iterrx #1#2#3#4#5%
{%
    \expandafter\XINT_expr_iterry\romannumeral0#1(#5)\expandafter\relax
    \expandafter{\romannumeral0\xintapply \XINT_expr_lockit
       {\xintRevWithBraces{\xintCSVtoListNonStripped{\XINT_expr_unlock #3}}}}%
    #3#4#2%
}%
```

11.49.2 \XINT_expr_iterry

#1=valeurs pour variable (locked), #2=initial values (reversed, one (braced) token each) #3=toutes les valeurs initiales (csv,locked), #4=variable, #5=expr, #6=\xintbareeval ou \xintbarefloateval ou \xintbareiieval

```
\def\XINT_expr_iterry #1#2#3#4#5#6%
{%
    \expandafter \XINT_expr_getop
    \csname .=%
    \expandafter\XINT_expr_iterr:_aa
    \romannumeral`&&@\XINT_expr_unlock #1!{#6#5\relax !#4}{#2}\endcsname
}%
\def\XINT_expr_iterr:_aa #1{\if +#1\expandafter\XINT_expr_iterr:_A\else
                                  \expandafter\XINT_expr_iterr:_a\fi #1}%
```

11.49.3 \XINT_expr_iterr:_a etc...

```
\def\XINT_expr_iterr:_a #1!#2#3{\XINT_expr_iterr:_b {#3}{#2}#1,^,}%
\def\XINT_expr_iterr:_b #1#2#3#4,{%
     \if ,#3\xint_dothis\XINT_expr_iterr:_noop\fi
     \if ^#3\xint_dothis\XINT_expr_iterr:_end\fi
     \xint_orthat{\expandafter\XINT_expr_iterr:_c}%
     \csname.=#3#4\endcsname {#1}{#2}%
}%
\def\XINT_expr_iterr:_noop\csname.=,#1\endcsname #2#3{\XINT_expr_iterr:_b {#2}{#3}#1,}%
\def\XINT_expr_iterr:_end \csname.=^\endcsname #1#2%
   {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
          {,\XINT_expr:_unlock}{\xintReverseOrder{#1\space}}}%
\def\XINT_expr_iterr:_c #1#2#3%
   {\expandafter\XINT_expr_iterr:_d\romannumeral`&&@#3#1~#2?{#3}}%
\def\XINT_expr_iterr:_d #1{%
    \if ^#1\xint_dothis\XINT_expr_iterr:_abort\fi
    \if ?#1\xint_dothis\XINT_expr_iterr:_break\fi
    \if !#1\xint_dothis\XINT_expr_iterr:_omit\fi
    \xint_orthat{\XINT_expr_iterr:_goon #1}%
}%
\def\XINT_expr_iterr:_goon  #1!#2#3~#4?#5{\expandafter\XINT_expr_iterr:_b\expandafter
        {\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
\def\XINT_expr_iterr:_omit  #1!#2#3~{\XINT_expr_iterr:_b }%
\def\XINT_expr_iterr:_abort #1!#2#3~#4?#5#6^,%
   {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
          {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}}}%
\def\XINT_expr_iterr:_break #1!#2#3~#4?#5#6^,%
   {\expandafter\xint_gobble_iv\romannumeral0\xintapplyunbraced
```

```
          {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}},#1}%
\def\XINT_expr:_unlock #1{\XINT_expr_unlock #1}%
```

11.49.4 `\XINT_expr_iterr:_A` etc...

n++ for iterr. ? is of catcode 3 here.

```
\def\XINT_expr_iterr:_A +#1!#2#3{\XINT_expr_iterr:_D #1{#3}{#2}}%
\def\XINT_expr_iterr:_D #1#2#3%
   {\expandafter\XINT_expr_iterr:_E\romannumeral`&&@#3#1~#2?{#3}}%
\def\XINT_expr_iterr:_Goon  #1!#2#3~#4?#5%
   {\expandafter\XINT_expr_iterr:_D
    \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
    \expandafter{\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
\def\XINT_expr_iterr:_Omit  #1!#2#3~%#4?#5%
    {\expandafter\XINT_expr_iterr:_D
     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
\def\XINT_expr_iterr:_Abort #1!#2#3~#4?#5%
   {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
          {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}}}%
\def\XINT_expr_iterr:_Break #1!#2#3~#4?#5%
   {\expandafter\xint_gobble_iv\romannumeral0\xintapplyunbraced
          {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}},#1}%
\def\XINT_expr_iterr:_E #1{\if #1^\xint_dothis\XINT_expr_iterr:_Abort\fi
                          \if #1?\xint_dothis\XINT_expr_iterr:_Break\fi
                          \if #1!\xint_dothis\XINT_expr_iterr:_Omit\fi
                          \xint_orthat{\XINT_expr_iterr:_Goon #1}}%
\catcode`? 11
```

11.50 Macros handling csv lists for functions with multiple comma separated arguments in expressions

These macros are used inside \csname...\endcsname. These things are not initiated by a \romannumeral in general, but in some cases they are, especially when involved in an \xintNewExpr. They will then be protected against expansion and expand only later in contexts governed by an initial \romannumeral-`0. There each new item may need to be expanded, which would not be the case in the use for the _func_ things.

1.2g adds (to be continued)

11.50.1 \xintANDof:csv

1.09a. For use by \xintexpr inside \csname. 1.1, je remplace ifTrueAelseB par iiNotZero pour des raisons d'optimisations.

```
\def\xintANDof:csv #1{\expandafter\XINT_andof:_a\romannumeral`&&@#1,,^}%
\def\XINT_andof:_a #1{\if ,#1\expandafter\XINT_andof:_e
                      \else\expandafter\XINT_andof:_c\fi #1}%
\def\XINT_andof:_c #1,{\xintiiifNotZero {#1}{\XINT_andof:_a}{\XINT_andof:_no}}%
\def\XINT_andof:_no #1^{0}%
\def\XINT_andof:_e  #1^{1}% works with empty list
```

11.50.2 \xintORof:csv

1.09a. For use by \xintexpr.

```
\def\xintORof:csv #1{\expandafter\XINT_orof:_a\romannumeral`&&@#1,,^}%
\def\XINT_orof:_a #1{\if ,#1\expandafter\XINT_orof:_e
                      \else\expandafter\XINT_orof:_c\fi #1}%
\def\XINT_orof:_c #1,{\xintiiifNotZero{#1}{\XINT_orof:_yes}{\XINT_orof:_a}}%
\def\XINT_orof:_yes #1^{1}%
\def\XINT_orof:_e   #1^{0}% works with empty list
```

11.50.3 \xintXORof:csv

1.09a. For use by \xintexpr (inside a \csname..\endcsname).

```
\def\xintXORof:csv #1{\expandafter\XINT_xorof:_a\expandafter 0\romannumeral`&&@#1,,^}%
\def\XINT_xorof:_a #1#2,{\XINT_xorof:_b #2,#1}%
\def\XINT_xorof:_b #1{\if ,#1\expandafter\XINT_xorof:_e
                      \else\expandafter\XINT_xorof:_c\fi #1}%
\def\XINT_xorof:_c #1,#2%
           {\xintiiifNotZero {#1}{\if #20\xint_afterfi{\XINT_xorof:_a 1}%
                                  \else\xint_afterfi{\XINT_xorof:_a 0}\fi}%
                                 {\XINT_xorof:_a #2}%
           }%
\def\XINT_xorof:_e ,#1#2^{#1}% allows empty list (then returns 0)
```

11.50.4 Generic csv routine (\XINT_oncsv:_a)

1.1. generic routine. up to the loss of some efficiency, especially for Sum:csv and Prod:csv, where \XINTinFloat will be done twice for each argument.

```
\def\XINT_oncsv:_empty  #1,^,#2{#2}%
\def\XINT_oncsv:_end    ^,#1#2#3#4{#1}%
\def\XINT_oncsv:_a #1#2#3%
   {\if ,#3\expandafter\XINT_oncsv:_empty\else\expandafter\XINT_oncsv:_b\fi #1#2#3}%
\def\XINT_oncsv:_b #1#2#3,%
   {\expandafter\XINT_oncsv:_c \expandafter{\romannumeral`&&@#2{#3}}#1#2}%
\def\XINT_oncsv:_c #1#2#3#4,{\expandafter\XINT_oncsv:_d \romannumeral`&&@#4,{#1}#2#3}%
\def\XINT_oncsv:_d #1%
   {\if ^#1\expandafter\XINT_oncsv:_end\else\expandafter\XINT_oncsv:_e\fi #1}%
\def\XINT_oncsv:_e #1,#2#3#4%
   {\expandafter\XINT_oncsv:_c\expandafter {\romannumeral`&&@#3{#4{#1}}{#2}}#3#4}%
```

11.50.5 \xintMaxof:csv, \xintiiMaxof:csv

1.09i. Rewritten for 1.1. Compatible avec liste vide donnant valeur par défaut. Pas compatible avec items manquants. ah je m'aperçois au dernier moment que je n'ai pas en effet de \xintiiMax. Je devrais le rajouter. En tout cas ici c'est uniquement pour xintiiexpr, dans il faut bien sûr ne pas faire de xintNum, donc il faut un iimax.

```
\def\xintMaxof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmax
                      \expandafter\xint_firstofone\romannumeral`&&@#1,^,{0/1[0]}}%
\def\xintiiMaxof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiimax
                        \expandafter\xint_firstofone\romannumeral`&&@#1,^,0}%
```

11.50.6 \xintMinof:csv, \xintiiMinof:csv

1.09i. Rewritten for 1.1. For use by \xintiiexpr.

