The xint source code

Jean-François Burnol
jfbu (at) free (dot) fr
Package version: 1.2j (2016/12/22); documentation date: 2016/12/22.
From source file xint.dtx. Time-stamp: <22-12-2016 at 22:59:29 CET>.

Contents

This is 1.2j of 2016/12/22.

- Some portions 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.
- Release 1.2i of 2016/12/13 has rewritten some legacy macros like \xintDSR or \xintDecSplit in the style of the techniques of 1.2. But this means also that they are now 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 has entirely rewritten the core arithmetic routines in xintcore. Many macros benefit indirectly from the faster core routines. The new model is yet to be extended to other portions of the code: for example the routines of xintbinhex could be made faster for very big inputs if they adopted some techniques from the implementation of the basic arithmetic routines. The parser of xintexpr is faster and does not have a limit at 5000 digits per number anymore.

- 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.

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.

1 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.

First appeared as a separate package with release 1.1.

1.2i adds \xintreplicate, \xintgobble and \xintLengthUpTo.

1.1 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.

Starting with version 1.06 of the package, also ` must be catcode-protected, because we replace everywhere in the code the twice-expansion done with \expandafter by the systematic use of \romannumeral-`0.

Starting with version 1.06b I decide that I suffer from an indigestion of @ signs, so I replace them all with underscores _, à la LaTeX3.

Release 1.09b is more economical: some macros are defined already in xint.sty (now in xintkernel.sty) and re-used in other modules. All catcode changes have been unified and \XINT_storecatcodes will be used by each module to redefine \XINT_restorecatcodes_endinput in case catcodes have changed in-between the loading of xint.sty (now xintkernel.sty) and the module (not very probable but...).

```
\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%
\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   % ,
```

```
63            \catcode35=\the\catcode35   % #
64            \catcode95=\the\catcode95   % _
65            \catcode125=\the\catcode125 % }
66            \catcode123=\the\catcode123 % {
67            \endlinechar=\the\endlinechar
68            \catcode13=\the\catcode13   % ^^M
69            \catcode32=\the\catcode32   %
70            \catcode61=\the\catcode61\relax   % =
71      }%
72      \edef\XINT_restorecatcodes_endinput
73      {%
74            \XINT_restorecatcodes\noexpand\endinput %
75      }%
76      \def\XINT_setcatcodes
77      {%
78        \catcode61=12   % =
79        \catcode32=10   % space
80        \catcode13=5    % ^^M
81        \endlinechar=13 %
82        \catcode123=1   % {
83        \catcode125=2   % }
84        \catcode95=11   % _ LETTER
85        \catcode35=6    % #
86        \catcode44=12   % ,
87        \catcode45=12   % -
88        \catcode46=12   % .
89        \catcode58=11   % : LETTER
90        \catcode60=12   % <
91        \catcode62=12   % >
92        \catcode43=12   % +
93        \catcode42=12   % *
94        \catcode40=12   % (
95        \catcode41=12   % )
96        \catcode47=12   % /
97        \catcode96=12   % `
98        \catcode94=11   % ^ LETTER
99        \catcode36=3    % $
100       \catcode91=12   % [
101       \catcode93=12   % ]
102       \catcode33=12   % !
103       \catcode64=11   % @ LETTER
104       \catcode38=7    % & for \romannumeral`&&@ trick.
105       \catcode124=12  % |
106       \catcode63=11   % ? LETTER
107       \catcode34=12   % "
108       \catcode39=12   % '
109       \catcode126=3   % ~ MATH
110       \catcode59=12   % ;
111       \catcode0=12    % for \romannumeral`&&@ trick
112     }%
113     \XINT_setcatcodes
114 }%
```

```
\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
}%
```

1.2 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}%
  [2016/12/22 1.2j Paraphernalia for the xint packages (JFB)]%
```

1.3 Constants

1.2 decides to move them to xintkernel from xintcore and xint. The \count's are left in their respective packages.

```
\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_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
```

1.4 Token management utilities

```
\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{}%
\long\def\xint_firstofone  #1{#1}%
\long\def\xint_firstoftwo  #1#2{#1}%
\long\def\xint_secondoftwo #1#2{#2}%
\long\def\xint_gobble_thenstop  #1{ }%
\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}}%
```

1.5 “gob til” macros and UD style fork

Some moved here from xintcore by release 1.2.
1.2i finally makes all gobbling macros long.

```
\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_exclam #1!{}% catcode 12 exclam
\long\def\xint_gob_til_dot    #1.{}%
\long\def\xint_gob_til_G     #1G{}%
\long\def\xint_gob_til_minus #1-{}%
\long\def\xint_gob_til_relax #1\relax {}%
\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_relax\relax
\def\xint_brelax {\xint_relax }%
\long\def\xint_gob_til_xint_relax #1\xint_relax {}%
```

1.6 \xint_afterfi

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

1.7 \xint_bye, \xint_Bye

\xint_Bye is new with 1.2i 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 { }%
```

1.8 \xintdothis, \xintorthat

New with 1.1. Public names without underscores with 1.2. Used as \if..\xint_dothis{..}\fi <multiple times> followed by \xint_orthat{...}. To be used with less probable things first.

```
\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
```

1.9 \xint_zapspaces

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

\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, and not 12\relax. Imagine also:

\the\numexpr 1 2\expandafter.\the\numexpr ...

The spaces will stop the \numexpr, and the \expandafter will not be immediately executed. Thus we have to get rid of spaces in contexts where arguments are fetched by delimited macros and fed to \numexpr (or for any reason can contain spaces). I apply this corrective treatment so far only in xintexpr but perhaps I should in xintfrac too. As said above, perhaps the zapspaces should force expansion too, but I leave it standing.

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

Made long with 1.2i.

```
\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}%
```

1.10 \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
```

1.11 \xintReverseOrder

\xintReverseOrder: does not expand its argument. Thus one must use some \expandafter if argument is a macro. Attention: removes braces.

A faster reverse, but only usable with digits, is provided by \xintReverseDigits macro from 1.2 xintcore.sty.

1.2g has (not user documented) \xintCSVReverse in xinttools.sty.

```
\def\xintReverseOrder {\romannumeral0\xintreverseorder }%
\long\def\xintreverseorder #1%
{%
    \XINT_rord_main {}#1%
      \xint_relax
        \xint_bye\xint_bye\xint_bye\xint_bye
        \xint_bye\xint_bye\xint_bye\xint_bye
      \xint_relax
}%
\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}%
}%
\long\edef\XINT_rord_cleanup\xint_bye\XINT_rord_main #1#2\xint_relax
```

```
{%
    \noexpand\expandafter\space\noexpand\xint_gob_til_xint_relax #1%
}%
```

1.12 \xintLength

\xintLength does not expand its argument. See \xintNthElt{0} from xinttools.sty which f-expands its argument.
1.2g has (not user documented) \xintCSVLength in xinttools.sty.
1.2i has rewritten this venerable macro. New code is about 40% faster across all lengths. Was again slightly changed for 1.2j (cosmetic).

```
\def\xintLength {\romannumeral0\xintlength }%
\long\edef\xintlength #1%
{%
    \noexpand\expandafter\space
    \noexpand\the\numexpr\noexpand\XINT_length_loop
    #1\xint_relax\xint_relax\xint_relax\xint_relax
      \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
      \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_\noexpand\xint_bye
    \relax
}%
\long\def\XINT_length_loop #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_xint_relax #9\XINT_length_finish_a\xint_relax
    \xint_c_ix+\XINT_length_loop
}%
\def\XINT_length_finish_a\xint_relax\xint_c_ix+\XINT_length_loop
    #1#2#3#4#5#6#7#8#9%
{%
    #9\xint_bye
}%
```

1.13 \xintLastItem

New with 1.2i (2016/12/10). Output empty if input empty. One level of braces removed in output.

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

1.14 \xintLengthUpTo

1.2i for use by \xintKeep and \xintTrim.
  \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)).
  Does not expand its second argument (it is used by \xintKeep and \xintTrim with already expanded argument). Not a user macro. 1.2j rewrote the ending and changed the loop interface. The argument N **must be non-negative**.

```
\def\xintLengthUpTo {\romannumeral0\xintlengthupto}%
\long\def\xintlengthupto #1#2%
{%
    \expandafter\XINT_lengthupto_loop
    \the\numexpr#1.#2\xint_relax\xint_relax\xint_relax\xint_relax
         \xint_relax\xint_relax\xint_relax\xint_relax
         \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_relax #9\XINT_lengthupto_finish_a\xint_relax
    \expandafter\XINT_lengthupto_loop_a\the\numexpr #1-\xint_c_viii.%
}%
\def\XINT_lengthupto_finish_a\xint_relax\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
}%
```

1.15 \xintreplicate

Added with 1.2i. This is cloned from \prg_replicate:nn from expl3, 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 |#1| replications when input #1 is negative and then activates an error triggering macro; here we simply do nothing when #1 is negative.

When #1 is already explicit digits (even #1=0, but non-negative) one can call the macro directly as \romannumeral\XINT_rep #1\endcsname {foo} to skip the \numexpr.

Expansion must be triggered by a \romannumeral.

```
\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}%
```

1.16 \xintgobble

Added with 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.

Like for \xintreplicate, a \romannumeral is needed to trigger expansion.

I wrote in a similar spirit an \xintcount. But it proved slower than the upgraded 1.2i \xintLength in all the range up to thousands of tokens.

```
\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}%
```

1.17 \xintMessage, \ifxintverbose

1.2c added it 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
\XINT_restorecatcodes_endinput%
```

2 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.

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   % }
  \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
```

2.2 Package identification

```
\XINT_providespackage
\ProvidesPackage{xinttools}%
  [2016/12/22 1.2j 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!
```

2.3 \xintgodef, \xintgoodef, \xintgfdef

1.09i. For use in \xintAssign.

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

2.4 \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_relax, 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_relax\xint_relax\xint_relax\xint_relax
                        \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
}%
\long\def\xintrevwithbracesnoexpand #1%
{%
    \XINT_revwbr_loop {}%
    #1\xint_relax\xint_relax\xint_relax\xint_relax
      \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
}%
\long\def\XINT_revwbr_loop #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_xint_relax #9\XINT_revwbr_finish_a\xint_relax
    \XINT_revwbr_loop {{#9}{#8}{#7}{#6}{#5}{#4}{#3}{#2}#1}%
}%
\long\def\XINT_revwbr_finish_a\xint_relax\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.

```
\edef\XINT_revwbr_finish_c #1#2\Z {\noexpand\expandafter\space #1}%
```

2.5 \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 }%
```

defined via an \edef in order to inject space tokens inside.

```
\long\edef\xintzapfirstspaces #1%
  {\noexpand\XINT_zapbsp_a \space #1\xint_relax \space\space\xint_relax }%
\xint_firstofone {\long\edef\XINT_zapbsp_a #1 } %<- space token here
{%
```

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_relax. 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_relax<sp><sp>\xint_relax

```
  \noexpand\XINT_zapbsp_again? #1\noexpand\xint_bye\noexpand\XINT_zapbsp_b #1\space\space
}%
\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_relax #2\xint_relax {#1}%
```

2.6 \xintZapLastSpaces

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

```
\def\xintZapLastSpaces {\romannumeral0\xintzaplastspaces }%
```

Next macro is defined via an \edef for the space tokens.

```
\long\edef\xintzaplastspaces #1{\noexpand\XINT_zapesp_a {}\noexpand\empty#1%
                                \space\space\noexpand\xint_bye\xint_relax}%
```

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_relax. 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_relax
    {\XINT_zapesp_end? #3\XINT_zapesp_e {#2#1}\empty #3\xint_relax }%
```

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_relax { #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.

```
\long\edef\XINT_zapesp_e #1{\noexpand \XINT_zapesp_a {#1\space\space}}%
```

2.7 \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 }%
\long\edef\xintzapspaces #1% like \xintZapFirstSpaces.
                    {\noexpand\XINT_zapsp_a \space #1\xint_relax \space\space\xint_relax }%
\xint_firstofone {\long\edef\XINT_zapsp_a #1 } %
  {\noexpand\XINT_zapsp_again? #1\noexpand\xint_bye\noexpand\XINT_zapsp_b #1\space\space}%
\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 }%
\long\edef\XINT_zapsp_c #1\xint_relax #2\xint_relax {\noexpand\XINT_zapesp_a
    {}\noexpand \empty #1\space\space\noexpand\xint_bye\xint_relax }%
```

2.8 \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_relax\xint_relax
                          \xint_bye\xintzapspaces {#1}}%
\long\def\XINT_zapspb_one? #1#2%
   {\xint_gob_til_xint_relax #1\XINT_zapspb_onlyspaces\xint_relax
    \xint_gob_til_xint_relax #2\XINT_zapspb_bracedorone\xint_relax
    \xint_bye {#1}}%
\def\XINT_zapspb_onlyspaces\xint_relax
    \xint_gob_til_xint_relax\xint_relax\XINT_zapspb_bracedorone\xint_relax
    \xint_bye #1\xint_bye\xintzapspaces #2{ }%
\long\def\XINT_zapspb_bracedorone\xint_relax
    \xint_bye #1\xint_relax\xint_bye\xintzapspaces #2{ #1}%
```

2.9 \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,\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}}%
```

2.10 \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.

```
\def\xintListWithSep {\romannumeral0\xintlistwithsep }%
\def\xintListWithSepNoExpand {\romannumeral0\xintlistwithsepnoexpand }%
\long\def\xintlistwithsep #1#2%
    {\expandafter\XINT_lws\expandafter {\romannumeral`&&@#2}{#1}}%
\long\def\XINT_lws #1#2{\XINT_lws_start {#2}#1\xint_bye }%
\long\def\xintlistwithsepnoexpand #1#2{\XINT_lws_start {#1}#2\xint_bye }%
\long\def\XINT_lws_start #1#2%
{%
    \xint_bye #2\XINT_lws_dont\xint_bye
    \XINT_lws_loop_a {#2}{#1}%
}%
\long\def\XINT_lws_dont\xint_bye\XINT_lws_loop_a #1#2{ }%
\long\def\XINT_lws_loop_a #1#2#3%
{%
    \xint_bye #3\XINT_lws_end\xint_bye
    \XINT_lws_loop_b {#1}{#2#3}{#2}%
}%
\long\def\XINT_lws_loop_b #1#2{\XINT_lws_loop_a {#1#2}}%
\long\def\XINT_lws_end\xint_bye\XINT_lws_loop_b #1#2#3{ #1}%
```

2.11 \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_relax\xint_relax\xint_relax\xint_relax
      \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
      \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_relax\xint_relax\xint_relax\xint_relax\xint_relax
       \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
    \xint_bye
}%
\def\XINT_nthelt_pos_done #1{%
\long\def\XINT_nthelt_pos_done ##1##2\xint_bye{%
  \xint_gob_til_xint_relax##1\expandafter#1\xint_gobble_ii\xint_relax#1##1}%
}\XINT_nthelt_pos_done{ }%
```

2.12 \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_relax\xint_relax\xint_relax\xint_relax
      \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
      \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_relax\xint_relax\xint_relax\xint_relax
         \xint_relax\xint_relax\xint_relax\xint_relax
         \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}%
```

2.13 \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_relax\xint_relax\xint_relax\xint_relax
         \xint_relax\xint_relax\xint_relax\xint_relax
         \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}%
```

2.14 \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_relax\xint_relax\xint_relax\xint_relax
      \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
```

```
      \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
}%
\edef\XINT_trim_trimall {\noexpand\expandafter\space\noexpand\xint_bye}%
```

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_relax\xint_relax\xint_relax\xint_relax\xint_relax
       \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
    \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_relax #9\XINT_trim_toofew\xint_relax-\xint_c_ix.%
}%
\def\XINT_trim_toofew\xint_relax{*\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_relax #2\xint_bye {#1}%
```

2.15 \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_relax\xint_relax\xint_relax\xint_relax
      \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
      \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
}%
```

2.16 \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}%
```

2.17 \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}%
```

2.18 \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}}}%
```

2.19 \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}}%
```

2.20 \xintiloop, \xintiloopindex, \xintouteriloopindex, \xintbreakiloop, \xintbreakiloopanddo, \xintiloopskiptonext, \xintiloopskipandredo

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

```
\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\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\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}}%
```

2.21 \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
}%
```

2.22 \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
}%
```

2.23 \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}%
}%
```

2.24 \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}%
}%
```

2.25 \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
```

2.26 \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_relax\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_relax\to #2%
{%
    \XINT_assign_def #2{#1}%
}%
\long\def\XINT_assign_c #1%
{%
    \def\xint_temp {#1}%
    \ifx\xint_temp\xint_brelax
        \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_relax
    \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_brelax\xint_temp
       \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 {\xintError:ArrayIndexIsNegative\space }%
```

```
1076         \else
1077         \xint_afterfi {%
1078               \ifnum ##1>#2
1079                   \xint_afterfi {\xintError:ArrayIndexBeyondLimit\space }%
1080               \else\xint_afterfi
1081           {\expandafter\expandafter\expandafter\space\csname #3##1\endcsname}%
1082               \fi}%
1083         \fi
1084      }%
1085 }%
1086 \let\xintDigitsOf\xintAssignArray
```

2.27 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.

2.27.1 \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_relax,\xint_relax,\xint_relax,\xint_relax,%
      \xint_relax,\xint_relax,\xint_relax,\xint_relax,\xint_relax,%
      \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_relax #1\xint_c_\xint_bye\xint_relax
    \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_relax #9\XINT_length:f:csv_finish\xint_relax
    \xint_c_ix+\XINT_length:f:csv_loop
}%
\def\XINT_length:f:csv_finish\xint_relax\xint_c_ix+\XINT_length:f:csv_loop
    #1,#2,#3,#4,#5,#6,#7,#8,#9,{#9\xint_bye}%
```

2.27.2 \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_relax,\xint_relax,\xint_relax,\xint_relax,%
         \xint_relax,\xint_relax,\xint_relax,\xint_relax,%
```

```
          \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_relax #2\XINT_lengthupto:f:csv_empty\xint_relax
    \XINT_lengthupto:f:csv_loop_b #1.#2%
}%
\def\XINT_lengthupto:f:csv_empty\xint_relax
    \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_relax #9\XINT_lengthupto:f:csv_finish_a\xint_relax
    \expandafter\XINT_lengthupto:f:csv_loop_a\the\numexpr #1-\xint_c_viii.%
}%
\def\XINT_lengthupto:f:csv_finish_a\xint_relax
    \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
}%
```

2.27.3 \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_relax,\xint_relax,\xint_relax,\xint_relax,%
      \xint_relax,\xint_relax,\xint_relax,\xint_relax,\xint_relax,%
      \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_relax,\xint_relax,\xint_relax,\xint_relax,%
         \xint_relax,\xint_relax,\xint_relax,\xint_relax,%
         \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}%
```

2.27.4 \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_relax,\xint_relax,\xint_relax,\xint_relax,%
      \xint_relax,\xint_relax,\xint_relax,\xint_relax,\xint_relax,%
      \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_relax,\xint_relax,\xint_relax,\xint_relax,\xint_relax,%
       \xint_relax,\xint_relax,\xint_relax,\xint_relax,\xint_relax\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_relax #9\XINT_trim:f:csv_toofew\xint_relax-\xint_c_ix.%
}%
\def\XINT_trim:f:csv_toofew\xint_relax{*\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_relax #2\xint_bye{#1}%
```

2.27.5 \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_relax,\xint_relax,\xint_relax,\xint_relax,%
      \xint_relax,\xint_relax,\xint_relax,\xint_relax,\xint_relax,%
      \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_relax,\xint_relax,\xint_relax,\xint_relax,\xint_relax,%
       \xint_relax,\xint_relax,\xint_relax,\xint_relax,\xint_relax,\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_relax##1\XINT_nthelt:f:csv_pos_cleanup\xint_relax#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_relax (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_relax} %
   #1\xint_relax{ #1}%
```

2.27.6 \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_relax,%
        \xint_bye,\xint_bye,\xint_bye,\xint_bye,%
        \xint_bye,\xint_bye,\xint_bye,\xint_bye,%
      \xint_relax
}%
\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_relax
{%
    \XINT_reverse:f:csv_finish #1%
}%
\long\def\XINT_reverse:f:csv_finish #1\xint_relax,{ }%
```

2.27.7 Public names for the undocumented csv macros

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\XINT_tmpa\relax \let\XINT_tmpb\relax \let\XINT_tmpc\relax
\XINT_restorecatcodes_endinput%
```

3 Package xintcore implementation

Got split off from xint with release 1.1, which also added the new macro \xintiiDivRound.

> The core arithmetic routines have been entirely rewritten for release 1.2.
> The commenting continues (2016/12/22) 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.

Also, starting with 1.2, \xintAdd etc... are defined only via xintfrac. Only \xintiAdd and \xintiiAdd (etc...) are provided via xintcore.

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@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
```

3.2 Package identification

```
\XINT_providespackage
\ProvidesPackage{xintcore}%
  [2016/12/22 1.2j Expandable arithmetic on big integers (JFB)]%
```

3.3 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
\newcount\xint_c_x^viii                  \xint_c_x^viii   100000000
\newcount\xint_c_x^ix                      \xint_c_x^ix  1000000000
\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
\newcount\xint_c_xii_e_viii_mone\xint_c_xii_e_viii_mone  1199999999
```

3.4 \xintNum

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

```
\def\xintiNum {\romannumeral0\xintinum }%
\def\xintinum #1%
{%
    \expandafter\XINT_num_loop
    \romannumeral`&&@#1\xint_relax\xint_relax\xint_relax\xint_relax
                       \xint_relax\xint_relax\xint_relax\xint_relax\Z
```

```
}%
\let\xintNum\xintiNum \let\xintnum\xintinum
\def\XINT_num #1%
{%
    \XINT_num_loop #1\xint_relax\xint_relax\xint_relax\xint_relax
                       \xint_relax\xint_relax\xint_relax\xint_relax\Z
}%
\def\XINT_num_loop #1#2#3#4#5#6#7#8%
{%
    \xint_gob_til_xint_relax #8\XINT_num_end\xint_relax
    \XINT_num_NumEight #1#2#3#4#5#6#7#8%
}%
\edef\XINT_num_end\xint_relax\XINT_num_NumEight #1\xint_relax #2\Z
{%
    \noexpand\expandafter\space\noexpand\the\numexpr #1+\xint_c_\relax
}%
\def\XINT_num_NumEight #1#2#3#4#5#6#7#8%
{%
    \ifnum \numexpr #1#2#3#4#5#6#7#8+\xint_c_= \xint_c_
      \xint_afterfi {\expandafter\XINT_num_keepsign_a
                     \the\numexpr #1#2#3#4#5#6#7#81\relax}%
    \else
      \xint_afterfi {\expandafter\XINT_num_finish
                     \the\numexpr #1#2#3#4#5#6#7#8\relax}%
    \fi
}%
\def\XINT_num_keepsign_a #1%
{%
    \xint_gob_til_one#1\XINT_num_gobacktoloop 1\XINT_num_keepsign_b
}%
\def\XINT_num_gobacktoloop 1\XINT_num_keepsign_b {\XINT_num_loop }%
\def\XINT_num_keepsign_b #1{\XINT_num_loop -}%
\def\XINT_num_finish #1\xint_relax #2\Z { #1}%
```

3.5 Zeroes

Everything had to be changed for 1.2 as it does computations by blocks of eight digits rather than four.

Currently many macros are launched by a \romannumeral0. Perhaps I should have used \romannumeral and end expansion by \z@ (\xint_c_).

\XINT_cuz_small removes leading zeroes from the first eight digits. Supposed to have been launched by a \romannumeral0. At least one digit is produced.

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

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

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 only now. Thus, one could as well use \ifnum test here. Besides the tests were not exactly for a situation like here where \XINT_cuz_z has two 00000000 blocks to grab.

```
\def\XINT_cuz #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_R #9\XINT_cuz_e \R
    \xint_gob_til_eightzeroes #1#2#3#4#5#6#7#8\XINT_cuz_z 00000000%
    \XINT_cuz_done #1#2#3#4#5#6#7#8#9%
}%
\def\XINT_cuz_z 00000000\XINT_cuz_done 00000000{\XINT_cuz }%
\edef\XINT_cuz_done #1#2#3#4#5#6#7#8#9\R
   {\noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7#8\relax #9}%
\edef\XINT_cuz_e\R #1\XINT_cuz_done #2\R
   {\noexpand\expandafter\space\noexpand\the\numexpr #2\relax }%
```

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

```
\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}%
```

3.6 Blocks of eight digits

Lingua of release 1.2.
  \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
}%
\edef\XINT_zeroes_forviii_end\R\XINT_zeroes_forviii #1#2#3#4#5#6#7#8#9\W
{%
    \noexpand\expandafter\space\noexpand\xint_gob_til_one #2#3#4#5#6#7#8%
}%
```

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>.1<8digits>!... where the original digits are organized by eight, and the order inside successive pairs of blocks separated by ! has been reversed. The output ends with a final 1U. or 1V. The former happens when we had an even number of eight blocks, the latter an odd number: 1<8d>!1<8d>.1U. or 1<8d>!1<8d>.1<8d>.1V.

```
\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.\the\numexpr 1\XINT_rsepbyviii
}%
\def\XINT_rsepbyviii_end_B #1\relax #2#3{#2.}%
\def\XINT_rsepbyviii_end_A #11#2\expandafter #3\relax #4#5{#2.1#5.}%
```

This is used typically as
  \romannumeral0\expandafter\XINT_sepandrev <8Ndigits>%
  \XINT_rsepbyviii_end_A 2345678%
  \XINT_rsepbyviii_end_B 2345678\relax UV\R.\R.\R.\R.\R.\R.\R.\W
and will produce 1<8digits>!1<8digits>!1<8digits>!... where the blocks have been globally reversed. The UV here are only place holders 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.#3.#4.#5.#6.#7.#8.#9.%
{%
    \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!{ }%
```

This is used typically as
  \romannumeral0\expandafter\XINT_sepandrev_andcount
  \the\numexpr1\XINT_rsepbyviii <8Ndigits>%
  \XINT_rsepbyviii_end_A 2345678%
  \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_iii
  \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
  \R.\xint_c_ii\R.\xint_c_i\R.\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.#4.#5.#6.#7.#8.#9.%
{%
    \xint_gob_til_R #9\XINT_sepandrev_andcount_end\R
    \expandafter\XINT_sepandrev_andcount_b \the\numexpr #1+\xint_c_xiv.%
    {#9!#8!#7!#6!#5!#4!#3!#2}%
}%
\def\XINT_sepandrev_andcount_end\R
    \expandafter\XINT_sepandrev_andcount_b\the\numexpr #1+\xint_c_xiv.#2#3#4\W
{\expandafter\XINT_sepandrev_andcount_done\the\numexpr \xint_c_ii*#3+#1.#2}%
\edef\XINT_sepandrev_andcount_done #1.#21#3!%
    {\noexpand\expandafter\space\noexpand\the\numexpr #1-#3.}%
```

```
Used as \romannumeral0\XINT_unrevbyviii 1<8d>!....1<8d>! terminated by
1\Z!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). Used by subtraction during \XINT_sub_out, in par-
ticular.

\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}%
}%
\edef\XINT_unrevbyviii_a\R\XINT_unrevbyviii #1#2\W
    {\noexpand\expandafter\space
     \noexpand\romannumeral`&&@\noexpand\xint_gob_til_Z #1}%

Can work with shorter ending pattern: 1\Z!1\R!1\R!1\R!1\R!1\R!1\R!\W but the longer one of un-
revbyviii is ok here too. Used currently (1.2) only by addition, now (1.2c) with long ending pat-
tern. 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_Z #8#7#6#5#4#3#2#1%
}%

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.<count of blocks>. Used by \XINT_div_prepare_g for
\XINT_div_prepare_h.

\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_.}%
\def\XINT_sepbyviii_andcount_b #1.#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.%
}%
\def\XINT_sepbyviii_andcount_end #1\XINT_sepbyviii_andcount_b\the\numexpr
    #2+\xint_c_viii.#3#4\W {\expandafter.\the\numexpr #2+#3.}%

This is used as
\romannumeral0\XINT_rev_nounsep {}<blocks 1<8d>!>\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. Attention does not make disappear a 1!.

\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}%

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

\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 {\relax\Z!}%

This is used as
\romannumeral0\XINT_unsep_cuzsmall {}<blocks of 1<8d>!>1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
  En fait le {} est optionnel, s'il est absent le premier #1 sera vide, tout simplement. It removes
the 1's and !'s, and removes the leading zeroes *of the first block*. This could have been done with
\numexpr and a \cleanup but would have restricted due to maximal expansion depth. Probably there
where already O(N^2) macros, thus I decided that this one would be too.

\def\XINT_unsep_cuzsmall #11#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
{%
    \xint_gob_til_R #9\XINT_unsep_cuzsmall_end\R
    \XINT_unsep_cuzsmall {#1#2#3#4#5#6#7#8#9}%
}%
\def\XINT_unsep_cuzsmall_end\R
    \XINT_unsep_cuzsmall #1{\XINT_unsep_cuzsmall_done #1}%
\def\XINT_unsep_cuzsmall_done  #1\R #2\W{\XINT_cuz_small #1}%
\def\XINT_unsep_delim {1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W}%

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 intial block of 8
zeroes is possible and thus if present must be removed. Then the next eight digits must be cleaned
of leading zeroes.

\def\XINT_div_unsepQ #11#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
{%
    \xint_gob_til_R #9\XINT_div_unsepQ_end\R
    \XINT_div_unsepQ {#1#2#3#4#5#6#7#8#9}%
}%
\def\XINT_div_unsepQ_end\R\XINT_div_unsepQ #1{\XINT_div_unsepQ_x #1}%
\def\XINT_div_unsepQ_x #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_R #9\XINT_div_unsepQ_e \R
    \xint_gob_til_eightzeroes #1#2#3#4#5#6#7#8\XINT_div_unsepQ_y 00000000%
    \expandafter\XINT_div_unsepQ_done \the\numexpr #1#2#3#4#5#6#7#8.#9%
```

```
}%
\def\XINT_div_unsepQ_e\R\xint_gob_til_eightzeroes #1\XINT_div_unsepQ_y #2\W
    {\the\numexpr #1\relax \Z}%
\def\XINT_div_unsepQ_y #1.#2\R #3\W{\XINT_cuz_small #2\Z}%
\def\XINT_div_unsepQ_done #1.#2\R #3\W { #1#2\Z}%
```

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. Terminator was 1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W

```
\def\XINT_div_unsepR #11#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
{%
    \xint_gob_til_R #9\XINT_div_unsepR_end\R
    \XINT_div_unsepR {#1#2#3#4#5#6#7#8#9}%
}%
\def\XINT_div_unsepR_end\R\XINT_div_unsepR #1{\XINT_div_unsepR_done #1}%
\def\XINT_div_unsepR_done  #1\R #2\W {\XINT_cuz #1\R}%
```

3.7 \xintReverseDigits

1.2.

```
\def\XINT_microrevsep #1#2#3#4#5#6#7#8%
{%
    1#8#7#6#5#4#3#2#1\expandafter!\the\numexpr\XINT_microrevsep
}%
\def\XINT_microrevsep_end #1\W #2\expandafter #3\Z{#2!}%
\def\xintReverseDigits {\romannumeral0\xintreversedigits }%
\def\xintreversedigits #1{\expandafter\XINT_reversedigits\romannumeral`&&@#1\Z}%
\def\XINT_reversedigits #1%
{%
    \xint_UDsignfork
      #1{\expandafter-\romannumeral0\XINT_reversedigits_a}%
       -{\XINT_reversedigits_a #1}%
    \krof
}%
\def\XINT_reversedigits_a #1\Z
{%
    \expandafter\XINT_revdigits_a\the\numexpr\expandafter\XINT_microrevsep
    \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\Z
    1\Z!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
}%
\def\XINT_revdigits_a {\XINT_revdigits_b {}}%
\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}%
}%
\edef\XINT_revdigits_end\R\XINT_revdigits_b #1#2\W
   {\noexpand\expandafter\space\noexpand\xint_gob_til_Z #1}%
```

3.8 \xintSgn, \xintiiSgn, \XINT_Sgn, \XINT_cntSgn

xintfrac.sty will overwrite \xintsgn with use of \xintraw rather than \xintnum, naturally.

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

3.9 \xintiOpp, \xintiiOpp

```
\def\xintiiOpp {\romannumeral0\xintiiopp }%
\def\xintiiopp #1%
{%
    \expandafter\XINT_opp \romannumeral`&&@#1%
}%
\def\xintiOpp {\romannumeral0\xintiopp }%
\def\xintiopp #1%
{%
    \expandafter\XINT_opp \romannumeral0\xintnum{#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
}%
```

3.10 \xintiAbs, \xintiiAbs

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

3.11 \xintFDg, \xintiiFDg

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

3.12 \xintLDg, \xintiiLDg

Rewritten for 1.2i (2016/12/10). Surprisingly perhaps, faster than \xintLastItem despite the \numexpr operations.

```
\def\xintLDg   {\romannumeral0\xintldg }%
\def\xintldg #1{\expandafter\XINT_ldg_fork\romannumeral0\xintnum{#1}%
    \XINT_ldg_c{}{}{}{}{}{}{}{}\xint_bye\relax}%
\def\xintiiLDg {\romannumeral0\xintiildg }%
\def\xintiildg #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
}%
\edef\XINT_ldg #1#2#3#4#5#6#7#8#9%
   {\noexpand\expandafter\space
    \noexpand\the\numexpr#9#8#7#6#5#4#3#2#1*\xint_c_+\noexpand\XINT_ldg_a#9}%
\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}%
```

3.13 \xintDouble

1.08. Rewritten for 1.2. Again rewritten for 1.2i (one year later...)!

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

3.14 \xintHalf

1.08. Rewritten for 1.2. Again rewritten for 1.2i.

```
\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 #1}%
    \krof
}%
\def\XINT_half_neg{\xintiiopp\XINT_half}%
\def\XINT_half #1#2#3#4#5#6#7#8%
   {\expandafter\space\the\numexpr(#1#2#3#4#5#6#7#8\XINT_half_a}%
```

```
\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}%
```

3.15 \xintInc

1.2i much delayed complete rewrite in 1.2 style. I should have done that at the time of 1.2 release (I modified \xintInc at that time but wasn't very lucid after all the work done on 1.2).
  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.

```
\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 #1}%
    \krof
}%
\def\XINT_inc_neg #1\xint_bye#2\relax
   {\xintiiopp\XINT_dec #1\XINT_dec_bye234567890\xint_bye}%
\def\XINT_inc #1#2#3#4#5#6#7#8#9%
   {\expandafter\space\the\numexpr#1#2#3#4#5#6#7#8#9\XINT_inc_a}%
\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}%
```

3.16 \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.

```
\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 #1}%
    \krof
}%
\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#2#3#4#5#6#7#8#9%
   {\expandafter\space\the\numexpr#1#2#3#4#5#6#7#8#9\XINT_dec_a}%
\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}%
```

3.17 \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.

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

3.18 \xintDSR

Decimal shift right, truncates towards zero. Rewritten for 1.2i. Limited to 22483 digits on 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 #1}%
    \krof
}%
\def\XINT_dsr_neg{\xintiiopp\XINT_dsr}%
\def\XINT_dsr #1#2#3#4#5#6#7#8#9%
   {\expandafter\space\the\numexpr(#1#2#3#4#5#6#7#8#9\XINT_dsr_a}%
\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}%
```

3.19 \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!

```
\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 #1}%
    \krof
}%
\def\XINT_dsrr_neg{\xintiiopp\XINT_dsrr}%
```

```
\def\XINT_dsrr #1#2#3#4#5#6#7#8#9%
   {\expandafter\space\the\numexpr#1#2#3#4#5#6#7#8#9\XINT_dsrr_a}%
\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
```

3.20 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.

3.21 \xintiAdd, \xintiiAdd

```
\def\xintiAdd    {\romannumeral0\xintiadd }%
\def\xintiadd  #1{\expandafter\XINT_iadd\romannumeral0\xintnum{#1}\Z }%
\def\xintiiAdd   {\romannumeral0\xintiiadd }%
\def\xintiiadd #1{\expandafter\XINT_iiadd\romannumeral`&&@#1\Z  }%
\def\XINT_iiadd #1#2\Z #3%
{%
    \expandafter\XINT_add_nfork\expandafter #1\romannumeral`&&@#3\Z #2\Z
}%
\def\XINT_iadd #1#2\Z #3%
{%
    \expandafter\XINT_add_nfork\expandafter #1\romannumeral0\xintnum{#3}\Z #2\Z
}%
\def\XINT_add_fork #1#2\Z #3\Z {\XINT_add_nfork #1#3\Z #2\Z}%
\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\Z #4\Z { #2#3}%
\def\XINT_add_secondiszero #1\krof #20#3\Z #4\Z { #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\Z
{%
  \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_iii
        \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
        \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
   \X #1%
}%
\let\XINT_add_plusplus \XINT_add_pp_a
\def\XINT_add_pp_b #1.#2\X #3\Z
{%
    \expandafter\XINT_add_checklengths
    \the\numexpr #1\expandafter.%
    \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_iii
        \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
        \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
     1\Z!1\Z!1\Z!1\Z!\W #21\Z!1\Z!1\Z!1\Z!\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.#2.%
{%
    \ifnum #2>#1
       \expandafter\XINT_add_exchange
    \else
       \expandafter\XINT_add_A
    \fi
    #1.#2.%
}%
\def\XINT_add_exchange #1.#2.#3\W #4\W
{%
    \XINT_add_A #2.#1.#4\W #3\W
}%
\def\XINT_add_A #1.#2.%
{%
    \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\Z!, replacing the \Z! used in 1.2.

