The xint source code

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

Package version: 1.1c (2015/09/12); documentation date: 2015/09/17.
From source file xint.dtx. Time-stamp: <17-09-2015 at 11:09:20 CEST>.

Contents

This is 1.1c of 2015/09/12.

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.

xinttools is not loaded anymore by xint, nor by xintfrac. It is loaded by xintexpr.

1 Package xintkernel implementation

This package provides the common minimal code base for loading management and catcode control and also a few programming utilities. It is loaded by both xintcore.sty and xinttools.sty hence by all other packages.

First appeared as a separate package with release 1.1.

1.1 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
  \expandafter
    \ifx\csname numexpr\endcsname\relax
     \y{xintkernel}{\numexpr not available, aborting input}%
     \aftergroup\endinput
    \else
    \expandafter
     \ifx\csname XINTsetupcatcodes\endcsname\relax
      \else
       \y{xintkernel}{I was already loaded, aborting input}%
       \aftergroup\endinput
      \fi
    \fi
  \def\SetCatcodesIfInputNotAborted
  {%
      \endgroup
      \def\XINT_restorecatcodes
      {% takes care of all, to allow more economical code in modules
           \catcode59=\the\catcode59   % ; xintexpr
           \catcode126=\the\catcode126 % ~ xintexpr
           \catcode39=\the\catcode39   % ' xintexpr
           \catcode34=\the\catcode34   % " xintbinhex, and xintexpr
           \catcode63=\the\catcode63   % ? xintexpr
```

```
        \catcode124=\the\catcode124 % | xintexpr
        \catcode38=\the\catcode38   % & xintexpr
        \catcode64=\the\catcode64   % @ xintexpr
        \catcode33=\the\catcode33   % ! xintexpr
        \catcode93=\the\catcode93   % ] -, xintfrac, xintseries, xintcfrac
        \catcode91=\the\catcode91   % [ -, xintfrac, xintseries, xintcfrac
        \catcode36=\the\catcode36   % $ xintgcd only
        \catcode94=\the\catcode94   % ^
        \catcode96=\the\catcode96   % `
        \catcode47=\the\catcode47   % /
        \catcode41=\the\catcode41   % )
        \catcode40=\the\catcode40   % (
        \catcode42=\the\catcode42   % *
        \catcode43=\the\catcode43   % +
        \catcode62=\the\catcode62   % >
        \catcode60=\the\catcode60   % <
        \catcode58=\the\catcode58   % :
        \catcode46=\the\catcode46   % .
        \catcode45=\the\catcode45   % -
        \catcode44=\the\catcode44   % ,
        \catcode35=\the\catcode35   % #
        \catcode95=\the\catcode95   % _
        \catcode125=\the\catcode125 % }
        \catcode123=\the\catcode123 % {
        \endlinechar=\the\endlinechar
        \catcode13=\the\catcode13   % ^^M
        \catcode32=\the\catcode32   %
        \catcode61=\the\catcode61\relax   % =
    }%
    \edef\XINT_restorecatcodes_endinput
    {%
        \XINT_restorecatcodes\noexpand\endinput %
    }%
    \def\XINT_setcatcodes
    {%
      \catcode61=12   % =
      \catcode32=10   % space
      \catcode13=5    % ^^M
      \endlinechar=13 %
      \catcode123=1   % {
      \catcode125=2   % }
      \catcode95=11   % _ LETTER
      \catcode35=6    % #
      \catcode44=12   % ,
      \catcode45=12   % -
      \catcode46=12   % .
      \catcode58=11   % : LETTER
      \catcode60=12   % <
      \catcode62=12   % >
      \catcode43=12   % +
      \catcode42=12   % *
      \catcode40=12   % (
```

```
        \catcode41=12   % )
        \catcode47=12   % /
        \catcode96=12   % `
        \catcode94=11   % ^ LETTER
        \catcode36=3    % $
        \catcode91=12   % [
        \catcode93=12   % ]
        \catcode33=11   % ! LETTER
        \catcode64=11   % @ LETTER
        \catcode38=12   % &
        \catcode124=12  % |
        \catcode63=11   % ? LETTER
        \catcode34=12   % "
        \catcode39=12   % '
        \catcode126=3   % ~
        \catcode59=12   % ;
      }%
      \XINT_setcatcodes
  }%
\SetCatcodesIfInputNotAborted
```

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}%
  [2015/09/12 v1.1c Paraphernalia for the xint packages (jfB)]%
```