```
\def\xintMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmin
                        \expandafter\xint_firstofone\romannumeral`&&@#1,^,{0/1[0]}}%
\def\xintiiMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiimin
                        \expandafter\xint_firstofone\romannumeral`&&@#1,^,0}%
```

11.50.7 \xintSum:csv, \xintiiSum:csv

1.09a. Rewritten for 1.1. For use by \xintexpr.

```
\def\xintSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintadd
                     \expandafter\xint_firstofone\romannumeral`&&@#1,^,{0/1[0]}}%
\def\xintiiSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiadd
                       \expandafter\xint_firstofone\romannumeral`&&@#1,^,0}%
```

11.50.8 \xintPrd:csv, \xintiiPrd:csv

1.09a. Rewritten for 1.1. For use by \xintexpr.

```
\def\xintPrd:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmul
                     \expandafter\xint_firstofone\romannumeral`&&@#1,^,{1/1[0]}}%
\def\xintiiPrd:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiimul
                       \expandafter\xint_firstofone\romannumeral`&&@#1,^,1}%
```

11.50.9 \xintGCDof:csv, \xintLCMof:csv

1.09a. Rewritten for 1.1. For use by \xintexpr. Expansion réinstaurée pour besoins de xintNewExpr de version 1.1

```
\def\xintGCDof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintgcd
                       \expandafter\xint_firstofone\romannumeral`&&@#1,^,1}%
\def\xintLCMof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintlcm
                       \expandafter\xint_firstofone\romannumeral`&&@#1,^,0}%
```

11.50.10 `\xintiiGCDof:csv`, `\xintiiLCMof:csv`

1.1a pour `\xintiiexpr`. Ces histoires de ii sont pénibles à la fin.

```
\def\xintiiGCDof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiigcd
                       \expandafter\xint_firstofone\romannumeral`&&@#1,^,1}%
\def\xintiiLCMof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiilcm
                       \expandafter\xint_firstofone\romannumeral`&&@#1,^,0}%
```

11.50.11 `\XINTinFloatdigits`, `\XINTinFloatSqrtdigits`, `\XINTinFloatFacdigits`

for `\xintNewExpr` matters, mainly.

```
\def\XINTinFloatdigits     {\XINTinFloat    [\XINTdigits]}%
\def\XINTinFloatSqrtdigits {\XINTinFloatSqrt[\XINTdigits]}%
\def\XINTinFloatFacdigits  {\XINTinFloatFac [\XINTdigits]}%
```

11.50.12 `\XINTinFloatMaxof:csv`, `\XINTinFloatMinof:csv`

1.09a. Rewritten for 1.1. For use by `\xintfloatexpr`. Name changed in 1.09h

```
\def\XINTinFloatMaxof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmax
                         \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^,{0[0]}}%
\def\XINTinFloatMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmin
                         \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^,{0[0]}}%
```

11.50.13 `\XINTinFloatSum:csv`, `\XINTinFloatPrd:csv`

1.09a. Rewritten for 1.1. For use by `\xintfloatexpr`.

```
\def\XINTinFloatSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\XINTinfloatadd
                        \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^,{0[0]}}%
\def\XINTinFloatPrd:csv #1{\expandafter\XINT_oncsv:_a\expandafter\XINTinfloatmul
                        \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^,{1[0]}}%
```

11.51 Auxiliary wrappers for function macros

```
\def\XINT:expr:one:and:opt #1,#2,#3!#4#5%
{%
    \if\relax#3\relax\expandafter\xint_firstoftwo\else
                     \expandafter\xint_secondoftwo\fi
    {#4}{#5[\xintNum{#2}]}{#1}%
}%
\def\XINT:expr:tacitzeroifonearg #1,#2,#3!#4#5%
{%
    \if\relax#3\relax\expandafter\xint_firstoftwo\else
                     \expandafter\xint_secondoftwo\fi
    {#4{0}}{#5{\xintNum{#2}}}{#1}%
}%
\def\XINT:iiexpr:tacitzeroifonearg #1,#2,#3!#4%
{%
    \if\relax#3\relax\expandafter\xint_firstoftwo\else
                     \expandafter\xint_secondoftwo\fi
    {#4{0}}{#4{#2}}{#1}%
}%
\def\XINT:expr:totwo #1#2{#1,#2}%
\def\XINT:expr:two:to:two #1,#2,!#3%
{%
    \expandafter\XINT:expr:totwo\romannumeral`&&@%
    #3{#1}{#2}%
}%
\def\XINT:expr:two:to:one #1,#2,!#3%
{%
    #3{#1}{#2}%
}%
\def\XINT:flexpr:two:to:one #1,#2,!#3%
{%
    #3{#1}{#2}%
}%
\let\XINT:flexpr:two:to:two\XINT:flexpr:two:to:one
```

11.52 The `num()`, `reduce()`, `preduce()`, `abs()`, `sgn()`, `frac()`, `floor()`, `ceil()`, `sqr()`, `sqrt()`, `sqrtr()`, `float()`, `round()`, `trunc()`, `mod()`, `quo()`, `rem()`, `divmod()`, `gcd()`, `lcm()`, `max()`, `min()`, ``` `+`() ```, ``` `*`() ```, `?()`, `!()`, `not()`, `all()`, `any()`, `xor()`, `if()`, `ifsgn()`, `ifint()`, `ifone()`, `even()`, `odd()`, `first()`, `last()`, `len()`, `reversed()`, `factorial()`, `binomial()` and `randrange()` functions

```
\def\XINT_expr_func_num #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:one\xintNum{\XINT_expr_unlock #3}\endcsname
}%
\let\XINT_flexpr_func_num\XINT_expr_func_num
\let\XINT_iiexpr_func_num\XINT_expr_func_num
\def\XINT_expr_func_reduce #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:one\xintIrr{\XINT_expr_unlock #3}[0]\endcsname
}%
\let\XINT_flexpr_func_reduce\XINT_expr_func_reduce
\def\XINT_expr_func_preduce #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:one\xintPIrr{\XINT_expr_unlock #3}\endcsname
}%
\let\XINT_flexpr_func_preduce\XINT_expr_func_preduce
\def\XINT_expr_func_abs #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:one\xintAbs{\XINT_expr_unlock #3}\endcsname
}%
\let\XINT_flexpr_func_abs\XINT_expr_func_abs
\def\XINT_iiexpr_func_abs #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:one\xintiiAbs{\XINT_expr_unlock #3}\endcsname
```

```
}%
\def\XINT_expr_func_sgn #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:one\xintSgn{\XINT_expr_unlock #3}\endcsname
}%
\let\XINT_flexpr_func_sgn\XINT_expr_func_sgn
\def\XINT_iiexpr_func_sgn #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:one\xintiiSgn{\XINT_expr_unlock #3}\endcsname
}%
\def\XINT_expr_func_frac #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:one\xintTFrac{\XINT_expr_unlock #3}\endcsname
}%
\def\XINT_flexpr_func_frac #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:one\XINTinFloatFracdigits{\XINT_expr_unlock #3}\endcsname
}%
```

no `\XINT_iiexpr_func_frac`

```
\def\XINT_expr_func_floor #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:one\xintFloor{\XINT_expr_unlock #3}\endcsname
}%
\let\XINT_flexpr_func_floor\XINT_expr_func_floor
```

The floor and ceil functions in `\xintiiexpr` require `protect(a/b)` or, better, `\qfrac(a/b)`; else the / will be executed first and do an integer rounded division.

```
\def\XINT_iiexpr_func_floor #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:one\xintiFloor{\XINT_expr_unlock #3}\endcsname
}%
\def\XINT_expr_func_ceil #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:one\xintCeil{\XINT_expr_unlock #3}\endcsname
}%
\let\XINT_flexpr_func_ceil\XINT_expr_func_ceil
\def\XINT_iiexpr_func_ceil #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:one\xintiCeil{\XINT_expr_unlock #3}\endcsname
}%
\def\XINT_expr_func_sqr #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:one\xintSqr{\XINT_expr_unlock #3}\endcsname
```

```
}%
\def\XINTinFloatSqr#1{\XINTinFloatMul{#1}{#1}}% revoir après
\def\XINT_flexpr_func_sqr #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:one\XINTinFloatSqr{\XINT_expr_unlock #3}\endcsname
}%
\def\XINT_iiexpr_func_sqr #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:one\xintiiSqr{\XINT_expr_unlock #3}\endcsname
}%
\def\XINT_expr_func_? #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:one\xintiiIsNotZero{\XINT_expr_unlock #3}\endcsname
}%
\let\XINT_flexpr_func_? \XINT_expr_func_?
\let\XINT_iiexpr_func_? \XINT_expr_func_?
\def\XINT_expr_func_! #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:one\xintiiIsZero{\XINT_expr_unlock #3}\endcsname
}%
\let\XINT_flexpr_func_! \XINT_expr_func_!
\let\XINT_iiexpr_func_! \XINT_expr_func_!
\def\XINT_expr_func_not #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:one\xintiiIsZero{\XINT_expr_unlock #3}\endcsname
}%
\let\XINT_flexpr_func_not \XINT_expr_func_not
\let\XINT_iiexpr_func_not \XINT_expr_func_not
\def\XINT_expr_func_odd #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:one\xintOdd{\XINT_expr_unlock #3}\endcsname
}%
\let\XINT_flexpr_func_odd\XINT_expr_func_odd
\def\XINT_iiexpr_func_odd #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:one\xintiiOdd{\XINT_expr_unlock #3}\endcsname
}%
\def\XINT_expr_func_even #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:one\xintEven{\XINT_expr_unlock #3}\endcsname
}%
\let\XINT_flexpr_func_even\XINT_expr_func_even
\def\XINT_iiexpr_func_even #1#2#3%
{%
```