Call: \the\numexpr\XINT_add_a 2#11\Z!1\Z!1\Z!1\Z!\W #21\Z!1\Z!1\Z!1\Z!\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\Z!. In recursive algotithm this 1\Z! 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_Z #2\XINT_add_bi \Z
    \expandafter\XINT_add_c\the\numexpr#1+1#2#3+#4-\xint_c_ii.%
}%
\def\XINT_add_bi\Z\expandafter\XINT_add_c
    \the\numexpr#1+#2+#3-\xint_c_ii.#4!#5!#6!#7!#8!#9!\W
{%
    \XINT_add_k #1#3!#5!#7!#9!%
}%
\def\XINT_add_c #1#2.%
{%
    1#2\expandafter!\the\numexpr\XINT_add_d #1%
}%
\def\XINT_add_d #11#2#3!#4!%
{%
    \xint_gob_til_Z #2\XINT_add_di \Z
    \expandafter\XINT_add_e\the\numexpr#1+1#2#3+#4-\xint_c_ii.%
}%
\def\XINT_add_di\Z\expandafter\XINT_add_e
    \the\numexpr#1+#2+#3-\xint_c_ii.#4!#5!#6!#7!#8\W
{%
    \XINT_add_k #1#3!#5!#7!%
}%
\def\XINT_add_e #1#2.%
{%
    1#2\expandafter!\the\numexpr\XINT_add_f #1%
}%
\def\XINT_add_f #11#2#3!#4!%
{%
    \xint_gob_til_Z #2\XINT_add_fi \Z
    \expandafter\XINT_add_g\the\numexpr#1+1#2#3+#4-\xint_c_ii.%
}%
\def\XINT_add_fi\Z\expandafter\XINT_add_g
    \the\numexpr#1+#2+#3-\xint_c_ii.#4!#5!#6\W
{%
    \XINT_add_k #1#3!#5!%
```

```
}%
\def\XINT_add_g #1#2.%
{%
    1#2\expandafter!\the\numexpr\XINT_add_h #1%
}%
\def\XINT_add_h #11#2#3!#4!%
{%
    \xint_gob_til_Z #2\XINT_add_hi \Z
    \expandafter\XINT_add_i\the\numexpr#1+1#2#3+#4-\xint_c_ii.%
}%
\def\XINT_add_hi\Z
    \expandafter\XINT_add_i\the\numexpr#1+#2+#3-\xint_c_ii.#4\W
{%
    \XINT_add_k #1#3!%
}%
\def\XINT_add_i #1#2.%
{%
    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\Z #2\W {\XINT_add_kf #11\Z!}%
\def\XINT_add_kf 1{1\relax }%
\def\XINT_add_l 1#1#2{\xint_gob_til_Z #1\XINT_add_lf \Z \XINT_add_m 1#1#2}%
\def\XINT_add_lf #1\W {1\relax 00000001!1\Z!}%
\def\XINT_add_m #1!{\expandafter\XINT_add_n\the\numexpr\xint_c_i+#1.}%
\def\XINT_add_n #1#2.{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}%
```

3.22 \xintiSub, \xintiiSub

Entirely rewritten for 1.2.

```
\def\xintiiSub   {\romannumeral0\xintiisub }%
\def\xintiisub #1{\expandafter\XINT_iisub\romannumeral`&&@#1\Z  }%
\def\XINT_iisub #1#2\Z #3%
{%
    \expandafter\XINT_sub_nfork\expandafter #1\romannumeral`&&@#3\Z #2\Z
}%
\def\xintiSub    {\romannumeral0\xintisub }%
\def\xintisub #1{\expandafter\XINT_isub\romannumeral0\xintnum{#1}\Z }%
\def\XINT_isub #1#2\Z #3%
{%
   \expandafter\XINT_sub_nfork\expandafter #1\romannumeral0\xintnum{#3}\Z #2\Z
}%
\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\Z #4\Z {\XINT_opp #2#3}%
\def\XINT_sub_secondiszero #1\krof #20#3\Z #4\Z { #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\Z
{%
  \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_iii
      \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
      \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
  \X #1%
}%
\def\XINT_sub_mm_b #1.#2\X #3\Z
{%
    \expandafter\XINT_sub_checklengths
    \the\numexpr #1\expandafter.%
    \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_iii
      \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
      \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
    \Z!\Z!\Z!\Z!\W #2\Z!\Z!\Z!\Z!\W
}%
\def\XINT_sub_checklengths #1.#2.%
{%
    \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 {} after \XINT_unrevbyviii could be removed, but attention then at \XINT_sub_startrescue

which must be modified (no need for #1).

```
\def\XINT_sub_out #1\Z #2#3\W
{%
    \if-#2\expandafter\XINT_sub_startrescue\fi
    \expandafter\XINT_cuz_small
    \romannumeral0\XINT_unrevbyviii {}#11\Z!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
}%
```

1 as first token of #1 stands for "no carry", 0 will mean a carry.
  Call: \the\numexpr \XINT_sub_a 1#1\Z!\Z!\Z!\Z!\W #2\Z!\Z!\Z!\Z!\W where #1 and #2 are blocks of 1<8d>!, and #1 *must* be at most as long as #2.
  The routine wants to compute #2-#1.
  Notice that currently the terminators on input differ from those for addition. Also, currently (1.2f) the routine can not be called without final reversal and clean up of the result.
  \numexpr governed expansion stops with various possibilities:
  1. #1 was shorter (in number of 8 digits blocks) than #2.
  *1a There may be no carry in which case we end up with
1<8d>!...1<8d>!\Z!\Z!\Z!\Z!\W
  If there is a carry things are more complicated.
  *1b If the first hit block of #2 is > 1 no problem we are like in the no-carry case.
  *1c If it is exactly 1 then we will have leading zeros; but there may be also before that arbitrarily many produced zeros, all these leading zeros will have to be cleaned up. This is done via ending the expansion with the shape
  1<8d>!...1<8d>!1\XINT_cuz_byviii!\Z 0\W\R
  *1d If the block value is zero, subtraction produces 99999999 and goes on. This is the only situation where the carry can propagate. This case can never produce extra blocks of leading zeros but may well end up with an ending zero block. In this subcase, the \numexpr is then made to stop with a 1!. This 1! will disappear during final reverse.
  2a. #1 was of same length as #2, but <= #2. Then we end up expansion with
1<8d>!...1<8d>!1\XINT_cuz_byviii!\Z 0\W\R
and the blocks will have to cleaned up of leading zeroes after reversal.
  2b. #1 was of same length as #2, but > #2. Then we end up with blocks 1<8d>!...1<8d>! followed by -1\Z-\W
  Thus \XINT_sub_out examines the token after the first \Z, which may be ! or 0 or -. If ! or 0, \XINT_unrevbyviii will be executed (gobbling a possible final 1!), and followed in case 2a or 1c by \XINT_cuz_byviii (note the extra \R which terminates it), and then in both 1* and 2a by \XINT_cuz_small.
  If we were in 2b we proceed to \XINT_sub_startrescue which I will comment another day (the extra -1 at the end from -1\Z-\W will become a -1! and the - will serve in \XINT_sub_rescue_d as loop terminator).
  Currently (1.2f) we can not easily use these low level routines in a binary split approach due to the fact that first input must be at most as long as second but also because the final reversal is not in a common second stage, due to the separate treatment for case 2b.
  For the record: subtraction was correct (I think) in xint releases up to 1.2, but 1.2 had a broken treatment of the 1d case. For example \xintiiSub {10000000112345678}{12345679} produced 99999999. This got fixed in 1.2c, but that fix broke the 1c case :((, for example \xintiiSub {10000000000000000}{9999999999999997} was now returning 000000003. Alas.
  This was only realized later on 2016/02/29 (in fact it impacted \xintiiSqrt). Hopefully 1.2f got it right at last.

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

```
}%
\def\XINT_sub_hi\Z
    \expandafter\XINT_sub_i\the\numexpr#1+1#2-#3.#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 is just 1<00000001>!, we will have those eight zeros to clean up. (but we have to be careful that if we encounter 1<00000001>! and this is not the final block, we should not make something silly either).

There is the possibility that A has exactly one more <eight-digits> block than B and that this block is exactly 1. In that case there can be arbitrarily many leading zeros to clean up from A-B. This was done correctly up to 1.2b but got broken in 1.2c. Belatedly fixed in 1.2f.

If we have simultaneously reached the end of A, then if B was smaller there might be arbitrarily many zeroes to clean up, if it was larger, we will have to rescue the whole thing.

```
\def\XINT_sub_k #1#2%
{%
    \xint_gob_til_Z #2\XINT_sub_p\Z \XINT_sub_l #1#2%
}%
\def\XINT_sub_l #1{\xint_UDzerofork #1\XINT_sub_l_carry 0\XINT_sub_l_nocarry\krof}%
\def\XINT_sub_l_nocarry 1{1\relax }%
\def\XINT_sub_l_carry 1#1!{\ifcase #1
    \expandafter \XINT_sub_l_zeroa\or\expandafter\XINT_sub_l_one\else
    \expandafter \XINT_sub_l_done\fi 1#1!}%
\def\XINT_sub_l_done {-\xint_c_i+}%
\def\XINT_sub_l_one 1#1!#2%
{%
    \xint_gob_til_Z #2\XINT_sub_l_oneone\Z 1\relax 00000000!#2%
}%
\def\XINT_sub_l_oneone #1\W {1\relax \XINT_cuz_byviii!\Z 0\W\R }%
\def\XINT_sub_l_zeroa 1#1!{199999999\expandafter!\the\numexpr \XINT_sub_l_zerob }%
\def\XINT_sub_l_zerob 1#1!{\ifcase #1
    \expandafter \XINT_sub_l_zeroa\or\expandafter\XINT_sub_l_zone\else
    \expandafter \XINT_sub_l_done\fi 1#1!}%
\def\XINT_sub_l_zone 1#1!#2%
{%
    \xint_gob_til_Z #2\XINT_sub_l_zoneone\Z 1\relax 00000000!#2%
}%
\def\XINT_sub_l_zoneone\Z 1\relax 00000000{1}%
```

Here we are in the situation were the two inputs had the same length in base 10^8. If #1=0 we bitterly discover that first input was greater than second input despite having same length (in base 10^8). The \numexpr will expand beyond the -1 or 1. If #1=1 we had no carry but perhaps the result will have plenty of zeroes to clean-up. The result might even be simply zero.

```
\def\XINT_sub_p\Z\XINT_sub_l #1#2\W
{%
    \xint_UDzerofork
       #1{-1\relax\Z -\W}%
```

```
        0{1\relax \XINT_cuz_byviii!\Z 0\W\R }%
    \krof
}%
```

We arrive here if #2-#1 concluded #1>#2 (both of the same length in base 10^8). To be commented. Here also before the \XINT_sub_rescue_finish there will be an ending 1! which will disappear only due to \XINT_unrevbyviii. The final \R is for \XINT_cuz.

```
\def\XINT_sub_startrescue\expandafter\XINT_cuz_small
    \romannumeral0\XINT_unrevbyviii #1#2\Z!#3\W
{%
    \expandafter\XINT_sub_rescue_finish
    \the\numexpr\XINT_sub_rescue_a #2!%
    1\Z!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W \R
}%
\def\XINT_sub_rescue_finish
   {\expandafter-\romannumeral0\expandafter\XINT_cuz\romannumeral0\XINT_unrevbyviii {}}%
\def\XINT_sub_rescue_a #1!%
{%
    \expandafter\XINT_sub_rescue_c\the\numexpr \xint_c_xii_e_viii-#1.%
}%
\def\XINT_sub_rescue_c 1#1#2.%
{%
    1#2\expandafter!\the\numexpr\XINT_sub_rescue_d #1%
}%
\def\XINT_sub_rescue_d #1#2#3!%
{%
    \xint_gob_til_minus #2\XINT_sub_rescue_z -%
    \expandafter\XINT_sub_rescue_c\the\numexpr \xint_c_xii_e_viii_mone-#2#3+#1.%
}%
\def\XINT_sub_rescue_z #1.{1!}%
```

3.23 \xintiMul, \xintiiMul

Completely rewritten for 1.2.

```
\def\xintiMul {\romannumeral0\xintimul }%
\def\xintimul #1%
{%
    \expandafter\XINT_imul\romannumeral0\xintnum{#1}\Z
}%
\def\XINT_imul #1#2\Z #3%
{%
    \expandafter\XINT_mul_nfork\expandafter #1\romannumeral0\xintnum{#3}\Z #2\Z
}%
\def\xintiiMul {\romannumeral0\xintiimul }%
\def\xintiimul #1%
{%
    \expandafter\XINT_iimul\romannumeral`&&@#1\Z
}%
\def\XINT_iimul #1#2\Z #3%
{%
    \expandafter\XINT_mul_nfork\expandafter #1\romannumeral`&&@#3\Z #2\Z
}%
```

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

```
\def\XINT_mul_fork #1#2\Z #3\Z{\XINT_mul_nfork #1#3\Z #2\Z}%
\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\Z #4\Z { 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\Z
{%
  \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_iii
        \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
        \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
  \W #1%
}%
\def\XINT_mul_pre_b #1.#2\W #3\Z
{%
    \expandafter\XINT_mul_checklengths
    \the\numexpr #1\expandafter.%
    \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_iii
        \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
        \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
     1\Z!\W #21\Z!%
    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.#2.%
{%
    \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\Z!\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\Z!%
{%
    \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.#1!}%
\def\XINT_mul_exchange #1\XINT_mul_start #2\W #31\Z!%
   {\fi\fi\XINT_mul_start #31\Z!\W #2}%
```

1.2c: earlier version of addition had sometimes a final 1!, but not in all cases. Version 1.2c of \XINT_add_a always has an ending 1\Z!, which is thus expected by \XINT_mul_loop.

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

The 1.2 \XINT_mul_loop could *not* be called directly with a small multiplicand, due to problems caused in case the addition done in \XINT_mul_a produced only 1 block the second one being either empty or a 1! which had to be handled by \XINT_mul_loop and \XINT_mul_e. But \XINT_mul_loop was only called via \xintiiMul for arguments with at least 2 digits in base 10^8, thus no problem. But this made it annoying for \xintiiPow and \xintiiSqr which had to check if the intended multiplier had only 1 digit in base 10^8. It also made it annoying to create recursive algorithms which did multiplications maintaining the result reverses, for iterative use of output as input.

Finally on 2015/11/14 during 1.2c preparation I modified the addition to *always* have the ending 1\Z!.\numexpr expands even through spaces to find operators and even something like 1<space>\Z will try to expand the \Z. Thus we have to not forget that #2 in \XINT_mul_e might be \Z! (a #2=1\Z! in \XINT_mul_a hence \XINT_add_a is no problem). Again this can only happen if we use \XINT_mul_loop directly with a small first argument (in place of smallmul). Anyway, now the routine \XINT_mul_loop can handle a small #2, with no black magic with delimiters and checking if #1 empty, although it never happens when called via \xintiiMul.

```
  The delimiting patterns for addition was changed to use 1\Z! to fit what is used on output (by
necessity).
  Call: \the\numexpr \XINT_mul_loop 100000000!1\Z!\W #11\Z!\W #21\Z!
where #1 and #2 are (globally reversed) blocks 1<8d>!. Its is generally more efficient to have #1
as the shorter one, but a better recipe is implemented in \XINT_mul_checklengths which as exe-
cuted earlier. 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\Z!. The most significant digit block is the last
one. It can not be 100000000! except if naturally 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.

924 \def\XINT_mul_loop #1\W #2\W 1#3!%
925 {%
926     \xint_gob_til_Z #3\XINT_mul_e \Z
927     \expandafter\XINT_mul_a\the\numexpr \XINT_smallmul 1#3!#2\W
928     #1\W #2\W
929 }%

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

930 \def\XINT_mul_a #1\W #2\W
931 {%
932     \expandafter\XINT_mul_b\the\numexpr
933     \XINT_add_a \xint_c_ii #21\Z!1\Z!1\Z!\W #11\Z!1\Z!1\Z!\W\W
934 }%
935 \def\XINT_mul_b 1#1!{1#1\expandafter!\the\numexpr\XINT_mul_loop }%
936 \def\XINT_mul_e\Z #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 rou-
tines. But division, float factorial, etc.. have their own variants as they need output with spe-
cific 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\Z!
  Multiplication by zero will produce blocks of zeros.

937 \def\XINT_minimulwc_a 1#1.#2.#3!#4#5#6#7#8.%
938 {%
939     \expandafter\XINT_minimulwc_b
940     \the\numexpr \xint_c_x^ix+#1+#3*#8.#3*#4#5#6#7+#2*#8.#2*#4#5#6#7.%
941 }%
942 \def\XINT_minimulwc_b 1#1#2#3#4#5#6.#7.%
943 {%
944     \expandafter\XINT_minimulwc_c
945     \the\numexpr \xint_c_x^ix+#1#2#3#4#5+#7.#6.%
946 }%
947 \def\XINT_minimulwc_c 1#1#2#3#4#5#6.#7.#8.%
948 {%
949     1#6#7\expandafter!%
950     \the\numexpr\expandafter\XINT_smallmul_a
951     \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8.%
```

```
}%
\def\XINT_smallmul 1#1#2#3#4#5!{\XINT_smallmul_a 100000000.#1#2#3#4.#5!}%
\def\XINT_smallmul_a #1.#2.#3!1#4!%
{%
    \xint_gob_til_Z #4\XINT_smallmul_e\Z
    \XINT_minimulwc_a #1.#2.#3!#4.#2.#3!%
}%
\def\XINT_smallmul_e\Z\XINT_minimulwc_a 1#1.#2\Z #3!%
    {\xint_gob_til_eightzeroes #1\XINT_smallmul_f 000000001\relax #1!1\Z!}%
\def\XINT_smallmul_f 000000001\relax 00000000!1{1\relax}%
```

This is multiplication by 1 up to 21. Last time I checked it is never called with a wasteful multiplicand of 1. Here also always the output terminated by a 1\Z! and the last block of digits is not zero. I imagine multiplication by zero produces blocks of zeroes. Will check another day.

```
\def\XINT_verysmallmul #1.#2!1#3!%
{%
    \xint_gob_til_Z #3\XINT_verysmallmul_e\Z
    \expandafter\XINT_verysmallmul_a
    \the\numexpr #2*#3+#1.#2!%
}%
\def\XINT_verysmallmul_e\Z\expandafter\XINT_verysmallmul_a\the\numexpr
    #1+#2#3.#4!%
{\xint_gob_til_zero #2\XINT_verysmallmul_f 0\xint_c_x^viii+#2#3!1\Z!}%
\def\XINT_verysmallmul_f #1!1{1\relax}%
\def\XINT_verysmallmul_a #1#2.%
{%
    \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.%
}%
\def\XINT_verysmallmul_bj{\expandafter\XINT_verysmallmul_cj }%
\def\XINT_verysmallmul_cj 1#1#2.%
    {1#2\expandafter!\the\numexpr\XINT_verysmallmul #1.}%
\def\XINT_verysmallmul_bi\the\numexpr\xint_c_x^ix+#1#2#3.%
    {1#3\expandafter!\the\numexpr\XINT_verysmallmul #1#2.}%
```

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

```
    1#6#7\expandafter!\the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8!%
}%
```

3.24 \xintiSqr, \xintiiSqr

Rewritten for 1.2.

```
\def\xintiiSqr {\romannumeral0\xintiisqr }%
\def\xintiisqr #1%
{%
    \expandafter\XINT_sqr\romannumeral0\xintiiabs{#1}\Z
}%
\def\xintiSqr {\romannumeral0\xintisqr }%
\def\xintisqr #1%
{%
    \expandafter\XINT_sqr\romannumeral0\xintiabs{#1}\Z
}%
\def\XINT_sqr #1\Z
{%
    \expandafter\XINT_sqr_a
      \romannumeral0\expandafter\XINT_sepandrev_andcount
      \romannumeral0\XINT_zeroes_forviii #1\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_iii
        \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
        \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
      \Z
}%
```

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.%
{%
    \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!\Z
{%
    \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.#5!#1#2#3#4#5!%
}%
\edef\XINT_sqr_verysmall
    \expandafter\XINT_sqr_small_out\the\numexpr\XINT_minimul_a #1!#2!%
    {\noexpand\expandafter\space\noexpand\the\numexpr #2*#2\relax}%
\def\XINT_sqr_small_out 1#1!1#2!%
{%
    \XINT_cuz #2#1\R
}%
```

An ending 1\Z! 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\Z
{%
    \expandafter\XINT_mul_out
    \the\numexpr\XINT_mul_loop 100000000!1\Z!\W #11\Z!\W #11\Z!%
    1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
}%
```

3.25 \xintiPow, \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.

```
\def\xintiiPow {\romannumeral0\xintiipow }%
\def\xintiipow #1#2%
{%
    \expandafter\xint_pow\the\numexpr #2\expandafter.\romannumeral`&&@#1\Z%
}%
\def\xintiPow  {\romannumeral0\xintipow }%
\def\xintipow #1#2%
{%
    \expandafter\xint_pow\the\numexpr #2\expandafter.\romannumeral0\xintnum{#1}\Z%
}%
\def\xint_pow #1.#2%#3\Z
{%
    \xint_UDzerominusfork
      #2-\XINT_pow_AisZero
      0#2\XINT_pow_Aneg
      0-{\XINT_pow_Apos #2}%
    \krof {#1}%
}%
\def\XINT_pow_AisZero #1#2\Z
{%
     \ifcase\XINT_cntSgn #1\Z
         \xint_afterfi { 1}%
     \or
         \xint_afterfi { 0}%
     \else
         \xint_afterfi {\xintError:DivisionByZero\space 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_Z #3\XINT_pow_Apos_short\Z
    \XINT_pow_AatleastTwo {#1}#2#3%
}%
\def\XINT_pow_Apos_short\Z\XINT_pow_AatleastTwo #1#2\Z
{%
    \ifcase #2
         \xintError:thiscannothappen!
    \or  \expandafter\XINT_pow_AisOne
    \else\expandafter\XINT_pow_AatleastTwo
    \fi {#1}#2\Z
}%
\def\XINT_pow_AisOne #1\Z{ 1}%
\def\XINT_pow_AatleastTwo #1%
{%
    \ifcase\XINT_cntSgn #1\Z
        \expandafter\XINT_pow_BisZero
    \or
        \expandafter\XINT_pow_I_in
    \else
        \expandafter\XINT_pow_BisNegative
    \fi
    {#1}%
}%
\edef\XINT_pow_BisNegative #1\Z
    {\noexpand\xintError:FractionRoundedToZero\space 0}%
\def\XINT_pow_BisZero #1\Z{ 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\Z
{%
    \expandafter\XINT_pow_I_loop
    \the\numexpr #1\expandafter.%
    \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.\R.\R.\R.\R.\R.\R.\W  1\Z!\W
    1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
}%
\def\XINT_pow_I_loop #1.%
{%
    \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.%
}%
\def\XINT_pow_I_exit \ifodd #1\fi #2.#3\W {\XINT_mul_out #3}%
```

The 1.2c \XINT_mul_loop can be called directly even with small arguments, hence the "butcheckifs-

```
mall" 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.#2\W%
{%
    \expandafter\XINT_pow_I_loop
    \the\numexpr #1/\xint_c_ii\expandafter.%
    \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W
}%
\def\XINT_pow_mulbutcheckifsmall #1!1#2%
{%
    \xint_gob_til_Z #2\XINT_pow_mul_small\Z
    \XINT_mul_loop 100000000!1\Z!\W #1!1#2%
}%
\def\XINT_pow_mul_small\Z \XINT_mul_loop 100000000!1\Z!\W 1#1!1\Z!\W
{%
    \XINT_smallmul 1#1!%
}%
\def\XINT_pow_II_in #1.#2\W
{%
    \expandafter\XINT_pow_II_loop
    \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter.%
    \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W #2\W
}%
\def\XINT_pow_II_loop #1.%
{%
    \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.%
}%
\def\XINT_pow_II_exit\ifodd #1\fi #2.#3\W #4\W
{%
    \expandafter\XINT_mul_out
    \the\numexpr\XINT_pow_mulbutcheckifsmall #4\W #3%
}%
\def\XINT_pow_II_even #1.#2\W
{%
    \expandafter\XINT_pow_II_loop
    \the\numexpr #1/\xint_c_ii\expandafter.%
    \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W
}%
\def\XINT_pow_II_odda #1.#2\W #3\W
{%
    \expandafter\XINT_pow_II_oddb
    \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter.%
    \the\numexpr\XINT_pow_mulbutcheckifsmall #3\W #2\W #2\W
}%
\def\XINT_pow_II_oddb #1.#2\W #3\W
{%
    \expandafter\XINT_pow_II_loop
```

```
1178     \the\numexpr #1\expandafter.%
1179     \the\numexpr\XINT_pow_mulbutcheckifsmall #3\W #3\W #2\W
1180 }%
```

3.26 \xintiFac, \xintiiFac

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

The routine has been partially rewritten with release 1.2 to exploit the new inner structure of multiplication. I impose an intrinsic limit of the argument at maximal value 9999 (1.2f sets it at 10000, there was no reason for 9999 and not 10000). Anyhow with current default settings of the etex memory and the current 1.2 routine (last commit: eada1b1), the maximal possible computation is 5971! (which has 19956 digits). Also, I add \xintiiFac which does only \romannumeral-`0 and not \numexpr on its argument. This is for a silly slight optimization of the \xintiiexpr (and \bnumexpr) parsers. If the argument is >=2^31 an arithmetic overflow will occur in the \ifnum. This is not as good as in the \numexpr, but well.

2015/11/14 added note on the implementation: we can roughly estimate for big n that we do n/2 multiplications alpha*X where alpha=(k+1)(k+2)<10^8 and X=k! has size of order k log(k), with k along a step 2 arithmetic sequence up to n. Each small multiplication should have a linear cost hence O(k log(k)) (as we maintain the reversed representation) hence a total cost of O(n^2 log(n)); on computing n! for n=100, 200, ..., 2000 I obtained a good fit (only roughly 20% variation) of the computation time with the square of the length of n! -- to the extent that the big variability of \pdfelapsedtime allows to draw any conclusion -- I did not repeat the computations as many times as I should have. I currently do not quite understand why in this range it seems computation times are better fitted by O(n^2 log^2 n) than by O(n^2 log n). True, final reverse is O(N^2) with N of order n log n, but for this range of n's this is marginal (and I tested also with this final reverse skipped).

On the other hand with an approach based on binary splitting n!=AB and A=[n/2]! each of A and B will be of size n/2 log(n), but xint schoolbook multiplication in TeX is worse than quadratic due to penalty when TeX needs to fetch arguments and it didn't seem promising. I didn't even test. Binary splitting is good when a fast multiplication is available.

No wait! incredibly a very naive recursive implementation with five lines of code via a binary splitting approach with \xintiiMul is only about 1.6x--2x slower in the range N=200 to 2000 ! this seems to say that the reversing done by \xintiiMul both on input and for output is quite efficient. The best case seems to be around N=1000, hence multiplication of 500 digits numbers, after that the impact of over-quadratic computation time seems to show: for N=4000, the naive binary splitting approach is about 3.4x slower than the naive iterated small multiplications as here (naturally with sub-quadratic multiplication that would be otherwise).

2015/11/29 for 1.2f: no more a \xintFac, only \xintiFac/\xintiiFac. I could not go on like this with \xintFac/\xintiFac/\xintiiFac.

```
1181 \def\xintiiFac {\romannumeral0\xintiifac }%
1182 \def\xintiifac #1{\expandafter\XINT_fac_fork\the\numexpr#1.}%
1183 \def\xintiFac  {\romannumeral0\xintifac }%
1184 \let\xintifac\xintiifac
```

Vieux style. Bon je modifie pour 1.2f. Le cas négatif devrait faire un 1/0 et créer un Inf.

```
1185 \def\XINT_fac_fork #1#2.%
1186 {%
1187     \xint_UDzerominusfork
1188      #1-\XINT_fac_zero
1189      0#1\XINT_fac_neg
1190       0-\XINT_fac_checksize
```

```
    \krof #1#2.%
}%
\def\XINT_fac_zero #1.{ 1}%
\edef\XINT_fac_neg #1.{\noexpand\xintError:FactorialOfNegative\space 1}%
\def\XINT_fac_checksize #1.%
{%
    \ifnum #1>\xint_c_x^iv
         \xint_dothis{\expandafter\xintError:TooBigFactorial
                      \expandafter\space\expandafter 1\xint_gob_til_W }\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_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\Z!%
}%
\expandafter\def\csname XINT_fac_smallloop_-2\endcsname #1.%
{%
    \XINT_fac_smallloop_loop 3.#1.100000002!1\Z!%
}%
\expandafter\def\csname XINT_fac_smallloop_-1\endcsname #1.%
{%
    \XINT_fac_smallloop_loop 4.#1.100000006!1\Z!%
}%
\expandafter\def\csname XINT_fac_smallloop_0\endcsname #1.%
{%
    \XINT_fac_smallloop_loop 5.#1.1000000024!1\Z!%
}%
\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\Z!#3{#3#2\Z!}%
```

3.27 \xintiDivision, \xintiQuo, \xintiRem, \xintiiDivision, \xintiiQuo, \xintiiRem

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 sit-
  uation 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.
    1.2 2015/10/15 had a bad bug which got corrected in 1.2b of 2015/10/29: a divisor starting with
  99999999xyz... would cause a failure, simply because it was attempted to use the \XINT_div_mini
  routine with a divisor of 1+99999999=100000000 having 9 digits. Fortunately the origin of the bug
  was easy to find out. Too bad that my obviously very deficient test files did not detect it.

1287 \def\xintiiQuo {\romannumeral0\xintiiquo }%
1288 \def\xintiiRem {\romannumeral0\xintiirem }%
1289 \def\xintiiquo {\expandafter\xint_firstoftwo_thenstop\romannumeral0\xintiidivision }%
1290 \def\xintiirem {\expandafter\xint_secondoftwo_thenstop\romannumeral0\xintiidivision }%
1291 \def\xintiQuo {\romannumeral0\xintiquo }%
1292 \def\xintiRem {\romannumeral0\xintirem }%
1293 \def\xintiquo {\expandafter\xint_firstoftwo_thenstop\romannumeral0\xintidivision }%
1294 \def\xintirem {\expandafter\xint_secondoftwo_thenstop\romannumeral0\xintidivision }%
1295 %%\let\xintQuo\xintiQuo\let\xintquo\xintiquo % now removed
1296 %%\let\xintRem\xintiRem\let\xintrem\xintirem % now removed

  #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|.

1297 \def\xintiDivision {\romannumeral0\xintidivision }%
1298 \def\xintidivision #1{\expandafter\XINT_idivision\romannumeral0\xintnum{#1}\Z }%
1299 \def\XINT_idivision #1#2\Z #3{\expandafter\XINT_iidivision_a\expandafter #1%
1300                               \romannumeral0\xintnum{#3}\Z #2\Z }%
1301 \def\xintiiDivision   {\romannumeral0\xintiidivision }%
1302 \def\xintiidivision  #1{\expandafter\XINT_iidivision \romannumeral`&&@#1\Z }%
1303 \def\XINT_iidivision #1#2\Z #3{\expandafter\XINT_iidivision_a\expandafter #1%
1304                               \romannumeral`&&@#3\Z #2\Z }%

  On regarde les signes de A et de B.

1305 \def\XINT_iidivision_a #1#2% #1 de A, #2 de B.
1306 {%
1307     \if0#2\xint_dothis\XINT_iidivision_divbyzero\fi
1308     \if0#1\xint_dothis\XINT_iidivision_aiszero\fi
1309     \if-#2\xint_dothis{\expandafter\XINT_iidivision_bneg
1310                        \romannumeral0\XINT_iidivision_bpos #1}\fi
1311     \xint_orthat{\XINT_iidivision_bpos #1#2}%
1312 }%
1313 \def\XINT_iidivision_divbyzero #1\Z #2\Z {\xintError:DivisionByZero{0}{0}}%
1314 \def\XINT_iidivision_aiszero #1\Z #2\Z {{0}{0}}%
1315 \def\XINT_iidivision_bneg #1% q->-q, r unchanged
1316                          {\expandafter{\romannumeral0\XINT_opp #1}}%
1317 \def\XINT_iidivision_bpos #1%
1318 {%
1319     \xint_UDsignfork
1320             #1\XINT_iidivision_aneg
```

```
1321              -{\XINT_iidivision_apos #1}%
1322     \krof
1323 }%
```

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.

```
1324 \def\XINT_iidivision_apos #1#2\Z #3\Z{\XINT_div_prepare {#2}{#1#3}}%
1325 \def\XINT_iidivision_aneg #1\Z #2\Z
1326    {\expandafter
1327     \XINT_iidivision_aneg_b\romannumeral0\XINT_div_prepare {#1}{#2}{#1}}%
1328 \def\XINT_iidivision_aneg_b #1#2{\if0\XINT_Sgn #2\Z
1329                                 \expandafter\XINT_iidivision_aneg_rzero
1330                              \else
1331                                 \expandafter\XINT_iidivision_aneg_rpos
1332                              \fi {#1}{#2}}%
1333 \def\XINT_iidivision_aneg_rzero #1#2#3{{-#1}{0}}% necessarily q was >0
1334 \def\XINT_iidivision_aneg_rpos #1%
1335 {%
1336     \expandafter\XINT_iidivision_aneg_end\expandafter
1337                 {\expandafter-\romannumeral0\xintinc {#1}}% q-> -(1+q)
1338 }%
1339 \def\XINT_iidivision_aneg_end #1#2#3%
1340 {%
1341      \expandafter\xint_exchangetwo_keepbraces
1342      \expandafter{\romannumeral0\XINT_sub_mm_a {}{}#3\Z #2\Z}{#1}% r-> b-r
1343 }%
```

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.

```
1344 \def\XINT_div_prepare #1%
1345 {%
1346     \XINT_div_prepare_a #1\R\R\R\R\R\R\R\R {10}0000001\W !{#1}%
1347 }%
1348 \def\XINT_div_prepare_a #1#2#3#4#5#6#7#8#9%
1349 {%
1350     \xint_gob_til_R #9\XINT_div_prepare_small\R
1351     \XINT_div_prepare_b #9%
1352 }%
```

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).

```
1353 \def\XINT_div_prepare_small\R #1!#2%
1354 {%
1355     \ifcase #2
1356     \or\expandafter\XINT_div_BisOne
1357     \or\expandafter\XINT_div_BisTwo
1358     \else\expandafter\XINT_div_small_a
1359     \fi {#2}%
```

```
}%
\def\XINT_div_BisOne #1#2{{#2}{0}}%
\def\XINT_div_BisTwo #1#2%
{%
    \expandafter\expandafter\expandafter\XINT_div_BisTwo_a
    \ifodd\xintiiLDg{#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
    .\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 #1.1#2!#3%
{%
    \expandafter\XINT_div_smallsmallend
    \the\numexpr (#3+#1)/#2-\xint_c_i.#2.#3.%
```

```
}%
\def\XINT_div_smallsmallend #1.#2.#3.{\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\Z! 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\Z! 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
    .\the\numexpr (#1#2#3#4#5#6#7#8+\xint_c_i)/\xint_c_ii\expandafter
    .\the\numexpr #1#2#3#4#5#6#7#8\expandafter
    .\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_iii
        \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
        \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
    \X
}%
\def\XINT_div_prepare_g #1.#2.#3.#4.#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.#2.#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
    #31\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W .%
}%
\def\XINT_div_zeroQ_end #1.#2%
    {\expandafter{\expandafter0\expandafter}\XINT_div_cleanR #1#2.}%
```

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}.#3.{#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_div_finish_b #2.{#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.#1#2{{#1}{0}}%
\def\XINT_div_finish_bRpos #1.#2#3%
{%
    \expandafter\xint_exchangetwo_keepbraces\XINT_div_cleanR  #1#3.{#2}%
}%
\def\XINT_div_cleanR #100000000.{{#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.#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!.%
}%
\def\XINT_div_start_cb #1%
   {\csname XINT_div_start_c_\romannumeral\numexpr#1\endcsname}%
\def\XINT_div_start_c_i   #1.#2!%
    {\XINT_div_start_c_   #1#2!.}%
\def\XINT_div_start_c_ii  #1.#2!#3!%
    {\XINT_div_start_c_   #1#2!#3!.}%
\def\XINT_div_start_c_iii #1.#2!#3!#4!%
    {\XINT_div_start_c_   #1#2!#3!#4!.}%
\def\XINT_div_start_c_iv  #1.#2!#3!#4!#5!%
    {\XINT_div_start_c_   #1#2!#3!#4!#5!.}%
\def\XINT_div_start_c_v   #1.#2!#3!#4!#5!#6!%
    {\XINT_div_start_c_   #1#2!#3!#4!#5!#6!.}%
\def\XINT_div_start_c_vi  #1.#2!#3!#4!#5!#6!#7!%
    {\XINT_div_start_c_   #1#2!#3!#4!#5!#6!#7!.}%
```

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

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

```
\def\XINT_div_I_da #1.%
{%
    \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.%
{%
    \expandafter\XINT_div_I_b\the\numexpr #1-\xint_c_i.%
}%
\def\XINT_div_I_db #1.#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!\W}%
       {\the\numexpr\XINT_div_verysmallmul #1!#51\Z!}%
    \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.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.#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!\W}%
      {\the\numexpr\XINT_div_verysmallmul #1!#51\Z!}%
    }%
}%
```

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.#6.{{#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.#3.#4#5%
{%
    1\expandafter\expandafter\expandafter!\expandafter\XINT_unsep_delim
    \romannumeral0\XINT_div_unsepR #2#31\R!1\R!1\R!1\R!1\R!1\R!1\R!\W.%
}%
```

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.#2!#3.#4#5%
{%
    \XINT_div_II_b #1#2!.{#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.#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>.
  ATTENTION!
  2015/10/29 :j'avais introduit un bug ici dans 1.2 2015/10/15, car \XINT_div_mini veut un diviseur de huit chiffres, or si le dénominateur B débute par x=99999999, on aura x'=100000000, d'où évidemment un bug. Bon il faut intercepter x'=100000000.
  I need to recognize x'=100000000 in some not too penalizing way. Anyway, will try to optimize some other day.
```

```
\def\XINT_div_II_c #1#2#3#4%
{%
     \expandafter\XINT_div_II_d\the\numexpr\XINT_div_xmini
     #1.#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 100000000.50000000!#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.#9%
{%
    \expandafter\XINT_div_II_e
    \romannumeral0\expandafter\XINT_div_sub\expandafter
      {\romannumeral0\XINT_rev_nounsep {}#8\R!\R!\R!\R!\R!\R!\R!\W}%
```

```
      {\the\numexpr\XINT_div_smallmul_a 100000000.#1#2#3#4.#5!#91\Z!}%
    .{#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.#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_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.#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
    .\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!\W}%
       {\the\numexpr\XINT_div_smallmul_a 100000000.#1#2#3#4.#5!#71\Z!}}%
    {#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.#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.{#3}#1{#2}#5.%
}%
\def\XINT_div_II_l #1.#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.#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\Z! --,
but keeps ever a vanishing ending carry.
```

```
\def\XINT_div_minimulwc_a 1#1.#2.#3!#4#5#6#7#8.%
{%
    \expandafter\XINT_div_minimulwc_b
    \the\numexpr \xint_c_x^ix+#1+#3*#8.#3*#4#5#6#7+#2*#8.#2*#4#5#6#7.%
}%
\def\XINT_div_minimulwc_b 1#1#2#3#4#5#6.#7.%
{%
    \expandafter\XINT_div_minimulwc_c
    \the\numexpr \xint_c_x^ix+#1#2#3#4#5+#7.#6.%
}%
\def\XINT_div_minimulwc_c 1#1#2#3#4#5#6.#7.#8.%
{%
    1#6#7\expandafter!%
    \the\numexpr\expandafter\XINT_div_smallmul_a
    \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8.%
}%
\def\XINT_div_smallmul_a #1.#2.#3!1#4!%
{%
    \xint_gob_til_Z #4\XINT_div_smallmul_e\Z
    \XINT_div_minimulwc_a #1.#2.#3!#4.#2.#3!%
}%
\def\XINT_div_smallmul_e\Z\XINT_div_minimulwc_a 1#1.#2\Z #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\Z! 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.#1}%
\def\XINT_div_verysmallisone 1\XINT_div_verysmallmul_a 0.1!1#11\Z!%
   {1\relax #1100000000!}%
```