1.3 Token management utilities

```
\long\def\xint_gobble_     {}%
\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_firstofone_thenstop  #1{ #1}%
\long\def\xint_firstoftwo_thenstop  #1#2{ #1}%
\long\def\xint_secondoftwo_thenstop #1#2{ #2}%
```

1.4 gob til macros and UD style fork

```
\def\xint_gob_til_zero #10{}%
\def\xint_UDzerominusfork #10-#2#3\krof {#2}%
\long\def\xint_gob_til_R #1\R {}%
\long\def\xint_gob_til_W #1\W {}%
\long\def\xint_gob_til_Z #1\Z {}%
\let\xint_relax\relax
\def\xint_brelax {\xint_relax }%
\long\def\xint_gob_til_xint_relax #1\xint_relax {}%
```

1.5 \xint_afterfi

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

1.6 \xint_bye

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

1.7 \xint_dothis, \xint_orthat

New with 1.1. 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}% v1.1
\let\xint_orthat \xint_firstofone
```

1.8 \xint_zapspaces

1.1. Zaps leading, intermediate, trailing, spaces in completely expanding context (\edef, \csname . . . \endcsname). To be used as:

```
 \xint_zapspaces foo \xint_gobble_i % notice the mandatory space after foo
```

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.

Code comment from 1.1 release said to do:

```
\xint_zapspaces foo \xint_bye\xint_bye
```

perhaps because it was pretty. It works also, but \xint_gobble_i is one token less. Compatible with an empty foo.

```
\def\xint_zapspaces #1 #2{#1#2\xint_zapspaces }% v1.1
```

1.9 Constants

```
\chardef\xint_c_     0
\chardef\xint_c_i    1
\chardef\xint_c_ii   2
\chardef\xint_c_iii  3
\chardef\xint_c_iv   4
\chardef\xint_c_v    5
\chardef\xint_c_vi   6
\chardef\xint_c_vii  7
\chardef\xint_c_viii 8
```

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-`0#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.

```
\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.

```
\def\xintLength {\romannumeral0\xintlength }%
\long\def\xintlength #1%
{%
    \XINT_length_loop
    0.#1\xint_relax\xint_relax\xint_relax\xint_relax
        \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
}%
\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
    \expandafter\XINT_length_loop\the\numexpr #1+\xint_c_viii.%
}%
\def\XINT_length_finish_a\xint_relax\expandafter\XINT_length_loop
    \the\numexpr #1+\xint_c_viii.#2\xint_bye
{%
    \XINT_length_finish_b #2\W\W\W\W\W\W\W\Z {#1}%
}%
\def\XINT_length_finish_b #1#2#3#4#5#6#7#8\Z
{%
    \xint_gob_til_W
            #1\XINT_length_finish_c \xint_c_
            #2\XINT_length_finish_c \xint_c_i
            #3\XINT_length_finish_c \xint_c_ii
            #4\XINT_length_finish_c \xint_c_iii
            #5\XINT_length_finish_c \xint_c_iv
            #6\XINT_length_finish_c \xint_c_v
            #7\XINT_length_finish_c \xint_c_vi
            \W\XINT_length_finish_c \xint_c_vii\Z
}%
\edef\XINT_length_finish_c #1#2\Z #3%
   {\noexpand\expandafter\space\noexpand\the\numexpr #3+#1\relax}%
\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}%
  [2015/09/12 v1.1c 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-`0#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-`0#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-`0#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.
  \xintNthElt {i}{stuff f-expanding to {a}{b}...{z}} (or `tokens' abcd...z)returns the i th element (one pair of braces removed). The list is first f-expanded. The \xintNthEltNoExpand does no expansion of its second argument. Both variants expand the first argument inside \numexpr.
  With i = 0, the number of items is returned. This is different from \xintLen which is only for numbers (particularly, it checks the sign) and different from \xintLength which does not f-expand its argument.
  Negative values return the |i|th element from the end. Release 1.09m rewrote the initial bits of the code (which checked the sign of #1 and expanded or not #2), ome `improvements' made earlier in 1.09c were quite sub-efficient. Now uses \xint_UDzerominusfork, moved from xint.sty.