```
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:one\xintiiEven{\XINT_expr_unlock #3}\endcsname
}%
% REVOIR nuple
\def\XINT_expr_func_nuple #1#2#3%
   {\expandafter #1\expandafter #2\csname.=\XINT_expr_unlock #3\endcsname }%
\let\XINT_flexpr_func_nuple\XINT_expr_func_nuple
\let\XINT_iiexpr_func_nuple\XINT_expr_func_nuple
\def\XINT_expr_func_factorial #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\XINT:expr:one:and:opt
    \romannumeral`&&@\XINT_expr_unlock#3,,!\xintFac\XINTinFloatFac
    \endcsname
}%
\def\XINT_flexpr_func_factorial #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\XINT:expr:one:and:opt
    \romannumeral`&&@\XINT_expr_unlock#3,,!\XINTinFloatFacdigits\XINTinFloatFac
    \endcsname
}%
\def\XINT_iiexpr_func_factorial #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:one\xintiiFac{\XINT_expr_unlock #3}\endcsname
}%
\def\XINT_expr_func_sqrt #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\XINT:expr:one:and:opt
    \romannumeral`&&@\XINT_expr_unlock#3,,!\XINTinFloatSqrtdigits\XINTinFloatSqrt
    \endcsname
}%
\let\XINT_flexpr_func_sqrt\XINT_expr_func_sqrt
\def\XINT_iiexpr_func_sqrt #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:one\xintiiSqrt{\XINT_expr_unlock #3}\endcsname
}%
\def\XINT_iiexpr_func_sqrtr #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:one\xintiiSqrtR{\XINT_expr_unlock #3}\endcsname
}%
\def\XINT_expr_func_round #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\XINT:expr:tacitzeroifonearg
    \romannumeral`&&@\XINT_expr_unlock #3,,!\xintiRound\xintRound
    \endcsname
}%
```

```
\let\XINT_flexpr_func_round\XINT_expr_func_round
\def\XINT_iiexpr_func_round #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\XINT:iiexpr:tacitzeroifonearg
    \romannumeral`&&@\XINT_expr_unlock #3,,!\xintiRound
    \endcsname
}%
\def\XINT_expr_func_trunc #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\XINT:expr:tacitzeroifonearg
    \romannumeral`&&@\XINT_expr_unlock #3,,!\xintiTrunc\xintTrunc
    \endcsname
}%
\let\XINT_flexpr_func_trunc\XINT_expr_func_trunc
\def\XINT_iiexpr_func_trunc #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\XINT:iiexpr:tacitzeroifonearg
    \romannumeral`&&@\XINT_expr_unlock #3,,!\xintiTrunc
    \endcsname
}%
\def\XINT_expr_func_float #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\XINT:expr:one:and:opt
    \romannumeral`&&@\XINT_expr_unlock #3,,!\XINTinFloatdigits\XINTinFloat
    \endcsname
}%
\let\XINT_flexpr_func_float\XINT_expr_func_float
% \XINT_iiexpr_func_float not defined
\def\XINT_expr_func_divmod #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\XINT:expr:two:to:two
    \romannumeral`&&@\XINT_expr_unlock #3,!\xintDivMod
    \endcsname
}%
\def\XINT_flexpr_func_divmod #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\XINT:flexpr:two:to:two
    \romannumeral`&&@\XINT_expr_unlock #3,!\XINTinFloatDivMod
    \endcsname
}%
\def\XINT_iiexpr_func_divmod #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\XINT:expr:two:to:two
    \romannumeral`&&@\XINT_expr_unlock #3,!\xintiiDivMod
    \endcsname
```

```
}%
\def\XINT_expr_func_mod #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\XINT:expr:two:to:one
    \romannumeral`&&@\XINT_expr_unlock #3,!\xintMod
    \endcsname
}%
\def\XINT_flexpr_func_mod #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\XINT:flexpr:two:to:one
    \romannumeral`&&@\XINT_expr_unlock #3,!\XINTinFloatMod
    \endcsname
}%
\def\XINT_iiexpr_func_mod #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\XINT:expr:two:to:one
    \romannumeral`&&@\XINT_expr_unlock #3,!\xintiiMod
    \endcsname
}%
\def\XINT_expr_func_binomial #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\XINT:expr:two:to:one
    \romannumeral`&&@\XINT_expr_unlock #3,!\xintBinomial
    \endcsname
}%
\def\XINT_flexpr_func_binomial #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\XINT:flexpr:two:to:one
    \romannumeral`&&@\XINT_expr_unlock #3,!\XINTinFloatBinomial
    \endcsname
}%
\def\XINT_iiexpr_func_binomial #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\XINT:expr:two:to:one
    \romannumeral`&&@\XINT_expr_unlock #3,!\xintiiBinomial
    \endcsname
}%
\def\XINT_expr_func_pfactorial #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\XINT:expr:two:to:one
    \romannumeral`&&@\XINT_expr_unlock #3,!\xintPFactorial
    \endcsname
}%
\def\XINT_flexpr_func_pfactorial #1#2#3%
{%
```

```
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\XINT:flexpr:two:to:one
    \romannumeral`&&@\XINT_expr_unlock #3,!\XINTinFloatPFactorial
    \endcsname
}%
\def\XINT_iiexpr_func_pfactorial #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\XINT:expr:two:to:one
    \romannumeral`&&@\XINT_expr_unlock #3,!\xintiiPFactorial
    \endcsname
}%
\def\XINT_expr_func_randrange #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\XINT:expr:randrange
    \romannumeral`&&@\XINT_expr_unlock #3,,!%
    \endcsname
}%
\let\XINT_flexpr_func_randrange\XINT_expr_func_randrange
\def\XINT_iiexpr_func_randrange #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\XINT:iiexpr:randrange
    \romannumeral`&&@\XINT_expr_unlock #3,,!%
    \endcsname
}%
\def\XINT:expr:randrange #1,#2,#3!%
{%
    \if\relax#3\relax\expandafter\xint_firstoftwo\else
                     \expandafter\xint_secondoftwo\fi
    {\xintiiRandRange{\XINT:NEhook:one\xintNum{#1}}}%
    {\xintiiRandRangeAtoB{\XINT:NEhook:one\xintNum{#1}}%
                         {\XINT:NEhook:one\xintNum{#2}}}%
}%
\def\XINT:iiexpr:randrange #1,#2,#3!%
{%
    \if\relax#3\relax\expandafter\xint_firstoftwo\else
                     \expandafter\xint_secondoftwo\fi
    {\xintiiRandRange{#1}}{\xintiiRandRangeAtoB{#1}{#2}}%
}%
\def\XINT_expr_func_quo #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\XINT:expr:two:to:one
    \romannumeral`&&@\XINT_expr_unlock #3,!\xintiQuo
    \endcsname
}%
\let\XINT_flexpr_func_quo\XINT_expr_func_quo
\def\XINT_iiexpr_func_quo #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\XINT:expr:two:to:one
    \romannumeral`&&@\XINT_expr_unlock #3,!\xintiiQuo
    \endcsname
}%
\def\XINT_expr_func_rem #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\XINT:expr:two:to:one
    \romannumeral`&&@\XINT_expr_unlock #3,!\xintiRem
    \endcsname
}%
\let\XINT_flexpr_func_rem\XINT_expr_func_rem
\def\XINT_iiexpr_func_rem #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\XINT:expr:two:to:one
    \romannumeral`&&@\XINT_expr_unlock #3,!\xintiiRem
    \endcsname
}%
\def\XINT_expr_func_gcd #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:csv\xintGCDof:csv{\XINT_expr_unlock #3}\endcsname
}%
\let\XINT_flexpr_func_gcd\XINT_expr_func_gcd
\def\XINT_iiexpr_func_gcd #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:csv\xintiiGCDof:csv{\XINT_expr_unlock #3}\endcsname
}%
\def\XINT_expr_func_lcm #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:csv\xintLCMof:csv{\XINT_expr_unlock #3}\endcsname
}%
\let\XINT_flexpr_func_lcm\XINT_expr_func_lcm
\def\XINT_iiexpr_func_lcm #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:csv\xintiiLCMof:csv{\XINT_expr_unlock #3}\endcsname
}%
\def\XINT_expr_func_max #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:csv\xintMaxof:csv{\XINT_expr_unlock #3}\endcsname
}%
\def\XINT_iiexpr_func_max #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:csv\xintiiMaxof:csv{\XINT_expr_unlock #3}\endcsname
}%
\def\XINT_flexpr_func_max #1#2#3%
```

```
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:csv\XINTinFloatMaxof:csv{\XINT_expr_unlock #3}\endcsname
}%
\def\XINT_expr_func_min #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:csv\xintMinof:csv{\XINT_expr_unlock #3}\endcsname
}%
\def\XINT_iiexpr_func_min #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:csv\xintiiMinof:csv{\XINT_expr_unlock #3}\endcsname
}%
\def\XINT_flexpr_func_min #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:csv\XINTinFloatMinof:csv{\XINT_expr_unlock #3}\endcsname
}%
\expandafter
\def\csname XINT_expr_func_+\endcsname #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:csv\xintSum:csv{\XINT_expr_unlock #3}\endcsname
}%
\expandafter
\def\csname XINT_flexpr_func_+\endcsname #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:csv\XINTinFloatSum:csv{\XINT_expr_unlock #3}\endcsname
}%
\expandafter
\def\csname XINT_iiexpr_func_+\endcsname #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:csv\xintiiSum:csv{\XINT_expr_unlock #3}\endcsname
}%
\expandafter
\def\csname XINT_expr_func_*\endcsname #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:csv\xintPrd:csv{\XINT_expr_unlock #3}\endcsname
}%
\expandafter
\def\csname XINT_flexpr_func_*\endcsname #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:csv\XINTinFloatPrd:csv{\XINT_expr_unlock #3}\endcsname
}%
\expandafter
\def\csname XINT_iiexpr_func_*\endcsname #1#2#3%
{%
```