```
\def\XINT_div_verysmallmul_a #1.#2!1#3!%
{%
    \xint_gob_til_Z #3\XINT_div_verysmallmul_e\Z
    \expandafter\XINT_div_verysmallmul_b
    \the\numexpr \xint_c_x^ix+#2*#3+#1.#2!%
}%
\def\XINT_div_verysmallmul_b 1#1#2.%
    {1#2\expandafter!\the\numexpr\XINT_div_verysmallmul_a #1.}%
\def\XINT_div_verysmallmul_e\Z #1\Z +#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\Z!\Z!\Z!\Z!\W #1\Z!\Z!\Z!\Z!\Z!\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!\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_Z #4\XINT_div_sub_bi \Z
    \expandafter\XINT_div_sub_c\the\numexpr#1-#3+1#4-\xint_c_i.%
}%
\def\XINT_div_sub_c 1#1#2.%
{%
    1#2\expandafter!\the\numexpr\XINT_div_sub_d #1%
}%
\def\XINT_div_sub_d #1#2#3!#4!%
{%
    \xint_gob_til_Z #4\XINT_div_sub_di \Z
    \expandafter\XINT_div_sub_e\the\numexpr#1-#3+1#4-\xint_c_i.%
}%
\def\XINT_div_sub_e 1#1#2.%
{%
    1#2\expandafter!\the\numexpr\XINT_div_sub_f #1%
}%
\def\XINT_div_sub_f #1#2#3!#4!%
{%
    \xint_gob_til_Z #4\XINT_div_sub_fi \Z
    \expandafter\XINT_div_sub_g\the\numexpr#1-#3+1#4-\xint_c_i.%
}%
\def\XINT_div_sub_g 1#1#2.%
```

```
{%
    1#2\expandafter!\the\numexpr\XINT_div_sub_h #1%
}%
\def\XINT_div_sub_h #1#2#3!#4!%
{%
    \xint_gob_til_Z #4\XINT_div_sub_hi \Z
    \expandafter\XINT_div_sub_i\the\numexpr#1-#3+1#4-\xint_c_i.%
}%
\def\XINT_div_sub_i 1#1#2.%
{%
    1#2\expandafter!\the\numexpr\XINT_div_sub_a #1%
}%
\def\XINT_div_sub_bi\Z
    \expandafter\XINT_div_sub_c\the\numexpr#1-#2+#3.#4!#5!#6!#7!#8!#9!\Z !\W
{%
    \XINT_div_sub_l #1#2!#5!#7!#9!%
}%
\def\XINT_div_sub_di\Z
    \expandafter\XINT_div_sub_e\the\numexpr#1-#2+#3.#4!#5!#6!#7!#8\W
{%
    \XINT_div_sub_l #1#2!#5!#7!%
}%
\def\XINT_div_sub_fi\Z
    \expandafter\XINT_div_sub_g\the\numexpr#1-#2+#3.#4!#5!#6\W
{%
    \XINT_div_sub_l #1#2!#5!%
}%
\def\XINT_div_sub_hi\Z
    \expandafter\XINT_div_sub_i\the\numexpr#1-#2+#3.#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\Z!
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\Z!#2!%
    {\expandafter
```

```
      {\romannumeral0\XINT_unsep_cuzsmall#11\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W}%
      {#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 #1.1#2!1#3!%
{%
    \expandafter\XINT_smalldivx_b
    \the\numexpr (#3+#1)/#2-\xint_c_i!#1.#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.#3!#4!%
{%
    \expandafter\XINT_smalldiv_d\the\numexpr #4-#1*#3!#2.#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. Ce dernier, malgré le fait qu'on soit dans le bon ordre déjà fait une macro dans le style O(N^2) car sinon le nombre maximal de chiffres serait moitié moins à cause des nettoyages nécessaires après \numexpr. Je suis obligé de faire un nettoyage final car comme l'expansion est engendrée par \numexpr, elle me boufferait des leading zeros si je ne mettais pas un 1 devant chaque bloc en sortie de Q.

```
\def\XINT_smalldiv_d #1!#2!1#3#4!%
{%
    \xint_gob_til_Z #3\XINT_smalldiv_end \Z
    \XINT_smalldiv_e #1!#2!1#3#4!%
}%
\def\XINT_smalldiv_end\Z\XINT_smalldiv_e #1!#2!1\Z!{1!\Z!#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.#3!%
{%
    \expandafter\XINT_smalldiv_f\the\numexpr
    \xint_c_xi_e_viii_mone+#1*\xint_c_x^viii/#3!#2.#3!#1!%
}%
\def\XINT_smalldiv_f 1#1#2#3#4#5#6!#7.#8!%
{%
     \xint_gob_til_zero #1\XINT_smalldiv_fz 0%
     \expandafter\XINT_smalldiv_g
     \the\numexpr\XINT_minimul_a #2#3#4#5.#6!#8!#2#3#4#5#6!#7.#8!%
}%
\def\XINT_smalldiv_fz 0%
    \expandafter\XINT_smalldiv_g\the\numexpr\XINT_minimul_a
```

```
    9999.9999!#1!99999999!#2!0!1#3!%
{%
    \XINT_smalldiv_i .#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.#2!#5!#3!#4!%
}%
\def\XINT_smalldiv_h 1#1#2.#3!#4!%
{%
    \expandafter\XINT_smalldiv_i\the\numexpr #4-#3+#1-\xint_c_i.#2!%
}%
\def\XINT_smalldiv_i #1.#2!#3!#4.#5!%
{%
    \expandafter\XINT_smalldiv_j\the\numexpr (#1#2+#4)/#5-\xint_c_i!#3!#1#2!#4.#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.#4!%
{%
    \expandafter\XINT_smalldiv_d\the\numexpr #2-#1*#4!#3.#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.#2!1#3!%
{%
    \expandafter\XINT_div_mini_a\the\numexpr
    \xint_c_xi_e_viii_mone+#3*\xint_c_x^viii/#1!#1.#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.#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.#6!#7!#2#3#4#5#6!#7.#8!%
```

```
}%
\def\XINT_div_mini_w 0%
    \expandafter\XINT_div_mini_b\the\numexpr\XINT_minimul_a
    9999.9999!#1!99999999!#2.#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.#2!#5!#3!#4!%
}%
\def\XINT_div_mini_c 1#1#2.#3!#4!%
{%
    \expandafter\XINT_div_mini_d
    \the\numexpr #4-#3+#1-\xint_c_i.#2!%
}%
\def\XINT_div_mini_d #1.#2!#3!#4.#5!%
{%
    \xint_c_x^viii_mone+#3+(#1#2+#5)/#4!%
}%
```

3.28 \xintiDivRound, \xintiiDivRound

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

```
\def\xintiDivRound    {\romannumeral0\xintidivround }%
\def\xintidivround  #1%
   {\expandafter\XINT_idivround\romannumeral0\xintnum{#1}\Z }%
\def\xintiiDivRound   {\romannumeral0\xintiidivround }%
\def\xintiidivround #1{\expandafter\XINT_iidivround \romannumeral`&&@#1\Z }%
\def\XINT_idivround #1#2\Z #3%
    {\expandafter\XINT_iidivround_a\expandafter #1%
                  \romannumeral0\xintnum{#3}\Z #2\Z }%
\def\XINT_iidivround #1#2\Z #3%
    {\expandafter\XINT_iidivround_a\expandafter #1\romannumeral`&&@#3\Z #2\Z }%
\def\XINT_iidivround_a #1#2% #1 de A, #2 de B.
{%
    \if0#2\xint_dothis\XINT_iidivround_divbyzero\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\Z #2\Z {\xintError:DivisionByZero\space 0}%
\def\XINT_iidivround_aiszero   #1\Z #2\Z { 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\Z #3\Z
{%
    \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
}%
```

3.29 \xintiDivTrunc, \xintiiDivTrunc

```
\def\xintiDivTrunc    {\romannumeral0\xintidivtrunc }%
\def\xintidivtrunc  #1{\expandafter\XINT_iidivtrunc\romannumeral0\xintnum{#1}\Z }%
\def\xintiiDivTrunc   {\romannumeral0\xintiidivtrunc }%
\def\xintiidivtrunc #1{\expandafter\XINT_iidivtrunc \romannumeral`&&@#1\Z }%
\def\XINT_iidivtrunc #1#2\Z #3{\expandafter\XINT_iidivtrunc_a\expandafter #1%
                               \romannumeral`&&@#3\Z #2\Z }%
\def\XINT_iidivtrunc_a #1#2% #1 de A, #2 de B.
{%
    \if0#2\xint_dothis\XINT_iidivround_divbyzero\fi
    \if0#1\xint_dothis\XINT_iidivround_aiszero\fi
    \if-#2\xint_dothis{\XINT_iidivtrunc_bneg #1}\fi
          \xint_orthat{\XINT_iidivtrunc_bpos #1#2}%
}%
\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\Z #3\Z%
    {\expandafter\xint_firstoftwo_thenstop
     \romannumeral0\XINT_div_prepare {#2}{#1#3}}%
```

3.30 \xintiMod, \xintiiMod

```
\def\xintiMod    {\romannumeral0\xintimod }%
\def\xintimod  #1{\expandafter\XINT_iimod\romannumeral0\xintnum{#1}\Z }%
\def\xintiiMod   {\romannumeral0\xintiimod }%
\def\xintiimod #1{\expandafter\XINT_iimod \romannumeral`&&@#1\Z }%
```

```
\def\XINT_iimod #1#2\Z #3{\expandafter\XINT_iimod_a\expandafter #1%
                           \romannumeral`&&@#3\Z #2\Z }%
\def\XINT_iimod_a #1#2% #1 de A, #2 de B.
{%
    \if0#2\xint_dothis\XINT_iidivround_divbyzero\fi
    \if0#1\xint_dothis\XINT_iidivround_aiszero\fi
    \if-#2\xint_dothis{\XINT_iimod_bneg #1}\fi
          \xint_orthat{\XINT_iimod_bpos #1#2}%
}%
\def\XINT_iimod_bpos #1%
{%
    \xint_UDsignfork
             #1{\xintiiopp\XINT_iimod_pos {}}%
              -{\XINT_iimod_pos #1}%
    \krof
}%
\def\XINT_iimod_bneg #1%
{%
    \xint_UDsignfork
             #1{\xintiiopp\XINT_iimod_pos {}}%
              -{\XINT_iimod_pos #1}%
    \krof
}%
\def\XINT_iimod_pos #1#2\Z #3\Z%
    {\expandafter\xint_secondoftwo_thenstop\romannumeral0\XINT_div_prepare
      {#2}{#1#3}}%
```

3.31 "Load xintfrac" macros

Originally was used in \xintiiexpr. Transferred from xintfrac for 1.1.

```
\catcode`! 11
\def\xintAbs {\Did_you_mean_iiAbs?or_load_xintfrac!}%
\def\xintOpp {\Did_you_mean_iiOpp?or_load_xintfrac!}%
\def\xintAdd {\Did_you_mean_iiAdd?or_load_xintfrac!}%
\def\xintSub {\Did_you_mean_iiSub?or_load_xintfrac!}%
\def\xintMul {\Did_you_mean_iiMul?or_load_xintfrac!}%
\def\xintPow {\Did_you_mean_iiPow?or_load_xintfrac!}%
\def\xintSqr {\Did_you_mean_iiSqr?or_load_xintfrac!}%
\def\xintQuo {\Removed!use_xintiQuo_or_xintiiQuo!}%
\def\xintRem {\Removed!use_xintiRem_or_xintiiRem!}%
\XINT_restorecatcodes_endinput%
```

4 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.

```
\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)
```

4.1 Package identification

```
\XINT_providespackage
\ProvidesPackage{xint}%
  [2016/12/22 1.2j Expandable operations on big integers (JFB)]%
```

4.2 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}%
\edef\xint_cleanupzeros_andstop #1#2#3#4%
{%
    \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4\relax
}%
```

4.3 \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
}%
```

4.4 \xintIsOne, \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\xintIsOne   {\romannumeral0\xintisone }%
\def\xintisone   #1{\expandafter\XINT_isone\romannumeral0\xintnum{#1}XY}%
\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_iSone#1XY}%
\def\XINT_iSone #1#2#3Y%
{%
    \unless\if#2X\xint_dothis0\fi
    \unless\if#11\xint_dothis0\fi
    \xint_orthat1%
}%
```

4.5 \xintRev

\xintRev: expands fully its argument \romannumeral-`0, and checks the sign. However this last aspect does not appear like a very useful thing. And despite the fact that a special check is made for a sign, actually the input is not given to \xintnum, contrarily to \xintLen. This is all a bit incoherent. Should be fixed.
  1.2 has \xintReverseDigits and I thus make \xintRev an alias. Remarks above not addressed.

```
\let\xintRev\xintReverseDigits
```

4.6 \xintLen

\xintLen is ONLY for (possibly long) integers. Gets extended to fractions by xintfrac.sty

```
\def\xintLen {\romannumeral0\xintlen }%
\def\xintlen #1%
{%
    \expandafter\XINT_len_fork
    \romannumeral0\xintnum{#1}\xint_relax\xint_relax\xint_relax\xint_relax
      \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
      \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
}%
\edef\XINT_len_fork #1%
{%
    \noexpand\expandafter\space
    \unexpanded{\the\numexpr\expandafter
                \XINT_length_loop\xint_UDsignfork#1{}-{#1}\krof}%
}%
```

4.7 \xintBool, \xintToggle

1.09c

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

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

4.9 \xintifZero, \xintifNotZero, \xintiiifZero, \xintiiifNotZero

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.

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

4.10 \xintifOne,\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
}%
\def\xintifOne {\romannumeral0\xintifone }%
\def\xintifone #1%
{%
    \if1\xintIsOne{#1}%
       \expandafter\xint_firstoftwo_thenstop
    \else
       \expandafter\xint_secondoftwo_thenstop
    \fi
}%
```

4.11 \xintifTrueAelseB, \xintifFalseAelseB

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

```
\let\xintifTrueAelseB\xintifNotZero
```

```
\let\xintifFalseAelseB\xintifZero
%%\let\xintifTrue\xintifNotZero      % now removed
%%\let\xintifTrueFalse\xintifNotZero % now removed
```

4.12 \xintifCmp, \xintiiifCmp

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

```
\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\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
}%
```

4.13 \xintifEq, \xintiiifEq

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

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

4.14 \xintifGt, \xintiiifGt

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

```
\def\xintifGt {\romannumeral0\xintifgt }%
```

```
\def\xintifgt #1#2%
{%
    \if1\xintCmp{#1}{#2}%
                \expandafter\xint_firstoftwo_thenstop
          \else\expandafter\xint_secondoftwo_thenstop
    \fi
}%
\def\xintiiifGt {\romannumeral0\xintiiifgt }%
\def\xintiiifgt #1#2%
{%
    \if1\xintiiCmp{#1}{#2}%
                \expandafter\xint_firstoftwo_thenstop
          \else\expandafter\xint_secondoftwo_thenstop
    \fi
}%
```

4.15 \xintifLt, \xintiiifLt

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

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

4.16 \xintifOdd, \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
}%
\def\xintifOdd {\romannumeral0\xintifodd }%
\def\xintifodd #1%
{%
```

```
    \if\xintOdd{#1}1%
       \expandafter\xint_firstoftwo_thenstop
    \else
       \expandafter\xint_secondoftwo_thenstop
    \fi
}%
```

4.17 \xintCmp, \xintiiCmp

Faster than doing the full subtraction.

```
\def\xintCmp    {\romannumeral0\xintcmp }%
\def\xintcmp   #1{\expandafter\XINT_icmp\romannumeral0\xintnum{#1}\Z }%
\def\xintiiCmp   {\romannumeral0\xintiicmp }%
\def\xintiicmp #1{\expandafter\XINT_iicmp\romannumeral`&&@#1\Z  }%
\def\XINT_iicmp #1#2\Z #3%
{%
    \expandafter\XINT_cmp_nfork\expandafter #1\romannumeral`&&@#3\Z #2\Z
}%
\let\XINT_Cmp    \xintiiCmp
\def\XINT_icmp #1#2\Z #3%
{%
    \expandafter\XINT_cmp_nfork\expandafter #1\romannumeral0\xintnum{#3}\Z #2\Z
}%
\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\Z #4\Z
{%
    \xint_UDzerominusfork
      #2-{ 0}%
      0#2{ 1}%
       0-{ -1}%
    \krof
}%
\def\XINT_cmp_secondiszero #1\krof #20#3\Z #4\Z
{%
    \xint_UDzerominusfork
      #2-{ 0}%
      0#2{ -1}%
       0-{ 1}%
    \krof
```

```
}%
\def\XINT_cmp_plusminus    #1\Z #2\Z{ 1}%
\def\XINT_cmp_minusplus    #1\Z #2\Z{ -1}%
\def\XINT_cmp_minusminus
    --{\expandafter\XINT_opp\romannumeral0\XINT_cmp_plusplus {}{}}%
\def\XINT_cmp_plusplus  #1#2#3\Z
{%
  \expandafter\XINT_cmp_pp
      \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_iii
      \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
      \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
  \X #1%
}%
\def\XINT_cmp_pp #1.#2\X #3\Z
{%
    \expandafter\XINT_cmp_checklengths
    \the\numexpr #1\expandafter.%
    \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_iii
      \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
      \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
    \Z!\Z!\Z!\Z!\Z!\W #2\Z!\Z!\Z!\Z!\Z!\W
}%
\def\XINT_cmp_checklengths #1.#2.%
{%
    \ifnum #1=#2
       \expandafter\xint_firstoftwo
    \else
       \expandafter\xint_secondoftwo
    \fi
    \XINT_cmp_aa {\XINT_cmp_distinctlengths {#1}{#2}}%
}%
\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_aa {\expandafter\XINT_cmp_w\the\numexpr\XINT_cmp_a \xint_c_i }%
\def\XINT_cmp_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
{%
    \XINT_cmp_b #1!#6!#2!#7!#3!#8!#4!#9!#5\W
}%
\def\XINT_cmp_b #1#2#3!#4!%
```

```
{%
    \xint_gob_til_Z #2\XINT_cmp_bi \Z
    \expandafter\XINT_cmp_c\the\numexpr#1+1#4-#3-\xint_c_i.%
}%
\def\XINT_cmp_c 1#1#2.%
{%
    1#2\expandafter!\the\numexpr\XINT_cmp_d #1%
}%
\def\XINT_cmp_d #1#2#3!#4!%
{%
    \xint_gob_til_Z #2\XINT_cmp_di \Z
    \expandafter\XINT_cmp_e\the\numexpr#1+1#4-#3-\xint_c_i.%
}%
\def\XINT_cmp_e 1#1#2.%
{%
    1#2\expandafter!\the\numexpr\XINT_cmp_f #1%
}%
\def\XINT_cmp_f #1#2#3!#4!%
{%
    \xint_gob_til_Z #2\XINT_cmp_fi \Z
    \expandafter\XINT_cmp_g\the\numexpr#1+1#4-#3-\xint_c_i.%
}%
\def\XINT_cmp_g 1#1#2.%
{%
    1#2\expandafter!\the\numexpr\XINT_cmp_h #1%
}%
\def\XINT_cmp_h #1#2#3!#4!%
{%
    \xint_gob_til_Z #2\XINT_cmp_hi \Z
    \expandafter\XINT_cmp_i\the\numexpr#1+1#4-#3-\xint_c_i.%
}%
\def\XINT_cmp_i 1#1#2.%
{%
    1#2\expandafter!\the\numexpr\XINT_cmp_a #1%
}%
\def\XINT_cmp_bi\Z
    \expandafter\XINT_cmp_c\the\numexpr#1+1#2-#3.#4!#5!#6!#7!#8!#9!\Z !\W
{%
    \XINT_cmp_k #1#2!#5!#7!#9!%
}%
\def\XINT_cmp_di\Z
    \expandafter\XINT_cmp_e\the\numexpr#1+1#2-#3.#4!#5!#6!#7!#8\W
{%
    \XINT_cmp_k #1#2!#5!#7!%
}%
\def\XINT_cmp_fi\Z
    \expandafter\XINT_cmp_g\the\numexpr#1+1#2-#3.#4!#5!#6\W
{%
    \XINT_cmp_k #1#2!#5!%
}%
\def\XINT_cmp_hi\Z
    \expandafter\XINT_cmp_i\the\numexpr#1+1#2-#3.#4\W
```

```
{%
    \XINT_cmp_k #1#2!%
}%
\def\XINT_cmp_k #1#2\W
{%
   \xint_UDzerofork
      #1{-1\relax \XINT_cmp_greater}%
       0{-1\relax \XINT_cmp_lessorequal}%
   \krof
}%
\def\XINT_cmp_w #1-1#2{#2#11\Z!\W}%
\def\XINT_cmp_greater #1\Z!\W{ 1}%
\def\XINT_cmp_lessorequal 1#1!%
    {\xint_gob_til_Z #1\XINT_cmp_equal\Z
     \xint_gob_til_eightzeroes #1\XINT_cmp_continue 00000000%
     \XINT_cmp_less }%
\def\XINT_cmp_less #1\W { -1}%
\def\XINT_cmp_continue 00000000\XINT_cmp_less {\XINT_cmp_lessorequal }%
\def\XINT_cmp_equal\Z\xint_gob_til_eightzeroes\Z\XINT_cmp_continue
    00000000\XINT_cmp_less\W { 0}%
```

4.18 \xintEq, \xintGt, \xintLt

1.09a.

```
\def\xintEq {\romannumeral0\xinteq }\def\xinteq #1#2{\xintifeq{#1}{#2}{1}{0}}%
\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}}%
```

4.19 \xintNeq, \xintGtorEq, \xintLtorEq

1.1. Pour xintexpr. No lowercase macros

```
\def\xintLtorEq #1#2{\romannumeral0\xintifgt {#1}{#2}{0}{1}}%
\def\xintGtorEq #1#2{\romannumeral0\xintiflt {#1}{#2}{0}{1}}%
\def\xintNeq    #1#2{\romannumeral0\xintifeq {#1}{#2}{0}{1}}%
```

4.20 \xintiiEq, \xintiiGt, \xintiiLt

1.1a Pour \xintiiexpr. No lowercase macros.

```
\def\xintiiEq #1#2{\romannumeral0\xintiiifeq{#1}{#2}{1}{0}}%
\def\xintiiGt #1#2{\romannumeral0\xintiiifgt{#1}{#2}{1}{0}}%
\def\xintiiLt #1#2{\romannumeral0\xintiiiflt{#1}{#2}{1}{0}}%
```

4.21 \xintiiNeq, \xintiiGtorEq, \xintiiLtorEq

1.1a. Pour \xintiiexpr. No lowercase macros.

```
\def\xintiiLtorEq #1#2{\romannumeral0\xintiiifgt {#1}{#2}{0}{1}}%
\def\xintiiGtorEq #1#2{\romannumeral0\xintiiiflt {#1}{#2}{0}{1}}%
\def\xintiiNeq    #1#2{\romannumeral0\xintiiifeq {#1}{#2}{0}{1}}%
```

4.22 \xintIsZero, \xintIsNotZero, \xintiiIsZero, \xintiiIsNotZero

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

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

4.23 \xintIsTrue, \xintNot, \xintIsFalse

1.09c

```
\let\xintIsTrue\xintIsNotZero
\let\xintNot\xintIsZero
\let\xintIsFalse\xintIsZero
```

4.24 \xintAND, \xintOR, \xintXOR

1.09a. Embarrasing bugs in \xintAND and \xintOR which inserted a space token corrected in 1.09i. \xintxor restyled with \if (faster) in 1.09i

```
\def\xintAND {\romannumeral0\xintand }%
\def\xintand #1#2{\if0\xintSgn{#1}\expandafter\xint_firstoftwo
                              \else\expandafter\xint_secondoftwo\fi
                  { 0}{\xintisnotzero{#2}}}%
\def\xintOR {\romannumeral0\xintor }%
\def\xintor #1#2{\if0\xintSgn{#1}\expandafter\xint_firstoftwo
                            \else\expandafter\xint_secondoftwo\fi
                 {\xintisnotzero{#2}}{ 1}}%
\def\xintXOR {\romannumeral0\xintxor }%
\def\xintxor #1#2{\if\xintIsZero{#1}\xintIsZero{#2}%
                     \xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi }%
```

4.25 \xintANDof

New with 1.09a. \xintANDof works also with an empty list.

```
\def\xintANDof      {\romannumeral0\xintandof }%
\def\xintandof    #1{\expandafter\XINT_andof_a\romannumeral`&&@#1\relax }%
\def\XINT_andof_a #1{\expandafter\XINT_andof_b\romannumeral`&&@#1\Z }%
\def\XINT_andof_b #1%
           {\xint_gob_til_relax #1\XINT_andof_e\relax\XINT_andof_c #1}%
\def\XINT_andof_c #1\Z
           {\xintifTrueAelseB {#1}{\XINT_andof_a}{\XINT_andof_no}}%
\def\XINT_andof_no #1\relax { 0}%
\def\XINT_andof_e #1\Z { 1}%
```

4.26 \xintORof

New with 1.09a. Works also with an empty list.

```
\def\xintORof      {\romannumeral0\xintorof }%
\def\xintorof    #1{\expandafter\XINT_orof_a\romannumeral`&&@#1\relax }%
\def\XINT_orof_a #1{\expandafter\XINT_orof_b\romannumeral`&&@#1\Z }%
\def\XINT_orof_b #1%
           {\xint_gob_til_relax #1\XINT_orof_e\relax\XINT_orof_c #1}%
\def\XINT_orof_c #1\Z
           {\xintifTrueAelseB {#1}{\XINT_orof_yes}{\XINT_orof_a}}%
\def\XINT_orof_yes #1\relax { 1}%
\def\XINT_orof_e #1\Z { 0}%
```

4.27 \xintXORof

New with 1.09a. Works with an empty list, too. \XINT_xorof_c more efficient in 1.09i

```
\def\xintXORof      {\romannumeral0\xintxorof }%
\def\xintxorof    #1{\expandafter\XINT_xorof_a\expandafter
                     0\romannumeral`&&@#1\relax }%
\def\XINT_xorof_a #1#2{\expandafter\XINT_xorof_b\romannumeral`&&@#2\Z #1}%
\def\XINT_xorof_b #1%
           {\xint_gob_til_relax #1\XINT_xorof_e\relax\XINT_xorof_c #1}%
\def\XINT_xorof_c #1\Z #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\Z #2{ #2}%
```

4.28 \xintGeq, \xintiiGeq

PLUS GRAND OU ÉGAL attention compare les **valeurs absolues**

```
\def\xintGeq    {\romannumeral0\xintgeq }%
\def\xintgeq   #1{\expandafter\XINT_geq\romannumeral0\xintnum{#1}\Z }%
\def\xintiiGeq   {\romannumeral0\xintiigeq }%
\def\xintiigeq #1{\expandafter\XINT_iigeq\romannumeral`&&@#1\Z  }%
\def\XINT_iigeq #1#2\Z #3%
{%
    \expandafter\XINT_geq_fork\expandafter #1\romannumeral`&&@#3\Z #2\Z
}%
\let\XINT_geq_pre \xintiigeq % TEMPORAIRE (oui, mais depuis quand ?)
\let\XINT_Geq \xintGeq       % TEMPORAIRE ATTENTION FAIT xintNum (et alors?)
\def\XINT_geq #1#2\Z #3%
{%
    \expandafter\XINT_geq_fork\expandafter #1\romannumeral0\xintnum{#3}\Z #2\Z
}%
\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\Z #4\Z
                             {\xint_UDzerofork #2{ 1}0{ 0}\krof }%
\def\XINT_geq_secondiszero #1\krof #20#3\Z #4\Z { 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 #1#2#3\Z #4\Z {\XINT_geq_pp #1#4\Z #2#3\Z }%
\def\XINT_geq_pp #1\Z
{%
  \expandafter\XINT_geq_pp_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_iii
      \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
      \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
  \X
}%
\def\XINT_geq_pp_a #1.#2\X #3\Z
{%
    \expandafter\XINT_geq_checklengths
    \the\numexpr #1\expandafter.%
    \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_iii
      \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
      \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
    \Z!\Z!\Z!\Z!\Z!\W #2\Z!\Z!\Z!\Z!\Z!\W
}%
\def\XINT_geq_checklengths #1.#2.%
{%
    \ifnum #1=#2
       \expandafter\xint_firstoftwo
    \else
       \expandafter\xint_secondoftwo
    \fi
    \XINT_geq_aa {\XINT_geq_distinctlengths {#1}{#2}}%
}%
\def\XINT_geq_distinctlengths #1#2#3\W #4\W
{%
    \ifnum #1>#2
```

```
        \expandafter\xint_firstoftwo
    \else
        \expandafter\xint_secondoftwo
    \fi
    { 1}{ 0}%
}%
\def\XINT_geq_aa {\expandafter\XINT_geq_w\the\numexpr\XINT_geq_a \xint_c_i }%
\def\XINT_geq_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
{%
    \XINT_geq_b #1!#6!#2!#7!#3!#8!#4!#9!#5\W
}%
\def\XINT_geq_b #1#2#3!#4!%
{%
    \xint_gob_til_Z #2\XINT_geq_bi \Z
    \expandafter\XINT_geq_c\the\numexpr#1+1#4-#3-\xint_c_i.%
}%
\def\XINT_geq_c 1#1#2.%
{%
    1#2\expandafter!\the\numexpr\XINT_geq_d #1%
}%
\def\XINT_geq_d #1#2#3!#4!%
{%
    \xint_gob_til_Z #2\XINT_geq_di \Z
    \expandafter\XINT_geq_e\the\numexpr#1+1#4-#3-\xint_c_i.%
}%
\def\XINT_geq_e 1#1#2.%
{%
    1#2\expandafter!\the\numexpr\XINT_geq_f #1%
}%
\def\XINT_geq_f #1#2#3!#4!%
{%
    \xint_gob_til_Z #2\XINT_geq_fi \Z
    \expandafter\XINT_geq_g\the\numexpr#1+1#4-#3-\xint_c_i.%
}%
\def\XINT_geq_g 1#1#2.%
{%
    1#2\expandafter!\the\numexpr\XINT_geq_h #1%
}%
\def\XINT_geq_h #1#2#3!#4!%
{%
    \xint_gob_til_Z #2\XINT_geq_hi \Z
    \expandafter\XINT_geq_i\the\numexpr#1+1#4-#3-\xint_c_i.%
}%
\def\XINT_geq_i 1#1#2.%
{%
    1#2\expandafter!\the\numexpr\XINT_geq_a #1%
}%
\def\XINT_geq_bi\Z
    \expandafter\XINT_geq_c\the\numexpr#1+1#2-#3.#4!#5!#6!#7!#8!#9!\Z !\W
{%
    \XINT_geq_k #1#2!#5!#7!#9!%
}%
```

```
\def\XINT_geq_di\Z
    \expandafter\XINT_geq_e\the\numexpr#1+1#2-#3.#4!#5!#6!#7!#8\W
{%
    \XINT_geq_k #1#2!#5!#7!%
}%
\def\XINT_geq_fi\Z
    \expandafter\XINT_geq_g\the\numexpr#1+1#2-#3.#4!#5!#6\W
{%
    \XINT_geq_k #1#2!#5!%
}%
\def\XINT_geq_hi\Z
    \expandafter\XINT_geq_i\the\numexpr#1+1#2-#3.#4\W
{%
    \XINT_geq_k #1#2!%
}%
\def\XINT_geq_k #1#2\W
{%
   \xint_UDzerofork
      #1{-1\relax { 0}}%
       0{-1\relax { 1}}%
   \krof
}%
\def\XINT_geq_w #1-1#2{#2}%
```

4.29 \xintiMax, \xintiiMax

```
1.2 REMOVES \xintMax, \xintMin, \xintMaxof, \xintMinof.
```

```
\def\xintiMax {\romannumeral0\xintimax }%
\def\xintimax #1%
{%
    \expandafter\xint_max\expandafter {\romannumeral0\xintnum{#1}}%
}%
\def\xint_max #1#2%
{%
    \expandafter\XINT_max_pre\expandafter {\romannumeral0\xintnum{#2}}{#1}%
}%
\def\xintiiMax {\romannumeral0\xintiimax }%
\def\xintiimax #1%
{%
    \expandafter\xint_iimax\expandafter {\romannumeral`&&@#1}%
}%
\def\xint_iimax #1#2%
{%
    \expandafter\XINT_max_pre\expandafter {\romannumeral`&&@#2}{#1}%
}%
\def\XINT_max_pre #1#2{\XINT_max_fork #1\Z #2\Z {#2}{#1}}%
\def\XINT_Max #1#2{\romannumeral0\XINT_max_fork #2\Z #1\Z {#1}{#2}}%
```

```
#3#4 vient du *premier*, #1#2 vient du *second*
```

```
\def\XINT_max_fork #1#2\Z #3#4\Z
{%
```

```
    \xint_UDsignsfork
          #1#3\XINT_max_minusminus  % A < 0, B < 0
           #1-\XINT_max_minusplus   % B < 0, A >= 0
           #3-\XINT_max_plusminus   % A < 0, B >= 0
            --{\xint_UDzerosfork
                      #1#3\XINT_max_zerozero % A = B = 0
                       #10\XINT_max_zeroplus % B = 0, A > 0
                       #30\XINT_max_pluszero % A = 0, B > 0
                        00\XINT_max_plusplus % A, B > 0
                      \krof }%
    \krof
    {#2}{#4}#1#3%
}%
```

A = #4#2, B = #3#1

```
\def\XINT_max_zerozero  #1#2#3#4{\xint_firstoftwo_thenstop }%
\def\XINT_max_zeroplus  #1#2#3#4{\xint_firstoftwo_thenstop }%
\def\XINT_max_pluszero  #1#2#3#4{\xint_secondoftwo_thenstop }%
\def\XINT_max_minusplus #1#2#3#4{\xint_firstoftwo_thenstop }%
\def\XINT_max_plusminus #1#2#3#4{\xint_secondoftwo_thenstop }%
\def\XINT_max_plusplus  #1#2#3#4%
{%
    \ifodd\XINT_Geq {#4#2}{#3#1}
      \expandafter\xint_firstoftwo_thenstop
    \else
      \expandafter\xint_secondoftwo_thenstop
    \fi
}%
```

#3=-, #4=-, #1 = |B| = -B, #2 = |A| = -A

```
\def\XINT_max_minusminus #1#2#3#4%
{%
    \ifodd\XINT_Geq {#1}{#2}
      \expandafter\xint_firstoftwo_thenstop
    \else
      \expandafter\xint_secondoftwo_thenstop
    \fi
}%
```

4.30 \xintiMaxof, \xintiiMaxof

New with 1.09a. 1.2 has NO MORE \xintMaxof, requires \xintfracname. 1.2a adds \xintiiMaxof, as \xintiiMaxof:csv is not public.