```
\def\xintNthElt         {\romannumeral0\xintnthelt }%
\def\xintNthEltNoExpand {\romannumeral0\xintntheltnoexpand }%
\def\xintnthelt #1#2%
{%
    \expandafter\XINT_nthelt_a\the\numexpr #1\expandafter.%
    \expandafter{\romannumeral-`0#2}%
}%
\def\xintntheltnoexpand #1%
{%
    \expandafter\XINT_nthelt_a\the\numexpr #1.%
}%
\def\XINT_nthelt_a #1#2.%
{%
    \xint_UDzerominusfork
        #1-{\XINT_nthelt_bzero}%
        0#1{\XINT_nthelt_bneg {#2}}%
         0-{\XINT_nthelt_bpos {#1#2}}%
    \krof
}%
\long\def\XINT_nthelt_bzero #1%
{%
     \XINT_length_loop 0.#1\xint_relax\xint_relax\xint_relax\xint_relax
                 \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
}%
\long\def\XINT_nthelt_bneg #1#2%
{%
     \expandafter\XINT_nthelt_loop_a\expandafter {\the\numexpr #1\expandafter}%
     \romannumeral0\xintrevwithbracesnoexpand {#2}%
          \xint_relax\xint_relax\xint_relax\xint_relax
          \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
}%
\long\def\XINT_nthelt_bpos #1#2%
{%
     \XINT_nthelt_loop_a {#1}#2\xint_relax\xint_relax\xint_relax\xint_relax
                     \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
}%
\def\XINT_nthelt_loop_a #1%
{%
    \ifnum #1>\xint_c_viii
        \expandafter\XINT_nthelt_loop_b
```

```
    \else
        \XINT_nthelt_getit
    \fi
    {#1}%
}%
\long\def\XINT_nthelt_loop_b #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_xint_relax #9\XINT_nthelt_silentend\xint_relax
    \expandafter\XINT_nthelt_loop_a\expandafter{\the\numexpr #1-\xint_c_viii}%
}%
\def\XINT_nthelt_silentend #1\xint_bye { }%
\def\XINT_nthelt_getit\fi #1%
{%
    \fi\expandafter\expandafter\expandafter\XINT_nthelt_finish
    \csname xint_gobble_\romannumeral\numexpr#1-\xint_c_i\endcsname
}%
\long\edef\XINT_nthelt_finish #1#2\xint_bye
   {\noexpand\xint_gob_til_xint_relax #1\noexpand\expandafter\space
                                \noexpand\xint_gobble_ii\xint_relax\space #1}%
```

2.12 \xintKeep

First included in release 1.09m.
  \xintKeep {i}{stuff f-expanding to {a}{b}...{z}} (or `tokens' abcd...z, but each naked token ends up braced in the output) returns (in two expansion steps) the first i elements from the list, which is first f-expanded. The i is expanded inside \numexpr. The variant \xintKeepNoExpand does not expand the list argument.
  With i = 0, the empty sequence is returned.
  With i<0, the last |i| elements are returned (in the same order as in the original list).
  With |i| equal to or bigger than the length of the (f-expanded) list, the full list is returned.

```
\def\xintKeep          {\romannumeral0\xintkeep }%
\def\xintKeepNoExpand {\romannumeral0\xintkeepnoexpand }%
\def\xintkeep #1#2%
{%
    \expandafter\XINT_keep_a\the\numexpr #1\expandafter.%
    \expandafter{\romannumeral-`0#2}%
}%
\def\xintkeepnoexpand #1%
{%
    \expandafter\XINT_keep_a\the\numexpr #1.%
}%
\def\XINT_keep_a #1#2.%
{%
    \xint_UDzerominusfork
        #1-{\expandafter\space\xint_gobble_i }%
        0#1{\XINT_keep_bneg_a {#2}}%
         0-{\XINT_keep_bpos {#1#2}}%
    \krof
}%
\long\def\XINT_keep_bneg_a #1#2%
{%
    \expandafter\XINT_keep_bneg_b \the\numexpr \xintLength{#2}-#1.{#2}%
```