```
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:csv\xintiiPrd:csv{\XINT_expr_unlock #3}\endcsname
}%
\def\XINT_expr_func_all #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:csv\xintANDof:csv{\XINT_expr_unlock #3}\endcsname
}%
\let\XINT_flexpr_func_all\XINT_expr_func_all
\let\XINT_iiexpr_func_all\XINT_expr_func_all
\def\XINT_expr_func_any #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:csv\xintORof:csv{\XINT_expr_unlock #3}\endcsname
}%
\let\XINT_flexpr_func_any\XINT_expr_func_any
\let\XINT_iiexpr_func_any\XINT_expr_func_any
\def\XINT_expr_func_xor #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:csv\xintXORof:csv{\XINT_expr_unlock #3}\endcsname
}%
\let\XINT_flexpr_func_xor\XINT_expr_func_xor
\let\XINT_iiexpr_func_xor\XINT_expr_func_xor
\def\XINT_expr_func_len #1#2#3%
{%
    \expandafter#1\expandafter#2\csname.=%
    \XINT:NEhook:csv\xintLength:f:csv{\XINT_expr_unlock#3}\endcsname
}%
\let\XINT_flexpr_func_len \XINT_expr_func_len
\let\XINT_iiexpr_func_len \XINT_expr_func_len
```

1.2k has \xintFirstItem:f:csv for improved \xintNewExpr compatibility.

```
\def\XINT_expr_func_first #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:csv\xintFirstItem:f:csv{\XINT_expr_unlock #3}\endcsname
}%
\let\XINT_flexpr_func_first\XINT_expr_func_first
\let\XINT_iiexpr_func_first\XINT_expr_func_first
```

1.2k has \xintLastItem:f:csv for efficiency and improved \xintNewExpr compatibility.

```
\def\XINT_expr_func_last #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:csv\xintLastItem:f:csv{\XINT_expr_unlock #3}\endcsname
}%
\let\XINT_flexpr_func_last\XINT_expr_func_last
\let\XINT_iiexpr_func_last\XINT_expr_func_last
```

1.2c I hesitated but left the function "reversed" from 1.1 with this name, not "reverse". But the inner not public macro got renamed into \xintReverse::csv. 1.2g opts for the name \xintReverse:f:csv, and rewrites it for direct handling of csv lists. 2016/03/17.

```
\def\XINT_expr_func_reversed #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \XINT:NEhook:csv\xintReverse:f:csv{\XINT_expr_unlock #3}\endcsname
}%
\let\XINT_flexpr_func_reversed\XINT_expr_func_reversed
\let\XINT_iiexpr_func_reversed\XINT_expr_func_reversed
\def\xintiiifNotZero: #1,#2,#3,{\xintiiifNotZero{#1}{#2}{#3}}%
\def\XINT_expr_func_if #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\xintiiifNotZero:%
    \romannumeral`&&@\XINT_expr_unlock #3,\endcsname
}%
\let\XINT_flexpr_func_if\XINT_expr_func_if
\let\XINT_iiexpr_func_if\XINT_expr_func_if
\def\xintifInt: #1,#2,#3,{\xintifInt{#1}{#2}{#3}}%
\def\XINT_expr_func_ifint #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\xintifInt:%
    \romannumeral`&&@\XINT_expr_unlock #3,\endcsname
}%
\let\XINT_iiexpr_func_ifint\XINT_expr_func_ifint
\def\xintifFloatInt: #1,#2,#3,{\xintifFloatInt{#1}{#2}{#3}}%
\def\XINT_flexpr_func_ifint #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\xintifFloatInt:%
    \romannumeral`&&@\XINT_expr_unlock #3,\endcsname
}%
\def\xintifOne: #1,#2,#3,{\xintifOne{#1}{#2}{#3}}%
\def\XINT_expr_func_ifone #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\xintifOne:%
    \romannumeral`&&@\XINT_expr_unlock #3,\endcsname
}%
\let\XINT_flexpr_func_ifone\XINT_expr_func_ifone
\def\xintiiifOne: #1,#2,#3,{\xintiiifOne{#1}{#2}{#3}}%
\def\XINT_iiexpr_func_ifone #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\xintiiifOne:%
    \romannumeral`&&@\XINT_expr_unlock #3,\endcsname
}%
\def\xintiiifSgn: #1,#2,#3,#4,{\xintiiifSgn{#1}{#2}{#3}{#4}}%
\def\XINT_expr_func_ifsgn #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=%
    \expandafter\xintiiifSgn:%
    \romannumeral`&&@\XINT_expr_unlock #3,\endcsname
}%
\let\XINT_flexpr_func_ifsgn\XINT_expr_func_ifsgn
\let\XINT_iiexpr_func_ifsgn\XINT_expr_func_ifsgn
```

11.53 f-expandable versions of the `\xintSeqB::csv` and alike routines, for `\xintNewExpr`

11.53.1 `\xintSeqB:f:csv`

Produces in f-expandable way. If the step is zero, gives empty result except if start and end coincide.

```
\def\xintSeqB:f:csv #1#2%
   {\expandafter\XINT_seqb:f:csv \expandafter{\romannumeral0\xintraw{#2}}{#1}}%
\def\XINT_seqb:f:csv #1#2{\expandafter\XINT_seqb:f:csv_a\romannumeral`&&@#2#1!}%
\def\XINT_seqb:f:csv_a #1#2;#3;#4!{%
   \expandafter\xint_gobble_i\romannumeral`&&@%
   \xintifCmp {#3}{#4}\XINT_seqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_seqb:f:csv_bg
   #1{#3}{#4}{}{#2}}%
\def\XINT_seqb:f:csv_be #1#2#3#4#5{,#2}%
\def\XINT_seqb:f:csv_bl #1{\if #1p\expandafter\XINT_seqb:f:csv_pa\else
                                   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
\def\XINT_seqb:f:csv_pa #1#2#3#4{\expandafter\XINT_seqb:f:csv_p\expandafter
                                  {\romannumeral0\xintadd{#4}{#1}}{#2}{#3,#1}{#4}}%
\def\XINT_seqb:f:csv_p #1#2%
{%
   \xintifCmp {#1}{#2}\XINT_seqb:f:csv_pa\XINT_seqb:f:csv_pb\XINT_seqb:f:csv_pc
   {#1}{#2}%
}%
\def\XINT_seqb:f:csv_pb #1#2#3#4{#3,#1}%
\def\XINT_seqb:f:csv_pc #1#2#3#4{#3}%
\def\XINT_seqb:f:csv_bg #1{\if #1n\expandafter\XINT_seqb:f:csv_na\else
                                   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
\def\XINT_seqb:f:csv_na #1#2#3#4{\expandafter\XINT_seqb:f:csv_n\expandafter
                                  {\romannumeral0\xintadd{#4}{#1}}{#2}{#3,#1}{#4}}%
\def\XINT_seqb:f:csv_n #1#2%
{%
   \xintifCmp {#1}{#2}\XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_seqb:f:csv_na
   {#1}{#2}%
}%
\def\XINT_seqb:f:csv_nb #1#2#3#4{#3,#1}%
\def\XINT_seqb:f:csv_nc #1#2#3#4{#3}%
```

11.53.2 `\xintiiSeqB:f:csv`

Produces in f-expandable way. If the step is zero, gives empty result except if start and end coincide.

2015/11/11. I correct a typo dating back to release 1.1 (2014/10/29): the macro name had a "b" rather than "B", hence was not functional (causing \xintNewIIExpr to fail on inputs such as #1..[1]..#2).

```
\def\xintiiSeqB:f:csv #1#2%
   {\expandafter\XINT_iiseqb:f:csv \expandafter{\romannumeral`&&@#2}{#1}}%
\def\XINT_iiseqb:f:csv #1#2{\expandafter\XINT_iiseqb:f:csv_a\romannumeral`&&@#2#1!}%
\def\XINT_iiseqb:f:csv_a #1#2;#3;#4!{%
   \expandafter\xint_gobble_i\romannumeral`&&@%
   \xintSgnFork{\XINT_Cmp {#3}{#4}}%
                 \XINT_iiseqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_iiseqb:f:csv_bg
   #1{#3}{#4}{}{#2}}%
\def\XINT_iiseqb:f:csv_bl #1{\if #1p\expandafter\XINT_iiseqb:f:csv_pa\else
                                  \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
\def\XINT_iiseqb:f:csv_pa #1#2#3#4{\expandafter\XINT_iiseqb:f:csv_p\expandafter
                               {\romannumeral0\xintiiadd{#4}{#1}}{#2}{#3,#1}{#4}}%
\def\XINT_iiseqb:f:csv_p #1#2%
{%
    \xintSgnFork{\XINT_Cmp {#1}{#2}}%
    \XINT_iiseqb:f:csv_pa\XINT_iiseqb:f:csv_pb\XINT_iiseqb:f:csv_pc {#1}{#2}%
}%
\def\XINT_iiseqb:f:csv_pb #1#2#3#4{#3,#1}%
\def\XINT_iiseqb:f:csv_pc #1#2#3#4{#3}%
\def\XINT_iiseqb:f:csv_bg #1{\if #1n\expandafter\XINT_iiseqb:f:csv_na\else
                                  \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
\def\XINT_iiseqb:f:csv_na #1#2#3#4{\expandafter\XINT_iiseqb:f:csv_n\expandafter
                               {\romannumeral0\xintiiadd{#4}{#1}}{#2}{#3,#1}{#4}}%
\def\XINT_iiseqb:f:csv_n #1#2%
{%
    \xintSgnFork{\XINT_Cmp {#1}{#2}}%
    \XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_iiseqb:f:csv_na {#1}{#2}%
}%
```

11.53.3 \XINTinFloatSeqB:f:csv

Produces in f-expandable way. If the step is zero, gives empty result except if start and end coincide. This is all for \xintNewExpr.