```
\def\xintiMaxof      {\romannumeral0\xintimaxof }%
\def\xintimaxof    #1{\expandafter\XINT_imaxof_a\romannumeral`&&@#1\relax }%
\def\XINT_imaxof_a #1{\expandafter\XINT_imaxof_b\romannumeral0\xintnum{#1}\Z }%
\def\XINT_imaxof_b #1\Z #2%
           {\expandafter\XINT_imaxof_c\romannumeral`&&@#2\Z {#1}\Z}%
\def\XINT_imaxof_c #1%
           {\xint_gob_til_relax #1\XINT_imaxof_e\relax\XINT_imaxof_d #1}%
\def\XINT_imaxof_d #1\Z
```

```
            {\expandafter\XINT_imaxof_b\romannumeral0\xintimax {#1}}%
\def\XINT_imaxof_e #1\Z #2\Z { #2}%
\def\xintiiMaxof      {\romannumeral0\xintiimaxof }%
\def\xintiimaxof    #1{\expandafter\XINT_iimaxof_a\romannumeral`&&@#1\relax }%
\def\XINT_iimaxof_a #1{\expandafter\XINT_iimaxof_b\romannumeral`&&@#1\Z }%
\def\XINT_iimaxof_b #1\Z #2%
            {\expandafter\XINT_iimaxof_c\romannumeral`&&@#2\Z {#1}\Z}%
\def\XINT_iimaxof_c #1%
            {\xint_gob_til_relax #1\XINT_iimaxof_e\relax\XINT_iimaxof_d #1}%
\def\XINT_iimaxof_d #1\Z
            {\expandafter\XINT_iimaxof_b\romannumeral0\xintiimax {#1}}%
\def\XINT_iimaxof_e #1\Z #2\Z { #2}%
```

4.31 \xintiMin, \xintiiMin

```
\xintnum added New with 1.09a. I add \xintiiMin in 1.1 and mark as deprecated \xintMin, re-
named \xintiMin. \xintMin NOW REMOVED (1.2, as \xintMax, \xintMaxof), only provided by \xint-
fracnameimp.
```

```
\def\xintiMin {\romannumeral0\xintimin }%
\def\xintimin #1%
{%
    \expandafter\xint_min\expandafter {\romannumeral0\xintnum{#1}}%
}%
\def\xint_min #1#2%
{%
    \expandafter\XINT_min_pre\expandafter {\romannumeral0\xintnum{#2}}{#1}%
}%
\def\xintiiMin {\romannumeral0\xintiimin }%
\def\xintiimin #1%
{%
    \expandafter\xint_iimin\expandafter {\romannumeral`&&@#1}%
}%
\def\xint_iimin #1#2%
{%
    \expandafter\XINT_min_pre\expandafter {\romannumeral`&&@#2}{#1}%
}%
\def\XINT_min_pre #1#2{\XINT_min_fork #1\Z #2\Z {#2}{#1}}%
\def\XINT_Min #1#2{\romannumeral0\XINT_min_fork #2\Z #1\Z {#1}{#2}}%
```

```
#3#4 vient du *premier*, #1#2 vient du *second*
```

```
\def\XINT_min_fork #1#2\Z #3#4\Z
{%
    \xint_UDsignsfork
          #1#3\XINT_min_minusminus  % A < 0, B < 0
           #1-\XINT_min_minusplus   % B < 0, A >= 0
           #3-\XINT_min_plusminus   % A < 0, B >= 0
            --{\xint_UDzerosfork
                      #1#3\XINT_min_zerozero % A = B = 0
                       #10\XINT_min_zeroplus % B = 0, A > 0
                       #30\XINT_min_pluszero % A = 0, B > 0
                        00\XINT_min_plusplus % A, B > 0
```

```
                      \krof }%
    \krof
    {#2}{#4}#1#3%
}%
```

A = #4#2, B = #3#1

```
\def\XINT_min_zerozero  #1#2#3#4{\xint_firstoftwo_thenstop }%
\def\XINT_min_zeroplus  #1#2#3#4{\xint_secondoftwo_thenstop }%
\def\XINT_min_pluszero  #1#2#3#4{\xint_firstoftwo_thenstop }%
\def\XINT_min_minusplus #1#2#3#4{\xint_secondoftwo_thenstop }%
\def\XINT_min_plusminus #1#2#3#4{\xint_firstoftwo_thenstop }%
\def\XINT_min_plusplus  #1#2#3#4%
{%
    \ifodd\XINT_Geq {#4#2}{#3#1}
      \expandafter\xint_secondoftwo_thenstop
    \else
      \expandafter\xint_firstoftwo_thenstop
    \fi
}%
```

#3=-, #4=-, #1 = |B| = -B, #2 = |A| = -A

```
\def\XINT_min_minusminus #1#2#3#4%
{%
    \ifodd\XINT_Geq {#1}{#2}
      \expandafter\xint_secondoftwo_thenstop
    \else
      \expandafter\xint_firstoftwo_thenstop
    \fi
}%
```

4.32 \xintiMinof, \xintiiMinof

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

```
\def\xintiMinof      {\romannumeral0\xintiminof }%
\def\xintiminof    #1{\expandafter\XINT_iminof_a\romannumeral`&&@#1\relax }%
\def\XINT_iminof_a #1{\expandafter\XINT_iminof_b\romannumeral0\xintnum{#1}\Z }%
\def\XINT_iminof_b #1\Z #2%
           {\expandafter\XINT_iminof_c\romannumeral`&&@#2\Z {#1}\Z}%
\def\XINT_iminof_c #1%
           {\xint_gob_til_relax #1\XINT_iminof_e\relax\XINT_iminof_d #1}%
\def\XINT_iminof_d #1\Z
           {\expandafter\XINT_iminof_b\romannumeral0\xintimin {#1}}%
\def\XINT_iminof_e #1\Z #2\Z { #2}%
\def\xintiiMinof      {\romannumeral0\xintiiminof }%
\def\xintiiminof    #1{\expandafter\XINT_iiminof_a\romannumeral`&&@#1\relax }%
\def\XINT_iiminof_a #1{\expandafter\XINT_iiminof_b\romannumeral`&&@#1\Z }%
\def\XINT_iiminof_b #1\Z #2%
           {\expandafter\XINT_iiminof_c\romannumeral`&&@#2\Z {#1}\Z}%
\def\XINT_iiminof_c #1%
           {\xint_gob_til_relax #1\XINT_iiminof_e\relax\XINT_iiminof_d #1}%
\def\XINT_iiminof_d #1\Z
```

```
            {\expandafter\XINT_iiminof_b\romannumeral0\xintiimin {#1}}%
\def\XINT_iiminof_e #1\Z #2\Z { #2}%
```

4.33 \xintiiSum

```
\xintiiSum {{a}{b}...{z}}, \xintiiSumExpr {a}{b}...{z}\relax
```

```
\def\xintiiSum {\romannumeral0\xintiisum }%
\def\xintiisum #1{\xintiisumexpr #1\relax }%
\def\xintiiSumExpr {\romannumeral0\xintiisumexpr }%
\def\xintiisumexpr {\expandafter\XINT_sumexpr\romannumeral`&&@}%
\def\XINT_sumexpr {\XINT_sum_loop_a 0\Z }%
\def\XINT_sum_loop_a #1\Z #2%
    {\expandafter\XINT_sum_loop_b \romannumeral`&&@#2\Z #1\Z \Z}%
\def\XINT_sum_loop_b #1%
    {\xint_gob_til_relax #1\XINT_sum_finished\relax\XINT_sum_loop_c #1}%
\def\XINT_sum_loop_c
    {\expandafter\XINT_sum_loop_a\romannumeral0\XINT_add_fork }%
\def\XINT_sum_finished #1\Z #2\Z \Z { #2}%
```

4.34 \xintiiPrd

```
\xintiiPrd {{a}...{z}}, \xintiiPrdExpr {a}...{z}\relax
```

```
\def\xintiiPrd {\romannumeral0\xintiiprd }%
\def\xintiiprd #1{\xintiiprdexpr #1\relax }%
\def\xintiiPrdExpr {\romannumeral0\xintiiprdexpr }%
\def\xintiiprdexpr {\expandafter\XINT_prdexpr\romannumeral`&&@}%
\def\XINT_prdexpr {\XINT_prod_loop_a 1\Z }%
\def\XINT_prod_loop_a #1\Z #2%
    {\expandafter\XINT_prod_loop_b  \romannumeral`&&@#2\Z #1\Z \Z}%
\def\XINT_prod_loop_b #1%
    {\xint_gob_til_relax #1\XINT_prod_finished\relax\XINT_prod_loop_c #1}%
\def\XINT_prod_loop_c
    {\expandafter\XINT_prod_loop_a\romannumeral0\XINT_mul_fork }%
\def\XINT_prod_finished\relax\XINT_prod_loop_c #1\Z #2\Z \Z { #2}%
```

```
-----------------------------------------------------------------
-----------------------------------------------------------------
DECIMAL OPERATIONS: FIRST DIGIT, LASTDIGIT, (<- moved to xintcore because xintiiLDg needed by di-
vision macros) ODDNESS, MULTIPLICATION BY TEN, QUOTIENT BY TEN, (moved to xintcore 1.2i) QUOTIENT
OR MULTIPLICATION BY POWER OF TEN, SPLIT OPERATION.
```

4.35 \xintMON, \xintMMON, \xintiiMON, \xintiiMMON

```
MINUS ONE TO THE POWER N and (-1)^{N-1}
```

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

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

4.36 \xintOdd, \xintiiOdd, \xintEven, \xintiiEven

```
\def\xintiiOdd {\romannumeral0\xintiiodd }%
\def\xintiiodd #1%
{%
    \ifodd\xintiiLDg{#1} %<- intentional space
        \xint_afterfi{ 1}%
    \else
        \xint_afterfi{ 0}%
    \fi
}%
\def\xintiiEven {\romannumeral0\xintiieven }%
\def\xintiieven #1%
{%
    \ifodd\xintiiLDg{#1} %<- intentional space
        \xint_afterfi{ 0}%
    \else
        \xint_afterfi{ 1}%
    \fi
}%
\def\xintOdd {\romannumeral0\xintodd }%
```

```
\def\xintodd #1%
{%
    \ifodd\xintLDg{#1} %<- intentional space
        \xint_afterfi{ 1}%
    \else
        \xint_afterfi{ 0}%
    \fi
}%
\def\xintEven {\romannumeral0\xinteven }%
\def\xinteven #1%
{%
    \ifodd\xintLDg{#1} %<- intentional space
        \xint_afterfi{ 0}%
    \else
        \xint_afterfi{ 1}%
    \fi
}%
```

4.37 \xintDSH, \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
}%
\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
}%
```

4.38 \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
```

```
952         0-\XINT_dsx_AisPos
953     \krof #1.#2%
954 }%
955 \def\XINT_dsx_AisZero #1;{{0}{0}}%
956 \def\XINT_dsx_AisNeg #1.-#2;%
957 {%
958     \expandafter\XINT_dsx_AisNeg_checkiffirstempty
959     \romannumeral0\XINT_split_xfork #1.#2\xint_bye2345678\xint_bye..%
960 }%
961 \def\XINT_dsx_AisNeg_checkiffirstempty #1%
962 {%
963     \xint_gob_til_dot #1\XINT_dsx_AisNeg_finish_zero.%
964     \XINT_dsx_AisNeg_finish_notzero #1%
965 }%
966 \def\XINT_dsx_AisNeg_finish_zero.\XINT_dsx_AisNeg_finish_notzero.#1.%
967 {%
968     \expandafter\XINT_dsx_end
969     \expandafter {\romannumeral0\XINT_num {-#1}}{0}%
970 }%
971 \def\XINT_dsx_AisNeg_finish_notzero #1.#2.%
972 {%
973     \expandafter\XINT_dsx_end
974     \expandafter {\romannumeral0\XINT_num {#2}}{-#1}%
975 }%
976 \def\XINT_dsx_AisPos #1.#2;%
977 {%
978     \expandafter\XINT_dsx_AisPos_finish
979     \romannumeral0\XINT_split_xfork #1.#2\xint_bye2345678\xint_bye..%
980 }%
981 \def\XINT_dsx_AisPos_finish #1.#2.%
982 {%
983     \expandafter\XINT_dsx_end
984     \expandafter {\romannumeral0\XINT_num {#2}}%
985                  {\romannumeral0\XINT_num {#1}}%
986 }%
987 \def\XINT_dsx_end #1#2{\expandafter{#2}{#1}}%
```

4.39 \xintDecSplit, \xintDecSplitL, \xintDecSplitR

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).

```
988 \def\xintDecSplit {\romannumeral0\xintdecsplit }%
989 \def\xintdecsplit #1#2%
990 {%
```

```
    \expandafter\XINT_split_finish
    \romannumeral0\expandafter\XINT_split_xfork
    \the\numexpr #1\expandafter.\romannumeral`&&@#2%
    \xint_bye2345678\xint_bye..%
}%
\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\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_split_finish  #1.#2.{{#1}{#2}}%
\def\XINT_splitl_finish #1.#2.{ #1}%
\def\XINT_splitr_finish #1.#2.{ #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_relax\xint_relax\xint_relax\xint_relax
      \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
      \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.}%
```

4.40 \xintiiSqrt, \xintiiSqrtR, \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.
    Attention to impact here of some 1.2i changes to macros (\xintDSx, \xintDecSplit and their in-
  terfaces).

1080 \def\xintiiSqrt  {\romannumeral0\xintiisqrt  }%
1081 \def\xintiiSqrtR {\romannumeral0\xintiisqrtr }%
1082 \def\xintiiSquareRoot {\romannumeral0\xintiisquareroot }%
1083 \def\xintiSqrt        {\romannumeral0\xintisqrt        }%
1084 \def\xintiSquareRoot  {\romannumeral0\xintisquareroot  }%
1085 \def\xintisqrt   {\expandafter\XINT_sqrt_post\romannumeral0\xintisquareroot   }%
1086 \def\xintiisqrt  {\expandafter\XINT_sqrt_post\romannumeral0\xintiisquareroot  }%
1087 \def\xintiisqrtr {\expandafter\XINT_sqrtr_post\romannumeral0\xintiisquareroot }%
1088 \def\XINT_sqrt_post #1#2{\XINT_dec #1\XINT_dec_bye234567890\xint_bye}%

  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.

1089 \def\XINT_sqrtr_post #1#2{\xintiiifLt {#2}{#1}%
1090                           { #1}{\XINT_dec #1\XINT_dec_bye234567890\xint_bye}}%
1091 \def\xintisquareroot #1%
1092    {\expandafter\XINT_sqrt_checkin\romannumeral0\xintnum{#1}\xint_relax }%
1093 \def\xintiisquareroot #1{\expandafter\XINT_sqrt_checkin\romannumeral`&&@#1\xint_relax }%
1094 \def\XINT_sqrt_checkin #1%
1095 {%
1096     \xint_UDzerominusfork
1097      #1-\XINT_sqrt_iszero
1098      0#1\XINT_sqrt_isneg
1099       0-{\XINT_sqrt #1}%
1100     \krof
1101 }%
1102 \def\XINT_sqrt_iszero #1\xint_relax { 11}%
1103 \edef\XINT_sqrt_isneg #1\xint_relax {\noexpand\xintError:RootOfNegative\space 11}%
1104 \def\XINT_sqrt #1\xint_relax
1105 {%
1106     \expandafter\XINT_sqrt_start\romannumeral0\xintlength {#1}.#1.%
1107 }%
1108 \def\XINT_sqrt_start #1.%
1109 {%
1110     \ifnum #1<\xint_c_x\xint_dothis\XINT_sqrt_small_a\fi
1111     \xint_orthat\XINT_sqrt_big_a #1.%
1112 }%
1113 \def\XINT_sqrt_small_a #1.{\XINT_sqrt_a #1.\XINT_sqrt_small_d }%
1114 \def\XINT_sqrt_big_a   #1.{\XINT_sqrt_a #1.\XINT_sqrt_big_d   }%
1115 \def\XINT_sqrt_a #1.%
1116 {%
1117    \ifodd #1
1118      \expandafter\XINT_sqrt_bO
```

```
1119    \else
1120      \expandafter\XINT_sqrt_bE
1121    \fi
1122    #1.%
1123 }%
1124 \def\XINT_sqrt_bE #1.#2#3#4%
1125 {%
1126     \XINT_sqrt_c {#3#4}#2{#1}#3#4%
1127 }%
1128 \def\XINT_sqrt_bO #1.#2#3%
1129 {%
1130     \XINT_sqrt_c #3#2{#1}#3%
1131 }%
1132 \def\XINT_sqrt_c #1#2%
1133 {%
1134     \expandafter #2%
1135     \the\numexpr \ifnum #1>\xint_c_ii
1136                  \ifnum #1>\xint_c_vi
1137                  \ifnum #1>12 \ifnum #1>20 \ifnum #1>30
1138                  \ifnum #1>42 \ifnum #1>56 \ifnum #1>72
1139                  \ifnum #1>90
1140       10\else 9\fi \else 8\fi \else 7\fi \else 6\fi \else 5\fi
1141         \else 4\fi \else 3\fi \else 2\fi \else 1\fi .%
1142 }%
1143 \def\XINT_sqrt_small_d #1.#2%
1144 {%
1145    \expandafter\XINT_sqrt_small_e
1146    \the\numexpr #1\ifcase \numexpr #2/\xint_c_ii-\xint_c_i\relax
1147                     \or 0\or 00\or 000\or 0000\fi .%
1148 }%
1149 \def\XINT_sqrt_small_e #1.#2.%
1150 {%
1151    \expandafter\XINT_sqrt_small_ea\the\numexpr #1*#1-#2.#1.%
1152 }%
1153 \def\XINT_sqrt_small_ea #1%
1154 {%
1155     \if0#1\xint_dothis\XINT_sqrt_small_ez\fi
1156     \if-#1\xint_dothis\XINT_sqrt_small_eb\fi
1157     \xint_orthat\XINT_sqrt_small_f #1%
1158 }%
1159 \def\XINT_sqrt_small_ez 0.#1.{\expandafter{\the\numexpr#1+\xint_c_i
1160          \expandafter}\expandafter{\the\numexpr #1*\xint_c_ii+\xint_c_i}}%
1161 \def\XINT_sqrt_small_eb -#1.#2.%
1162 {%
1163     \expandafter\XINT_sqrt_small_ec \the\numexpr
1164     (#1-\xint_c_i+#2)/(\xint_c_ii*#2).#1.#2.%
1165 }%
1166 \def\XINT_sqrt_small_ec #1.#2.#3.%
1167 {%
1168     \expandafter\XINT_sqrt_small_f \the\numexpr
1169       -#2+\xint_c_ii*#3*#1+#1*#1\expandafter.\the\numexpr #3+#1.%
1170 }%
```

```
\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\Z
          \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}}%
```

4.41 \xintiiBinomial, \xintiBinomial

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.

```
\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.%
}%
\def\xintiBinomial{\romannumeral0\xintibinomial}%
\let\xintibinomial\xintiibinomial
```

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{\xintError:OutOfRangeBinomial\space 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{\xintError:OutOfRangeBinomial\space 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 1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W 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!\Z!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\Z!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!\Z!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\Z!#2%
    \R!\R!\R!\R!\R!\R!\R!\W
    \R!\R!\R!\R!\R!\R!\R!\W
    1\Z!%
}%
\def\XINT_binom_div #1!1\Z!%
{%
    \expandafter\XINT_smalldivx_a
    \the\numexpr #1/\xint_c_ii\expandafter.%
    \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\Z!0!%
   {\XINT_unsep_cuzsmall #11\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W }%
```

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!%
}%
\edef\XINT_binom_vsmallfinish 1#1!1!\Z!0!%
    {\noexpand\expandafter\space\noexpand\the\numexpr #1\relax}%
```

4.42 \xintiiPFactorial, \xintiPFactorial

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\xintiPFactorial{\romannumeral0\xintipfactorial}%
\let\xintipfactorial\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.{\xintError:OutOfRangePFac\space 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.#2.10000000!!\Z!%
    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!%
}%
```

4.43 \xintiiE

Originally was used in \xintiiexpr. Transferred from xintfrac for 1.1. Code rewritten for 1.2i.

```
\def\xintiiE {\romannumeral0\xintiie }% used in \xintMod.
\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%
}%
\def\XINT_iie_a #1.%
 {\expandafter\XINT_dsx_append\romannumeral\XINT_rep #1\endcsname 0.}%
\def\XINT_iie_neg #1.#2;{ #2}%
```

4.44 "Load xintfrac" macros

Originally was used in \xintiiexpr. Transferred from xintfrac for 1.1.

```
\catcode`! 11
\def\xintMax {\Did_you_mean_iiMax?or_load_xintfrac!}%
\def\xintMin {\Did_you_mean_iiMin?or_load_xintfrac!}%
\def\xintMaxof {\Did_you_mean_iMaxof?or_load_xintfrac!}%
\def\xintMinof {\Did_you_mean_iMinof?or_load_xintfrac!}%
\def\xintSum {\Did_you_mean_iiSum?or_load_xintfrac!}%
\def\xintPrd {\Did_you_mean_iiPrd?or_load_xintfrac!}%
\def\xintPrdExpr {\Did_you_mean_iiPrdExpr?or_load_xintfrac!}%
\def\xintSumExpr {\Did_you_mean_iiSumExpr?or_load_xintfrac!}%
\XINT_restorecatcodes_endinput%
```

5 Package xintbinhex implementation

The commenting is currently (2016/12/22) very sparse.

5.1 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@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{xintbinhex}{\numexpr not available, aborting input}%
     \aftergroup\endinput
  \else
    \ifx\x\relax   % plain-TeX, first loading of xintbinhex.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 % xintbinhex already loaded.
```

```
      \fi
    \fi
  \fi
\z%
\XINTsetupcatcodes% defined in xintkernel.sty
```

5.2 Package identification

```
\XINT_providespackage
\ProvidesPackage{xintbinhex}%
  [2016/12/22 1.2j Expandable binary and hexadecimal conversions (JFB)]%
```

5.3 Constants, etc...

1.08

```
\newcount\xint_c_ii^xv  \xint_c_ii^xv   32768
\newcount\xint_c_ii^xvi \xint_c_ii^xvi  65536
\newcount\xint_c_x^v    \xint_c_x^v    100000
\def\XINT_tmpa #1{\ifx\relax#1\else
  \expandafter\edef\csname XINT_sdth_#1\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_sdtb_#1\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_sbtd_0000\endcsname {0}%
\expandafter\def\csname XINT_sbtd_0001\endcsname {1}%
\expandafter\def\csname XINT_sbtd_0010\endcsname {2}%
\expandafter\def\csname XINT_sbtd_0011\endcsname {3}%
\expandafter\def\csname XINT_sbtd_0100\endcsname {4}%
\expandafter\def\csname XINT_sbtd_0101\endcsname {5}%
\expandafter\def\csname XINT_sbtd_0110\endcsname {6}%
\expandafter\def\csname XINT_sbtd_0111\endcsname {7}%
\expandafter\def\csname XINT_sbtd_1000\endcsname {8}%
\expandafter\def\csname XINT_sbtd_1001\endcsname {9}%
\expandafter\def\csname XINT_sbtd_1010\endcsname {10}%
\expandafter\def\csname XINT_sbtd_1011\endcsname {11}%
\expandafter\def\csname XINT_sbtd_1100\endcsname {12}%
\expandafter\def\csname XINT_sbtd_1101\endcsname {13}%
\expandafter\def\csname XINT_sbtd_1110\endcsname {14}%
\expandafter\def\csname XINT_sbtd_1111\endcsname {15}%
\expandafter\let\csname XINT_sbth_0000\expandafter\endcsname
                \csname XINT_sbtd_0000\endcsname
\expandafter\let\csname XINT_sbth_0001\expandafter\endcsname
                \csname XINT_sbtd_0001\endcsname
\expandafter\let\csname XINT_sbth_0010\expandafter\endcsname
```

```
                \csname XINT_sbtd_0010\endcsname
\expandafter\let\csname XINT_sbth_0011\expandafter\endcsname
                \csname XINT_sbtd_0011\endcsname
\expandafter\let\csname XINT_sbth_0100\expandafter\endcsname
                \csname XINT_sbtd_0100\endcsname
\expandafter\let\csname XINT_sbth_0101\expandafter\endcsname
                \csname XINT_sbtd_0101\endcsname
\expandafter\let\csname XINT_sbth_0110\expandafter\endcsname
                \csname XINT_sbtd_0110\endcsname
\expandafter\let\csname XINT_sbth_0111\expandafter\endcsname
                \csname XINT_sbtd_0111\endcsname
\expandafter\let\csname XINT_sbth_1000\expandafter\endcsname
                \csname XINT_sbtd_1000\endcsname
\expandafter\let\csname XINT_sbth_1001\expandafter\endcsname
                \csname XINT_sbtd_1001\endcsname
\expandafter\def\csname XINT_sbth_1010\endcsname {A}%
\expandafter\def\csname XINT_sbth_1011\endcsname {B}%
\expandafter\def\csname XINT_sbth_1100\endcsname {C}%
\expandafter\def\csname XINT_sbth_1101\endcsname {D}%
\expandafter\def\csname XINT_sbth_1110\endcsname {E}%
\expandafter\def\csname XINT_sbth_1111\endcsname {F}%
\expandafter\def\csname XINT_shtb_0\endcsname {0000}%
\expandafter\def\csname XINT_shtb_1\endcsname {0001}%
\expandafter\def\csname XINT_shtb_2\endcsname {0010}%
\expandafter\def\csname XINT_shtb_3\endcsname {0011}%
\expandafter\def\csname XINT_shtb_4\endcsname {0100}%
\expandafter\def\csname XINT_shtb_5\endcsname {0101}%
\expandafter\def\csname XINT_shtb_6\endcsname {0110}%
\expandafter\def\csname XINT_shtb_7\endcsname {0111}%
\expandafter\def\csname XINT_shtb_8\endcsname {1000}%
\expandafter\def\csname XINT_shtb_9\endcsname {1001}%
\def\XINT_shtb_A {1010}%
\def\XINT_shtb_B {1011}%
\def\XINT_shtb_C {1100}%
\def\XINT_shtb_D {1101}%
\def\XINT_shtb_E {1110}%
\def\XINT_shtb_F {1111}%
\def\XINT_shtb_G {}%
\def\XINT_smallhex #1%
{%
    \expandafter\XINT_smallhex_a\expandafter
    {\the\numexpr (#1+\xint_c_viii)/\xint_c_xvi-\xint_c_i}{#1}%
}%
\def\XINT_smallhex_a #1#2%
{%
    \csname XINT_sdth_#1\expandafter\expandafter\expandafter\endcsname
    \csname XINT_sdth_\the\numexpr #2-\xint_c_xvi*#1\endcsname
}%
\def\XINT_smallbin #1%
{%
    \expandafter\XINT_smallbin_a\expandafter
    {\the\numexpr (#1+\xint_c_viii)/\xint_c_xvi-\xint_c_i}{#1}%
```

```
}%
\def\XINT_smallbin_a #1#2%
{%
    \csname XINT_sdtb_#1\expandafter\expandafter\expandafter\endcsname
    \csname XINT_sdtb_\the\numexpr #2-\xint_c_xvi*#1\endcsname
}%
```

5.4 \XINT_OQ

Moved with release 1.2 from xintcore 1.1 as it is used only here. Will be probably suppressed once I review the code of xintbinhex.

```
\def\XINT_OQ #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_R #9\XINT_OQ_end_a\R\XINT_OQ {#9#8#7#6#5#4#3#2#1}%
}%
\def\XINT_OQ_end_a\R\XINT_OQ #1#2\Z
{%
    \XINT_OQ_end_b #1\Z
}%
\def\XINT_OQ_end_b #1#2#3#4#5#6#7#8%
{%
    \xint_gob_til_R
            #8\XINT_OQ_end_viii
            #7\XINT_OQ_end_vii
            #6\XINT_OQ_end_vi
            #5\XINT_OQ_end_v
            #4\XINT_OQ_end_iv
            #3\XINT_OQ_end_iii
            #2\XINT_OQ_end_ii
            \R\XINT_OQ_end_i
            \Z #2#3#4#5#6#7#8%
}%
\def\XINT_OQ_end_viii #1\Z #2#3#4#5#6#7#8#9\Z { #9}%
\def\XINT_OQ_end_vii  #1\Z #2#3#4#5#6#7#8#9\Z { #8#90000000}%
\def\XINT_OQ_end_vi   #1\Z #2#3#4#5#6#7#8#9\Z { #7#8#9000000}%
\def\XINT_OQ_end_v    #1\Z #2#3#4#5#6#7#8#9\Z { #6#7#8#900000}%
\def\XINT_OQ_end_iv   #1\Z #2#3#4#5#6#7#8#9\Z { #5#6#7#8#90000}%
\def\XINT_OQ_end_iii  #1\Z #2#3#4#5#6#7#8#9\Z { #4#5#6#7#8#9000}%
\def\XINT_OQ_end_ii   #1\Z #2#3#4#5#6#7#8#9\Z { #3#4#5#6#7#8#900}%
\def\XINT_OQ_end_i      \Z #1#2#3#4#5#6#7#8\Z { #1#2#3#4#5#6#7#80}%
```

5.5 \xintDecToHex, \xintDecToBin

1.08

```
\def\xintDecToHex {\romannumeral0\xintdectohex }%
\def\xintdectohex #1%
        {\expandafter\XINT_dth_checkin\romannumeral`&&@#1\W\W\W\W \T}%
\def\XINT_dth_checkin #1%
{%
    \xint_UDsignfork
       #1\XINT_dth_N
```

```
        -{\XINT_dth_P #1}%
     \krof
}%
\def\XINT_dth_N {\expandafter-\romannumeral0\XINT_dth_P }%
\def\XINT_dth_P {\expandafter\XINT_dth_III\romannumeral`&&@\XINT_dtbh_I {0.}}%
\def\xintDecToBin {\romannumeral0\xintdectobin }%
\def\xintdectobin #1%
        {\expandafter\XINT_dtb_checkin\romannumeral`&&@#1\W\W\W\W \T }%
\def\XINT_dtb_checkin #1%
{%
    \xint_UDsignfork
       #1\XINT_dtb_N
        -{\XINT_dtb_P #1}%
     \krof
}%
\def\XINT_dtb_N {\expandafter-\romannumeral0\XINT_dtb_P }%
\def\XINT_dtb_P {\expandafter\XINT_dtb_III\romannumeral`&&@\XINT_dtbh_I {0.}}%
\def\XINT_dtbh_I #1#2#3#4#5%
{%
    \xint_gob_til_W #5\XINT_dtbh_II_a\W\XINT_dtbh_I_a  {}{#2#3#4#5}#1\Z.%
}%
\def\XINT_dtbh_II_a\W\XINT_dtbh_I_a #1#2{\XINT_dtbh_II_b #2}%
\def\XINT_dtbh_II_b #1#2#3#4%
{%
    \xint_gob_til_W
      #1\XINT_dtbh_II_c
      #2\XINT_dtbh_II_ci
      #3\XINT_dtbh_II_cii
      \W\XINT_dtbh_II_ciii #1#2#3#4%
}%
\def\XINT_dtbh_II_c \W\XINT_dtbh_II_ci
                    \W\XINT_dtbh_II_cii
                    \W\XINT_dtbh_II_ciii \W\W\W\W {{}}%
\def\XINT_dtbh_II_ci #1\XINT_dtbh_II_ciii #2\W\W\W
   {\XINT_dtbh_II_d {}{#2}{0}}%
\def\XINT_dtbh_II_cii\W\XINT_dtbh_II_ciii #1#2\W\W
   {\XINT_dtbh_II_d {}{#1#2}{00}}%
\def\XINT_dtbh_II_ciii #1#2#3\W
   {\XINT_dtbh_II_d {}{#1#2#3}{000}}%
\def\XINT_dtbh_I_a #1#2#3.%
{%
    \xint_gob_til_Z #3\XINT_dtbh_I_z\Z
    \expandafter\XINT_dtbh_I_b\the\numexpr #2+#30000.{#1}%
}%
\def\XINT_dtbh_I_b #1.%
{%
    \expandafter\XINT_dtbh_I_c\the\numexpr
    (#1+\xint_c_ii^xv)/\xint_c_ii^xvi-\xint_c_i.#1.%
}%
\def\XINT_dtbh_I_c #1.#2.%
{%
    \expandafter\XINT_dtbh_I_d\expandafter
```

```
    {\the\numexpr #2-\xint_c_ii^xvi*#1}{#1}%
}%
\def\XINT_dtbh_I_d #1#2#3{\XINT_dtbh_I_a {#3#1.}{#2}}%
\def\XINT_dtbh_I_z\Z\expandafter\XINT_dtbh_I_b\the\numexpr #1+#2.%
{%
    \ifnum #1=\xint_c_ \expandafter\XINT_dtbh_I_end_zb\fi
    \XINT_dtbh_I_end_za {#1}%
}%
\def\XINT_dtbh_I_end_za #1#2{\XINT_dtbh_I {#2#1.}}%
\def\XINT_dtbh_I_end_zb\XINT_dtbh_I_end_za #1#2{\XINT_dtbh_I {#2}}%
\def\XINT_dtbh_II_d #1#2#3#4.%
{%
    \xint_gob_til_Z #4\XINT_dtbh_II_z\Z
    \expandafter\XINT_dtbh_II_e\the\numexpr #2+#4#3.{#1}{#3}%
}%
\def\XINT_dtbh_II_e #1.%
{%
    \expandafter\XINT_dtbh_II_f\the\numexpr
        (#1+\xint_c_ii^xv)/\xint_c_ii^xvi-\xint_c_i.#1.%
}%
\def\XINT_dtbh_II_f #1.#2.%
{%
    \expandafter\XINT_dtbh_II_g\expandafter
    {\the\numexpr #2-\xint_c_ii^xvi*#1}{#1}%
}%
\def\XINT_dtbh_II_g #1#2#3{\XINT_dtbh_II_d {#3#1.}{#2}}%
\def\XINT_dtbh_II_z\Z\expandafter\XINT_dtbh_II_e\the\numexpr #1+#2.%
{%
    \ifnum #1=\xint_c_ \expandafter\XINT_dtbh_II_end_zb\fi
    \XINT_dtbh_II_end_za {#1}%
}%
\def\XINT_dtbh_II_end_za #1#2#3{{}#2#1.\Z.}%
\def\XINT_dtbh_II_end_zb\XINT_dtbh_II_end_za #1#2#3{{}#2\Z.}%
\def\XINT_dth_III #1#2.%
{%
    \xint_gob_til_Z #2\XINT_dth_end\Z
    \expandafter\XINT_dth_III\expandafter
    {\romannumeral`&&@\XINT_dth_small #2.#1}%
}%
\def\XINT_dth_small #1.%
{%
    \expandafter\XINT_smallhex\expandafter
    {\the\numexpr (#1+\xint_c_ii^vii)/\xint_c_ii^viii-\xint_c_i\expandafter}%
    \romannumeral`&&@\expandafter\XINT_smallhex\expandafter
    {\the\numexpr
    #1-((#1+\xint_c_ii^vii)/\xint_c_ii^viii-\xint_c_i)*\xint_c_ii^viii}%
}%
\def\XINT_dth_end\Z\expandafter\XINT_dth_III\expandafter #1#2\T
{%
    \XINT_dth_end_b #1%
}%
\def\XINT_dth_end_b #1.{\XINT_dth_end_c }%
```

```
\def\XINT_dth_end_c #1{\xint_gob_til_zero #1\XINT_dth_end_d 0\space #1}%
\def\XINT_dth_end_d 0\space 0#1%
{%
    \xint_gob_til_zero #1\XINT_dth_end_e 0\space #1%
}%
\def\XINT_dth_end_e 0\space 0#1%
{%
    \xint_gob_til_zero #1\XINT_dth_end_f 0\space #1%
}%
\def\XINT_dth_end_f 0\space 0{ }%
\def\XINT_dtb_III #1#2.%
{%
    \xint_gob_til_Z #2\XINT_dtb_end\Z
    \expandafter\XINT_dtb_III\expandafter
    {\romannumeral`&&@\XINT_dtb_small #2.#1}%
}%
\def\XINT_dtb_small #1.%
{%
    \expandafter\XINT_smallbin\expandafter
    {\the\numexpr (#1+\xint_c_ii^vii)/\xint_c_ii^viii-\xint_c_i\expandafter}%
    \romannumeral`&&@\expandafter\XINT_smallbin\expandafter
    {\the\numexpr
    #1-((#1+\xint_c_ii^vii)/\xint_c_ii^viii-\xint_c_i)*\xint_c_ii^viii}%
}%
\def\XINT_dtb_end\Z\expandafter\XINT_dtb_III\expandafter #1#2\T
{%
    \XINT_dtb_end_b #1%
}%
\def\XINT_dtb_end_b #1.{\XINT_dtb_end_c }%
\def\XINT_dtb_end_c #1#2#3#4#5#6#7#8%
{%
    \expandafter\XINT_dtb_end_d\the\numexpr #1#2#3#4#5#6#7#8\relax
}%
\edef\XINT_dtb_end_d #1#2#3#4#5#6#7#8#9%
{%
    \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7#8#9\relax
}%
```

5.6 \xintHexToDec

1.08

```
\def\xintHexToDec {\romannumeral0\xinthextodec }%
\def\xinthextodec #1%
        {\expandafter\XINT_htd_checkin\romannumeral`&&@#1\W\W\W\W \T }%
\def\XINT_htd_checkin #1%
{%
    \xint_UDsignfork
       #1\XINT_htd_neg
        -{\XINT_htd_I {0000}#1}%
     \krof
}%
\def\XINT_htd_neg {\expandafter-\romannumeral0\XINT_htd_I {0000}}%
```

```
\def\XINT_htd_I #1#2#3#4#5%
{%
    \xint_gob_til_W #5\XINT_htd_II_a\W
    \XINT_htd_I_a  {}{"#2#3#4#5}#1\Z\Z\Z\Z
}%
\def\XINT_htd_II_a \W\XINT_htd_I_a #1#2{\XINT_htd_II_b #2}%
\def\XINT_htd_II_b "#1#2#3#4%
{%
    \xint_gob_til_W
      #1\XINT_htd_II_c
      #2\XINT_htd_II_ci
      #3\XINT_htd_II_cii
      \W\XINT_htd_II_ciii #1#2#3#4%
}%
\def\XINT_htd_II_c \W\XINT_htd_II_ci
                   \W\XINT_htd_II_cii
                   \W\XINT_htd_II_ciii \W\W\W\W #1\Z\Z\Z\Z\T
{%
    \expandafter\xint_cleanupzeros_andstop
    \romannumeral0\XINT_rord_main {}#1%
      \xint_relax
        \xint_bye\xint_bye\xint_bye\xint_bye
        \xint_bye\xint_bye\xint_bye\xint_bye
      \xint_relax
}%
\def\XINT_htd_II_ci #1\XINT_htd_II_ciii
                      #2\W\W\W {\XINT_htd_II_d {}{"#2}{\xint_c_xvi}}%
\def\XINT_htd_II_cii\W\XINT_htd_II_ciii
                      #1#2\W\W {\XINT_htd_II_d {}{"#1#2}{\xint_c_ii^viii}}%
\def\XINT_htd_II_ciii #1#2#3\W {\XINT_htd_II_d {}{"#1#2#3}{\xint_c_ii^xii}}%
\def\XINT_htd_I_a #1#2#3#4#5#6%
{%
    \xint_gob_til_Z #3\XINT_htd_I_end_a\Z
    \expandafter\XINT_htd_I_b\the\numexpr
    #2+\xint_c_ii^xvi*#6#5#4#3+\xint_c_x^ix\relax {#1}%
}%
\def\XINT_htd_I_b 1#1#2#3#4#5#6#7#8#9{\XINT_htd_I_c {#1#2#3#4#5}{#9#8#7#6}}%
\def\XINT_htd_I_c #1#2#3{\XINT_htd_I_a {#3#2}{#1}}%
\def\XINT_htd_I_end_a\Z\expandafter\XINT_htd_I_b\the\numexpr #1+#2\relax
{%
    \expandafter\XINT_htd_I_end_b\the\numexpr \xint_c_x^v+#1\relax
}%
\def\XINT_htd_I_end_b 1#1#2#3#4#5%
{%
    \xint_gob_til_zero #1\XINT_htd_I_end_bz0%
    \XINT_htd_I_end_c #1#2#3#4#5%
}%
\def\XINT_htd_I_end_c #1#2#3#4#5#6{\XINT_htd_I {#6#5#4#3#2#1000}}%
\def\XINT_htd_I_end_bz0\XINT_htd_I_end_c 0#1#2#3#4%
{%
    \xint_gob_til_zeros_iv #1#2#3#4\XINT_htd_I_end_bzz 0000%
    \XINT_htd_I_end_D {#4#3#2#1}%
```

```
}%
\def\XINT_htd_I_end_D  #1#2{\XINT_htd_I {#2#1}}%
\def\XINT_htd_I_end_bzz 0000\XINT_htd_I_end_D #1{\XINT_htd_I }%
\def\XINT_htd_II_d #1#2#3#4#5#6#7%
{%
    \xint_gob_til_Z #4\XINT_htd_II_end_a\Z
    \expandafter\XINT_htd_II_e\the\numexpr
    #2+#3*#7#6#5#4+\xint_c_x^viii\relax {#1}{#3}%
}%
\def\XINT_htd_II_e 1#1#2#3#4#5#6#7#8{\XINT_htd_II_f {#1#2#3#4}{#5#6#7#8}}%
\def\XINT_htd_II_f #1#2#3{\XINT_htd_II_d {#2#3}{#1}}%
\def\XINT_htd_II_end_a\Z\expandafter\XINT_htd_II_e
    \the\numexpr #1+#2\relax #3#4\T
{%
    \XINT_htd_II_end_b #1#3%
}%
\edef\XINT_htd_II_end_b #1#2#3#4#5#6#7#8%
{%
    \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7#8\relax
}%
```