```
}%
\def\XINT_keep_bneg_b #1#2.%
{%
    \xint_UDzerominusfork
        #1-{\xint_firstofone_thenstop }%
        0#1{\xint_firstofone_thenstop }%
         0-{\XINT_trim_bpos {#1#2}}%
    \krof
}%
\long\def\XINT_keep_bpos #1#2%
{%
     \XINT_keep_loop_a {#1}{}#2\xint_relax\xint_relax\xint_relax\xint_relax
          \xint_relax\xint_relax\xint_relax\xint_bye
}%
\def\XINT_keep_loop_a #1%
{%
    \ifnum #1>\xint_c_vi
        \expandafter\XINT_keep_loop_b
    \else
        \XINT_keep_finish
    \fi
    {#1}%
}%
\long\def\XINT_keep_loop_b #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_xint_relax #9\XINT_keep_enda\xint_relax
    \expandafter\XINT_keep_loop_c\expandafter{\the\numexpr #1-\xint_c_vii}%
    {{#3}{#4}{#5}{#6}{#7}{#8}{#9}}{#2}%
}%
\long\def\XINT_keep_loop_c #1#2#3{\XINT_keep_loop_a {#1}{#3#2}}%
\long\def\XINT_keep_enda\xint_relax
     \expandafter\XINT_keep_loop_c\expandafter #1#2#3#4\xint_bye
{%
     \XINT_keep_endb #4\W\W\W\W\W\W\Z #2{#3}%
}%
\def\XINT_keep_endb #1#2#3#4#5#6#7\Z
{%
    \xint_gob_til_W
    #1\XINT_keep_endc_
    #2\XINT_keep_endc_i
    #3\XINT_keep_endc_ii
    #4\XINT_keep_endc_iii
    #5\XINT_keep_endc_iv
    #6\XINT_keep_endc_v
    \W\XINT_keep_endc_vi\Z
}%
\long\def\XINT_keep_endc_    #1\Z #2#3#4#5#6#7#8#9{ #9}%
\long\def\XINT_keep_endc_i   #1\Z #2#3#4#5#6#7#8#9{ #9{#2}}%
\long\def\XINT_keep_endc_ii  #1\Z #2#3#4#5#6#7#8#9{ #9{#2}{#3}}%
\long\def\XINT_keep_endc_iii #1\Z #2#3#4#5#6#7#8#9{ #9{#2}{#3}{#4}}%
\long\def\XINT_keep_endc_iv  #1\Z #2#3#4#5#6#7#8#9{ #9{#2}{#3}{#4}{#5}}%
\long\def\XINT_keep_endc_v   #1\Z #2#3#4#5#6#7#8#9{ #9{#2}{#3}{#4}{#5}{#6}}%
```

```
\long\def\XINT_keep_endc_vi\Z #1#2#3#4#5#6#7#8{ #8{#1}{#2}{#3}{#4}{#5}{#6}}%
\long\def\XINT_keep_finish\fi #1#2#3#4#5#6#7#8#9\xint_bye
{%
    \fi\XINT_keep_finish_loop_a {#1}{}{#3}{#4}{#5}{#6}{#7}{#8}\Z {#2}%
}%
\def\XINT_keep_finish_loop_a #1%
{%
    \xint_gob_til_zero #1\XINT_keep_finish_z0%
    \expandafter\XINT_keep_finish_loop_b\expandafter
       {\the\numexpr #1-\xint_c_i}%
}%
\long\def\XINT_keep_finish_z0%
     \expandafter\XINT_keep_finish_loop_b\expandafter #1#2#3\Z #4{ #4#2}%
\long\def\XINT_keep_finish_loop_b #1#2#3%
{%
    \xint_gob_til_xint_relax #3\XINT_keep_finish_exit\xint_relax
    \XINT_keep_finish_loop_c {#1}{#2}{#3}%
}%
\long\def\XINT_keep_finish_exit\xint_relax
         \XINT_keep_finish_loop_c #1#2#3\Z #4{ #4#2}%
\long\def\XINT_keep_finish_loop_c #1#2#3%
        {\XINT_keep_finish_loop_a {#1}{#2{#3}}}%
```

2.13 \xintTrim

First included in release 1.09m.
  \xintTrim {i}{stuff f-expanding to {a}{b}...{z}} (or `tokens' abcd...z, but each naked token ends up braced in the output) returns (in two expansion steps) the sequence with the first i elements omitted. The list is first f-expanded. The i is expanded inside \numexpr. Variant \xintTrimNoExpand does not expand the list argument.
  With i = 0, the original (expanded) list is returned.
  With i<0, the last |i| elements from the tail are suppressed.
  With |i| equal to or bigger than the length of the (f-expanded) list, the empty list is returned.