```
\def\XINTinFloatSeqB:f:csv #1#2{\expandafter\XINT_flseqb:f:csv \expandafter
   {\romannumeral0\XINTinfloat [\XINTdigits]{#2}}{#1}}%
\def\XINT_flseqb:f:csv #1#2{\expandafter\XINT_flseqb:f:csv_a\romannumeral`&&@#2#1!}%
\def\XINT_flseqb:f:csv_a #1#2;#3;#4!{%
   \expandafter\xint_gobble_i\romannumeral`&&@%
   \xintifCmp {#3}{#4}\XINT_flseqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_flseqb:f:csv_bg
   #1{#3}{#4}{}{#2}}%
\def\XINT_flseqb:f:csv_bl #1{\if #1p\expandafter\XINT_flseqb:f:csv_pa\else
                                  \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
\def\XINT_flseqb:f:csv_pa #1#2#3#4{\expandafter\XINT_flseqb:f:csv_p\expandafter
                         {\romannumeral0\XINTinfloatadd{#4}{#1}}{#2}{#3,#1}{#4}}%
\def\XINT_flseqb:f:csv_p #1#2%
{%
    \xintifCmp {#1}{#2}%
    \XINT_flseqb:f:csv_pa\XINT_flseqb:f:csv_pb\XINT_flseqb:f:csv_pc {#1}{#2}%
}%
\def\XINT_flseqb:f:csv_pb #1#2#3#4{#3,#1}%
\def\XINT_flseqb:f:csv_pc #1#2#3#4{#3}%
\def\XINT_flseqb:f:csv_bg #1{\if #1n\expandafter\XINT_flseqb:f:csv_na\else
```

```
                                   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
\def\XINT_flseqb:f:csv_na #1#2#3#4{\expandafter\XINT_flseqb:f:csv_n\expandafter
                           {\romannumeral0\XINTinfloatadd{#4}{#1}}{#2}{#3,#1}{#4}}%
\def\XINT_flseqb:f:csv_n #1#2%
{%
    \xintifCmp {#1}{#2}%
    \XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_flseqb:f:csv_na {#1}{#2}%
}%
```

11.54 User defined functions: \xintdeffunc, \xintdefiifunc, \xintdeffloatfunc

1.2c (2015/11/12).

Note: it is possible to have same name assigned both to a variable and a function: things such as add(f(f), f=1..10) are possible.

1.2f (2016/03/08).

Comma separated expressions allowed (formerly this required using parenthesis \xintdeffunc foo(x,..):=(.., .., ..);

1.3c (2018/06/17).

Usage of \xintexprSafeCatcodes to be compatible with an active semi-colon at time of use; the colon was not a problem (see ##3) already.

```
\def\XINT_tmpa #1#2#3#4%
{%
  \def #1##1(##2)##3=##4;{%
   \edef\XINT_expr_tmpa {##1}%
   \edef\XINT_expr_tmpa {\xint_zapspaces_o \XINT_expr_tmpa}%
   \def\XINT_expr_tmpb {0}%
   \def\XINT_expr_tmpc {(##4)}%
   \xintFor ####1 in {##2} \do
      {\edef\XINT_expr_tmpb {\the\numexpr\XINT_expr_tmpb+\xint_c_i}%
       \edef\XINT_expr_tmpc {subs(\unexpanded\expandafter{\XINT_expr_tmpc},%
                           ####1=################\XINT_expr_tmpb)}%
      }%
   \expandafter\XINT_expr_defuserfunc
     \csname XINT_#2_func_\XINT_expr_tmpa\expandafter\endcsname
     \expandafter{\XINT_expr_tmpa}{#2}%
   \expandafter#3\csname XINT_#2_userfunc_\XINT_expr_tmpa\endcsname
                               [\XINT_expr_tmpb]{\XINT_expr_tmpc}%
   \ifxintverbose\xintMessage {xintexpr}{Info}
        {Function \XINT_expr_tmpa\space for \string\xint #4 parser
         associated to \string\XINT_#2_userfunc_\XINT_expr_tmpa\space
         with \ifxintglobaldefs global \fi meaning \expandafter\meaning
         \csname XINT_#2_userfunc_\XINT_expr_tmpa\endcsname}%
   \fi
   \xintexprRestoreCatcodes
  }%
}%
\def\xintdeffunc      {\xintexprSafeCatcodes\xintdeffunc_a}%
\def\xintdefiifunc    {\xintexprSafeCatcodes\xintdefiifunc_a}%
\def\xintdeffloatfunc {\xintexprSafeCatcodes\xintdeffloatfunc_a}%
\XINT_tmpa\xintdeffunc_a     {expr}  \XINT_NewFunc     {expr}%
```

```
\XINT_tmpa\xintdefiifunc_a   {iiexpr}\XINT_NewIIFunc   {iiexpr}%
\XINT_tmpa\xintdeffloatfunc_a{flexpr}\XINT_NewFloatFunc{floatexpr}%
\def\XINT_expr_defuserfunc #1#2#3%
{%
    \XINT_global
    \def #1##1##2##3{\expandafter ##1\expandafter ##2%
     \csname .=\XINT:expr:userfunc{#3}{#2}{\XINT_expr_unlock ##3}\endcsname
    }%
}%
\def\XINT:expr:userfunc #1#2#3%
   {\csname XINT_#1_userfunc_#2\expandafter\endcsname
    \romannumeral0\xintcsvtolistnonstripped{#3}}%
\catcode`~ 12
\def\XINT:newexpr:userfunc #1#2#3%
   {~xintExpandArgs{XINT_#1_userfunc_#2}{\xintCSVtoListNonStripped{#3}}}%
\catcode`~ 3
```

11.55 \xintNewFunction

1.2h (2016/11/20). Syntax is \xintNewFunction{<name>}[nb of arguments]{expression with #1, #2,... as in \xintNewExpr}. This defines a function for all three parsers but the expression parsing is delayed until function execution. Hence the expression admits all constructs, contrarily to \xintNewExpr or \xintdeffunc.