5.7 \xintBinToDec

1.08

```
\def\xintBinToDec {\romannumeral0\xintbintodec }%
\def\xintbintodec #1{\expandafter\XINT_btd_checkin
                     \romannumeral`&&@#1\W\W\W\W\W\W\W\W \T }%
\def\XINT_btd_checkin #1%
{%
    \xint_UDsignfork
       #1\XINT_btd_neg
        -{\XINT_btd_I {000000}#1}%
     \krof
}%
\def\XINT_btd_neg {\expandafter-\romannumeral0\XINT_btd_I {000000}}%
\def\XINT_btd_I #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_W #9\XINT_btd_II_a {#2#3#4#5#6#7#8#9}\W
    \XINT_btd_I_a {}{\csname XINT_sbtd_#2#3#4#5\endcsname*\xint_c_xvi+%
                     \csname XINT_sbtd_#6#7#8#9\endcsname}%
    #1\Z\Z\Z\Z\Z\Z
}%
\def\XINT_btd_II_a #1\W\XINT_btd_I_a #2#3{\XINT_btd_II_b #1}%
\def\XINT_btd_II_b #1#2#3#4#5#6#7#8%
{%
    \xint_gob_til_W
      #1\XINT_btd_II_c
      #2\XINT_btd_II_ci
      #3\XINT_btd_II_cii
      #4\XINT_btd_II_ciii
      #5\XINT_btd_II_civ
      #6\XINT_btd_II_cv
```

```
      #7\XINT_btd_II_cvi
      \W\XINT_btd_II_cvii #1#2#3#4#5#6#7#8%
}%
\def\XINT_btd_II_c #1\XINT_btd_II_cvii \W\W\W\W\W\W\W\W #2\Z\Z\Z\Z\Z\Z\T
{%
    \expandafter\XINT_btd_II_c_end
    \romannumeral0\XINT_rord_main {}#2%
      \xint_relax
        \xint_bye\xint_bye\xint_bye\xint_bye
        \xint_bye\xint_bye\xint_bye\xint_bye
      \xint_relax
}%
\edef\XINT_btd_II_c_end #1#2#3#4#5#6%
{%
    \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6\relax
}%
\def\XINT_btd_II_ci  #1\XINT_btd_II_cvii #2\W\W\W\W\W\W\W
   {\XINT_btd_II_d {}{#2}{\xint_c_ii }}%
\def\XINT_btd_II_cii #1\XINT_btd_II_cvii #2\W\W\W\W\W\W
   {\XINT_btd_II_d {}{\csname XINT_sbtd_00#2\endcsname }{\xint_c_iv }}%
\def\XINT_btd_II_ciii #1\XINT_btd_II_cvii #2\W\W\W\W\W
   {\XINT_btd_II_d {}{\csname XINT_sbtd_0#2\endcsname }{\xint_c_viii }}%
\def\XINT_btd_II_civ #1\XINT_btd_II_cvii #2\W\W\W\W
   {\XINT_btd_II_d {}{\csname XINT_sbtd_#2\endcsname}{\xint_c_xvi }}%
\def\XINT_btd_II_cv #1\XINT_btd_II_cvii #2#3#4#5#6\W\W\W
{%
    \XINT_btd_II_d {}{\csname XINT_sbtd_#2#3#4#5\endcsname*\xint_c_ii+%
                           #6}{\xint_c_ii^v }%
}%
\def\XINT_btd_II_cvi #1\XINT_btd_II_cvii #2#3#4#5#6#7\W\W
{%
    \XINT_btd_II_d {}{\csname XINT_sbtd_#2#3#4#5\endcsname*\xint_c_iv+%
                         \csname XINT_sbtd_00#6#7\endcsname}{\xint_c_ii^vi }%
}%
\def\XINT_btd_II_cvii #1#2#3#4#5#6#7\W
{%
    \XINT_btd_II_d {}{\csname XINT_sbtd_#1#2#3#4\endcsname*\xint_c_viii+%
                         \csname XINT_sbtd_0#5#6#7\endcsname}{\xint_c_ii^vii }%
}%
\def\XINT_btd_II_d #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_Z #4\XINT_btd_II_end_a\Z
    \expandafter\XINT_btd_II_e\the\numexpr
    #2+(\xint_c_x^ix+#3*#9#8#7#6#5#4)\relax {#1}{#3}%
}%
\def\XINT_btd_II_e 1#1#2#3#4#5#6#7#8#9{\XINT_btd_II_f {#1#2#3}{#4#5#6#7#8#9}}%
\def\XINT_btd_II_f #1#2#3{\XINT_btd_II_d {#2#3}{#1}}%
\def\XINT_btd_II_end_a\Z\expandafter\XINT_btd_II_e
    \the\numexpr #1+(#2\relax #3#4\T
{%
    \XINT_btd_II_end_b #1#3%
}%
```

```
\edef\XINT_btd_II_end_b #1#2#3#4#5#6#7#8#9%
{%
    \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7#8#9\relax
}%
\def\XINT_btd_I_a #1#2#3#4#5#6#7#8%
{%
    \xint_gob_til_Z #3\XINT_btd_I_end_a\Z
    \expandafter\XINT_btd_I_b\the\numexpr
    #2+\xint_c_ii^viii*#8#7#6#5#4#3+\xint_c_x^ix\relax {#1}%
}%
\def\XINT_btd_I_b 1#1#2#3#4#5#6#7#8#9{\XINT_btd_I_c {#1#2#3}{#9#8#7#6#5#4}}%
\def\XINT_btd_I_c #1#2#3{\XINT_btd_I_a {#3#2}{#1}}%
\def\XINT_btd_I_end_a\Z\expandafter\XINT_btd_I_b
    \the\numexpr #1+\xint_c_ii^viii #2\relax
{%
    \expandafter\XINT_btd_I_end_b\the\numexpr 1000+#1\relax
}%
\def\XINT_btd_I_end_b 1#1#2#3%
{%
    \xint_gob_til_zeros_iii #1#2#3\XINT_btd_I_end_bz 000%
    \XINT_btd_I_end_c #1#2#3%
}%
\def\XINT_btd_I_end_c #1#2#3#4{\XINT_btd_I {#4#3#2#1000}}%
\def\XINT_btd_I_end_bz 000\XINT_btd_I_end_c 000{\XINT_btd_I }%
```

5.8 \xintBinToHex

1.08

```
\def\xintBinToHex {\romannumeral0\xintbintohex }%
\def\xintbintohex #1%
{%
    \expandafter\XINT_bth_checkin
                      \romannumeral0\expandafter\XINT_num_loop
                      \romannumeral`&&@#1\xint_relax\xint_relax
                                         \xint_relax\xint_relax
                      \xint_relax\xint_relax\xint_relax\xint_relax\Z
    \R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W\W
}%
\def\XINT_bth_checkin #1%
{%
    \xint_UDsignfork
       #1\XINT_bth_N
        -{\XINT_bth_P #1}%
     \krof
}%
\def\XINT_bth_N {\expandafter-\romannumeral0\XINT_bth_P }%
\def\XINT_bth_P {\expandafter\XINT_bth_I\expandafter{\expandafter}%
                 \romannumeral0\XINT_OQ {}}%
\def\XINT_bth_I #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_W #9\XINT_bth_end_a\W
    \expandafter\expandafter\expandafter
```

```
    \XINT_bth_I
    \expandafter\expandafter\expandafter
    {\csname XINT_sbth_#9#8#7#6\expandafter\expandafter\expandafter\endcsname
     \csname XINT_sbth_#5#4#3#2\endcsname #1}%
}%
\def\XINT_bth_end_a\W \expandafter\expandafter\expandafter
    \XINT_bth_I       \expandafter\expandafter\expandafter #1%
{%
    \XINT_bth_end_b #1%
}%
\def\XINT_bth_end_b  #1\endcsname #2\endcsname #3%
{%
    \xint_gob_til_zero #3\XINT_bth_end_z 0\space #3%
}%
\def\XINT_bth_end_z0\space 0{ }%
```

5.9 \xintHexToBin

1.08

```
\def\xintHexToBin {\romannumeral0\xinthextobin }%
\def\xinthextobin #1%
{%
    \expandafter\XINT_htb_checkin\romannumeral`&&@#1GGGGGGGG\T
}%
\def\XINT_htb_checkin #1%
{%
    \xint_UDsignfork
       #1\XINT_htb_N
        -{\XINT_htb_P #1}%
     \krof
}%
\def\XINT_htb_N {\expandafter-\romannumeral0\XINT_htb_P }%
\def\XINT_htb_P {\XINT_htb_I_a {}}%
\def\XINT_htb_I_a #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_G #9\XINT_htb_II_a G%
    \expandafter\expandafter\expandafter
    \XINT_htb_I_b
    \expandafter\expandafter\expandafter
    {\csname XINT_shtb_#2\expandafter\expandafter\expandafter\endcsname
     \csname XINT_shtb_#3\expandafter\expandafter\expandafter\endcsname
     \csname XINT_shtb_#4\expandafter\expandafter\expandafter\endcsname
     \csname XINT_shtb_#5\expandafter\expandafter\expandafter\endcsname
     \csname XINT_shtb_#6\expandafter\expandafter\expandafter\endcsname
     \csname XINT_shtb_#7\expandafter\expandafter\expandafter\endcsname
     \csname XINT_shtb_#8\expandafter\expandafter\expandafter\endcsname
     \csname XINT_shtb_#9\endcsname }{#1}%
}%
\def\XINT_htb_I_b #1#2{\XINT_htb_I_a {#2#1}}%
\def\XINT_htb_II_a G\expandafter\expandafter\expandafter\XINT_htb_I_b
{%
    \expandafter\expandafter\expandafter \XINT_htb_II_b
```

```
}%
\def\XINT_htb_II_b #1#2#3\T
{%
    \XINT_num_loop #2#1%
    \xint_relax\xint_relax\xint_relax\xint_relax
    \xint_relax\xint_relax\xint_relax\xint_relax\Z
}%
```

5.10 \xintCHexToBin

1.08

```
\def\xintCHexToBin {\romannumeral0\xintchextobin }%
\def\xintchextobin #1%
{%
    \expandafter\XINT_chtb_checkin\romannumeral`&&@#1%
    \R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W\W
}%
\def\XINT_chtb_checkin #1%
{%
    \xint_UDsignfork
       #1\XINT_chtb_N
        -{\XINT_chtb_P #1}%
     \krof
}%
\def\XINT_chtb_N {\expandafter-\romannumeral0\XINT_chtb_P }%
\def\XINT_chtb_P {\expandafter\XINT_chtb_I\expandafter{\expandafter}%
                  \romannumeral0\XINT_OQ {}}%
\def\XINT_chtb_I #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_W #9\XINT_chtb_end_a\W
    \expandafter\expandafter\expandafter
    \XINT_chtb_I
    \expandafter\expandafter\expandafter
    {\csname XINT_shtb_#9\expandafter\expandafter\expandafter\endcsname
     \csname XINT_shtb_#8\expandafter\expandafter\expandafter\endcsname
     \csname XINT_shtb_#7\expandafter\expandafter\expandafter\endcsname
     \csname XINT_shtb_#6\expandafter\expandafter\expandafter\endcsname
     \csname XINT_shtb_#5\expandafter\expandafter\expandafter\endcsname
     \csname XINT_shtb_#4\expandafter\expandafter\expandafter\endcsname
     \csname XINT_shtb_#3\expandafter\expandafter\expandafter\endcsname
     \csname XINT_shtb_#2\endcsname
     #1}%
}%
\def\XINT_chtb_end_a\W\expandafter\expandafter\expandafter
    \XINT_chtb_I\expandafter\expandafter\expandafter #1%
{%
    \XINT_chtb_end_b #1%
    \xint_relax\xint_relax\xint_relax\xint_relax
    \xint_relax\xint_relax\xint_relax\xint_relax\Z
}%
\def\XINT_chtb_end_b #1\W#2\W#3\W#4\W#5\W#6\W#7\W#8\W\endcsname
{%
```

```
    \XINT_num_loop
}%
\XINT_restorecatcodes_endinput%
```

6 Package xintgcd implementation

The commenting is currently (2016/12/22) 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.

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@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
```

6.2 Package identification

```
\XINT_providespackage
\ProvidesPackage{xintgcd}%
  [2016/12/22 1.2j Euclide algorithm with xint package (JFB)]%
```

6.3 \xintGCD, \xintiiGCD

```
\def\xintGCD {\romannumeral0\xintgcd }%
\def\xintgcd #1%
{%
    \expandafter\XINT_gcd\expandafter{\romannumeral0\xintiabs {#1}}%
}%
\def\XINT_gcd #1#2%
{%
    \expandafter\XINT_gcd_fork\romannumeral0\xintiabs {#2}\Z #1\Z
}%
\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

```
\def\XINT_gcd_loop #1#2%
{%
    \expandafter\expandafter\expandafter
        \XINT_gcd_CheckRem
    \expandafter\xint_secondoftwo
    \romannumeral0\XINT_div_prepare {#1}{#2}\Z
    {#1}%
}%
```

6.4 \xintLCM, \xintiiLCM

```
\def\xintLCM {\romannumeral0\xintlcm}%
\def\xintlcm #1%
{%
    \expandafter\XINT_lcm\expandafter{\romannumeral0\xintiabs {#1}}%
}%
\def\XINT_lcm #1#2%
{%
    \expandafter\XINT_lcm_fork\romannumeral0\xintiabs {#2}\Z #1\Z
}%
\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}}}%
```

6.5 \xintBezout

```
\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

\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
          #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 }%
    \krof
    {#2}{#4}#1#3{#3#4}{#1#2}% #1#2=B, #3#4=A
}%
\edef\XINT_bezout_botharezero #1#2#3#4#5#6%
{%
    \noexpand\xintError:NoBezoutForZeros\space {0}{0}{0}{0}{0}%
}%

attention première entrée doit être ici (-1)^n donc 1
#4#2 = 0 = A, B = #3#1

\def\XINT_bezout_firstiszero #1#2#3#4#5#6%
{%
    \xint_UDsignfork
      #3{ {0}{#3#1}{0}{1}{#1}}%
       -{ {0}{#3#1}{0}{-1}{#1}}%
    \krof
}%

#4#2 = A, B = #3#1 = 0

\def\XINT_bezout_secondiszero #1#2#3#4#5#6%
{%
    \xint_UDsignfork
       #4{ {#4#2}{0}{-1}{0}{#2}}%
        -{ {#4#2}{0}{1}{0}{#2}}%
    \krof
}%

#4#2= A < 0, #3#1 = B < 0

\def\XINT_bezout_minusminus #1#2#3#4%
{%
    \expandafter\XINT_bezout_mm_post
    \romannumeral0\XINT_bezout_loop_a 1{#1}{#2}1001%
}%
\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\XINT_bezout_mm_postc\expandafter {#2}{#1}%
}%
\edef\XINT_bezout_mm_postc #1#2#3#4#5%
{%
    \space {#4}{#5}{#1}{#2}{#3}%
}%
```

minusplus #4#2= A > 0, B < 0

```
\def\XINT_bezout_minusplus #1#2#3#4%
{%
    \expandafter\XINT_bezout_mp_post
    \romannumeral0\XINT_bezout_loop_a 1{#1}{#4#2}1001%
}%
\def\XINT_bezout_mp_post #1#2%
{%
    \expandafter\XINT_bezout_mp_postb\expandafter
      {\romannumeral0\xintiiopp {#2}}{#1}%
}%
\edef\XINT_bezout_mp_postb #1#2#3#4#5%
{%
    \space {#4}{#5}{#2}{#1}{#3}%
}%
```

plusminus A < 0, B > 0

```
\def\XINT_bezout_plusminus #1#2#3#4%
{%
    \expandafter\XINT_bezout_pm_post
    \romannumeral0\XINT_bezout_loop_a 1{#3#1}{#2}1001%
}%
\def\XINT_bezout_pm_post #1%
{%
    \expandafter \XINT_bezout_pm_postb \expandafter
        {\romannumeral0\xintiiopp{#1}}%
}%
\edef\XINT_bezout_pm_postb #1#2#3#4#5%
{%
    \space {#4}{#5}{#1}{#2}{#3}%
}%
```

plusplus

```
\def\XINT_bezout_plusplus #1#2#3#4%
{%
    \expandafter\XINT_bezout_pp_post
    \romannumeral0\XINT_bezout_loop_a 1{#3#1}{#4#2}1001%
}%
```

la parité (-1)^N est en #1, et on la jette ici.

```
\edef\XINT_bezout_pp_post #1#2#3#4#5%
{%
    \space {#4}{#5}{#1}{#2}{#3}%
}%
```

```
n = 0: 1BAalpha(0)beta(0)alpha(-1)beta(-1)
n général: {(-1)^n}{r(n-1)}{r(n-2)}{alpha(n-1)}{beta(n-1)}{alpha(n-2)}{beta(n-2)}
#2 = B, #3 = A
```

```
\def\XINT_bezout_loop_a #1#2#3%
{%
    \expandafter\XINT_bezout_loop_b
    \expandafter{\the\numexpr -#1\expandafter }%
    \romannumeral0\XINT_div_prepare {#2}{#3}{#2}%
}%
```

```
Le q(n) a ici une existence éphémère, dans le version Bezout Algorithm il faudra le conserver. On
voudra à la fin {{q(n)}{r(n)}{alpha(n)}{beta(n)}}. De plus ce n'est plus (-1)^n que l'on veut mais
n. (ou dans un autre ordre)
{-(-1)^n}{q(n)}{r(n)}{r(n-1)}{alpha(n-1)}{beta(n-1)}{alpha(n-2)}{beta(n-2)}
```

```
\def\XINT_bezout_loop_b #1#2#3#4#5#6#7#8%
{%
    \expandafter \XINT_bezout_loop_c \expandafter
        {\romannumeral0\xintiiadd{\XINT_mul_fork #5\Z #2\Z}{#7}}%
        {\romannumeral0\xintiiadd{\XINT_mul_fork #6\Z #2\Z}{#8}}%
    {#1}{#3}{#4}{#5}{#6}%
}%
```

```
{alpha(n)}{->beta(n)}{-(-1)^n}{r(n)}{r(n-1)}{alpha(n-1)}{beta(n-1)}
```

```
\def\XINT_bezout_loop_c #1#2%
{%
    \expandafter \XINT_bezout_loop_d \expandafter
        {#2}{#1}%
}%
```

```
{beta(n)}{alpha(n)}{(-1)^(n+1)}{r(n)}{r(n-1)}{alpha(n-1)}{beta(n-1)}
```

```
\def\XINT_bezout_loop_d #1#2#3#4#5%
{%
    \XINT_bezout_loop_e #4\Z {#3}{#5}{#2}{#1}%
}%
```

```
r(n)\Z {(-1)^(n+1)}{r(n-1)}{alpha(n)}{beta(n)}{alpha(n-1)}{beta(n-1)}
```

```
\def\XINT_bezout_loop_e #1#2\Z
{%
    \xint_gob_til_zero #1\xint_bezout_loop_exit0\XINT_bezout_loop_f
    {#1#2}%
}%
```

```
{r(n)}{(-1)^(n+1)}{r(n-1)}{alpha(n)}{beta(n)}{alpha(n-1)}{beta(n-1)}
```

```
\def\XINT_bezout_loop_f #1#2%
{%
    \XINT_bezout_loop_a {#2}{#1}%
}%
```

```
{(-1)^(n+1)}{r(n)}{r(n-1)}{alpha(n)}{beta(n)}{alpha(n-1)}{beta(n-1)} et itération
```

```
\def\xint_bezout_loop_exit0\XINT_bezout_loop_f #1#2%
{%
    \ifcase #2
    \or  \expandafter\XINT_bezout_exiteven
    \else\expandafter\XINT_bezout_exitodd
    \fi
}%
\edef\XINT_bezout_exiteven #1#2#3#4#5%
{%
    \space {#5}{#4}{#1}%
}%
\edef\XINT_bezout_exitodd #1#2#3#4#5%
{%
    \space {-#5}{-#4}{#1}%
}%
```

6.6 \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
```

```
\def\xintEuclideAlgorithm {\romannumeral0\xinteuclidealgorithm }%
\def\xinteuclidealgorithm #1%
{%
    \expandafter \XINT_euc \expandafter{\romannumeral0\xintiabs {#1}}%
}%
\def\XINT_euc #1#2%
{%
    \expandafter\XINT_euc_fork \romannumeral0\xintiabs {#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
    \expandafter {\the\numexpr #1+1\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_
    \romannumeral0%
    \XINT_rord_main {}#4{{#1}{#3}}%
    \xint_relax
      \xint_bye\xint_bye\xint_bye\xint_bye
      \xint_bye\xint_bye\xint_bye\xint_bye
    \xint_relax
}%
\edef\xint_euc_end_ #1#2#3%
{%
    \space {#1}{#3}{#2}%
}%
```

6.7 \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\xintiabs {#1}}%
}%
\def\XINT_bezalg #1#2%
```

```
{%
    \expandafter\XINT_bezalg_fork \romannumeral0\xintiabs {#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)}{al
division de #3 par #2

```
\def\XINT_bezalg_a #1#2#3%
{%
    \expandafter\XINT_bezalg_b
    \expandafter {\the\numexpr #1+1\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_
    \romannumeral0%
    \XINT_rord_main {}#8{{#1}{#3}}%
    \xint_relax
      \xint_bye\xint_bye\xint_bye\xint_bye
      \xint_bye\xint_bye\xint_bye\xint_bye
    \xint_relax
}%
```

```
{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}
```

```
\edef\xint_bezalg_end_ #1#2#3#4%
{%
    \space {#1}{#3}{0}{1}{#2}{#4}{1}{0}%
}%
```

6.8 \xintGCDof

```
\def\xintGCDof      {\romannumeral0\xintgcdof }%
\def\xintgcdof    #1{\expandafter\XINT_gcdof_a\romannumeral`&&@#1\relax }%
\def\XINT_gcdof_a #1{\expandafter\XINT_gcdof_b\romannumeral`&&@#1\Z }%
\def\XINT_gcdof_b #1\Z #2{\expandafter\XINT_gcdof_c\romannumeral`&&@#2\Z {#1}\Z}%
\def\XINT_gcdof_c #1{\xint_gob_til_relax #1\XINT_gcdof_e\relax\XINT_gcdof_d #1}%
\def\XINT_gcdof_d #1\Z {\expandafter\XINT_gcdof_b\romannumeral0\xintgcd {#1}}%
\def\XINT_gcdof_e #1\Z #2\Z { #2}%
```

6.9 \xintLCMof

```
New with 1.09a
```

```
\def\xintLCMof      {\romannumeral0\xintlcmof }%
\def\xintlcmof    #1{\expandafter\XINT_lcmof_a\romannumeral`&&@#1\relax }%
\def\XINT_lcmof_a #1{\expandafter\XINT_lcmof_b\romannumeral`&&@#1\Z }%
\def\XINT_lcmof_b #1\Z #2{\expandafter\XINT_lcmof_c\romannumeral`&&@#2\Z {#1}\Z}%
\def\XINT_lcmof_c #1{\xint_gob_til_relax #1\XINT_lcmof_e\relax\XINT_lcmof_d #1}%
\def\XINT_lcmof_d #1\Z {\expandafter\XINT_lcmof_b\romannumeral0\xintlcm {#1}}%
\def\XINT_lcmof_e #1\Z #2\Z { #2}%
```

6.10 \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
}%
```

6.11 \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%
```

7 Package xintfrac implementation

The commenting is currently (2016/12/22) very sparse.

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@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
```

7.2 Package identification

```
\XINT_providespackage
\ProvidesPackage{xintfrac}%
  [2016/12/22 1.2j Expandable operations on fractions (JFB)]%
```

7.3 \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
```

```
51          \else\expandafter\xint_firstofthree
52     \fi
53 }%
```

7.4 \xintLen

The used formula is disputable, the idea is that A/1 and A should have same length. Venerable code rewritten for 1.2i.

```
54 \def\xintLen {\romannumeral0\xintlen }%
55 \def\xintlen #1%
56 {%
57     \expandafter\XINT_flen\romannumeral0\XINT_infrac {#1}%
58 }%
59 \def\XINT_flen #1#2#3%
60 {%
61     \expandafter\space
62     \the\numexpr \XINT_abs#1+\XINT_length_loop
63     #2#3\xint_relax\xint_relax\xint_relax\xint_relax
64       \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
65       \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
66       \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye-\xint_c_i
67     \relax
68 }%
```

7.5 \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.

```
69 \def\XINT_outfrac #1#2#3%
70 {%
71     \ifcase\XINT_cntSgn #3\Z
72         \expandafter \XINT_outfrac_divisionbyzero
73     \or
74         \expandafter \XINT_outfrac_P
75     \else
76         \expandafter \XINT_outfrac_N
77     \fi
78     {#2}{#3}[#1]%
79 }%
80 \def\XINT_outfrac_divisionbyzero #1#2{\xintError:DivisionByZero\space #1/0}%
81 \edef\XINT_outfrac_P #1#2%
82 {%
83     \noexpand\if0\noexpand\XINT_Sgn #1\noexpand\Z
84         \noexpand\expandafter\noexpand\XINT_outfrac_Zero
85     \noexpand\fi
86     \space #1/#2%
87 }%
88 \def\XINT_outfrac_Zero #1[#2]{ 0/1[0]}%
89 \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}%
}%
```

7.6 \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.

```
\def\XINT_inFrac {\romannumeral0\XINT_infrac }%
\def\XINT_infrac #1%
```

```
{%
    \expandafter\XINT_infrac_fork\romannumeral`&&@#1/\XINT_W[\XINT_W\XINT_T
}%
\def\XINT_infrac_fork #1[#2%
{%
    \xint_UDXINTWfork
      #2\XINT_frac_gen
      \XINT_W\XINT_infrac_res_a % 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%
}%
\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
     \XINT_W\XINT_infrac_res_cb
    \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_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_W[\XINT_W\XINT_T
   {\expandafter{\the\numexpr #3}{#2}{#1}}%
```

7.7 \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 \fexpan sion 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
      \XINT_W\XINT_frac_gen_B
    \krof
    #1/#2%
}%
```

```
\def\XINT_frac_gen_A #1/\XINT_W [\XINT_W {\XINT_frac_gen_C 0~1!#1ee.\XINT_W }%
\def\XINT_frac_gen_B #1/#2/\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_relax\xint_relax\xint_relax\xint_relax
      \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
      \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\XINT_num_loop
     #2\xint_relax\xint_relax\xint_relax\xint_relax
      \xint_relax\xint_relax\xint_relax\xint_relax\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_relax\xint_relax\xint_relax\xint_relax
      \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
      \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\XINT_num_loop
     #3\xint_relax\xint_relax\xint_relax\xint_relax
      \xint_relax\xint_relax\xint_relax\xint_relax\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\XINT_num_loop
     #2\xint_relax\xint_relax\xint_relax\xint_relax
      \xint_relax\xint_relax\xint_relax\xint_relax\Z~#1~%
}%
\def\XINT_frac_gen_Gdivbyzero_a #1~#2~%
{%
    \xintError:DivisionByZero {#2}{#1}{0}%
}%
\def\XINT_frac_gen_G #1#2#3~#4~#5~%
{%
    \expandafter\XINT_frac_gen_Ga
    \romannumeral0\XINT_num_loop
      #1#5\xint_relax\xint_relax\xint_relax\xint_relax
      \xint_relax\xint_relax\xint_relax\xint_relax\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}}%
```

7.8 \XINT_factortens, \XINT_cuz_cnt

Old routines.

```
\def\XINT_factortens #1%
{%
    \expandafter\XINT_cuz_cnt_loop\expandafter
    {\expandafter}\romannumeral0\XINT_rord_main {}#1%
      \xint_relax
        \xint_bye\xint_bye\xint_bye\xint_bye
```

```
        \xint_bye\xint_bye\xint_bye\xint_bye
      \xint_relax
    \R\R\R\R\R\R\R\Z
}%
\def\XINT_cuz_cnt #1%
{%
    \XINT_cuz_cnt_loop {}#1\R\R\R\R\R\R\R\Z
}%
\def\XINT_cuz_cnt_loop #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_R #9\XINT_cuz_cnt_toofara \R
    \expandafter\XINT_cuz_cnt_checka\expandafter
    {\the\numexpr #1+8\relax}{#2#3#4#5#6#7#8#9}%
}%
\def\XINT_cuz_cnt_toofara\R
    \expandafter\XINT_cuz_cnt_checka\expandafter #1#2%
{%
    \XINT_cuz_cnt_toofarb {#1}#2%
}%
\def\XINT_cuz_cnt_toofarb #1#2\Z {\XINT_cuz_cnt_toofarc #2\Z {#1}}%
\def\XINT_cuz_cnt_toofarc #1#2#3#4#5#6#7#8%
{%
    \xint_gob_til_R #2\XINT_cuz_cnt_toofard 7%
            #3\XINT_cuz_cnt_toofard 6%
            #4\XINT_cuz_cnt_toofard 5%
            #5\XINT_cuz_cnt_toofard 4%
            #6\XINT_cuz_cnt_toofard 3%
            #7\XINT_cuz_cnt_toofard 2%
            #8\XINT_cuz_cnt_toofard 1%
            \Z #1#2#3#4#5#6#7#8%
}%
\def\XINT_cuz_cnt_toofard #1#2\Z #3\R #4\Z #5%
{%
    \expandafter\XINT_cuz_cnt_toofare
    \the\numexpr #3\relax \R\R\R\R\R\R\R\Z
    {\the\numexpr #5-#1\relax}\R\Z
}%
\def\XINT_cuz_cnt_toofare #1#2#3#4#5#6#7#8%
{%
    \xint_gob_til_R #2\XINT_cuz_cnt_stopc 1%
            #3\XINT_cuz_cnt_stopc 2%
            #4\XINT_cuz_cnt_stopc 3%
            #5\XINT_cuz_cnt_stopc 4%
            #6\XINT_cuz_cnt_stopc 5%
            #7\XINT_cuz_cnt_stopc 6%
            #8\XINT_cuz_cnt_stopc 7%
            \Z #1#2#3#4#5#6#7#8%
}%
\def\XINT_cuz_cnt_checka #1#2%
{%
    \expandafter\XINT_cuz_cnt_checkb\the\numexpr #2\relax \Z {#1}%
}%
```

```
\def\XINT_cuz_cnt_checkb #1%
{%
    \xint_gob_til_zero #1\expandafter\XINT_cuz_cnt_loop\xint_gob_til_Z
    0\XINT_cuz_cnt_stopa #1%
}%
\def\XINT_cuz_cnt_stopa #1\Z
{%
    \XINT_cuz_cnt_stopb #1\R\R\R\R\R\R\R\Z %
}%
\def\XINT_cuz_cnt_stopb #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_R #2\XINT_cuz_cnt_stopc 1%
            #3\XINT_cuz_cnt_stopc 2%
            #4\XINT_cuz_cnt_stopc 3%
            #5\XINT_cuz_cnt_stopc 4%
            #6\XINT_cuz_cnt_stopc 5%
            #7\XINT_cuz_cnt_stopc 6%
            #8\XINT_cuz_cnt_stopc 7%
            #9\XINT_cuz_cnt_stopc 8%
            \Z #1#2#3#4#5#6#7#8#9%
}%
\def\XINT_cuz_cnt_stopc #1#2\Z #3\R #4\Z #5%
{%
    \expandafter\XINT_cuz_cnt_stopd\expandafter
    {\the\numexpr #5-#1}#3%
}%
\def\XINT_cuz_cnt_stopd #1#2\R #3\Z
{%
    \expandafter\space\expandafter
     {\romannumeral0\XINT_rord_main {}#2%
      \xint_relax
        \xint_bye\xint_bye\xint_bye\xint_bye
        \xint_bye\xint_bye\xint_bye\xint_bye
      \xint_relax }{#1}%
}%
```

7.9 \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]}%
```

7.10 \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}%
}%
```

7.11 \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}%
```

7.12 \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}}%
```

7.13 \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}}}}%
```

7.14 \xintNumerator

```
\def\xintNumerator {\romannumeral0\xintnumerator }%
\def\xintnumerator
{%
    \expandafter\XINT_numer\romannumeral0\XINT_infrac
}%
\def\XINT_numer #1%
{%
    \ifcase\XINT_cntSgn #1\Z
      \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}%
```

7.15 \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;}%
```

7.16 \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 }%
```

7.17 \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
}%
```

7.18 \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%
}%
```

7.19 \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
}%
```

7.20 \xintREZ

Removes trailing zeros from A and B and adjust the N in A/B[N].

```
\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#4#5%
{%
    \expandafter\XINT_rez_E\expandafter
    {\the\numexpr #3+#4-#2}{#1}{#5}%
}%
\def\XINT_rez_E #1#2#3{ #3/#2[#1]}%
```

7.21 \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]}%
```

7.22 \xintIrr

```
\def\xintIrr {\romannumeral0\xintirr }%
\def\xintirr #1%
{%
    \expandafter\XINT_irr_start\romannumeral0\xintrawwithzeros {#1}\Z
}%
\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_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{\xintError:NaN\space 0/0}%
\def\XINT_irr_divisionbyzero #1#2#3#4#5{\xintError:DivisionByZero #5#2/0}%
\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
```

7.23 \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
}%
```

7.24 \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{\xintError:NaN\space 0/0}%
\def\XINT_jrr_divisionbyzero #1#2#3#4#5#6#7{\xintError:DivisionByZero #7#2/0}%
\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\Z #1\Z}{#6}}%
        {\romannumeral0\xintiiadd{\XINT_mul_fork #5\Z #1\Z}{#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}%
}%
```

7.25 \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}}%
```

7.26 \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\Z\xint_dothis\XINT_trunc_zero\fi
    \if1\XINT_iSone#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_relax\xint_relax\xint_relax\xint_relax
      \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
      \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%
}%
```

7.27 \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_}%
```

7.28 \xintNum

```
\let\xintNum \xintTTrunc
\let\xintnum \xintttrunc
```

7.29 \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\Z\xint_dothis\XINT_trunc_zero\fi
    \if1\XINT_iSone#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.}%
```

7.30 \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
    .\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
    .\the\numexpr (#1#2#3#4#5#6#7#8+\xint_c_i)/\xint_c_ii\expandafter
    .\the\numexpr #1#2#3#4#5#6#7#8\expandafter
    .\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_iii
        \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
        \R.\xint_c_ii\R.\xint_c_i\R.\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.{#1}%
}%
\def\XINT_xtrunc_I -#1.#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.#4.{#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.#2.#3#4#5#6%
{%
   \expandafter\XINT_xtrunc_IA_d
```

```
    \the\numexpr#2-\xintLength{#6}.{#6}%
    \expandafter\XINT_xtrunc_IA_xd
    \the\numexpr (#1+\xint_c_ii^v)/\xint_c_ii^vi-\xint_c_i.#1.{#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.#2%
{%
    \romannumeral0\XINT_split_fromleft
    #1.#2\xint_gobble_i\xint_bye2345678\xint_bye..%
}%
\def\XINT_xtrunc_IAB_e #1.#2%
{%
    0.\romannumeral\XINT_rep#1\endcsname0#2%
}%
\def\XINT_xtrunc_IA_xd #1.#2.%
{%
    \expandafter\XINT_xtrunc_IA_xe\the\numexpr #2-\xint_c_ii^vi*#1.#1.%
}%
\def\XINT_xtrunc_IA_xe #1.#2.#3#4%
{%
    \XINT_xtrunc_loop {#2}{#4}{#3}{#1}%
}%
\def\XINT_xtrunc_IB_c #1.#2.#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}.{#1}%
}%
\def\XINT_xtrunc_II #1.%
{%
    \expandafter\XINT_xtrunc_II_a\romannumeral\xintreplicate{#1}0.%
}%
\def\XINT_xtrunc_II_a #1.#2#3#4%
{%
    \expandafter\XINT_xtrunc_II_b
    \the\numexpr (#3+\xint_c_ii^v)/\xint_c_ii^vi-\xint_c_i\expandafter.%
    \the\numexpr #3\expandafter.\romannumeral0#2{#4#1}{#2}%
}%
\def\XINT_xtrunc_II_b #1.#2.%
{%
    \expandafter\XINT_xtrunc_II_c\the\numexpr #2-\xint_c_ii^vi*#1.#1.%
}%
\def\XINT_xtrunc_II_c #1.#2.#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.%
}%
\def\XINT_xtrunc_loop_a #1.#2#3%
{%
    \expandafter\XINT_xtrunc_loop_b\romannumeral0#3%
    {#2000000000000000000000000000000000000000000000000000000000000000}%
    {#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.#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.%
}%
\def\XINT_xtrunc_sp_I -#1.#2#3%
{%
    \expandafter\XINT_xtrunc_sp_I_a\the\numexpr #1-#3.#1.{#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.#2.#3#4%
{%
   \expandafter\XINT_xtrunc_sp_IA_c
   \the\numexpr#2-\xintLength{#4}.{#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.#2%
{%
    \romannumeral0\XINT_split_fromleft
    #1.#2\xint_gobble_i\xint_bye2345678\xint_bye..%
}%
\def\XINT_xtrunc_sp_IAB #1.#2%
{%
    0.\romannumeral\XINT_rep#1\endcsname0#2%
}%
\def\XINT_xtrunc_sp_IB_b #1.#2.#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}.{#1}%
}%
\def\XINT_xtrunc_sp_II #1.#2#3%
{%
    #2\romannumeral\XINT_rep#1\endcsname0.\romannumeral\XINT_rep#3\endcsname0%
}%
```

7.31 \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 }%
```

7.32 \xintAdd

```
\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_C #1#2#3%
{%
   \ifcase\romannumeral0\xintiicmp {#2}{#3} %<- intentional space here.
      \expandafter\XINT_fadd_eq
   \or\expandafter\XINT_fadd_D
   \else\expandafter\XINT_fadd_Da
   \fi {#2}{#3}{#1}%
}%
\def\XINT_fadd_eq #1#2#3#4%#5%
{%
   \expandafter\XINT_fadd_G
   \romannumeral0\xintiiadd {#3}{#4}/#1%[#5]%
}%
\def\XINT_fadd_D #1#2%
{%
   \expandafter\XINT_fadd_E\romannumeral0\XINT_div_prepare {#2}{#1}{#1}{#2}%
}%
\def\XINT_fadd_E #1#2%
{%
   \if0\XINT_Sgn #2\Z
        \expandafter\XINT_fadd_F
   \else\expandafter\XINT_fadd_K
   \fi {#1}%
}%
```

```
\def\XINT_fadd_F #1#2#3#4#5%#6%
{%
   \expandafter\XINT_fadd_G
   \romannumeral0\xintiiadd {\xintiiMul {#5}{#1}}{#4}/#2%[#6]%
}%
\def\XINT_fadd_Da #1#2%
{%
   \expandafter\XINT_fadd_Ea\romannumeral0\XINT_div_prepare {#1}{#2}{#1}{#2}%
}%
\def\XINT_fadd_Ea #1#2%
{%
   \if0\XINT_Sgn #2\Z
        \expandafter\XINT_fadd_Fa
   \else\expandafter\XINT_fadd_K
   \fi {#1}%
}%
\def\XINT_fadd_Fa #1#2#3#4#5%#6%
{%
   \expandafter\XINT_fadd_G
   \romannumeral0\xintiiadd {\xintiiMul {#4}{#1}}{#5}/#3%[#6]%
}%
\def\XINT_fadd_G #1{\if0#1\XINT_fadd_iszero\fi\space #1}%
\def\XINT_fadd_K #1#2#3#4#5%
{%
    \expandafter\XINT_fadd_L
    \romannumeral0\xintiiadd {\xintiiMul {#2}{#5}}{\xintiiMul {#3}{#4}}.%
    {{#2}{#3}}%
}%
\def\XINT_fadd_L #1{\if0#1\XINT_fadd_iszero\fi \XINT_fadd_M #1}%
\def\XINT_fadd_M #1.#2{\expandafter\XINT_fadd_N \expandafter
                       {\romannumeral0\xintiimul #2}{#1}}%
\def\XINT_fadd_N #1#2{ #2/#1}%
\edef\XINT_fadd_iszero\fi #1[#2]{\noexpand\fi\space 0/1[0]}% ou [#2] originel?
```

7.33 \xintSub

```
\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 }%
```

7.34 \xintSum

```
\def\xintSum {\romannumeral0\xintsum }%
\def\xintsum #1{\xintsumexpr #1\relax }%
\def\xintSumExpr {\romannumeral0\xintsumexpr }%
\def\xintsumexpr {\expandafter\XINT_fsumexpr\romannumeral`&&@}%
\def\XINT_fsumexpr {\XINT_fsum_loop_a {0/1[0]}}%
```

```
\def\XINT_fsum_loop_a #1#2%
{%
    \expandafter\XINT_fsum_loop_b \romannumeral`&&@#2\Z {#1}%
}%
\def\XINT_fsum_loop_b #1%
{%
    \xint_gob_til_relax #1\XINT_fsum_finished\relax
    \XINT_fsum_loop_c #1%
}%
\def\XINT_fsum_loop_c #1\Z #2%
{%
    \expandafter\XINT_fsum_loop_a\expandafter{\romannumeral0\xintadd {#2}{#1}}%
}%
\def\XINT_fsum_finished #1\Z #2{ #2}%
```

7.35 \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]}%
```

7.36 \xintSqr

1.1 modifs comme xintMul.
  ARRRRRGGGGGGH! I realize only on 2016/02/29 that this was broken since 1.1 of 2014/10/28 due to a typo in \XINT_fsqr_a, which was written \xint_fsqr_a :((((((((. My test files are highly deficient... (they test only the xint/xintcore version).

```
\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]}%
```

7.37 \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}\Z%
}%
\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}}%
```

7.38 \xintiFac

Note pour 1.2f: il y avait un peu de confusion avec \xintFac, \xintiFac, \xintiiFac, car \xintiFac aurait dû aussi utiliser \xintNum une fois xintfrac.sty chargé ce qu'elle ne faisait pas. \xintNum est nécessaire pour gérer des inputs fractionnaires ou avec [N], car il les transforme en entiers stricts, et la doc dit que les macros avec "i" l'utilise. Maintenant \xintiFac fait la chose correcte. \xintFac est synonyme.

2015/11/29: NO MORE a \xintFac, only \xintiFac/\xintiiFac.

```
\def\xintifac #1{\expandafter\XINT_fac_fork\the\numexpr \xintNum{#1}.}%
```

7.39 \xintiBinomial

1.2f. Binomial coefficients.

```
\def\xintibinomial #1#2%
{%
    \expandafter\XINT_binom_pre
    \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.%
}%
```

7.40 \xintiPFactorial

1.2f. Partial factorial.

```
\def\xintipfactorial #1#2%
{%
    \expandafter\XINT_pfac_fork
    \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.%
}%
```

7.41 \xintPrd

```
\def\xintPrd {\romannumeral0\xintprd }%
\def\xintprd #1{\xintprdexpr #1\relax }%
\def\xintPrdExpr {\romannumeral0\xintprdexpr }%
\def\xintprdexpr {\expandafter\XINT_fprdexpr \romannumeral`&&@}%
\def\XINT_fprdexpr {\XINT_fprod_loop_a {1/1[0]}}%
\def\XINT_fprod_loop_a #1#2%
{%
    \expandafter\XINT_fprod_loop_b \romannumeral`&&@#2\Z {#1}%
}%
\def\XINT_fprod_loop_b #1%
{%
    \xint_gob_til_relax #1\XINT_fprod_finished\relax
    \XINT_fprod_loop_c #1%
}%
\def\XINT_fprod_loop_c #1\Z #2%
{%
  \expandafter\XINT_fprod_loop_a\expandafter{\romannumeral0\xintmul {#1}{#2}}%
}%
\def\XINT_fprod_finished #1\Z #2{ #2}%
```

7.42 \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}}%
```

7.43 \xintDivFloor

1.1

```
\def\xintDivFloor     {\romannumeral0\xintdivfloor }%
\def\xintdivfloor #1#2{\xintfloor{\xintDiv {#1}{#2}}}%
```

7.44 \xintDivTrunc

1.1. \xintttrunc rather than \xintitrunc0 in 1.1a

```
\def\xintDivTrunc     {\romannumeral0\xintdivtrunc }%
\def\xintdivtrunc #1#2{\xintttrunc {\xintDiv {#1}{#2}}}%
```

7.45 \xintDivRound

1.1

```
\def\xintDivRound     {\romannumeral0\xintdivround }%
\def\xintdivround #1#2{\xintiround 0{\xintDiv {#1}{#2}}}%
```

7.46 \xintMod

1.1. \xintMod {q1}{q2} computes q2*t(q1/q2) with t(q1/q2) equal to the truncated division of two arbitrary fractions q1 and q2. We put some efforts into minimizing the amount of computations. Oui, et bien cela aurait été bien si j'avais aussi daigné commenté ce que je faisais.

```
\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\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}%
}%
\def\XINT_mod_bpos #1%
{%
    \xint_UDsignfork
            #1{\xintiiopp\XINT_mod_pos {}}%
             -{\XINT_mod_pos #1}%
    \krof
}%
\def\XINT_mod_bneg #1%
{%
    \xint_UDsignfork
            #1{\xintiiopp\XINT_mod_pos {}}%
             -{\XINT_mod_pos #1}%
    \krof
}%
\def\XINT_mod_divbyzero #1.{\xintError:DivisionByZero\space 0/1[0]}%
\def\XINT_mod_aiszero #1.{ 0/1[0]}%
\def\XINT_mod_pos #1#2/#3[#4]#5/#6[#7].%
{%
    \expandafter\XINT_mod_pos_a
    \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.\expandafter
    {\romannumeral0\xintiimul {#6}{#3}}%       n fois u
    {\xintiiE{\xintiiMul {#1#5}{#3}}{#7-#4}}%  m fois u
    {\xintiiE{\xintiiMul {#2}{#6}}{#4-#7}}%    t fois n
}%
\def\XINT_mod_pos_a #1.#2#3#4{\xintiirem {#3}{#4}/#2[#1]}%
```

7.47 \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\XINT_Cmp {#1}{#2}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%
```

7.48 \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\Z
    { 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\expandafter
    {\romannumeral0\XINT_dsx_addzeros {#1}#3;}{#2}%
}%
\def\XINT_fgeq_Fe #1#2{\XINT_geq_pre {#2}{#1}}%
\def\XINT_fgeq_Fn #1\Z #2#3%
{%
    \expandafter\XINT_geq_pre\expandafter
    {\romannumeral0\XINT_dsx_addzeros {#1}#2;}{#3}%
}%
```

7.49 \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
}%
```

7.50 \xintMaxof

```
\def\xintMaxof      {\romannumeral0\xintmaxof }%
\def\xintmaxof    #1{\expandafter\XINT_maxof_a\romannumeral`&&@#1\relax }%
\def\XINT_maxof_a #1{\expandafter\XINT_maxof_b\romannumeral0\xintraw{#1}\Z }%
\def\XINT_maxof_b #1\Z #2%
            {\expandafter\XINT_maxof_c\romannumeral`&&@#2\Z {#1}\Z}%
\def\XINT_maxof_c #1%
            {\xint_gob_til_relax #1\XINT_maxof_e\relax\XINT_maxof_d #1}%
\def\XINT_maxof_d #1\Z
            {\expandafter\XINT_maxof_b\romannumeral0\xintmax {#1}}%
\def\XINT_maxof_e #1\Z #2\Z { #2}%
```

7.51 \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
}%
```

7.52 \xintMinof

```
\def\xintMinof      {\romannumeral0\xintminof }%
\def\xintminof    #1{\expandafter\XINT_minof_a\romannumeral`&&@#1\relax }%
\def\XINT_minof_a #1{\expandafter\XINT_minof_b\romannumeral0\xintraw{#1}\Z }%
\def\XINT_minof_b #1\Z #2%
            {\expandafter\XINT_minof_c\romannumeral`&&@#2\Z {#1}\Z}%
\def\XINT_minof_c #1%
            {\xint_gob_til_relax #1\XINT_minof_e\relax\XINT_minof_d #1}%
\def\XINT_minof_d #1\Z
            {\expandafter\XINT_minof_b\romannumeral0\xintmin {#1}}%
\def\XINT_minof_e #1\Z #2\Z { #2}%
```

7.53 \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}%  1.08b had some [ and ] here!!!
\def\XINT_fcmp_firstzero  #1#2#3#4{ -1}% incredibly I never saw that until
\def\XINT_fcmp_secondzero #1#2#3#4{ 1}%  preparing 1.09a.
\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\Z
    { -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\expandafter
```

```
    {\romannumeral0\XINT_dsx_addzeros {#1}#3;}{#2}%
}%
\def\XINT_fcmp_Fe #1#2{\xintiicmp {#2}{#1}}%
\def\XINT_fcmp_Fn #1\Z #2#3%
{%
    \expandafter\xintiicmp\expandafter
    {\romannumeral0\XINT_dsx_addzeros {#1}#2;}{#3}%
}%
```

7.54 \xintAbs

```
\def\xintAbs   {\romannumeral0\xintabs }%
\def\xintabs #1{\expandafter\XINT_abs\romannumeral0\xintraw {#1}}%
```

7.55 \xintOpp

```
\def\xintOpp   {\romannumeral0\xintopp }%
\def\xintopp #1{\expandafter\XINT_opp\romannumeral0\xintraw {#1}}%
```

7.56 \xintSgn

```
\def\xintSgn   {\romannumeral0\xintsgn }%
\def\xintsgn #1{\expandafter\XINT_sgn\romannumeral0\xintraw {#1}\Z }%
```

7.57 Floating point macros

For a long time the float routines from releases 1.07/1.08a (May-June 2013) were not modified.

1.2 release did not touch either the floating point routines apart from adding the new \xintFloatFac.

1.2f added \xintFloatPFactorial and \xintFloatBinomial and improved the speed of \xintFloatPow and \xintFloatPower. And its \xintFloat tries to be more efficient in handling inputs which are not fractions to start with. (this has been improved again in 1.2g)

But some parts of the code in \xintFloat are still in the pre-1.2 style and could be improved, anyhow in the future quite probably xintfrac will have to adopt an inner format for floats with for example their precision P as a visible data first in front and things will have to be modified again.

Next major release will have made fondamental decisions regarding the handling of inputs having originally more than the target precision, or even worse are expressed as fractions.

Currently \xintFloat still handles fractional input A/B via an initial replacement to A'/B' (up to powers of ten) with A' and B' the A and B truncated to P+2 digits. In the future \xintFloat will produce the correctly rounded value with P digits of precision independently of the fraction representative A/B, which is not the case with the current procedure.

But already 1.2f has changed an important aspect: the four operations first round their arguments to P-floats, not (P+2)-floats as earlier.

7.58 \xintFloat

1.07. May 2013. The original macro did the exact rounding of the input fraction to P digits of float precision.

It was completely re-written in 1.08a (June 2013), with "spectacular speed gains", so said my comment back then and further: "The earlier version was seriously silly when dealing with inputs having a big power of ten. Again some modifications in 1.08b for a better treatment of cases with long explicit numerators or denominators."

2015/12. I finally add more comments two years later (for 1.2f).

The important thing I had forgotten to document was that the initial 1.07 version did the *exact* rounding of fractional inputs A/B whereas the 1.08 version **first truncated A and B to P+2

digits**: to round A/B 10^n to P digits of precision the routine first truncates A and B to P+2 digits and after that does the exact rounding of A/B. Naturally this means that it may then not necessarily compute the correct rounding of A/B.
  Example : \xintFloat {1/17597472569900621233}
with xintfrac 1.07: 5.682634230727187e-20
with xintfrac 1.08b or later: 5.682634230727188e-20
and the exact value is 5.68263423072718749992412 4...e-20.
  1.07 computed the exact rounding for all inputs but as explained from 1.08 on, \xintFloat first truncates the denominator to 16+2=18 digits, and here this increases 1/x enough to make the final rounding produce a result 1ulp higher.
  In fact already dropping the last digit is enough to make the quotient cross the border:
1/175974725699006212 33=5.68263423072718749992412 4...e-20
1/175974725699006212 3 =56.82634230727187500892894...e-20
  1.2f did some minor improvements to the code, there was in particular a never-used branch. And it tries to handle more swiftly the case of inputs which are not fractions. (improved again in 1.2g)
  This routine uses old macros \XINT_addm_A and \XINT_lenrord_loop. This could now be penalizing for P exceeding a few dozens, compared to doing it the 1.2 way.
  I have always hesitated about the policy of printing 10.00...0 in case of rounding upwards towards next power of ten. It does make sense because it tells the (higher) precision of the rounding and moreover it is not too hard to test on output, although it is a bit cumbersome not to be certain to have exactly P digits.
  Since 1.2f only \xintFloat but not \XINTinFloat may have P+1 digits in the mantissa on output. \XINTinFloat is the inner macro used for example by all operations in float expression to output their result, hence most of the time receive it in A[N] shape also, which is quicly parsed --- apart of course that each time length of A is computed again.
  1.2g adds \XINTinFloatS which may output A[N] with A having <P digits. This is for situations with large P's (say in the hundreds) and has a dramatic speed-up effect on things like 2x or x/3. Addition and subtraction will still use \XINTinFloat on inputs; anyway this will have to be changed again when inner structure will carry up front at least the length of mantissa as a data not to recompute.
  1.2g has also re-written both \xintFloat and \XINTinFloat start code to intercept more quickly and more satisfactorily B=1 case.
  1.2i simplifies code via use of \xintDSRr. No more use of \XINT_lenrord_loop and of \XINT_addm_A, which have now been deleted from the xint sources.

```
\def\xintFloat   {\romannumeral0\xintfloat }%
\def\xintfloat #1{\XINT_float_chkopt #1\xint_relax }%
\def\XINT_float_chkopt #1%
{%
    \ifx [#1\expandafter\XINT_float_opt
       \else\expandafter\XINT_float_noopt
    \fi  #1%
}%
\def\XINT_float_noopt #1\xint_relax
{%
    \expandafter\XINT_float_a\expandafter\XINTdigits\expandafter.%
    \romannumeral0\XINT_infrac {#1}\XINT_float_Q
}%
\def\XINT_float_opt [\xint_relax #1]#2%
{%
    \expandafter\XINT_float_a\the\numexpr #1\expandafter.%
    \romannumeral0\XINT_infrac {#2}\XINT_float_Q
}%
```

```
Note 2015/12/02. Le but de ce code de 1.08 (2013), jusqu'à l'exécution de \XINT_float_Q, est sim-
plement de tronquer numérateur et dénominateur à au plus P+2 chiffres en ajustant la partie déci-
male "n".
  2016/03/19. First thing to do is to intercept denominator=1 case.
```

```
\def\XINT_float_a #1.#2#3#4%
{%
    \if1\XINT_iSone#4XY\expandafter\XINT_float_sp
    \else\expandafter\XINT_float_fork\fi #3.{#1}{#2}{#4}%
}%
```

```
Special quick treatment of B=1 case. The \XINTinFloat variant has also optionally the non-addition
of zeroes to short inputs. Not for \xintFloat. The other difference is that \xintFloat may output
10.0...0eN
```

```
\def\XINT_float_sp #1%
{%
    \xint_UDzerominusfork
     #1-\XINT_float_spzero
     0#1\XINT_float_spneg
      0-{\XINT_float_sppos #1}%
    \krof
}%
\def\XINT_float_spzero .#1#2#3#4{ 0.e0}%
\def\XINT_float_spneg {\expandafter-\romannumeral0\XINT_float_sppos}%
\def\XINT_float_sppos #1.#2#3#4#5%
{%
    \expandafter\XINT_float_sp_b\the\numexpr#2-\xintLength{#1}.#1.#2.#3.%
}%
\def\XINT_float_sp_b #1%
{%
    \xint_UDzerominusfork
     #1-\XINT_float_sp_quick
     0#1\XINT_float_sp_c
      0-{\XINT_float_sp_addzeros #1}%
    \krof
}%
\def\XINT_float_sp_quick .#1.#2.#3.%
{%
    \expandafter\XINT_float_sp_done\the\numexpr #3+#2-\xint_c_i.#1;%
}%
\def\XINT_float_sp_addzeros #1.#2.#3.#4.%
{%
    \expandafter\XINT_float_sp_done
    \the\numexpr #4-#1+#3-\xint_c_i\expandafter.%
    \romannumeral0\XINT_dsx_addzeros {#1}#2;;%
}%
\def\XINT_float_sp_done #1.#2#3;{ #2.#3e#1}%
\def\XINT_float_sp_c #1.#2%
{%
    \if #29\xint_dothis {\XINT_float_sp_d\XINT_float_Wb }\fi
           \xint_orthat {\XINT_float_sp_d\XINT_float_Wa }#1.#2%
}%
```

```
\def\XINT_float_sp_d #1#2.#3.#4.#5.%
{%
    \expandafter\XINT_float_sp_e
    \the\numexpr #5+#2+#4-\xint_c_i\expandafter.%
    \romannumeral0\XINT_split_fromleft
    (\xint_c_i+#4).#3\xint_bye2345678\xint_bye..#1%
}%
```

1.2i uses the \xintDSRr quick \numexpr loop for faster treatment.
#1=exposant final.#2=P+1chiffres de A.#3=junk.#4=\XINT_float_Wb ou \XINT_float_Wa

```
\def\XINT_float_sp_e #1.#2.#3.#4%
{%
    \expandafter#4\romannumeral0\XINT_dsrr#2%
    \xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax e#1%
}%
```

A.{P}{n}{B}\XINT_float_Q avec B qui est >1, donc A=0 exclu.

```
\def\XINT_float_fork #1%
{%
    \xint_UDsignfork
     #1\XINT_float_J
     -{\XINT_float_K #1}%
    \krof
}%
\def\XINT_float_J {\expandafter-\romannumeral0\XINT_float_K }%
\def\XINT_float_K #1.#2%
{%
    \expandafter\XINT_float_L
    \the\numexpr\xintLength{#1}\expandafter.\the\numexpr #2+\xint_c_ii.{#1}{#2}%
}%
```

|A|.P+2.{A}{P}{n}{B}\XINT_float_Q. We check if A already has length <= P+2.

```
\def\XINT_float_L #1.#2.%
{%
    \ifnum #1>#2
      \expandafter\XINT_float_Ma
    \else
      \expandafter\XINT_float_Mb
    \fi #1.#2.%
}%
```

|A|.P+2.{A}{P}{n}{B}\XINT_float_Q. We will keep only the first P+2 digits of A. We use A' for notation.

```
\def\XINT_float_Ma #1.#2.#3%
{%
    \expandafter\XINT_float_MatoN
    \the\numexpr #1-#2\expandafter.%
    \romannumeral0\XINT_split_fromleft#2.#3\xint_bye2345678\xint_bye..{#2}%
}%
```

```
  |A|-(P+2).{A'=P+2 premiers chiffres de A}.{junk}.{P+2}{P}{n}{B}\XINT_float_Q devient |B|.n'=n+|A|-
  (P+2).P+2.{B}{P+2}{A'}{P}. Car ici P+2=|A'|.

1726 \def\XINT_float_MatoN #1.#2.#3.#4#5#6#7%
1727 {%
1728    \expandafter\XINT_float_N
1729    \the\numexpr\xintLength{#7}\expandafter.\the\numexpr #1+#6.#4.%
1730    {#7}{#4}{#2}{#5}%
1731 }%

  Dans cette branche A'=A. En entrée |A|.P+2.{A}{P}{n}{B}, en sortie |B|.n.P+2.{B}{|A|}{A}{P}

1732 \def\XINT_float_Mb #1.#2.#3#4#5#6%
1733 {%
1734    \expandafter\XINT_float_N
1735    \romannumeral0\xintlength{#6}.#5.#2.{#6}{#1}{#3}{#4}%
1736 }%

  Ce qu'on a fait avec A on le fait maintenant avec B. En entrée: |B|.n'.P+2.{B}{|A'|}{A'}{P}\XINT_float_Q
    1.2g has already filtered out the case B=1. We compare length |B| to P+2.

1737 \def\XINT_float_N #1.#2.#3.%
1738 {%
1739     \ifnum #1>#3
1740          \expandafter\XINT_float_N_Blong
1741     \else\expandafter\XINT_float_P\fi
1742     #1.#2.#3.%
1743 }%

  Ici B est de longueur > P+2. On va le tronquer. En entrée #1=|B|, #2=n', #3=P+2, #4=B. En sortie:
  n''=n'-(|B|-(P+2)).{B'}{junk}{P+2}

1744 \def\XINT_float_N_Blong #1.#2.#3.#4%
1745 {%
1746     \expandafter\XINT_float_NaP
1747     \the\numexpr #2-#1+#3\expandafter.%
1748     \romannumeral0\XINT_split_fromleft#3.#4\xint_bye2345678\xint_bye..{#3}%
1749 }%

  n''=n'-(|B|-(P+2)).{B'}.{junk}.{P+2}->0.P+2.n''.P+2.{B'}

1750 \def\XINT_float_NaP #1.#2.#3.#4{\XINT_float_P #4.#1.#4.{#2}}%

  Si B est de longueur <= P+2 on arrive ici avec en entrée |B|.n'.P+2.{B}, sinon avec P+2.n''.P+2.{B'},
  suivi dans les deux cas par {|A'|}{A'}{P} et #8=\XINT_float_Q. On va invoquer \XINT_float_Q, on
  aura donc \XINT_float_Q |B|-|A|+P+1.{A}{B}{P}{n} avec les nouveaux A, B, n. Ici on a doit passer
  au-dessus de A et |A| pour aller chercher #8. à revoir car pas satisfaisant.

1751 \def\XINT_float_P #1.#2.#3.#4#5#6#7#8%
1752 {%
1753     \expandafter #8\the\numexpr #1-#5+#3-\xint_c_i.{#6}{#4}{#7}{#2}%
1754 }%
```

```
  On arrive ici avec |B|-|A|+P+1.{A}{B}{P}{n}, les A et B étant ceux d'origine tronqués à au plus
  P+2 chiffres, le n a été ajusté si besoin.
    On calcule maintenant le quotient euclidien de A 10^{|B|-|A|+P+1} (qui a P+1 chiffres de plus
  que B) par B. Ce quotient Q aura P+1 ou P+2 chiffres.

1755 \def\XINT_float_Q #1.#2#3%
1756 {%
1757     \expandafter\XINT_float_Sa
1758     \romannumeral0\xintiiquo{\XINT_dsx_addzeros {#1}#2;}{#3}\Z {#1}%
1759 }%

  On a Q\Z {|B|-|A|+P+1}{P}{n}. Comme Q = trunc (A/B 10^x), x=|B|-|A|+P+1, Q peut avoir P+1 ou P+2
  chiffres. Ce qui compte c'est qu'il a au moins P+1 chiffres. On va examiner si le (P+1)e chiffre
  est 5. Mais on fait une sous-branche pour les cas exceptionnels qui donnent un arrondi est vers
  le haut vers la prochaine puissance de 10. Ceci ne pourra se produire que si le premier chiffre
  significatif de Q est un 9.

1760 \def\XINT_float_Sa #1%
1761 {%
1762     \if #19\xint_dothis {\XINT_float_Sb\XINT_float_Wb }\fi
1763           \xint_orthat {\XINT_float_Sb\XINT_float_Wa }#1%
1764 }%

  En entrée \XINT_float_W(a ou b) Q\Z {|B|-|A|+P+1}{P}{n}.
    Refait pour 1.2i, plus de reverse, mais emploi de \xintDSRr. Mais on doit savoir si Q est de
  longueur L=P+1 ou P+2.
    #1=Wa ou Wb, #2=Q, #3=|B|-|A|+P+1, #4=P, #5=n. On va calculer L-P.

1765 \def\XINT_float_Sb #1#2\Z #3#4%
1766 {%
1767     \expandafter\XINT_float_T
1768     \the\numexpr\XINT_length_loop
1769     #2\xint_relax\xint_relax\xint_relax\xint_relax
1770       \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
1771       \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
1772       \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye-#4.%
1773     {#2}#1{#3}{#4}%
1774 }%

  Si L>P+1, c'est que L=P+2. On partira en Ub puis Xb. Sinon Ua puis Xa En sortie on a {Q}\token
  {|B|-|A|+P+1}{P}{n}

1775 \def\XINT_float_T #1.%
1776 {%
1777     \if2#1\expandafter\XINT_float_Ub\else\expandafter\XINT_float_Ua\fi
1778 }%
1779 \def\XINT_float_Ua #1#2%
1780 {%
1781     \expandafter\XINT_float_Xa
1782     \romannumeral0\expandafter#2%
1783     \romannumeral0\XINT_dsrr
1784     #1\xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax;%
1785 }%
1786 \def\XINT_float_Ub #1#2%
```

```
{%
    \expandafter\XINT_float_Xb
    \romannumeral0\expandafter#2%
    \romannumeral0\expandafter\XINT_dsrr
    \romannumeral0\XINT_dsr
    #1\xint_bye\xint_Bye3456789\xint_bye+\xint_c_v)/\xint_c_x-\xint_c_i\relax
    \xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax;%
}%
```

Wa insère la virgule. Wb regarde si on a arrondi vers le haut vers une puissance de 10. Il n'est exécuté que si le premier chiffre était un 9, donc il regarde si maintenant le premier chiffre est devenu un 1. Est-il sûr que Wb trouve deux chiffres si P=1? Pour n'en avoir qu'un, il faudrait que le quotient Q aie eu seulement deux chiffres. Mais si on arrive en Wb le premier chiffre était 9 et si Q avait eu seulement deux chiffres il aurait été 95 au plus, et ici on serait avec 10 qui ne pose pas de problème.

```
\def\XINT_float_Wa #1{ #1.}%
\def\XINT_float_Wb #1#2{\if #11\xint_dothis{ 10.}\fi\xint_orthat{ #1.#2}}%
```

Il faut faire l'ajustement final de n. On doit regrabber notre mantisse, maintenant avec sa virgule. On a {|B|-|A|+P+1}{P}{n} après le ; On exécute _Xb si le quotient produit avait P+2 chiffres. On n'a pas à faire d'ajustement en cas d'arrondi vers un 10.00...0.

```
\def\XINT_float_Xa #1;#2#3#4%
{%
    \expandafter\XINT_float_Y\the\numexpr #3+#4-#2.{#1}%
}%
\def\XINT_float_Xb #1;#2#3#4%
{%
    \expandafter\XINT_float_Y\the\numexpr #3+#4+\xint_c_i-#2.{#1}%
}%
\def\XINT_float_Y #1.#2{ #2e#1}%
```

7.59 \XINTinFloat

1.07.

This routine is like \xintFloat but produces an output of the shape A[N] which is then parsed faster on input to other float macros.

For 1.2f I modify it : contrarily to current \xintFloat, in the exceptional case of rounding up to a power of ten, it does not produce anymore a mantissa 10^P with P+1 digits, but 10^{P-1}. Indeed not knowing for sure the number of digits of the mantissa caused various complications in other routines, and I really got tired of this. This means however that it is a tiny bit slower than earlier.

2016/03/11. 1.2f \XINTinFloat clones somes of the \XINT_infrac start code to handle more swiftly inputs of the shape A[N] (still allowing N to be an <expression> for \numexpr). This is done without yet introducting a private format for floats, as I want to conclude now and doing this would need some extra time.

As this is surgery on pre-existing code where a more complete rewrite would be needed it is a bit ugly though. 1.2g has redone it and does not anymore tap into \XINT_infrac.

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. But obviously we can not go one re-parsing each input that way, although the situation is better with 1.2f.

2016/03/18. 1.2g adds a variant \XINTinFloatS which allows its output to be shorter than the current precision. Indeed \XINTinFloat is used everywhere but for example it is silly to convert 2 into 2<499zeroes>[-499] if we need to do some multiplication or division...
2016/03/19. 1.2g uses non-patched \XINT_infrac but then immediately filters out all denominator=1 cases (only the A[N] cases were identified by 1.2f).
2016/12/11. 1.2i simplifies the coding via use of \xintDSRr.

```
\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}%
\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]}%
\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.{ }%
\def\XINT_infloat [#1]#2%
{%
    \expandafter\XINT_infloat_a\the\numexpr #1\expandafter.%
    \romannumeral0\XINT_infrac {#2}\XINT_infloat_Q
}%
\def\XINT_infloat_a #1.#2#3#4%
{%
    \if1\XINT_iSone#4XY\expandafter\XINT_infloat_sp
    \else\expandafter\XINT_float_fork\fi #3.{#1}{#2}{#4}%
}%
```

Special quick treatment of B=1 case (1.2f then redone 1.2g).

```
\def\XINT_infloat_sp #1%
{%
    \xint_UDzerominusfork
     #1-\XINT_infloat_spzero
     0#1\XINT_infloat_spneg
      0-{\XINT_infloat_sppos #1}%
    \krof
}%
\def\XINT_infloat_spzero .#1#2#3#4{ 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.-}%
\def\XINT_infloat_sppos #1.#2#3#4#5%
```

```
{%
    \expandafter\XINT_infloat_sp_b\the\numexpr#2-\xintLength{#1}.#1.#2.#3.%
}%
\def\XINT_infloat_sp_b #1%
{%
    \xint_UDzerominusfork
     #1-\XINT_infloat_sp_quick
     0#1\XINT_infloat_sp_c
      0-{\XINT_infloat_sp_needzeros #1}%
    \krof
}%
\def\XINT_infloat_sp_quick .#1.#2.#3.{ #1[#3]}%
\def\XINT_infloat_sp_needzeros #1.#2.#3.#4.{!#1.#2[#4]}%
```

I should re-use truncating/rounding routines but I wrote them too much time ago. Faster to do it again.

```
\def\XINT_infloat_sp_c #1.#2%
{%
    \if #29\xint_dothis {\XINT_infloat_sp_d\XINT_infloat_Wb }\fi
           \xint_orthat {\XINT_infloat_sp_d .}#1.#2%
}%
\def\XINT_infloat_sp_d #1#2.#3.#4.%#5.%
{%
    \expandafter\XINT_infloat_sp_e
    \romannumeral0\XINT_split_fromleft
    (\xint_c_i+#4).#3\xint_bye2345678\xint_bye..#1#2.%
}%
```

#1=first P+1 digits of A, #2=junk, #3=\XINT_infloat_Wb ou «.», #4=L-P, #5=N. Exposant final sera N+L-P ou N+L-P+1. Le +1 dans le cas où on il y a eu arrondi vers la puissance de 10 supérieure. On récupère le +1 par #3 éventuellement. C'est le #3 qui met un «.» pour terminer le \numexpr.

```
\def\XINT_infloat_sp_e #1.#2.#3#4.#5.%
{%
    \expandafter\XINT_infloat_done
    \the\numexpr#4+#5\expandafter#3\romannumeral0\XINT_dsrr
    #1\xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax;%
}%
```

General branch handling A/B possibly with [N] or scientific notation inputs.
  Since 1.2g this is always with B>1. And A is not zero.

```
\def\XINT_infloat_Q #1.#2#3%
{%
    \expandafter\XINT_infloat_Sa
    \romannumeral0\xintiiquo{\XINT_dsx_addzeros {#1}#2;}{#3}\Z {#1}%
}%
\def\XINT_infloat_Sa #1%
{%
    \if #19\xint_dothis {\XINT_infloat_Sb\XINT_infloat_Wb }\fi
           \xint_orthat {\XINT_infloat_Sb .}#1%
}%
% \lverb?Refait pour 1.2i: calcul de longueur, puis \xintDSRr.
```

```
%
% En entrée \XINT_infloat_Wb ou «.» Q\Z {|B|-|A|+P+1}{P}{n}. On
% commence par évaluer longueur de Q -P pour savoir si 1 ou 2. Plus besoin de
% P après. L'exposant final sera 1+n-(|B|-|A|+P+1) ou 2+n-(|B|-|A|+P+1).
% modulo ajustement avec encore un +1 éventuel de Wb.
% ?
\def\XINT_infloat_Sb #1#2\Z #3#4%
{%
    \expandafter\XINT_infloat_T
    \the\numexpr\XINT_length_loop
    #2\xint_relax\xint_relax\xint_relax\xint_relax
      \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
      \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-#4.%
    {#2}#1{#3}%
}%
```

Si L>P+1, c'est que L=P+2. Dans ce cas exposant final sera 2+n-(|B|-|A|+P+1) tandis que c'est 1+n-(|B|-|A|+P+1) dans le premier cas. Dans les deux ajustement éventuel par un +1 pouvant venir de Wb.

```
\def\XINT_infloat_T #1.%
{%
    \if2#1\expandafter\XINT_infloat_Ub\else\expandafter\XINT_infloat_Ua\fi
}%
\def\XINT_infloat_Ua #1#2#3#4%
{%
    \expandafter\XINT_infloat_done
    \the\numexpr\xint_c_i+#4-#3\romannumeral0\expandafter#2%
    \romannumeral0\XINT_dsrr
    #1\xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax;%
}%
\def\XINT_infloat_Ub #1#2#3#4%
{%
    \expandafter\XINT_infloat_done
    \the\numexpr\xint_c_ii+#4-#3\romannumeral0\expandafter#2%
    \romannumeral0\expandafter\XINT_dsrr
    \romannumeral0\XINT_dsr
    #1\xint_bye\xint_Bye3456789\xint_bye+\xint_c_v)/\xint_c_x-\xint_c_i\relax
    \xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax;%
}%
\def\XINT_infloat_Wb #1#2%
    {\if #11\xint_dothis{+\xint_c_i.1}\fi\xint_orthat{.#1#2}}%
```

7.60 \xintPFloat

1.1. This is a prettifying printing macro for floats.
2016/03/07.
1.2f modifies the macro: among the now obsoleted rules there was one which told it to employ decimal notation as soon as possible without extra zeroes, for example a...z<decimal mark>A...Z if the total length is at most the precision, but obviously for large precisions the human eye has difficulties with that, this was not a good choice.

The new rule is simply that 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. There is the exceptional cases where x=10, and yz..=0000.... from rounding up to the next power of ten.

```
\def\xintPFloat   {\romannumeral0\xintpfloat }%
\def\xintpfloat #1{\XINT_pfloat_chkopt #1\xint_relax }%
\def\XINT_pfloat_chkopt #1%
{%
    \ifx [#1\expandafter\XINT_pfloat_opt
       \else\expandafter\XINT_pfloat_noopt
    \fi  #1%
}%
\def\XINT_pfloat_noopt #1\xint_relax
{%
    \expandafter\XINT_pfloat_a
    \romannumeral0\xintfloat [\XINTdigits]{#1};\XINTdigits.%
}%
\def\XINT_pfloat_opt [\xint_relax #1]%#2%
{%
    \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 #1}%
    \krof
}%
\def\XINT_pfloat_zero #1;#2.{ 0.}%
\def\XINT_pfloat_neg {\expandafter-\romannumeral0\XINT_pfloat_pos }%
\def\XINT_pfloat_pos #1e#2;#3.%
{%
    \ifnum #2>\xint_c_v  \xint_dothis\XINT_pfloat_no\fi
    \ifnum #2<-\xint_c_v \xint_dothis\XINT_pfloat_no\fi
    \ifnum #2<\xint_c_   \xint_dothis\XINT_pfloat_N\fi
    \ifnum #2>\numexpr #3-\xint_c_i\relax \xint_dothis\XINT_pfloat_Ps\fi
    \xint_orthat\XINT_pfloat_P #1e#2;%
}%
\def\XINT_pfloat_no #1;{ #1}%
\def\XINT_pfloat_N #1#2.#3e#4;%
{%
    \csname XINT_pfloat_N\romannumeral-#4\endcsname #2#10.#3;%
}%
\def\XINT_pfloat_Ni  #1#2#3.#4;{ #2.#1#4}%
\def\XINT_pfloat_Nii #1#2#3.#4;{ 0.#2#1#4}%
\def\XINT_pfloat_Niii#1#2#3.#4;{ 0.0#2#1#4}%
\def\XINT_pfloat_Niv #1#2#3.#4;{ 0.00#2#1#4}%
```

```
\def\XINT_pfloat_Nv  #1#2#3.#4;{ 0.000#2#1#4}%
\def\XINT_pfloat_P #1#2.#3e#4;%
{%
    \csname XINT_pfloat_P_\romannumeral#4\endcsname #3.#1#2;%
}%
\def\XINT_pfloat_P_   #1.#2;{ #2.#1}%
\def\XINT_pfloat_P_i  #1#2.#3;{ #3#1.#2}%
\def\XINT_pfloat_P_ii #1#2#3.#4;{ #4#1#2.#3}%
\def\XINT_pfloat_P_iii#1#2#3#4.#5;{ #5#1#2#3.#4}%
\def\XINT_pfloat_P_iv #1#2#3#4#5.#6;{ #6#1#2#3#4.#5}%
\def\XINT_pfloat_P_v  #1#2#3#4#5#6.#7;{ #7#1#2#3#4#5.#6}%
\def\XINT_pfloat_Ps #1#2.#3e#4;%
{%
    \csname XINT_pfloat_Ps\romannumeral#4\endcsname #300000.#1#2;%
}%
\def\XINT_pfloat_Psi  #1#2.#3;{ #3#1.}%
\def\XINT_pfloat_Psii #1#2#3.#4;{ #4#1#2.}%
\def\XINT_pfloat_Psiii#1#2#3#4.#5;{ #5#1#2#3.}%
\def\XINT_pfloat_Psiv #1#2#3#4#5.#6;{ #6#1#2#3#4.}%
\def\XINT_pfloat_Psv  #1#2#3#4#5#6.#7;{ #7#1#2#3#4#5.}%
```

7.61 \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]}%
```

7.62 \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_relax }%
\def\XINTinFloatAdd    {\romannumeral0\XINTinfloatadd }%
\def\XINTinfloatadd  #1{\XINT_fladd_chkopt \XINTinfloatS #1\xint_relax }%
\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_relax #3%
{%
    #1[\XINTdigits]%
    {\expandafter\XINT_FL_add_a
     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{#3}}%
}%
\def\XINT_fladd_opt #1[\xint_relax #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]}%
}%
```

7.63 \xintFloatSub, \XINTinFloatSub

First done 1.07.
  1.2f does not use two extra rounding digits on inputs.

```
\def\xintFloatSub      {\romannumeral0\xintfloatsub }%
\def\xintfloatsub    #1{\XINT_flsub_chkopt \xintfloat #1\xint_relax }%
\def\XINTinFloatSub    {\romannumeral0\XINTinfloatsub }%
\def\XINTinfloatsub  #1{\XINT_flsub_chkopt \XINTinfloatS #1\xint_relax }%
\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_relax #3%
{%
    #1[\XINTdigits]%
    {\expandafter\XINT_FL_add_a
     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{\xintOpp{#3}}}%
}%
\def\XINT_flsub_opt #1[\xint_relax #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}}}%
}%
```

7.64 \xintFloatMul, \XINTinFloatMul

It is a long-standing issue here that I must at some point revise the code and avoid compute with 2P digits the exact intermediate result.

1.07.
1.2f does not use two extra rounding digits on inputs.
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.