```
\def\XINT_expr_wrapit #1{\expandafter\XINT_expr_wrap\csname.=#1\endcsname}%
\def\xintNewFunction #1#2[#3]#4%
{%
  \edef\XINT_expr_tmpa {#1}%
  \edef\XINT_expr_tmpa {\xint_zapspaces_o \XINT_expr_tmpa}%
  \def\XINT_expr_tmpb ##1##2##3##4##5##6##7##8##9{#4}%
  \begingroup
    \ifcase #3\relax
        \toks0{}%
    \or \toks0{##1}%
    \or \toks0{##1##2}%
    \or \toks0{##1##2##3}%
    \or \toks0{##1##2##3##4}%
    \or \toks0{##1##2##3##4##5}%
    \or \toks0{##1##2##3##4##5##6}%
    \or \toks0{##1##2##3##4##5##6##7}%
    \or \toks0{##1##2##3##4##5##6##7##8}%
    \else \toks0{##1##2##3##4##5##6##7##8##9}%
    \fi
    \expandafter
  \endgroup\expandafter
  \XINT_global\expandafter
  \def\csname XINT_expr_macrofunc_\XINT_expr_tmpa\expandafter\endcsname
  \the\toks0\expandafter{\XINT_expr_tmpb
    {\XINT_expr_wrapit{##1}}{\XINT_expr_wrapit{##2}}{\XINT_expr_wrapit{##3}}%
    {\XINT_expr_wrapit{##4}}{\XINT_expr_wrapit{##5}}{\XINT_expr_wrapit{##6}}%
    {\XINT_expr_wrapit{##7}}{\XINT_expr_wrapit{##8}}{\XINT_expr_wrapit{##9}}}%
  \expandafter\XINT_expr_newfunction
    \csname XINT_expr_func_\XINT_expr_tmpa\expandafter\endcsname
    \expandafter{\XINT_expr_tmpa}{eval}\xintbareeval
  \expandafter\XINT_expr_newfunction
    \csname XINT_iiexpr_func_\XINT_expr_tmpa\expandafter\endcsname
    \expandafter{\XINT_expr_tmpa}{iieval}\xintbareiieval
  \expandafter\XINT_expr_newfunction
    \csname XINT_flexpr_func_\XINT_expr_tmpa\expandafter\endcsname
    \expandafter{\XINT_expr_tmpa}{floateval}\xintbarefloateval
  \ifxintverbose
    \xintMessage {xintexpr}{Info}
        {Function \XINT_expr_tmpa\space for the expression parsers is
         associated to \string\XINT_expr_macrofunc_\XINT_expr_tmpa\space
         with \ifxintglobaldefs global \fi meaning \expandafter\meaning
         \csname XINT_expr_macrofunc_\XINT_expr_tmpa\endcsname}%
  \fi
}%
\def\XINT_expr_newfunction #1#2#3#4%
{%
    \XINT_global
    \def#1##1##2##3{\expandafter ##1\expandafter ##2\romannumeral0%
       \XINT:expr:macrofunc{#4}{#3}{#2}{\XINT_expr_unlock##3}}%
}%
\def\XINT:expr:macrofunc #1#2#3#4%
{%
    #1\csname XINT_expr_macrofunc_#3\expandafter\endcsname
      \romannumeral0\xintcsvtolistnonstripped{#4}\relax
}%
\catcode`~ 12
\def\XINT:newexpr:macrofunc #1{%
\def\XINT:newexpr:macrofunc ##1##2##3##4%
{%
    \expandafter#1\csname.=~XINT:newexpr:macrofunc:a{##2}{##3}%
    {\xintCSVtoListNonStripped{##4}}\endcsname
}%
}\XINT:newexpr:macrofunc { }%
\catcode`~ 3
\def\XINT:newexpr:macrofunc:a #1#2#3%
{%
    \expandafter\XINT_expr_unlock\romannumeral0\csname xintbare#1\endcsname
    \csname XINT_expr_macrofunc_#2\endcsname#3\relax
}%
```

11.56 \xintNewExpr, \xintNewIExpr, \xintNewFloatExpr, \xintNewIIExpr

There was an \xintNewExpr already in 1.07 from May 2013, which was modified in September 2013 to work with the # macro parameter character, and then refactored into a more powerful version in

June 2014 for 1.1 release of 2014/10/28. List handling causes special challenges, addressed by \xintApply::csv, \xintApply:::csv, ... next.

Comments finally added 2015/12/11 (with later edits):

The whole point is to expand completely macros when they have only numerical arguments and to inhibit this expansion if not. This is done in a recursive way: the catcode 12 ~ is used to register a macro name whose expansion must be inhibited. Any argument itself starting with such a ~ will force use of ~ for the macro which receives it.

In this context the catcode 12 $ is used to signal a "virtual list" argument. It triggers insertion of \xintApply::csv or \xintApply:::csv for delayed handling later. This succeeds into handling inputs such as [#1..[#2]..#3][#4:#5]...

A final \scantokens converts the "~" prefixed names into real control sequences.

For this whole mechanism we need to have everything expressed using exclusively f-expandable macros. We avoid \csname...\endcsname like construct, but if absolutely needed perhaps we will do it ultimately.

For the iterating loops seq, iter, etc..., and dummy variables, we have no macros to our disposal to handle the case where the list of indices is not explicit. Moreover omit, abort, break can not work with non numerical data. Thus the whole mechanism is currently not appicable to them. It does work when the macro parameters (or variables for \xintdeffunc) do not intervene in the list of values to iterate over. But we can not delay expansion of dummy variables.

Comments added 2018/02/28:

At 1.3 of February 2018, there was important refactoring. Earlier, \XINT_expr_redefinemacros was a very big macro which made aliases of the dozens of macros (most from xintfrac and some defined especially by xintexpr for acting on csv lists primarily) involved in the expression rendering and then redefined them all to expand to their original selves only when applied to purely numeric arguments. At 1.3 only very few such re-definitions are made, as what is redefined are a limited number of core wrapper macros.

Only when the original macros have one or two arguments is it examined if they can expand immediately (this includes case of function having possibly only one, or possibly two arguments). For macros applying to three or more or an undefined number of arguments, we don't complicate matters into checking if expansion is possible, and we delay that expansion automatically (but if() and ifsgn() do check if first argument is numeric and expand to suitable branch in that case).

In particular any function defined by \xintdeffunc or \xintNewFunction (it is then basically only a macro abstraction) when used in new function definitions will never be expanded immediately, because the detection of whether they are applied to only numerical data has not yet been added. (this might be added in future).

Some aspects of the 1.3 refactoring have made recursive definition via \xintdeffunc possible (of course they always were via \xintNewFunction as the latter is but a wrapper of a standard TeX macro definition, where \xintexpr parsing is not at all involved).

A somewhat complicated layer (not modified at 1.3) is devoted to making possible the parsing of constructs such as [#1..[#2]..#3][#4:#5] or [#1..#2]*#3 and it seems to work. At 1.3, even esoteric construct such as [divmod(#1,#2)]*#3 is parsable by \xintNewExpr. (In \xintNewFloatExpr, don't forget \empty token so that square brackets are not mistaken for optional argument of \xintthefloatexpr; same for \xintdeffloatfunc.)

Side note: I wonder if I really had a good idea to define these list operations [..]*foo or foo^[...] which do not seem to occur in other languages with the meanings I used. And they caused me lots of efforts for support at \xintNewExpr level...

The catcode 12 dollar sign is used to signal when a macro can not be expanded but would produce a csv list. Furthermore some cases require f-expandable macros as the original code expanding in \xintexpr is in \csname context and did not need f-expandability.

As mentioned above, currently syntax with dummy variables can not go through where the values iterated over are not explicit; and omit, abort, break mechanisms are not parsable with non purely numerical data, in part because they are not implemented internally via pure f-expansion.

11.56.1 \xintApply::csv and \xintApply:::csv

```
\def\xintApply::csv #1#2%
   {\expandafter\XINT_applyon::_a\expandafter {\romannumeral`&&@#2}{#1}}%
\def\XINT_applyon::_a #1#2{\XINT_applyon::_b {#2}{}#1,,}%
\def\XINT_applyon::_b #1#2#3,{\expandafter\XINT_applyon::_c \romannumeral`&&@#3,{#1}{#2}}%
\def\XINT_applyon::_c #1{\if #1,\expandafter\XINT_applyon::_end
                                 \else\expandafter\XINT_applyon::_d\fi #1}%
\def\XINT_applyon::_d #1,#2{\expandafter\XINT_applyon::_e\romannumeral`&&@#2{#1},{#2}}%
\def\XINT_applyon::_e #1,#2#3{\XINT_applyon::_b {#2}{#3, #1}}%
\def\XINT_applyon::_end #1,#2#3{\xint_secondoftwo #3}%
\def\xintApply:::csv #1#2#3%
   {\expandafter\XINT_applyon:::_a\expandafter{\romannumeral`&&@#2}{#1}{#3}}%
\def\XINT_applyon:::_a #1#2#3{\XINT_applyon:::_b {#2}{#3}{}#1,,}%
\def\XINT_applyon:::_b #1#2#3#4,%
   {\expandafter\XINT_applyon:::_c \romannumeral`&&@#4,{#1}{#2}{#3}}%
\def\XINT_applyon:::_c #1{\if #1,\expandafter\XINT_applyon:::_end
                      \else\expandafter\XINT_applyon:::_d\fi #1}%
\def\XINT_applyon:::_d #1,#2#3%
   {\expandafter\XINT_applyon:::_e\expandafter
    {\romannumeral`&&@\xintApply::csv {#2{#1}}{#3}},{#2}{#3}}%
\def\XINT_applyon:::_e #1,#2#3#4{\XINT_applyon:::_b {#2}{#3}{#4, #1}}%
\def\XINT_applyon:::_end #1,#2#3#4{\xint_secondoftwo #4}%
```

11.56.2 Mysterious stuff

~ and $ of catcode 12 in what follows.