```
\def\xintFloatMul   {\romannumeral0\xintfloatmul   }%
\def\xintfloatmul   #1{\XINT_flmul_chkopt \xintfloat #1\xint_relax }%
\def\XINTinFloatMul {\romannumeral0\XINTinfloatmul }%
\def\XINTinfloatmul #1{\XINT_flmul_chkopt \XINTinfloatS #1\xint_relax }%
\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_relax #3%
{%
    #1[\XINTdigits]%
    {\expandafter\XINT_FL_mul_a
     \romannumeral0\XINTinfloatS[\XINTdigits]{#2}\XINTdigits.{#3}}%
}%
\def\XINT_flmul_opt #1[\xint_relax #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]{\xintE{\xintiiMul {#3}{#1}}{#4+#2}}%
```

7.65 \xintFloatDiv, \XINTinFloatDiv

1.07.
  1.2f does not use two extra rounding digits on inputs.
  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.

```
\def\xintFloatDiv   {\romannumeral0\xintfloatdiv   }%
\def\xintfloatdiv   #1{\XINT_fldiv_chkopt \xintfloat #1\xint_relax }%
\def\XINTinFloatDiv {\romannumeral0\XINTinfloatdiv }%
\def\XINTinfloatdiv #1{\XINT_fldiv_chkopt \XINTinfloatS #1\xint_relax }%
\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_relax #3%
{%
    #1[\XINTdigits]%
    {\expandafter\XINT_FL_div_a
     \romannumeral0\XINTinfloatS[\XINTdigits]{#2}\XINTdigits.{#3}}%
}%
\def\XINT_fldiv_opt #1[\xint_relax #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]{#3}#1.{#4}}%
}%
\def\XINT_FL_div_a #1[#2]#3.#4%
{%
    \expandafter\XINT_FL_div_b\romannumeral0\XINTinfloatS[#3]{#4}#1[#2]%
}%
\def\XINT_FL_div_b #1[#2]#3[#4]{\xintE{#3/#1}{#4-#2}}%
```

7.66 \xintFloatPow, \XINTinFloatPow

1.07: initial version. 1.09j has re-organized the core loop.
  2015/12/07. I have hesitated to maintain the mapping of ^ in expressions to \xintFloatPow rather than \xintFloatPower. 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_relax }%
```

```
\def\XINTinFloatPow {\romannumeral0\XINTinfloatpow }%
\def\XINTinfloatpow #1{\XINT_flpow_chkopt \XINTinfloatS #1\xint_relax }%
\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_relax #3%
{%
   \expandafter\XINT_flpow_checkB_a
   \the\numexpr #3.\XINTdigits.{#2}{#1[\XINTdigits]}%
}%
\def\XINT_flpow_opt #1[\xint_relax #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%
{%
    \if 1#51\xint_afterfi {#6{0[0]}}\else
    \xint_afterfi {\xintError:DivisionByZero #6{1[2147483648]}}\fi
}%
\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 will inverse the Q-digits mantissa, keeping Q significant digits (exceptionally 10^Q) before the final rounding to P digits. Here Q is working precision. Releases prior to 1.2f trusted the final inverse to \xintFloat on output but this worked only with P+2 digits on denominator. Enough for 0.1 ulp extra error, but as our goal is to get <0.6ulp, and there is already 0.5ulp from rounding error, this was not enough. When \xintFloat will achieve correct rounding for arbitrary fractions, the step here will not be needed.

```
\def\XINT_flpow_III #1.#2.#3.#4.#5%
{%
    \expandafter\XINT_flpow_IIIend
    \xint_UDsignfork
        #5{{\xintNum{1/#3[\xint_c_ii*#4-\xint_c_i]}[\xint_c_i-\xint_c_ii*#4-#2]}}%
```

```
          -{{#3[#2]}}%
    \krof #1%
}%
\def\XINT_flpow_IIIend #1#2#3%
    {#3{\if#21\xint_afterfi{\expandafter-\romannumeral`&&@}\fi#1}}%
```

7.67 \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 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.

```
\def\xintFloatPower   {\romannumeral0\xintfloatpower}%
\def\xintfloatpower #1{\XINT_flpower_chkopt \xintfloat #1\xint_relax }%
\def\XINTinFloatPower {\romannumeral0\XINTinfloatpower }%
\def\XINTinfloatpower #1{\XINT_flpower_chkopt \XINTinfloatS #1\xint_relax }%
\def\XINTinFloatPowerH {\romannumeral0\XINTinfloatpowerH }%
\def\XINTinfloatpowerH #1#2%
   {\expandafter\XINT_flpowerh_a \romannumeral0\xinttrunc 1{#2}.0;%
    \XINTdigits.{#1}{\XINTinfloatS[\XINTdigits]}}%
\def\XINT_flpowerh_a #1.#2%
{%
    \ifnum#2>\xint_c_iv\xint_dothis\XINT_flpowerh_b\fi
    \xint_orthat\XINT_flpowerh_i #1.#2%
}%
\def\XINT_flpowerh_i #1.#2;%
   {\expandafter\XINT_flpower_checkB_a\romannumeral0\xintinum{#1}.}%
\def\XINT_flpowerh_b #1%
{%
    \if#1-\xint_dothis\XINT_flpowerh_bneg\fi
          \xint_orthat{\XINT_flpowerh_bpos #1}%
}%
\def\XINT_flpowerh_bpos #1.#2;\XINTdigits.#3#4%
{%
    \expandafter\XINT_flpower_checkB_a
    \romannumeral0\xintinc{\xintDouble{#1}}.%
    \XINTdigits.{#3}{\XINTinfloatsqrt[\XINTdigits]}%
}%
\def\XINT_flpowerh_bneg #1.#2;\XINTdigits.#3#4%
{%
    \expandafter\XINT_flpower_checkB_a
    \expandafter-\romannumeral0\xintinc{\xintDouble{#1}}.%
    \XINTdigits.{#3}{\XINTinfloatsqrt[\XINTdigits]}%
}%
\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_relax #3%
{%
   \expandafter\XINT_flpower_checkB_a
   \romannumeral0\xintnum{#3}.\XINTdigits.{#2}{#1[\XINTdigits]}%
}%
\def\XINT_flpower_opt #1[\xint_relax #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#1{\XINT_flpower_checkB_b -}%
       0-{\XINT_flpower_checkB_b {}#1}%
    \krof
}%
\def\XINT_flpower_BisZero .#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
    \if1\xintiiOdd {#1}\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
    \if1\xintiiOdd{#1}\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.%
}%
```

7.68 \xintFloatFac, \XINTFloatFac

```
\def\xintFloatFac     {\romannumeral0\xintfloatfac}%
\def\xintfloatfac   #1{\XINT_flfac_chkopt \xintfloat #1\xint_relax }%
\def\XINTinFloatFac   {\romannumeral0\XINTinfloatfac }%
\def\XINTinfloatfac #1{\XINT_flfac_chkopt \XINTinfloat #1\xint_relax }%
\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_relax
{%
   \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_relax #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{\expandafter\xintError:FactorialOfNegative #5{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{\expandafter\xintError:TooBigFactorial #5{1[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:

```
                     7.886578673647905035523632139321 9e374
```

and this is the correct rounding because for 40 digits it computes

```
                     7.8865786736479050355236321393218 50622951e374
```

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\Z![#2]#3{#3{\romannumeral0\XINT_mul_out
                                   #1\Z!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\Z!. 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\Z![-#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\Z!%
}%
\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_Z #3\XINT_FL_fac_smallmul_end\Z
    \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\Z\XINT_FL_fac_minimulwc_a #1!\Z!#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]}%
}%
```

7.69 \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_relax }%
\def\XINTinFloatPFactorial {\romannumeral0\XINTinfloatpfactorial }%
\def\XINTinfloatpfactorial #1{\XINT_flpfac_chkopt \XINTinfloat #1\xint_relax }%
\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_relax #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_relax #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{\xintError:OutOfRangePFac #5{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\Z![-#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!%
}%
```

7.70 \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_relax }%
\def\XINTinFloatBinomial   {\romannumeral0\XINTinfloatbinomial }%
\def\XINTinfloatbinomial #1{\XINT_flbinom_chkopt \XINTinfloat #1\xint_relax }%
\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_relax #3%
{%
    \expandafter\XINT_FL_binom_a
    \the\numexpr\xintNum{#2}\expandafter.\the\numexpr\xintNum{#3}.\XINTdigits.#1%
}%
\def\XINT_flbinom_opt #1[\xint_relax #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_outofrange\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_outofrange\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_outofrange #1.#2.#3.#4.#5%
       {\xintError:OutOfRangeBinomial #5[#4]{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]}}}%
}%
```

7.71 \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_relax }%
\def\XINTinFloatSqrt   {\romannumeral0\XINTinfloatsqrt }%
\def\XINTinfloatsqrt #1{\XINT_flsqrt_chkopt \XINTinfloat #1\xint_relax }%
\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_relax
{%
    \expandafter\XINT_FL_sqrt_a
                \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.#1%
}%
\def\XINT_flsqrt_opt #1[\xint_relax #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{\xintError:RootOfNegative #3[#2]{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.%
}%
```

7.72 \xintFloatE, \XINTinFloatE

```
\def\xintFloatE   {\romannumeral0\xintfloate }%
\def\xintfloate #1{\XINT_floate_chkopt #1\xint_relax }%
\def\XINT_floate_chkopt #1%
{%
    \ifx [#1\expandafter\XINT_floate_opt
       \else\expandafter\XINT_floate_noopt
    \fi  #1%
}%
\def\XINT_floate_noopt #1\xint_relax
{%
    \expandafter\XINT_floate_a\expandafter\XINTdigits
                \romannumeral0\XINT_infrac {#1}%
}%
\def\XINT_floate_opt [\xint_relax #1]#2%
{%
    \expandafter\XINT_floate_a\expandafter
    {\the\numexpr #1\expandafter}\romannumeral0\XINT_infrac {#2}%
}%
\def\XINT_floate_a #1#2#3#4#5%
{%
    \expandafter\XINT_float_a\the\numexpr#1\expandafter.%
    \expandafter{\the\numexpr #2+#5}{#3}{#4}\XINT_float_Q
}%
\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]}%
```

7.73 \XINTinFloatMod

```
\def\XINTinFloatMod {\romannumeral0\XINTinfloatmod [\XINTdigits]}%
\def\XINTinfloatmod [#1]#2#3{\expandafter\XINT_infloatmod\expandafter
                            {\romannumeral0\XINTinfloat[#1]{#2}}%
                            {\romannumeral0\XINTinfloat[#1]{#3}}{#1}}%
\def\XINT_infloatmod #1#2{\expandafter\XINT_infloatmod_a\expandafter {#2}{#1}}%
\def\XINT_infloatmod_a #1#2#3{\XINTinfloat [#3]{\xintMod {#2}{#1}}}%
\XINT_restorecatcodes_endinput%
```

8 Package xintseries implementation

The commenting is currently (2016/12/22) 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@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
```

8.2 Package identification

```
\XINT_providespackage
\ProvidesPackage{xintseries}%
  [2016/12/22 1.2j Expandable partial sums with xint package (JFB)]%
```

8.3 \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%
}%
```

8.4 \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%
}%
```

8.5 \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}%
}%
```

8.6 \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}%
}%
```

8.7 \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}%
}%
```

8.8 \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}}%
}%
```

8.9 \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]}%
}%
```

8.10 \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}%
}%
```

8.11 \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_relax }%
\def\XINT_flpowseries_chkopt #1%
{%
    \ifx [#1\expandafter\XINT_flpowseries_opt
```

```
        \else\expandafter\XINT_flpowseries_noopt
    \fi
    #1%
}%
\def\XINT_flpowseries_noopt  #1\xint_relax #2%
{%
    \expandafter\XINT_flpowseries\expandafter
    {\the\numexpr #1\expandafter}\expandafter
    {\the\numexpr #2}\XINTdigits
}%
\def\XINT_flpowseries_opt [\xint_relax #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}}%
}%
```

8.12 \xintFloatPowerSeriesX

1.08a

```
\def\xintFloatPowerSeriesX {\romannumeral0\xintfloatpowerseriesx }%
\def\xintfloatpowerseriesx #1{\XINT_flpowseriesx_chkopt #1\xint_relax }%
\def\XINT_flpowseriesx_chkopt #1%
{%
    \ifx [#1\expandafter\XINT_flpowseriesx_opt
       \else\expandafter\XINT_flpowseriesx_noopt
    \fi
    #1%
}%
\def\XINT_flpowseriesx_noopt  #1\xint_relax #2%
{%
    \expandafter\XINT_flpowseriesx\expandafter
    {\the\numexpr #1\expandafter}\expandafter
    {\the\numexpr #2}\XINTdigits
}%
\def\XINT_flpowseriesx_opt [\xint_relax #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%
```

9 Package xintcfrac implementation

The commenting is currently (2016/12/22) 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.

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@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
```

9.2 Package identification

```
\XINT_providespackage
\ProvidesPackage{xintcfrac}%
  [2016/12/22 1.2j Expandable continued fractions with xint package (JFB)]%
```

9.3 \xintCFrac

```
\def\xintCFrac {\romannumeral0\xintcfrac }%
\def\xintcfrac #1%
{%
    \XINT_cfrac_opt_a #1\xint_relax
}%
\def\XINT_cfrac_opt_a #1%
{%
    \ifx[#1\XINT_cfrac_opt_b\fi \XINT_cfrac_noopt #1%
}%
\def\XINT_cfrac_noopt #1\xint_relax
{%
    \expandafter\XINT_cfrac_A\romannumeral0\xintrawwithzeros {#1}\Z
    \relax\relax
}%
\def\XINT_cfrac_opt_b\fi\XINT_cfrac_noopt [\xint_relax #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}%
```

9.4 \xintGCFrac

```
\def\xintGCFrac {\romannumeral0\xintgcfrac }%
\def\xintgcfrac #1{\XINT_gcfrac_opt_a #1\xint_relax }%
\def\XINT_gcfrac_opt_a #1%
{%
```

```
    \ifx[#1\XINT_gcfrac_opt_b\fi \XINT_gcfrac_noopt #1%
}%
\def\XINT_gcfrac_noopt #1\xint_relax
{%
    \XINT_gcfrac #1+\xint_relax/\relax\relax
}%
\def\XINT_gcfrac_opt_b\fi\XINT_gcfrac_noopt [\xint_relax #1]%
{%
    \fi\csname XINT_gcfrac_opt#1\endcsname
}%
\def\XINT_gcfrac_optl #1%
{%
    \XINT_gcfrac #1+\xint_relax/\relax\hfill
}%
\def\XINT_gcfrac_optc #1%
{%
    \XINT_gcfrac #1+\xint_relax/\relax\relax
}%
\def\XINT_gcfrac_optr #1%
{%
    \XINT_gcfrac #1+\xint_relax/\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_xint_relax #3\XINT_gcfrac_endloop\xint_relax
    \XINT_gcfrac_loop {{#3}{#2}#1}%
}%
\def\XINT_gcfrac_endloop\xint_relax\XINT_gcfrac_loop #1#2#3%
{%
    \XINT_gcfrac_T #2#3#1\xint_relax\xint_relax
}%
\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_xint_relax #5\XINT_gcfrac_end\xint_relax\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_relax\XINT_gcfrac_U #1#2#3%
{%
    \XINT_gcfrac_end_b #3%
}%
\def\XINT_gcfrac_end_b #1\cfrac#2#3{ #3}%
```

9.5 \xintGGCFrac

New with 1.09m

```
\def\xintGGCFrac {\romannumeral0\xintggcfrac }%
\def\xintggcfrac #1{\XINT_ggcfrac_opt_a #1\xint_relax }%
\def\XINT_ggcfrac_opt_a #1%
{%
    \ifx[#1\XINT_ggcfrac_opt_b\fi \XINT_ggcfrac_noopt #1%
}%
\def\XINT_ggcfrac_noopt #1\xint_relax
{%
    \XINT_ggcfrac #1+\xint_relax/\relax\relax
}%
\def\XINT_ggcfrac_opt_b\fi\XINT_ggcfrac_noopt [\xint_relax #1]%
{%
    \fi\csname XINT_ggcfrac_opt#1\endcsname
}%
\def\XINT_ggcfrac_optl #1%
{%
    \XINT_ggcfrac #1+\xint_relax/\relax\hfill
}%
\def\XINT_ggcfrac_optc #1%
{%
    \XINT_ggcfrac #1+\xint_relax/\relax\relax
}%
\def\XINT_ggcfrac_optr #1%
{%
    \XINT_ggcfrac #1+\xint_relax/\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_xint_relax #3\XINT_ggcfrac_endloop\xint_relax
    \XINT_ggcfrac_loop {{#3}{#2}#1}%
}%
\def\XINT_ggcfrac_endloop\xint_relax\XINT_ggcfrac_loop #1#2#3%
{%
    \XINT_ggcfrac_T #2#3#1\xint_relax\xint_relax
}%
\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_xint_relax #5\XINT_ggcfrac_end\xint_relax\XINT_ggcfrac_U
              #1#2{#5+\cfrac{#1#4#2}{#3}}%
}%
\def\XINT_ggcfrac_end\xint_relax\XINT_ggcfrac_U #1#2#3%
{%
    \XINT_ggcfrac_end_b #3%
```

```
}%
\def\XINT_ggcfrac_end_b #1\cfrac#2#3{ #3}%
```

9.6 \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+\xint_relax/}%
\def\XINT_gctgcx_loop_a #1#2#3#4+#5/%
{%
    \xint_gob_til_xint_relax #5\XINT_gctgcx_end\xint_relax
    \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_relax\XINT_gctgcx_loop_b #1#2#3#4{ #1}%
```

9.7 \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}%
```

9.8 \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}}%
```

9.9 \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 {}}%
```

9.10 \xintFtoGC

```
\def\xintFtoGC {\romannumeral0\xintftogc }%
\def\xintftogc {\xintftocx {+1/}}%
```

9.11 \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%
{%
    \xintifZero {#1}{\expandafter\xint_firstofone\xint_gobble_iii}%
                    {\XINT_fgtc_f {#1}}%
}%
\def\XINT_fgtc_f #1#2%
{%
   \xintifZero {#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}%
}%
```

9.12 \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/}%
}%
```

9.13 \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}\xint_relax
}%
\def\xintCtoF {\romannumeral0\xintctof }%
\def\xintctof #1%
{%
    \expandafter\XINT_ctf_prep \romannumeral`&&@#1\xint_relax
}%
\def\XINT_ctf_prep
{%
    \XINT_ctf_loop_a 1001%
}%
\def\XINT_ctf_loop_a #1#2#3#4#5%
{%
    \xint_gob_til_xint_relax #5\XINT_ctf_end\xint_relax
    \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\Z #4\Z }%
    {\romannumeral0\XINT_mul_fork #2\Z #3\Z }%
    {\romannumeral0\xintiiadd {\XINT_mul_fork #2\Z #6\Z}{\XINT_mul_fork #1\Z #4\Z}}%
    {\romannumeral0\xintiiadd {\XINT_mul_fork #2\Z #5\Z}{\XINT_mul_fork #1\Z #3\Z}}%
}%
\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]
```

9.14 \xintiCstoF

```
\def\xintiCstoF {\romannumeral0\xinticstof }%
\def\xinticstof #1%
{%
    \expandafter\XINT_icstf_prep \romannumeral`&&@#1,\xint_relax,%
}%
\def\XINT_icstf_prep
{%
    \XINT_icstf_loop_a 1001%
}%
\def\XINT_icstf_loop_a #1#2#3#4#5,%
{%
    \xint_gob_til_xint_relax #5\XINT_icstf_end\xint_relax
    \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\Z #3\Z}}%
    {\romannumeral0\xintiiadd {#4}{\XINT_mul_fork #1\Z #2\Z}}%
    {#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]
```

9.15 \xintGCtoF

```
\def\xintGCtoF {\romannumeral0\xintgctof }%
\def\xintgctof #1%
```

```
{%
    \expandafter\XINT_gctf_prep \romannumeral`&&@#1+\xint_relax/%
}%
\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\Z #4\Z }%
    {\romannumeral0\XINT_mul_fork #2\Z #3\Z }%
    {\romannumeral0\xintiiadd {\XINT_mul_fork #2\Z #6\Z}{\XINT_mul_fork #1\Z #4\Z}}%
    {\romannumeral0\xintiiadd {\XINT_mul_fork #2\Z #5\Z}{\XINT_mul_fork #1\Z #3\Z}}%
}%
\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_xint_relax #2\XINT_gctf_end\xint_relax
    \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\Z #6\Z }%
    {\romannumeral0\XINT_mul_fork #1\Z #5\Z }%
    {\romannumeral0\XINT_mul_fork #2\Z #4\Z }%
    {\romannumeral0\XINT_mul_fork #2\Z #3\Z }%
}%
\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]
```

9.16 \xintiGCtoF

```
\def\xintiGCtoF {\romannumeral0\xintigctof }%
\def\xintigctof #1%
{%
    \expandafter\XINT_igctf_prep \romannumeral`&&@#1+\xint_relax/%
}%
\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\Z #3\Z}}%
    {\romannumeral0\xintiiadd {#4}{\XINT_mul_fork #1\Z #2\Z}}%
    {#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_xint_relax #4\XINT_igctf_end\xint_relax
    \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\Z #3\Z }%
    {\romannumeral0\XINT_mul_fork #1\Z #2\Z }%
}%
\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]
```

9.17 \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}\xint_relax
}%
\def\xintCtoCv {\romannumeral0\xintctocv }%
\def\xintctocv #1%
{%
    \expandafter\XINT_ctcv_prep\romannumeral`&&@#1\xint_relax
}%
\def\XINT_ctcv_prep
{%
    \XINT_ctcv_loop_a {}1001%
}%
\def\XINT_ctcv_loop_a #1#2#3#4#5#6%
{%
    \xint_gob_til_xint_relax #6\XINT_ctcv_end\xint_relax
    \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\Z #4\Z }%
    {\romannumeral0\XINT_mul_fork #2\Z #3\Z }%
    {\romannumeral0\xintiiadd {\XINT_mul_fork #2\Z #6\Z}{\XINT_mul_fork #1\Z #4\Z}}%
    {\romannumeral0\xintiiadd {\XINT_mul_fork #2\Z #5\Z}{\XINT_mul_fork #1\Z #3\Z}}%
}%
\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}%
```

9.18 \xintiCstoCv

```
\def\xintiCstoCv {\romannumeral0\xinticstocv }%
\def\xinticstocv #1%
{%
    \expandafter\XINT_icstcv_prep \romannumeral`&&@#1,\xint_relax,%
}%
\def\XINT_icstcv_prep
{%
    \XINT_icstcv_loop_a {}1001%
}%
\def\XINT_icstcv_loop_a #1#2#3#4#5#6,%
{%
    \xint_gob_til_xint_relax #6\XINT_icstcv_end\xint_relax
    \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\Z #3\Z}}%
    {\romannumeral0\xintiiadd {#4}{\XINT_mul_fork #1\Z #2\Z}}%
    {{#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]
```

9.19 \xintGCtoCv

```
\def\xintGCtoCv {\romannumeral0\xintgctocv }%
\def\xintgctocv #1%
{%
    \expandafter\XINT_gctcv_prep \romannumeral`&&@#1+\xint_relax/%
}%
\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\Z #4\Z }%
    {\romannumeral0\XINT_mul_fork #2\Z #3\Z }%
    {\romannumeral0\xintiiadd {\XINT_mul_fork #2\Z #6\Z}{\XINT_mul_fork #1\Z #4\Z}}%
    {\romannumeral0\xintiiadd {\XINT_mul_fork #2\Z #5\Z}{\XINT_mul_fork #1\Z #3\Z}}%
}%
\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_xint_relax #3\XINT_gctcv_end\xint_relax
    \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\Z #6\Z }%
    {\romannumeral0\XINT_mul_fork #1\Z #5\Z }%
    {\romannumeral0\XINT_mul_fork #2\Z #4\Z }%
    {\romannumeral0\XINT_mul_fork #2\Z #3\Z }%
}%
\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}%
```

9.20 \xintiGCtoCv

```
\def\xintiGCtoCv {\romannumeral0\xintigctocv }%
\def\xintigctocv #1%
{%
    \expandafter\XINT_igctcv_prep \romannumeral`&&@#1+\xint_relax/%
}%
\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\Z #3\Z}}%
    {\romannumeral0\xintiiadd {#4}{\XINT_mul_fork #1\Z #2\Z}}%
    {{#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_xint_relax #4\XINT_igctcv_end_a\xint_relax
    \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\Z #5\Z }%
    {\romannumeral0\XINT_mul_fork #1\Z #4\Z }%
    {{#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]
```

9.21 \xintFtoCv

Still uses \xinticstocv \xintFtoCs rather than \xintctocv \xintFtoC.

```
\def\xintFtoCv {\romannumeral0\xintftocv }%
\def\xintftocv #1%
{%
    \xinticstocv {\xintFtoCs {#1}}%
}%
```

9.22 \xintFtoCCv

```
\def\xintFtoCCv {\romannumeral0\xintftoccv }%
\def\xintftoccv #1%
{%
    \xintigctocv {\xintFtoCC {#1}}%
}%
```

9.23 \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%
}%
```

9.24 \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%
}%
```

9.25 \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
```

9.26 \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/{ }%
```

9.27 \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/{ }%
```

9.28 \xintCstoGC

```
\def\xintCstoGC {\romannumeral0\xintcstogc }%
\def\xintcstogc #1%
{%
    \expandafter\XINT_cstc_prep \romannumeral`&&@#1,\xint_relax,%
}%
\def\XINT_cstc_prep #1,{\XINT_cstc_loop_a {{#1}}}%
\def\XINT_cstc_loop_a #1#2,%
{%
    \xint_gob_til_xint_relax #2\XINT_cstc_end\xint_relax
    \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_relax\XINT_cstc_loop_b #1#2{ #1}%
```

9.29 \xintGCtoGC

```
\def\xintGCtoGC {\romannumeral0\xintgctogc }%
\def\xintgctogc #1%
{%
    \expandafter\XINT_gctgc_start \romannumeral`&&@#1+\xint_relax/%
}%
\def\XINT_gctgc_start {\XINT_gctgc_loop_a {}}%
\def\XINT_gctgc_loop_a #1#2+#3/%
{%
    \xint_gob_til_xint_relax #3\XINT_gctgc_end\xint_relax
    \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\xint_relax\expandafter\XINT_gctgc_loop_b
{%
    \expandafter\XINT_gctgc_end_b
}%
\def\XINT_gctgc_end_b #1#2#3{ #3{#1}}%
\XINT_restorecatcodes_endinput%
```

10 Package xintexpr implementation

Contents

This is release 1.2j of [2016/12/22].

10.1 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.

10.2 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.

```
1 \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%
```

10.3 Package identification

```
\XINT_providespackage
\ProvidesPackage{xintexpr}%
  [2016/12/22 1.2j Expandable expression parser (JFB)]%
\catcode`! 11
```

10.4 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...). The macro \xintHexToDec could probably be made faster by using techniques similar as the ones 1.2 uses in xintcore.sty.

```
\def\xint_gob_til_! #1!{}% catcode 11 ! default in xintexpr.sty code.
\edef\XINT_expr_lockscan#1!% not used for decimal numbers in xintexpr 1.2
    {\noexpand\expandafter\space\noexpand\csname .=#1\endcsname }%
\edef\XINT_expr_lockit
     #1{\noexpand\expandafter\space\noexpand\csname .=#1\endcsname }%
\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
```

10.5 \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 }%
```

10.6 \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!}%
```

10.7 \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}}%
```

10.8 \xintexpr, \xintiexpr, \xintfloatexpr, \xintiiexpr

```
\def\xintexpr       {\romannumeral0\xinteval      }%
\def\xintiexpr      {\romannumeral0\xintieval     }%
\def\xintfloatexpr  {\romannumeral0\xintfloateval }%
\def\xintiiexpr     {\romannumeral0\xintiieval    }%
```

10.9 \xinttheexpr, \xintttheiexpr, \xintthefloatexpr, \xintttheiiexpr

```
\def\xinttheexpr
   {\romannumeral`&&@\expandafter\XINT_expr_print\romannumeral0\xintbareeval  }%
\def\xintttheiexpr     {\romannumeral`&&@\xintthe\xintiexpr }%
\def\xintthefloatexpr {\romannumeral`&&@\xintthe\xintfloatexpr }%
\def\xintttheiiexpr
   {\romannumeral`&&@\expandafter\XINT_iiexpr_print\romannumeral0\xintbareiieval }%
```

10.10 \xintthe

```
\def\xintthe #1{\romannumeral`&&@\expandafter\xint_gobble_iii\romannumeral`&&@#1}%
```

10.11 \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    \xintttheiexpr
\let\thexintfloatexpr\xintthefloatexpr
\let\thexintiiexpr   \xintttheiiexpr
```

10.12 \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^{}%
```

10.13 \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 }%
```

10.14 \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}%
```

10.15 \xinteval, \xintiieval

```
\def\xinteval   {\expandafter\XINT_expr_wrap\romannumeral0\xintbareeval }%
\def\xintiieval {\expandafter\XINT_iiexpr_wrap\romannumeral0\xintbareiieval }%
```

10.16 \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
}%
```

10.17 \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!%
}%
\catcode`: 12
    \def\XINT_expr_unlock_sp #1.;#2.=#3!{{#2}{#3}}%
\catcode`: 11
```

10.18 \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 }%
```

10.19 \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}}%
```

10.20 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.

10.20.1 \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}%
```

10.20.2 \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.
```

10.20.3 \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}}%
```

10.20.4 \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}}%
```

10.20.5 \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}}%
```

10.20.6 \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}}%
```

10.20.7 \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_relax #2]{#1},{#2}}%
\def\XINT_pfloat::_e #1,#2#3{\XINT_pfloat::_b {#2}{#3, #1}}%
```

10.21 \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\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 }%
```

10.22 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_scanhex_I\fi
    \if `#1\xint_dothis {\XINT_expr_onlitteral_`}\fi
    \ifnum \xint_c_ix<1#1 \xint_dothis \XINT_expr_startint\fi
    \xint_orthat \XINT_expr_scanfunc #1%
}%
\def\XINT_expr_onlitteral_` #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}%
```

10.22.1 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.

```
\def\XINT_expr_scanint_c\string #1\XINT_expr_scanint_d
{%
    \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    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}%
}%
```

10.22.2 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    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\xint_c_i.%
{%
    \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}%
}%
```

10.22.3 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_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 {]\endcsname #1}%
```

10.22.4 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).

```
\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%
           \expandafter\xint_firstoftwo
       \else % gather what we got so far, leave catcode 12 #1 in stream
           \expandafter\xint_secondoftwo
```

```
        \fi
        {\expandafter\XINT_expr_scanhex_transition}%
        {\xint_afterfi {.>\endcsname\endcsname}}%
    \fi
    #1%
}%
\def\XINT_expr_scanhexI_b #1#2%
{%
    #1\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
       \xint_afterfi {\endcsname\endcsname}%
    \fi
    #1%
}%
\def\XINT_expr_scanhexII_b #1#2%
{%
    #1\expandafter\XINT_expr_scanhexII_a\romannumeral`&&@#2%
}%
```

10.22.5 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 "onlitteral" 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}}%
```

10.23 \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.

```
\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
    \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}%
```

10.24 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
```

10.25 |, ||, &, &&, <, >, =, ==, <=, >=, !=, +, −, *, /, ^, **, //, /:, .., ..[,]..,][,][:, :], and ++ operators

10.25.1 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 .=\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
```

10.25.2 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%
{%
  \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 .=#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_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}   {xintNeq}%
\XINT_expr_defbin_b {flexpr}{!=} {v}{vi}   {xintNeq}%
\XINT_expr_defbin_b {iiexpr}{!=} {v}{vi}   {xintiiNeq}%
\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}  {//} {vii}{vii}{xintDivTrunc}%
\XINT_expr_defbin_b {flexpr}{//} {vii}{vii}{xintDivTrunc}%
\XINT_expr_defbin_b {iiexpr}{//} {vii}{vii}{xintiiDivTrunc}%
\XINT_expr_defbin_b {expr}  {/:} {vii}{vii}{xintMod}%
\XINT_expr_defbin_b {flexpr}{/:} {vii}{vii}{xintMod}%
\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}%
\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}%
```

10.25.3 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_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_a\expandafter
      {\expandafter\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_a\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}%
```

10.25.4 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
}%
```

10.25.5 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
```

```
}%
```

10.26 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
         .=:\xintNum{\XINT_expr_unlock ##1};\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 .=]\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 \expandafter\noexpand
                          \csname XINT_expr_itself_][\endcsname #10\string :}%
\let\XINT_flexpr_op_][: \XINT_expr_op_][:
\let\XINT_iiexpr_op_][: \XINT_expr_op_][:
\catcode`[ 12 \catcode`] 12
```

10.26.1 \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:_nth #1;#2;{\xintNthEltPy:f:csv {\xintNum{#1}}{#2}}%
\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:_: #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;}%
```

10.26.2 \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_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;}%
```

10.26.3 \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
```

10.26.4 \xintKeep:f:csv

1.2g, code in xinttools.sty. Refactored in 1.2j.

10.26.5 \xintTrim:f:csv

1.2g, code in xinttools.sty. Refactored in 1.2j.

10.26.6 \xintNthEltPy:f:csv

1.2g, code in xinttools.sty. Refactored in 1.2j.

10.26.7 \xintLength:f:csv

1.2g, code in xinttools.sty. Refactored in 1.2j.

10.26.8 \xintReverse:f:csv

1.2g, code in xinttools.sty.

10.27 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.

10.27.1 \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,{ }%
```

10.27.2 \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,{ }%
```

10.28 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.

10.28.1 \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}%
```

10.28.2 \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}%
```

10.28.3 \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}%
```

10.28.4 \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}%
```

10.29 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}%
}%
```

10.30 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 .=#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}}%
```

10.31 ? 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}%
}%
```

10.32 ! as postfix factorial operator

A float version \xintFloatFac was at last done 2015/10/06 for 1.2. Attention 2015/11/29 for 1.2f: no more \xintFac, but \xintiFac.

```
\let\XINT_expr_precedence_! \xint_c_x
\def\XINT_expr_op_! #1{\expandafter\XINT_expr_getop
                                    \csname .=\xintiFac{\XINT_expr_unlock #1}\endcsname }%
\def\XINT_flexpr_op_! #1{\expandafter\XINT_expr_getop
                                    \csname .=\XINTinFloatFac{\XINT_expr_unlock #1}\endcsname }%
\def\XINT_iiexpr_op_! #1{\expandafter\XINT_expr_getop
                                   \csname .=\xintiiFac{\XINT_expr_unlock #1}\endcsname }%
```

10.33 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
\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
\catcode`* 12
```

10.34 \XINT_expr_op_` for recognizing functions

The "onlitteral" 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 "onlitteral" 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 onlitteral_<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_onlitteral_##1\endcsname
          \xint_dothis{\csname XINT_expr_onlitteral_##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 }%
```

10.35 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_onlitteral_bool #1)%
        {\expandafter\XINT_expr_getop\csname .=\xintBool{#1}\endcsname }%
\def\XINT_expr_onlitteral_togl #1)%
        {\expandafter\XINT_expr_getop\csname .=\xintToggle{#1}\endcsname }%
\def\XINT_expr_onlitteral_protect #1)%
        {\expandafter\XINT_expr_getop\csname .=\detokenize{#1}\endcsname }%
```

10.36 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
```

10.37 The qint, qfrac, qfloat "functions"

New with 1.2. Allows the user to hand over quickly a big number to the parser, spaces not immediately removed but should be harmless in general.

```
\def\XINT_expr_onlitteral_qint #1)%
        {\expandafter\XINT_expr_getop\csname .=\xintiNum{#1}\endcsname }%
\def\XINT_expr_onlitteral_qfrac #1)%
        {\expandafter\XINT_expr_getop\csname .=\xintRaw{#1}\endcsname }%
\def\XINT_expr_onlitteral_qfloat #1)%
        {\expandafter\XINT_expr_getop\csname .=\XINTinFloatdigits{#1}\endcsname }%
```

10.38 \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__
```

10.39 User defined variables: \xintdefvar, \xintdefiivar, \xintdeffloatvar

1.1 An active : character will be a pain with our delimited macros and I almost decided not to use := but rather = as assignation operator, but this is the same problem inside expressions with the modulo operator /:, or with babel+frenchb with all high punctuation ?, !, :, ;.

Variable names may contain letters, digits, underscores, and must not start with a digit. Names starting with @ or un underscore are reserved.

Note (2015/11/11): although defined since october 2014 with 1.1, they were only very briefly mentioned in the user documentation, I should have expanded more. I am now adding functions to variables, and will rewrite entirely the documentation of xintexpr.sty.

1.2c adds the "onlitteral" macros as we changed our tricks to disambiguate variables from functions if followed by a parenthesis, in order to allow function names to have precedence on variable names.

I don't issue warnings if a an attempt to define a variable name clashes with a pre-existing function name, as I would have to check expr, iiexpr and also floatexpr. And anyhow overloading a function name with a variable name is allowed, the only thing to know is that if an opening parenthesis follows it is the function meaning which prevails.

2015/11/13: I now first do an a priori complete expansion of #1, and then apply \detokenize to the result, and remove spaces.

2015/11/21: finally I do not detokenize the variable name. Because this complicated the \xintunassignvar if it did the same and we wanted to use it to redeclare a letter as dummy variable.

Documentation of 1.2d said that the tacit multiplication always was done with increased precedence, but I had not at that time made up my mind for the case of variable(stuff) and pushed to CTAN early because I need to fix the bug I had introduced in 1.2c which itself I had pushed to CTAN early because I had to fix the 1.2 bug with subtraction....

Finally I decide to do it indeed. Hence for 1.2e. This only impacts situations such as A/B(stuff), which are thus interpreted as A/(B*(stuff)).