```
\catcode`~ 12
\catcode`$ 12 % $
\def\xint_ddfork #1$$#2#3\krof {#2}% $$
\def\XINT:NE:RApply::csv #1#2#3#4%
   {~xintApply::csv{~expandafter #2~xint_exchangetwo_keepbraces{#4}}{#3}}%
\def\XINT:NE:LApply::csv #1#2#3{~xintApply::csv{#2{#3}}}%
\def\XINT:NE:RLApply:::csv #1{~xintApply:::csv}%
\def\XINT:NE:two#1{\XINT:NE:two_{#1}{\detokenize{#1}}}%
\def\XINT:NE:two_#1#2#3#4%
    {\expandafter\XINT:NE:two_a\romannumeral`&&@#4!{#3}{#1}{#2}}%
\def\XINT:NE:two_a#1#2!#3#4#5%
    {\expandafter\XINT:NE:two_b\romannumeral`&&@#3!#1{#4}{#5}{#1#2}}%
\def\XINT:NE:two_b#1#2!#3#4#5{\XINT:NE:two_fork_dd#1#3{#4}{#5}{#1#2}}%
\def\XINT:NE:two_fork_dd #1#2{%
    \xint_ddfork
      #1#2\XINT:NE:RLApply:::csv
      #1$\XINT:NE:RApply::csv% $
      $#2\XINT:NE:LApply::csv% $
      $${\XINT:NE:two_fork_nn #1#2}% $$
    \krof
}%
\def\XINT:NE:two_fork_nn #1#2#3#4{%
    \if #1##\xint_dothis{#4}\fi
    \if  #1~\xint_dothis{#4}\fi
    \if #2##\xint_dothis{#4}\fi
    \if  #2~\xint_dothis{#4}\fi
    \xint_orthat{#3}%
}%
\def\XINT:NE:one#1#2{\expandafter\XINT:NE:one_a\romannumeral`&&@#2!#1}%
\def\XINT:NE:one_a#1#2!#3{%
    \if ###1\xint_dothis {\detokenize{#3}}\fi
    \if  ~#1\xint_dothis {\detokenize{#3}}\fi
    \if  $#1\xint_dothis {~xintApply::csv{\detokenize{#3}}}\fi %$
    \xint_orthat #3{#1#2}%
}%
\def\XINT:NE:oneopt#1[#2]#3%
   {\expandafter\XINT:NE:oneopt_a\romannumeral`&&@#3!{#2}#1}%
\def\XINT:NE:oneopt_a#1#2!#3#4%
   {\expandafter\XINT:NE:oneopt_b\romannumeral`&&@#3!#1#4{#1#2}}%
\def\XINT:NE:oneopt_b#1#2!#3#4%
   {\expandafter\XINT:NE:oneopt_fork#1#3#4{#1#2}}%
\def\XINT:NE:oneopt_fork#1#2#3#4{%
    \if1\if###11\else\if~#11\else\if###21\else\if~#21\else0\fi\fi\fi\fi
           \xint_dothis {\detokenize{#3}[#4]}\fi
    \if $#2\xint_dothis {~xintApply::csv{\detokenize{#3}[#4]}}\fi %$
    \xint_orthat{#3[#4]}%
}% pas complétement général, mais bon
\def\XINT:NE:csv #1{\detokenize{#1}}% radicalement fainéant
\def\XINT:newexpr:one:and:opt #1,#2,#3!#4#5%
{%
    \if\relax#3\relax\expandafter\xint_firstoftwo\else
                     \expandafter\xint_secondoftwo\fi
    {\XINT:NE:one#4}{\XINT:NE:oneopt#5[\XINT:NE:one\xintNum{#2}]}{#1}%
}%
\def\XINT:newexpr:tacitzeroifonearg #1,#2,#3!#4#5%
{%
    \if\relax#3\relax\expandafter\xint_firstoftwo\else
                     \expandafter\xint_secondoftwo\fi
    {\XINT:NE:two#4{0}}{\XINT:NE:two#5{\XINT:NE:one\xintNum{#2}}}{#1}%
}%
\def\XINT:newiiexpr:tacitzeroifonearg #1,#2,#3!#4%
{%
    \if\relax#3\relax\expandafter\xint_firstoftwo\else
                     \expandafter\xint_secondoftwo\fi
    {\XINT:NE:two#4{0}}{\XINT:NE:two#4{#2}}{#1}%
}%
\def\XINT:newexpr:insertdollar~{$noexpand$}%
\def\XINT:newexpr:two:to:two #1,#2,!#3%
{%
    \XINT:NE:two_
    {\expandafter\XINT:expr:totwo\romannumeral`&&@#3}%
    {$noexpand$expandafter~XINT:expr:totwo~romannumeral-`0\detokenize{#3}}%
    {#1}{#2}%
}%
\def\XINT:newflexpr:two:to:two #1,#2,!#3%
{%
    \XINT:NE:two_
    {#3}%
    {\expandafter\XINT:newexpr:insertdollar\detokenize{#3}}%
    {#1}{#2}%
}%
\def\XINT:newexpr:two:to:one #1,#2,!#3%
{%
    \XINT:NE:two#3{#1}{#2}%
}%
\let\XINT:newflexpr:two:to:one\XINT:newexpr:two:to:one
\def\xintiiifNotZeroNE:#1#2,#3,#4,%
{%
    \if1\if###11\else\if~#11\else\if$#11\else0%$
        \fi\fi\fi
    \xint_dothis{~xintiiifNotZero}\fi
    \xint_orthat\xintiiifNotZero
    {#1#2}{#3}{#4}%
}%
\def\xintifIntNE:#1#2,#3,#4,%
{%
    \if1\if###11\else\if~#11\else\if$#11\else0%$
        \fi\fi\fi
    \xint_dothis{~xintifInt}\fi
    \xint_orthat\xintifInt
    {#1#2}{#3}{#4}%
}%
\def\xintifFloatIntNE:#1#2,#3,#4,%
{%
    \if1\if###11\else\if~#11\else\if$#11\else0%$
        \fi\fi\fi
    \xint_dothis{~xintifFloatInt}\fi
    \xint_orthat\xintifFloatInt
    {#1#2}{#3}{#4}%
}%
\def\xintiiifOneNE:#1#2,#3,#4,%
{%
    \if1\if###11\else\if~#11\else\if$#11\else0%$
        \fi\fi\fi
    \xint_dothis{~xintiiifOne}\fi
    \xint_orthat\xintiiifOne
    {#1#2}{#3}{#4}%
}%
\def\xintifOneNE:#1#2,#3,#4,%
{%
    \if1\if###11\else\if~#11\else\if$#11\else0%$
        \fi\fi\fi
    \xint_dothis{~xintifOne}\fi
    \xint_orthat\xintifOne
    {#1#2}{#3}{#4}%
}%
\def\xintiiifSgnNE:#1#2,#3,#4,#5,%
{%
    \if1\if###11\else\if~#11\else\if$#11\else0%$
        \fi\fi\fi
    \xint_dothis{~xintiiifSgn}\fi
    \xint_orthat\xintiiifSgn
    {#1#2}{#3}{#4}{#5}%
}%
```

11.56.3 \XINT_expr_redefinemacros

Completely refactored at 1.3.

```
\def\XINT_expr_redefinemacros {%
  \let\XINT:NEhook:one\XINT:NE:one
  \let\XINT:NEhook:two\XINT:NE:two
  \let\XINT:NEhook:csv\XINT:NE:csv
  \let\XINT:expr:one:and:opt         \XINT:newexpr:one:and:opt
  \let\XINT:expr:one:or:two:nums     \XINT:newexpr:one:or:two:nums
  \let\XINT:iiexpr:one:or:two:       \XINT:newiiexpr:one:or:two:
  \let\XINT:expr:tacitzeroifonearg   \XINT:newexpr:tacitzeroifonearg
  \let\XINT:iiexpr:tacitzeroifonearg \XINT:newiiexpr:tacitzeroifonearg
  \let\XINT:expr:two:to:two   \XINT:newexpr:two:to:two
  \let\XINT:expr:two:to:one   \XINT:newexpr:two:to:one
  \let\XINT:flexpr:two:to:one \XINT:newflexpr:two:to:one
  \let\XINT:expr:two:to:one   \XINT:newexpr:two:to:one
  \let\XINT:flexpr:two:to:two \XINT:newflexpr:two:to:two
  \let\XINT:flexpr:two:to:one \XINT:newflexpr:two:to:one
  \let\xintiiifNotZero:       \xintiiifNotZeroNE:
  \let\xintifInt:             \xintifIntNE:
  \let\xintifFloatInt:        \xintifFloatIntNE:
  \let\xintiiifOne:           \xintiiifOneNE:
  \let\xintifOne:             \xintifOneNE:
  \let\xintiiifSgn:           \xintiiifSgnNE:
  \let\xintSeqNumeric::csv         \xintSeq::csv
  \let\xintiiSeqNumeric::csv       \xintiiSeq::csv
  \let\XINTinFloatSeqNumeric::csv  \XINTinFloatSeq::csv
  \let\xintSeqBNumeric::csv        \xintSeqB::csv
  \let\xintiiSeqBNumeric::csv      \xintiiSeqB::csv
  \let\XINTinFloatSeqBNumeric::csv\XINTinFloatSeqB::csv
  \def\xintSeq::csv
    {\XINT:NE:two_\xintSeqNumeric::csv{$noexpand$xintSeq::csv}}%
  \def\xintiiSeq::csv
    {\XINT:NE:two_\xintiiSeqNumeric::csv{$noexpand$xintiiSeq::csv}}%
  \def\XINTinFloatSeq::csv
    {\XINT:NE:two_\XINTinFloatSeqNumeric::csv{$noexpand$XINTinFloatSeq::csv}}%
  \def\xintSeqB::csv
    {\XINT:NE:two_\xintSeqBNumeric::csv{$noexpand$xintSeqB:f:csv}}%
  \def\xintiiSeqB::csv
    {\XINT:NE:two_\xintiiSeqBNumeric::csv{$noexpand$xintiiSeqB:f:csv}}%
  \def\XINTinFloatSeqB::csv
    {\XINT:NE:two_\XINTinFloatSeqBNumeric::csv{$noexpand$XINTinFloatSeqB:f:csv}}%
  \def\xintListSel:x:csv  {~xintListSel:f:csv }%
  \let\XINT:expr:userfunc \XINT:newexpr:userfunc
  \let\XINT:expr:macrofunc\XINT:newexpr:macrofunc
  \def\XINTinRandomFloatSdigits{~XINTinRandomFloatSdigits }%
  \def\XINTinRandomFloatSixteen{~XINTinRandomFloatSixteen }%
  \def\xintiiRandRange{~xintiiRandRange }%
  \def\xintiiRandRangeAtoB{~xintiiRandRangeAtoB }%
}%
```

11.56.4 \XINT_expr_redefineprints

This is used by \xintNewExpr but not by \xintdeffunc.

```
\def\XINT_expr_redefineprints
{%
   \def\XINT_flexpr_noopt
     {\expandafter\XINT_flexpr_withopt_b\expandafter-\romannumeral0\xintbarefloateval }%
   \def\XINT_flexpr_withopt_b ##1##2%
     {\expandafter\XINT_flexpr_wrap\csname .;##1.=\XINT_expr_unlock  ##2\endcsname }%
   \def\XINT_expr_unlock_sp ##1.;##2##3.=##4!%
     {\if -##2\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
      \XINTdigits{{##2##3}}{##4}}%
   \def\XINT_expr_print ##1%
     {\expandafter\xintSPRaw::csv\expandafter
                 {\romannumeral`&&@\XINT_expr_unlock ##1}}%
   \def\XINT_iiexpr_print ##1%
     {\expandafter\xintCSV::csv\expandafter
                 {\romannumeral`&&@\XINT_expr_unlock ##1}}%
   \def\XINT_boolexpr_print ##1%
     {\expandafter\xintIsTrue::csv\expandafter
                 {\romannumeral`&&@\XINT_expr_unlock ##1}}%
   \def\xintCSV::csv     {~xintCSV::csv    }%
   \def\xintSPRaw::csv   {~xintSPRaw::csv  }%
   \def\xintPFloat::csv  {~xintPFloat::csv }%
   \def\xintIsTrue::csv  {~xintIsTrue::csv }%
   \def\xintRound::csv   {~xintRound::csv  }%
}%
```

11.56.5 \xintNewExpr, ..., at last.

1.2c modifications to accomodate \XINT_expr_deffunc_newexpr etc..
  1.2f adds token \XINT_newexpr_clean to be able to have a different \XINT_newfunc_clean.