```
\catcode`* 11
\def\XINT_expr_defvar #1#2#3;{%
   \edef\XINT_expr_tmpa{#2}%
   \edef\XINT_expr_tmpa {\xint_zapspaces_o\XINT_expr_tmpa}%
   \ifnum\expandafter\xintLength\expandafter{\XINT_expr_tmpa}=\z@
        \xintMessage {xintexpr}{Warning}
         {Error: impossible to declare variable with empty name.}%
   \else
     \edef\XINT_expr_tmpb {\romannumeral0#1#3\relax }%
     \expandafter\edef\csname XINT_expr_var_\XINT_expr_tmpa\endcsname
              {\expandafter\noexpand\XINT_expr_tmpb}%
     \expandafter\edef\csname XINT_expr_onlitteral_\XINT_expr_tmpa\endcsname
              {\XINT_expr_precedence_*** *\expandafter\noexpand\XINT_expr_tmpb (}%
     \ifxintverbose\xintMessage {xintexpr}{Info}
       {Variable "\XINT_expr_tmpa" defined with value
               \expandafter\XINT_expr_unlock\XINT_expr_tmpb.}%
     \fi
   \fi
}%
\catcode`* 12
\catcode`: 12
\def\xintdefvar      #1:={\XINT_expr_defvar\xintbareeval      {#1}}%
\def\xintdefiivar    #1:={\XINT_expr_defvar\xintbareiieval    {#1}}%
\def\xintdeffloatvar #1:={\XINT_expr_defvar\xintbarefloateval {#1}}%
\catcode`: 11
```

10.40 \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\XINT_expr_makedummy \XINT_expr_tmpa
         \ifxintverbose\xintMessage {xintexpr}{Info}
           {Character \XINT_expr_tmpa\space usable as dummy variable (if letter).}%
         \fi
       \else
       \expandafter\edef\csname XINT_expr_var_\XINT_expr_tmpa\endcsname
           {\csname .=0\endcsname\noexpand\XINT_expr_undefined {\XINT_expr_tmpa}}%
```

```
1427        \expandafter\edef\csname XINT_expr_onlitteral_\XINT_expr_tmpa\endcsname
1428            {\csname .=0\endcsname\noexpand\XINT_expr_undefined {\XINT_expr_tmpa}*}%
1429          \ifxintverbose\xintMessage {xintexpr}{Info}
1430            {Variable \XINT_expr_tmpa\space has been ``unassigned''.}%
1431          \fi
1432        \fi
1433    \else
1434        \xintMessage {xintexpr}{Warning}
1435            {Error: there was no such variable \XINT_expr_tmpa\space to unassign.}%
1436    \fi
1437 }%
1438 \def\XINT_expr_undefined #1{\xintError:replaced_by_zero\xint_gobble_i {#1}}%
```

10.41 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_onlitteral_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.

10.41.1 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_onlitteral_<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 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. Public interface is via \xintunassignvar which one can use with a letter to re-declare it as dummy variable. 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%
{%
   \expandafter\def\csname XINT_expr_var_#1\endcsname ##1\relax !#1##2%
      {##2##1\relax !#1##2}%
   \expandafter\def\csname XINT_expr_onlitteral_#1\endcsname ##1\relax !#1##2%
      {\XINT_expr_precedence_*** *##2(##1\relax !#1##2}%
}%
\xintApplyUnbraced \XINT_expr_makedummy {abcdefghijklmnopqrstuvwxyz}%
\xintApplyUnbraced \XINT_expr_makedummy {ABCDEFGHIJKLMNOPQRSTUVWXYZ}%
\edef\XINT_expr_var_nil  {\expandafter\noexpand\csname .= \endcsname}%
\edef\XINT_expr_onlitteral_nil
      {\XINT_expr_precedence_*** *\expandafter\noexpand\csname .= \endcsname (}%
\catcode`* 12
```

10.41.2 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_!?
```

10.41.3 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 "onlitteral" 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_onlitteral_@ #1~#2{\XINT_expr_precedence_*** *#2(#1~#2}%
\expandafter\let\csname XINT_expr_onlitteral_@1\endcsname \XINT_expr_onlitteral_@
\expandafter\def\csname XINT_expr_onlitteral_@2\endcsname #1~#2#3%
            {\XINT_expr_precedence_*** *#3(#1~#2#3}%
\expandafter\def\csname XINT_expr_onlitteral_@3\endcsname #1~#2#3#4%
            {\XINT_expr_precedence_*** *#4(#1~#2#3#4}%
\expandafter\def\csname XINT_expr_onlitteral_@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
```

10.41.4 \XINT_expr_onlitteral_seq

```
\def\XINT_expr_onlitteral_seq
 {\expandafter\XINT_expr_onlitteral_seq_f\romannumeral`&&@\XINT_expr_onlitteral_seq_a {}}%
\def\XINT_expr_onlitteral_seq_f #1#2{\xint_c_xviii `{seqx}#2)\relax #1}%
```

10.41.5 \XINT_expr_onlitteral_seq_a

```
\def\XINT_expr_onlitteral_seq_a #1#2,%
{%
    \ifcase\XINT_isbalanced_a \relax #1#2(\xint_bye)\xint_bye
           \expandafter\XINT_expr_onlitteral_seq_c
       \or\expandafter\XINT_expr_onlitteral_seq_b
     \else\expandafter\xintError:we_are_doomed
    \fi {#1#2},%
}%
\def\XINT_expr_onlitteral_seq_b #1,{\XINT_expr_onlitteral_seq_a {#1,}}%
\def\XINT_expr_onlitteral_seq_c #1,#2#3% #3 pour absorber le =
{%
    \XINT_expr_onlitteral_seq_d {#2{#1}}{}%
}%
\def\XINT_expr_onlitteral_seq_d #1#2#3)%
{%
    \ifcase\XINT_isbalanced_a \relax #2#3(\xint_bye)\xint_bye
        \or\expandafter\XINT_expr_onlitteral_seq_e
      \else\expandafter\xintError:we_are_doomed
    \fi
    {#1}{#2#3}%
}%
\def\XINT_expr_onlitteral_seq_e #1#2{\XINT_expr_onlitteral_seq_d {#1}{#2)}}%
```

10.41.6 \XINT_isbalanced_a for \XINT_expr_onlitteral_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 }%
```

10.41.7 \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}%
```

10.41.8 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? { }%
```

10.41.9 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}%
```

10.42 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_onlitteral_add
 {\expandafter\XINT_expr_onlitteral_add_f\romannumeral`&&@\XINT_expr_onlitteral_seq_a {}}%
\def\XINT_expr_onlitteral_add_f #1#2{\xint_c_xviii `{opxadd}#2)\relax #1}%
\def\XINT_expr_onlitteral_mul
 {\expandafter\XINT_expr_onlitteral_mul_f\romannumeral`&&@\XINT_expr_onlitteral_seq_a {}}%
\def\XINT_expr_onlitteral_mul_f #1#2{\xint_c_xviii `{opxmul}#2)\relax #1}%
```

10.42.1 \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`&&@#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}%
```

10.43 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_onlitteral_subs
 {\expandafter\XINT_expr_onlitteral_subs_f\romannumeral`&&@\XINT_expr_onlitteral_seq_a {}}%
\def\XINT_expr_onlitteral_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}%
```

10.44 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_onlitteral_seq_a {}%
}%
```

10.44.1 \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%
}%
```

10.44.2 \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}%
```

10.44.3 \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}%
```

10.44.4 \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}%
```

10.45 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_onlitteral_seq_a {}%
}%
```

10.45.1 \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%
}%
```

10.45.2 \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}%
```

10.45.3 \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}%
```

10.45.4 \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}%
```

10.46 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_onlitteral_seq_a {}%
}%
```

10.46.1 \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%
}%
```

10.46.2 \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 \xintthebarefloateval 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}%
```

10.46.3 \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}%
```

10.46.4 \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}}%
```

10.47 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_onlitteral_seq_a {}%
}%
```

10.47.1 \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%
}%
```

10.47.2 \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}%
```

10.47.3 \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}%
```

10.47.4 \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
```

10.48 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)

10.48.1 \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
```

10.48.2 \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
```

10.48.3 \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)
```

10.48.4 Generic csv routine

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}%
```

10.48.5 \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}%
```

10.48.6 \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}%
```

10.48.7 \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}%
```

10.48.8 \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}%
```

10.48.9 \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}%
```

10.48.10 \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}%
```

10.48.11 \XINTinFloatdigits, \XINTinFloatSqrtdigits, \XINTinFloatFacdigits

for \xintNewExpr matters, mainly.

```
\def\XINTinFloatdigits     {\XINTinFloat    [\XINTdigits]}%
\def\XINTinFloatSqrtdigits {\XINTinFloatSqrt[\XINTdigits]}%
\def\XINTinFloatFacdigits  {\XINTinFloatFac [\XINTdigits]}%
```

10.48.12 \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]}}%
```

10.48.13 \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]}}%
```

10.49 The num, reduce, abs, sgn, frac, floor, ceil, sqr, sqrt, sqrtr, float, round, trunc, mod, quo, rem, gcd, lcm, max, min, `+`, `*`, ?, !, not, all, any, xor, if, ifsgn, even, odd, first, last, len, reversed, factorial and binomial functions

```
\def\XINT_expr_twoargs #1,#2,{{#1}{#2}}%
\def\XINT_expr_argandopt #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_oneortwo #1#2#3,#4,#5.%
{%
    \if\relax#5\relax\expandafter\xint_firstoftwo\else
                       \expandafter\xint_secondoftwo\fi
    {#1{0}}{#2{\xintNum {#4}}}{#3}%
```

```
}%
\def\XINT_iiexpr_oneortwo #1#2,#3,#4.%
{%
    \if\relax#4\relax\expandafter\xint_firstoftwo\else
                       \expandafter\xint_secondoftwo\fi
    {#1{0}}{#1{#3}}{#2}%
}%
\def\XINT_expr_func_num #1#2#3%
    {\expandafter #1\expandafter #2\csname.=\xintNum {\XINT_expr_unlock #3}\endcsname }%
\let\XINT_flexpr_func_num\XINT_expr_func_num
\let\XINT_iiexpr_func_num\XINT_expr_func_num
% [0] added Oct 25. For interaction with SPRaw::csv
\def\XINT_expr_func_reduce #1#2#3%
  {\expandafter #1\expandafter #2\csname.=\xintIrr {\XINT_expr_unlock #3}[0]\endcsname }%
\let\XINT_flexpr_func_reduce\XINT_expr_func_reduce
% no \XINT_iiexpr_func_reduce
\def\XINT_expr_func_abs #1#2#3%
  {\expandafter #1\expandafter #2\csname.=\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.=\xintiiAbs {\XINT_expr_unlock #3}\endcsname }%
\def\XINT_expr_func_sgn #1#2#3%
  {\expandafter #1\expandafter #2\csname.=\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.=\xintiiSgn {\XINT_expr_unlock #3}\endcsname }%
\def\XINT_expr_func_frac #1#2#3%
  {\expandafter #1\expandafter #2\csname.=\xintTFrac {\XINT_expr_unlock #3}\endcsname }%
\def\XINT_flexpr_func_frac #1#2#3{\expandafter #1\expandafter #2\csname
    .=\XINTinFloatFracdigits {\XINT_expr_unlock #3}\endcsname }%
```

no \XINT_iiexpr_func_frac

```
\def\XINT_expr_func_floor #1#2#3%
  {\expandafter #1\expandafter #2\csname .=\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.=\xintiFloor {\XINT_expr_unlock #3}\endcsname }%
\def\XINT_expr_func_ceil #1#2#3%
  {\expandafter #1\expandafter #2\csname .=\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.=\xintiCeil {\XINT_expr_unlock #3}\endcsname }%
\def\XINT_expr_func_sqr #1#2#3%
    {\expandafter #1\expandafter #2\csname.=\xintSqr {\XINT_expr_unlock #3}\endcsname }%
\def\XINT_flexpr_func_sqr #1#2#3%
{%
    \expandafter #1\expandafter #2\csname
```

```
    .=\XINTinFloatMul{\XINT_expr_unlock #3}{\XINT_expr_unlock #3}\endcsname
}%
\def\XINT_expr_func_factorial #1#2#3%
{%
    \expandafter #1\expandafter #2\csname .=%
    \expandafter\XINT_expr_argandopt
    \romannumeral`&&@\XINT_expr_unlock#3,,.\xintiFac\XINTinFloatFac
    \endcsname
}%
\def\XINT_flexpr_func_factorial #1#2#3%
{%
    \expandafter #1\expandafter #2\csname .=%
    \expandafter\XINT_expr_argandopt
    \romannumeral`&&@\XINT_expr_unlock#3,,.\XINTinFloatFacdigits\XINTinFloatFac
    \endcsname
}%
\def\XINT_iiexpr_func_factorial #1#2#3%
{%
    \expandafter #1\expandafter #2\csname.=\xintiiFac{\XINT_expr_unlock #3}\endcsname
}%
\def\XINT_iiexpr_func_sqr #1#2#3%
  {\expandafter #1\expandafter #2\csname.=\xintiiSqr {\XINT_expr_unlock #3}\endcsname }%
\def\XINT_expr_func_sqrt #1#2#3%
{%
    \expandafter #1\expandafter #2\csname .=%
    \expandafter\XINT_expr_argandopt
    \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.=\xintiiSqrt {\XINT_expr_unlock #3}\endcsname }%
\def\XINT_iiexpr_func_sqrtr #1#2#3%
 {\expandafter #1\expandafter #2\csname.=\xintiiSqrtR {\XINT_expr_unlock #3}\endcsname }%
\def\XINT_expr_func_round #1#2#3%
{%
    \expandafter #1\expandafter #2\csname .=%
    \expandafter\XINT_expr_oneortwo
    \expandafter\xintiRound\expandafter\xintRound
    \romannumeral`&&@\XINT_expr_unlock #3,,.\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_oneortwo\expandafter\xintiRound
    \romannumeral`&&@\XINT_expr_unlock #3,,.\endcsname
}%
\def\XINT_expr_func_trunc #1#2#3%
{%
    \expandafter #1\expandafter #2\csname .=%
    \expandafter\XINT_expr_oneortwo
```

```
    \expandafter\xintiTrunc\expandafter\xintTrunc
    \romannumeral`&&@\XINT_expr_unlock #3,,.\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_oneortwo\expandafter\xintiTrunc
    \romannumeral`&&@\XINT_expr_unlock #3,,.\endcsname
}%
\def\XINT_expr_func_float #1#2#3%
{%
    \expandafter #1\expandafter #2\csname .=%
    \expandafter\XINT_expr_argandopt
    \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_mod #1#2#3%
{%
    \expandafter #1\expandafter #2\csname .=%
    \expandafter\expandafter\expandafter\xintMod
    \expandafter\XINT_expr_twoargs
    \romannumeral`&&@\XINT_expr_unlock #3,\endcsname
}%
\def\XINT_flexpr_func_mod #1#2#3%
{%
    \expandafter #1\expandafter #2\csname .=%
    \expandafter\XINTinFloatMod
    \romannumeral`&&@\expandafter\XINT_expr_twoargs
    \romannumeral`&&@\XINT_expr_unlock #3,\endcsname
}%
\def\XINT_iiexpr_func_mod #1#2#3%
{%
    \expandafter #1\expandafter #2\csname .=%
    \expandafter\expandafter\expandafter\xintiiMod
    \expandafter\XINT_expr_twoargs
    \romannumeral`&&@\XINT_expr_unlock #3,\endcsname
}%
\def\XINT_expr_func_binomial #1#2#3%
{%
    \expandafter #1\expandafter #2\csname .=%
    \expandafter\expandafter\expandafter\xintiBinomial
    \expandafter\XINT_expr_twoargs
    \romannumeral`&&@\XINT_expr_unlock #3,\endcsname
}%
\def\XINT_flexpr_func_binomial #1#2#3%
{%
    \expandafter #1\expandafter #2\csname .=%
    \expandafter\expandafter\expandafter\XINTinFloatBinomial
    \expandafter\XINT_expr_twoargs
```

```
    \romannumeral`&&@\XINT_expr_unlock #3,\endcsname
}%
\def\XINT_iiexpr_func_binomial #1#2#3%
{%
    \expandafter #1\expandafter #2\csname .=%
    \expandafter\expandafter\expandafter\xintiiBinomial
    \expandafter\XINT_expr_twoargs
    \romannumeral`&&@\XINT_expr_unlock #3,\endcsname
}%
\def\XINT_expr_func_pfactorial #1#2#3%
{%
    \expandafter #1\expandafter #2\csname .=%
    \expandafter\expandafter\expandafter\xintiPFactorial
    \expandafter\XINT_expr_twoargs
    \romannumeral`&&@\XINT_expr_unlock #3,\endcsname
}%
\def\XINT_flexpr_func_pfactorial #1#2#3%
{%
    \expandafter #1\expandafter #2\csname .=%
    \expandafter\expandafter\expandafter\XINTinFloatPFactorial
    \expandafter\XINT_expr_twoargs
    \romannumeral`&&@\XINT_expr_unlock #3,\endcsname
}%
\def\XINT_iiexpr_func_pfactorial #1#2#3%
{%
    \expandafter #1\expandafter #2\csname .=%
    \expandafter\expandafter\expandafter\xintiiPFactorial
    \expandafter\XINT_expr_twoargs
    \romannumeral`&&@\XINT_expr_unlock #3,\endcsname
}%
\def\XINT_expr_func_quo #1#2#3%
{%
    \expandafter #1\expandafter #2\csname .=%
    \expandafter\expandafter\expandafter\xintiQuo
    \expandafter\XINT_expr_twoargs
    \romannumeral`&&@\XINT_expr_unlock #3,\endcsname
}%
\let\XINT_flexpr_func_quo\XINT_expr_func_quo
\def\XINT_iiexpr_func_quo #1#2#3%
{%
    \expandafter #1\expandafter #2\csname .=%
    \expandafter\expandafter\expandafter\xintiiQuo
    \expandafter\XINT_expr_twoargs
    \romannumeral`&&@\XINT_expr_unlock #3,\endcsname
}%
\def\XINT_expr_func_rem #1#2#3%
{%
    \expandafter #1\expandafter #2\csname .=%
    \expandafter\expandafter\expandafter\xintiRem
    \expandafter\XINT_expr_twoargs
    \romannumeral`&&@\XINT_expr_unlock #3,\endcsname
}%
```

```
\let\XINT_flexpr_func_rem\XINT_expr_func_rem
\def\XINT_iiexpr_func_rem #1#2#3%
{%
    \expandafter #1\expandafter #2\csname .=%
    \expandafter\expandafter\expandafter\xintiiRem
    \expandafter\XINT_expr_twoargs
    \romannumeral`&&@\XINT_expr_unlock #3,\endcsname
}%
\def\XINT_expr_func_gcd #1#2#3%
   {\expandafter #1\expandafter #2\csname
                                      .=\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
                                      .=\xintiiGCDof:csv{\XINT_expr_unlock #3}\endcsname }%
\def\XINT_expr_func_lcm #1#2#3%
    {\expandafter #1\expandafter #2\csname
                                      .=\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
                                      .=\xintiiLCMof:csv{\XINT_expr_unlock #3}\endcsname }%
\def\XINT_expr_func_max #1#2#3%
    {\expandafter #1\expandafter #2\csname
                                      .=\xintMaxof:csv{\XINT_expr_unlock #3}\endcsname }%
\def\XINT_iiexpr_func_max #1#2#3%
    {\expandafter #1\expandafter #2\csname
                                     .=\xintiiMaxof:csv{\XINT_expr_unlock #3}\endcsname }%
\def\XINT_flexpr_func_max #1#2#3%
    {\expandafter #1\expandafter #2\csname
                                 .=\XINTinFloatMaxof:csv{\XINT_expr_unlock #3}\endcsname }%
\def\XINT_expr_func_min #1#2#3%
    {\expandafter #1\expandafter #2\csname
                                      .=\xintMinof:csv{\XINT_expr_unlock #3}\endcsname }%
\def\XINT_iiexpr_func_min #1#2#3%
    {\expandafter #1\expandafter #2\csname
                                     .=\xintiiMinof:csv{\XINT_expr_unlock #3}\endcsname }%
\def\XINT_flexpr_func_min #1#2#3%
    {\expandafter #1\expandafter #2\csname
                                 .=\XINTinFloatMinof:csv{\XINT_expr_unlock #3}\endcsname }%
\expandafter\def\csname XINT_expr_func_+\endcsname #1#2#3%
    {\expandafter #1\expandafter #2\csname
                                        .=\xintSum:csv{\XINT_expr_unlock #3}\endcsname }%
\expandafter\def\csname XINT_flexpr_func_+\endcsname #1#2#3%
    {\expandafter #1\expandafter #2\csname
                                   .=\XINTinFloatSum:csv{\XINT_expr_unlock #3}\endcsname }%
\expandafter\def\csname XINT_iiexpr_func_+\endcsname #1#2#3%
    {\expandafter #1\expandafter #2\csname
                                      .=\xintiiSum:csv{\XINT_expr_unlock #3}\endcsname }%
\expandafter\def\csname XINT_expr_func_*\endcsname #1#2#3%
    {\expandafter #1\expandafter #2\csname
                                        .=\xintPrd:csv{\XINT_expr_unlock #3}\endcsname }%
```

```
\expandafter\def\csname XINT_flexpr_func_*\endcsname #1#2#3%
    {\expandafter #1\expandafter #2\csname
                                  .=\XINTinFloatPrd:csv{\XINT_expr_unlock #3}\endcsname }%
\expandafter\def\csname XINT_iiexpr_func_*\endcsname #1#2#3%
    {\expandafter #1\expandafter #2\csname
                                       .=\xintiiPrd:csv{\XINT_expr_unlock #3}\endcsname }%
\def\XINT_expr_func_? #1#2#3%
    {\expandafter #1\expandafter #2\csname
                                    .=\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.=\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.=\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_all #1#2#3%
    {\expandafter #1\expandafter #2\csname
                                       .=\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
                                        .=\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
                                       .=\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\xintifNotZero: #1,#2,#3,{\xintiiifNotZero{#1}{#2}{#3}}%
\def\XINT_expr_func_if #1#2#3%
    {\expandafter #1\expandafter #2\csname
         .=\expandafter\xintifNotZero:\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\xintifSgn: #1,#2,#3,#4,{\xintiiifSgn{#1}{#2}{#3}{#4}}%
\def\XINT_expr_func_ifsgn #1#2#3%
{%
    \expandafter #1\expandafter #2\csname
         .=\expandafter\xintifSgn:\romannumeral`&&@\XINT_expr_unlock #3,\endcsname
}%
\let\XINT_flexpr_func_ifsgn\XINT_expr_func_ifsgn
\let\XINT_iiexpr_func_ifsgn\XINT_expr_func_ifsgn
\def\XINT_expr_func_len #1#2#3%
    {\expandafter#1\expandafter#2%
     \csname.=\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
\def\XINT_expr_func_first #1#2#3%
    {\expandafter #1\expandafter #2\csname.=\expandafter\XINT_expr_func_firsta
     \romannumeral`&&@\XINT_expr_unlock #3,^\endcsname }%
\def\XINT_expr_func_firsta #1,#2^{#1}%
\let\XINT_flexpr_func_first\XINT_expr_func_first
\let\XINT_iiexpr_func_first\XINT_expr_func_first
\def\XINT_expr_func_last #1#2#3% will not work in \xintNewExpr if macro param involved
    {\expandafter #1\expandafter #2\csname.=\expandafter\XINT_expr_func_lasta
     \romannumeral`&&@\XINT_expr_unlock #3,^\endcsname }%
\def\XINT_expr_func_lasta #1,#2%
   {\if ^#2 #1\expandafter\xint_gobble_ii\fi \XINT_expr_func_lasta #2}%
\let\XINT_flexpr_func_last\XINT_expr_func_last
\let\XINT_iiexpr_func_last\XINT_expr_func_last
\def\XINT_expr_func_odd #1#2#3%
    {\expandafter #1\expandafter #2\csname.=\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.=\xintiiOdd{\XINT_expr_unlock #3}\endcsname}%
\def\XINT_expr_func_even #1#2#3%
    {\expandafter #1\expandafter #2\csname.=\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.=\xintiiEven{\XINT_expr_unlock #3}\endcsname}%
\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
```

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 .=%
                    \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
```

10.50 f-expandable versions of the \xintSeqB::csv and alike routines, for \xintNewExpr

10.50.1 \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}%
```

10.50.2 \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}%
}%
```

10.50.3 \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}%
}%
```

10.51 User defined functions: \xintdeffunc, \xintdefiifunc, \xintdeffloatfunc

1.2c (November 11-12, 2015). It is possible to overload a variable name with a function name (and conversely). The function interpretation with be used only if followed by an opening parenthesis, disabling the tacit multiplication usually applied to variables. Crazy things such as add(f(f), f=1..10) are possible if there is a function "f". Or we can use "e" both for an exponential function and the Euler constant.

2015/11/13: function candidates names first completely expanded, then detokenized and cleaned of spaces.
2015/11/21: no more \detokenize on the function names. Also I use #1(#2)#3:=#4 rather than #1(#2):=#3. Ah, rather #1(#2)#3=#4, then I don't have to worry about active :.
2016/02/22: 1.2f la macro associée à la fonction ne débute plus par un \romannumeral, de toute façon est pour emploi dans \csname..\endcsname.
2016/03/08: adding a pair of braces thus allowing comma separated expressions; until then the user had to do \xintdeffunc foo(x,..):=(.., .., ..)\relax.

```
\catcode`: 12
\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#3\csname XINT_#2_userfunc_\XINT_expr_tmpa\endcsname
                               [\XINT_expr_tmpb]{\XINT_expr_tmpc}%
   \expandafter\XINT_expr_defuserfunc
     \csname XINT_#2_func_\XINT_expr_tmpa\expandafter\endcsname
     \csname XINT_#2_userfunc_\XINT_expr_tmpa\endcsname
   \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 meaning \expandafter\meaning
         \csname XINT_#2_userfunc_\XINT_expr_tmpa\endcsname}%
   \fi
  }%
}%
\catcode`: 11
\XINT_tmpa\xintdeffunc     {expr}  \XINT_NewFunc     {expr}%
\XINT_tmpa\xintdefiifunc   {iiexpr}\XINT_NewIIFunc   {iiexpr}%
\XINT_tmpa\xintdeffloatfunc{flexpr}\XINT_NewFloatFunc{floatexpr}%
\def\XINT_expr_defuserfunc #1#2{%
    \def #1##1##2##3{\expandafter ##1\expandafter ##2%
     \csname .=\expandafter #2\romannumeral`&&@\XINT_expr_unlock ##3,\endcsname
    }%
}%
```

10.52 \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
  \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
    \csname XINT_expr_macrofunc_\XINT_expr_tmpa\endcsname\xintbareeval
  \expandafter\XINT_expr_newfunction
    \csname XINT_iiexpr_func_\XINT_expr_tmpa\expandafter\endcsname
    \csname XINT_expr_macrofunc_\XINT_expr_tmpa\endcsname\xintbareiieval
  \expandafter\XINT_expr_newfunction
    \csname XINT_flexpr_func_\XINT_expr_tmpa\expandafter\endcsname
    \csname XINT_expr_macrofunc_\XINT_expr_tmpa\endcsname\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 meaning \expandafter\meaning
         \csname XINT_expr_macrofunc_\XINT_expr_tmpa\endcsname}%
  \fi
}%
\def\XINT_expr_newfunction #1#2#3%
{%
  \def#1##1##2##3{\expandafter ##1\expandafter ##2\romannumeral`&&@%
      #3\expandafter#2\romannumeral`&&@\XINT_expr_unlock##3,\relax}%
}%
```

10.53 \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.

10.53.1 \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}%
```

10.53.2 \xintApply:::csv

```
\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}%
```

10.53.3 \XINT_expr_RApply::csv, \XINT_expr_LApply::csv, \XINT_expr_RLApply:::csv

The #1 in _Rapply will start with a ~. No risk of glueing to previous ~expandafter during the \scantokens.

Attention here and next ~ has catcode 12 and not 3 like elsewhere in xintexpr.

```
\catcode`~ 12
\def\XINT_expr_RApply::csv #1#2#3#4%
   {~xintApply::csv{~expandafter#1~xint_exchangetwo_keepbraces{#4}}{#3}}%
\def\XINT_expr_LApply::csv #1#2#3#4{~xintApply::csv{#1{#3}}{#4}}%
\def\XINT_expr_RLApply:::csv #1#2{~xintApply:::csv{#1}}%
```

10.53.4 Mysterious stuff

~ and $ of catcode 12 in what follows.

```
\catcode`$ 12 % $
\def\XINT_xptwo_getab_b #1#2!#3%
   {\expandafter\XINT_xptwo_getab_c\romannumeral`&&@#3!#1{#1#2}}%
\def\XINT_xptwo_getab_c #1#2!#3#4#5#6{#1#3{#5}{#6}{#1#2}{#4}}%
\def\xint_ddfork #1$$#2#3\krof {#2}% $$
\def\XINT_NEfork #1#2{\xint_ddfork
                          #1#2\XINT_expr_RLApply:::csv
                           #1$\XINT_expr_RApply::csv% $
                           $#2\XINT_expr_LApply::csv% $
                            $${\XINT_NEfork_nn #1#2}% $$
                      \krof }%
\def\XINT_NEfork_nn #1#2#3#4{%
        \if #1##\xint_dothis{#3}\fi
        \if  #1~\xint_dothis{#3}\fi
        \if #2##\xint_dothis{#3}\fi
        \if  #2~\xint_dothis{#3}\fi
        \xint_orthat {\csname #4NE\endcsname }%
        }%
\def\XINT_NEfork_one #1#2!#3#4#5#6{%
    \if ###1\xint_dothis {#3}\fi
    \if  ~#1\xint_dothis {#3}\fi
    \if  $#1\xint_dothis {~xintApply::csv{#3#5}}\fi %$
    \xint_orthat {\csname #4NE\endcsname #6}{#1#2}%
}%
\toks0 {}%
\xintFor #1 in
       {DivTrunc,Mod,Round,Trunc,iRound,iTrunc,iQuo,iRem,
        iiDivTrunc,iiDivRound,iiMod,iiQuo,iiRem,%
        Lt,Gt,Eq,LtorEq,GtorEq,Neq,%
        iiLt,iiGt,iiEq,iiLtorEq,iiGtorEq,iiNeq,%
        Add,Sub,Mul,Div,Pow,E,%
        iiAdd,iiSub,iiMul,iiPow,iiE,%
        AND,OR,XOR,%
        SeqA::csv,iiSeqA::csv}\do
 {\toks0
  \expandafter{\the\toks0% no space!
  \expandafter\let\csname xint#1NE\expandafter\endcsname\csname xint#1\expandafter
```

```
  \endcsname\expandafter\def\csname xint#1\endcsname ####1####2{%
        \expandafter\XINT_NEfork
        \romannumeral`&&@\expandafter\XINT_xptwo_getab_b
        \romannumeral`&&@####2!{####1}{~xint#1}{xint#1}}%
  }%
}%
% cela aurait-il un sens d'ajouter Raw et iNum (à cause de qint, qfrac,
% qfloat?). Pas le temps d'y réfléchir. Je ne fais rien.
\xintFor #1 in {Num,Irr,Abs,iiAbs,Sgn,iiSgn,TFrac,Floor,iFloor,Ceil,iCeil,%
   Sqr,iiSqr,iiSqrt,iiSqrtR,iiIsZero,iiIsNotZero,iiifNotZero,iiifSgn,%
   Odd,Even,iiOdd,iiEven,Opp,iiOpp,iiifZero,iFac,iBinomial,%
   iPFactorial,iiFac,iiBinomial,iiPFactorial,Bool,Toggle}\do
{\toks0 \expandafter{\the\toks0%
     \expandafter\let\csname xint#1NE\expandafter\endcsname\csname
     xint#1\expandafter
  \endcsname\expandafter\def\csname xint#1\endcsname ####1{%
      \expandafter\XINT_NEfork_one\romannumeral`&&@####1!{~xint#1}{xint#1}{}{}}%
  }%
}%
\toks0
  \expandafter{\the\toks0
  \let\XINTinFloatFacNE\XINTinFloatFac
  \def\XINTinFloatFac ##1{%
        \expandafter\XINT_NEfork_one
        \romannumeral`&&@##1!{~XINTinFloatFac}{XINTinFloatFac}{}{}}%
  }%
\xintFor #1 in {Add,Sub,Mul,Div,Binomial,PFactorial,PowerH,E,Mod,SeqA::csv}\do
{\toks0
  \expandafter{\the\toks0%
  \expandafter\let\csname XINTinFloat#1NE\expandafter\endcsname
                                \csname XINTinFloat#1\expandafter\endcsname
  \expandafter\def\csname XINTinFloat#1\endcsname ####1####2{%
        \expandafter\XINT_NEfork
        \romannumeral`&&@\expandafter\XINT_xptwo_getab_b
        \romannumeral`&&@####2!{####1}{~XINTinFloat#1}{XINTinFloat#1}}%
  }%
}%
\xintFor #1 in {XINTinFloatdigits,XINTinFloatFracdigits,XINTinFloatSqrtdigits,XINTinFloatFacdigits}\do
{\toks0
  \expandafter{\the\toks0%
  \expandafter\let\csname #1NE\expandafter\endcsname\csname #1\expandafter
  \endcsname\expandafter\def\csname #1\endcsname ####1{\expandafter
       \XINT_NEfork_one\romannumeral`&&@####1!{~#1}{#1}{}{}}%
  }%
}%
\xintFor #1 in {xintSeq::csv,xintiiSeq::csv,XINTinFloatSeq::csv}\do
 {\toks0
  \expandafter{\the\toks0% no space
  \expandafter\let\csname #1NE\expandafter\endcsname\csname #1\expandafter
  \endcsname\expandafter\def\csname #1\endcsname ####1####2{%
        \expandafter\XINT_NEfork
        \romannumeral`&&@\expandafter\XINT_xptwo_getab_b
```

```
        \romannumeral`&&@####2!{####1}{$noexpand$#1}{#1}}%
  }%
}%
\xintFor #1 in {xintSeqB,xintiiSeqB,XINTinFloatSeqB}\do
 {\toks0
  \expandafter{\the\toks0% no space
  \expandafter\let\csname #1::csvNE\expandafter\endcsname\csname #1::csv\expandafter
  \endcsname\expandafter\def\csname #1::csv\endcsname ####1####2{%
        \expandafter\XINT_NEfork
        \romannumeral`&&@\expandafter\XINT_xptwo_getab_b
        \romannumeral`&&@####2!{####1}{$noexpand$#1:f:csv}{#1::csv}}%
  }%
}%
\toks0
  \expandafter{\the\toks0
  \let\XINTinFloatNE\XINTinFloat
  \def\XINTinFloat [##1]##2{% not ultimately general, but got tired
        \expandafter\XINT_NEfork_one
        \romannumeral`&&@##2!{~XINTinFloat[##1]}{XINTinFloat}{}{[##1]}}%
  \let\XINTinFloatSqrtNE\XINTinFloatSqrt
  \def\XINTinFloatSqrt [##1]##2{%
        \expandafter\XINT_NEfork_one
        \romannumeral`&&@##2!{~XINTinFloatSqrt[##1]}{XINTinFloatSqrt}{}{[##1]}}%
  \let\XINTinFloatFacNE\XINTinFloatFac
  \def\XINTinFloatFac [##1]##2{%
        \expandafter\XINT_NEfork_one
        \romannumeral`&&@##2!{~XINTinFloatFac[##1]}{XINTinFloatFac}{}{[##1]}}%
}%
\xintFor #1 in {ANDof,ORof,XORof,iiMaxof,iiMinof,iiSum,iiPrd,
                GCDof,LCMof,Sum,Prd,Maxof,Minof}\do
{\toks0
  \expandafter{\the\toks0\expandafter\def\csname xint#1:csv\endcsname {~xint#1:csv}}%
}%
\xintFor #1 in
   {XINTinFloatMaxof,XINTinFloatMinof,XINTinFloatSum,XINTinFloatPrd}\do
{\toks0
  \expandafter{\the\toks0\expandafter\def\csname #1:csv\endcsname {~#1:csv}}%
}%
```

~xintListSel:f:csv must have space after it, the reason being in \XINT_expr_until_:_b which inserts a : as fist token of something which will reappear later following ~xintListSel:f:csv. There must be some deep wisdom in previous sentence but I find it barely comprehensible now. Anyway, 1.2j has \xintListSel:x:csv which is thus mapped here to f-expandable variant.

```
\toks0 \expandafter{\the\toks0
                    \def\xintReverse:f:csv {~xintReverse:f:csv }%
                    \def\xintListSel:x:csv {~xintListSel:f:csv }%
}%
\odef\XINT_expr_redefinemacros {\the\toks0}% Not \edef ! (subtle)
\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  }%
}%
\toks0 {}%
```

10.53.5 \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
                                    \xintteboolexpr\XINT_newexpr_clean}%
\def\XINT_newexpr_clean #1>{\noexpand\romannumeral`&&@}%

 1.2c for \xintdeffunc, \xintdefiifunc, \xintdeffloatfunc.

\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. And
 _NewExpr itself receives two new parameters to treat both \xintNewExpr and \xintdeffunc.
   Up to and including 1.2c the definition was global. Starting with 1.2d it is done locally.

\def\XINT_NewExpr #1#2#3#4#5#6#7[#8]%
{%
 \begingroup
```

```
    \ifcase #8\relax
        \toks0 {\endgroup\def#6}%
    \or \toks0 {\endgroup\def#6##1#1}%
    \or \toks0 {\endgroup\def#6##1#1##2#1}%
    \or \toks0 {\endgroup\def#6##1#1##2#1##3#1}%
    \or \toks0 {\endgroup\def#6##1#1##2#1##3#1##4#1}%
    \or \toks0 {\endgroup\def#6##1#1##2#1##3#1##4#1##5#1}%
    \or \toks0 {\endgroup\def#6##1#1##2#1##3#1##4#1##5#1##6#1}%
    \or \toks0 {\endgroup\def#6##1#1##2#1##3#1##4#1##5#1##6#1##7#1}%
    \or \toks0 {\endgroup\def#6##1#1##2#1##3#1##4#1##5#1##6#1##7#1##8#1}%
    \or \toks0 {\endgroup\def#6##1#1##2#1##3#1##4#1##5#1##6#1##7#1##8#1##9#1}%
    \fi
    \xintexprSafeCatcodes
    \XINT_expr_redefinemacros
    #2%
    \XINT_NewExpr_a #3#4#5#6%
}%
```

For the 1.2a release I replaced all \romannumeral-`0 by a fancier \romannumeral`&&@ (with & of catcode 7). I got lucky here that it worked, despite @ being of catcode comment (anyhow \input xintexpr.sty would not have compiled if not, and I would have realized immediately). But to be honest I wouldn't have been 100% sure beforehand that &&@ worked also with @ comment character. I now know.

1.2d's \xintNewExpr makes a local definition. In earlier releases, the definition was global.

```
\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 meaning \meaning%4}@
       \fi}@
}@
\catcode`% 14
\let\xintexprRestoreCatcodes\empty
\def\xintexprSafeCatcodes
{%
    \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   % =
        \catcode32=\the\catcode32\relax % space
    }%
        \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  % =
        \catcode32=10  % space
}%
\let\XINT_tmpa\relax \let\XINT_tmpb\relax \let\XINT_tmpc\relax
\XINT_restorecatcodes_endinput%
```

```
xintkernel: 489.
 xinttools:1391.
  xintcore:2000.
      xint:1785.
xintbinhex: 629.
   xintgcd: 455.
  xintfrac:3003.
xintseries: 386.
 xintcfrac:1021.
  xintexpr:2767.
```

Total number of code lines: 13926. (but 3219 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 1.2j of 2016/12/22.