```
\def\xintNewExpr     {\XINT_NewExpr\XINT_expr_redefineprints\xint_firstofone
                                   \xinttheexpr\XINT_newexpr_clean}%
\def\xintNewFloatExpr{\XINT_NewExpr\XINT_expr_redefineprints\xint_firstofone
                                   \xintthefloatexpr\XINT_newexpr_clean}%
\def\xintNewIExpr    {\XINT_NewExpr\XINT_expr_redefineprints\xint_firstofone
                                   \xinttheiexpr\XINT_newexpr_clean}%
\def\xintNewIIExpr   {\XINT_NewExpr\XINT_expr_redefineprints\xint_firstofone
                                   \xinttheiiexpr\XINT_newexpr_clean}%
\def\xintNewBoolExpr {\XINT_NewExpr\XINT_expr_redefineprints\xint_firstofone
                                   \xinttheboolexpr\XINT_newexpr_clean}%
\def\XINT_newexpr_clean #1>{\noexpand\romannumeral`&&@}%
```

1.2c for \xintdeffunc, \xintdefiifunc, \xintdeffloatfunc.
At 1.3, NewFunc does not use a comma delimited pattern anymore.

```
\def\XINT_NewFunc
 {\XINT_NewExpr{}\xint_gobble_i\xintthebareeval\XINT_newfunc_clean}%
\def\XINT_NewFloatFunc
 {\XINT_NewExpr{}\xint_gobble_i\xintthebarefloateval\XINT_newfunc_clean}%
\def\XINT_NewIIFunc
 {\XINT_NewExpr{}\xint_gobble_i\xintthebareiieval\XINT_newfunc_clean}%
\def\XINT_newfunc_clean #1>{}%
```

1.2c adds optional logging. For this needed to pass to _NewExpr_a the macro name as parameter.

Up to and including 1.2c the definition was global. Starting with 1.2d it is done locally.

Modified at 1.3c so that \XINT_NewFunc et. al. do not execute the \xintexprSafeCatcodes, as it is now already done earlier by \xintdeffunc: and as already #2 was either \xint_firstofone (for \xintNewExpr et. al.) or \xint_gobble_i (for \XINT_NewFunc et. al.) we can use that #2. This is only to avoid doing twice the catcodes, as anyhow there is an \endgroup coming later, so external \xintexprRestoreCatcodes would not have been compromised.

```
\def\XINT_NewExpr #1#2#3#4#5#6[#7]%
{%
 \begingroup
    \ifcase #7\relax
        \toks0 {\endgroup\XINT_global\def#5}%
    \or \toks0 {\endgroup\XINT_global\def#5##1}%
    \or \toks0 {\endgroup\XINT_global\def#5##1##2}%
    \or \toks0 {\endgroup\XINT_global\def#5##1##2##3}%
    \or \toks0 {\endgroup\XINT_global\def#5##1##2##3##4}%
    \or \toks0 {\endgroup\XINT_global\def#5##1##2##3##4##5}%
    \or \toks0 {\endgroup\XINT_global\def#5##1##2##3##4##5##6}%
    \or \toks0 {\endgroup\XINT_global\def#5##1##2##3##4##5##6##7}%
    \or \toks0 {\endgroup\XINT_global\def#5##1##2##3##4##5##6##7##8}%
    \or \toks0 {\endgroup\XINT_global\def#5##1##2##3##4##5##6##7##8##9}%
    \fi
    #2\xintexprSafeCatcodes
    \XINT_expr_redefinemacros
    #1%
    \XINT_NewExpr_a #2#3#4#5%
}%
```

1.2d's \xintNewExpr makes a local definition. In earlier releases, the definition was global.

\the\toks0 inserts the \endgroup, but this will happen after \XINT_tmpa has already been expanded...

The 1 is \xint_firstofone for \xintNewExpr, \xint_gobble_i for \xintdeffunc.

```
\catcode`~ 13 \catcode`@ 14 \catcode`\% 6 \catcode`# 12 \catcode`$ 11 @ $
\def\XINT_NewExpr_a %1%2%3%4%5@
{@
    \def\XINT_tmpa %%1%%2%%3%%4%%5%%6%%7%%8%%9{%5}@
    \def~{$noexpand$}@
    \catcode`: 11 \catcode`_ 11
    \catcode`# 12 \catcode`~ 13 \escapechar 126
    \endlinechar -1 \everyeof {\noexpand }@
    \edef\XINT_tmpb
    {\scantokens\expandafter{\romannumeral`&&@\expandafter
     %2\XINT_tmpa{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}\relax}@
    }@
    \escapechar 92 \catcode`# 6 \catcode`$ 0 @ $
    \edef\XINT_tmpa %%1%%2%%3%%4%%5%%6%%7%%8%%9@
      {\scantokens\expandafter{\expandafter%3\meaning\XINT_tmpb}}@
    \the\toks0\expandafter
      {\XINT_tmpa{%%1}{%%2}{%%3}{%%4}{%%5}{%%6}{%%7}{%%8}{%%9}}@
    %1{\ifxintverbose
        \xintMessage{xintexpr}{Info}@
                    {\string%4\space now with @
                     \ifxintglobaldefs global \fi meaning \meaning%4}@
       \fi}@
}@
\catcode`% 14
```

11.56.6 \ifxintsafecatcodes, \xintexprSafeCatcodes, \xintexprRestoreCatcodes

1.3c (2018/06/17).

Added \ifxintsafecatcodes to allow nesting

```
\newif\ifxintexprsafecatcodes
\let\xintexprRestoreCatcodes\empty
\def\xintexprSafeCatcodes
{%
  \unless\ifxintexprsafecatcodes
    \edef\xintexprRestoreCatcodes  {%
        \catcode59=\the\catcode59   % ;
        \catcode34=\the\catcode34   % "
        \catcode63=\the\catcode63   % ?
        \catcode124=\the\catcode124 % |
        \catcode38=\the\catcode38   % &
        \catcode33=\the\catcode33   % !
        \catcode93=\the\catcode93   % ]
        \catcode91=\the\catcode91   % [
        \catcode94=\the\catcode94   % ^
        \catcode95=\the\catcode95   % _
        \catcode47=\the\catcode47   % /
        \catcode41=\the\catcode41   % )
        \catcode40=\the\catcode40   % (
        \catcode42=\the\catcode42   % *
        \catcode43=\the\catcode43   % +
        \catcode62=\the\catcode62   % >
        \catcode60=\the\catcode60   % <
        \catcode58=\the\catcode58   % :
        \catcode46=\the\catcode46   % .
        \catcode45=\the\catcode45   % -
        \catcode44=\the\catcode44   % ,
        \catcode61=\the\catcode61   % =
        \catcode96=\the\catcode96   % `
        \catcode32=\the\catcode32\relax % space
        \noexpand\xintexprsafecatcodesfalse
    }%
  \fi
  \xintexprsafecatcodestrue
        \catcode59=12  % ;
        \catcode34=12  % "
        \catcode63=12  % ?
        \catcode124=12 % |
        \catcode38=4   % &
        \catcode33=12  % !
        \catcode93=12  % ]
        \catcode91=12  % [
        \catcode94=7   % ^
        \catcode95=8   % _
        \catcode47=12  % /
        \catcode41=12  % )
        \catcode40=12  % (
        \catcode42=12  % *
        \catcode43=12  % +
        \catcode62=12  % >
        \catcode60=12  % <
        \catcode58=12  % :
        \catcode46=12  % .
        \catcode45=12  % -
        \catcode44=12  % ,
        \catcode61=12  % =
        \catcode96=12  % `
        \catcode32=10  % space
}%
\let\XINT_tmpa\relax \let\XINT_tmpb\relax \let\XINT_tmpc\relax
\XINT_restorecatcodes_endinput%
```

12 Per package indices of control sequences

12.1 Index of xintkernel

12.2 Index of xinttools

12.3 Index of xintcore

12.4 Index of xint

12.5 Index of xintbinhex

12.6 Index of xintgcd

12.7 Index of xintfrac

12.8 Index of xintseries

12.9 Index of xintcfrac

12.10 Index of xintexpr

Particularly for this package, there are quite a few macros which are defined via constructs such as \expandafter \def \csname ...\endcsname, and their indexing needs some extra mark-up which is yet to be added to the commented source code.

13 Cumulative index

Currently there are some macros which did not make it to the indices because they are defined via constructs such as `\expandafter\def\csname ...\endcsname`, thus their indexing needs some extra mark-up which is yet to be added. This is particularly true for some definitions done by xintexpr.

The first indicated page number is the one where the macro is first encountered, hence this is most likely also the page where it gets defined (if it is not one of those which are only used as delimiters).

14 Cumulative line count

xintkernel:	554.
xinttools:	1454.
xintcore:	2182.
xint:	1496.
xintbinhex:	472.
xintgcd:	434.
xintfrac:	3220.
xintseries:	386.
xintcfrac:	1029.
xintexpr:	3167.

Total number of code lines: 14394. (but 3325 lines among them start either with {% or with }%.)
Each package starts with circa 50 lines dealing with catcodes, package identification and reloading management, also for Plain TeX. Version 1.3c of 2018/06/17.