```
\def\xintTrim         {\romannumeral0\xinttrim }%
\def\xintTrimNoExpand {\romannumeral0\xinttrimnoexpand }%
\def\xinttrim #1#2%
{%
    \expandafter\XINT_trim_a\the\numexpr #1\expandafter.%
    \expandafter{\romannumeral-`0#2}%
}%
\def\xinttrimnoexpand #1%
{%
    \expandafter\XINT_trim_a\the\numexpr #1.%
}%
\def\XINT_trim_a #1#2.%
{%
    \xint_UDzerominusfork
        #1-{\xint_firstofone_thenstop }%
        0#1{\XINT_trim_bneg_a {#2}}%
         0-{\XINT_trim_bpos {#1#2}}%
    \krof
}%
```

```
\long\def\XINT_trim_bneg_a #1#2%
{%
    \expandafter\XINT_trim_bneg_b \the\numexpr \xintLength{#2}-#1.{#2}%
}%
\def\XINT_trim_bneg_b #1#2.%
{%
    \xint_UDzerominusfork
        #1-{\expandafter\space\xint_gobble_i }%
        0#1{\expandafter\space\xint_gobble_i }%
         0-{\XINT_keep_bpos {#1#2}}%
    \krof
}%
\long\def\XINT_trim_bpos #1#2%
{%
     \XINT_trim_loop_a {#1}#2\xint_relax\xint_relax\xint_relax\xint_relax
                      \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
}%
\def\XINT_trim_loop_a #1%
{%
    \ifnum #1>\xint_c_vii
        \expandafter\XINT_trim_loop_b
    \else
        \XINT_trim_finish
    \fi
    {#1}%
}%
\long\def\XINT_trim_loop_b #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_xint_relax #9\XINT_trim_silentend\xint_relax
    \expandafter\XINT_trim_loop_a\expandafter{\the\numexpr #1-\xint_c_viii}%
}%
\def\XINT_trim_silentend #1\xint_bye { }%
\def\XINT_trim_finish\fi #1%
{%
    \fi\expandafter\expandafter\expandafter\XINT_trim_finish_a
    \expandafter\expandafter\expandafter\space % avoids brace removal
    \csname xint_gobble_\romannumeral\numexpr#1\endcsname
}%
\long\def\XINT_trim_finish_a #1\xint_relax #2\xint_bye {#1}%
```

2.14 \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-`0#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-`0#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}%
```

2.15 \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-`0#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-`0#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}%
```

2.16 \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}}}%
```

2.17 \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}}%
```

2.18 \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}}%
```

2.19 \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-`0%
}%
\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-`0%
}%
\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
}%
```

2.20 \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
}%
```

2.21 \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.

```
\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
              \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\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\XINT_forx_d #1{#2}}%
  \XINT_xflet\XINT_for_last?
}%
\def\XINT_for_last?
{%
    \let\xintifForLast\xint_secondoftwo
    \ifx\XINT_token Z\let\xintifForLast\xint_firstoftwo
        \xint_afterfi{\xintBreakForAndDo{\XINT_x\xint_gobble_i Z}}\fi
    \the\XINT_toks
}%
```

2.22 \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-`0\xintrawwithzeros{\xintZapSpacesB{#2}}}%
          {\romannumeral-`0\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-`0#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
  \expandafter\XINT_forever_d\expandafter #1\romannumeral-`0#4{#2}{#3}#4{#5}%
}%
```

2.23 \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.

```
\catcode`j 3
\long\def\xintForpair #1#2#3in#4#5#6%
{%
    \let\xintifForFirst\xint_firstoftwo
    \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}%
  \let\xintifForLast\xint_secondoftwo
  \ifx #5j\expandafter\xint_firstoftwo
     \else\expandafter\xint_secondoftwo
  \fi
  {\let\xintifForLast\xint_firstoftwo
   \xintBreakForAndDo {\XINT_x \xint_gobble_i Z}}%
  \XINT_x
  \let\xintifForFirst\xint_secondoftwo\XINT_forpair_d #1{#2}%
}%
\long\def\xintForthree #1#2#3in#4#5#6%
{%
    \let\xintifForFirst\xint_firstoftwo
```

```
    \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}%
  \let\xintifForLast\xint_secondoftwo
  \ifx #5j\expandafter\xint_firstoftwo
     \else\expandafter\xint_secondoftwo
  \fi
  {\let\xintifForLast\xint_firstoftwo
   \xintBreakForAndDo {\XINT_x \xint_gobble_i Z}}%
  \XINT_x
  \let\xintifForFirst\xint_secondoftwo\XINT_forthree_d #1{#2}%
}%
\long\def\xintForfour #1#2#3in#4#5#6%
{%
    \let\xintifForFirst\xint_firstoftwo
    \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}%
  \let\xintifForLast\xint_secondoftwo
  \ifx #5j\expandafter\xint_firstoftwo
     \else\expandafter\xint_secondoftwo
  \fi
  {\let\xintifForLast\xint_firstoftwo
   \xintBreakForAndDo {\XINT_x \xint_gobble_i Z}}%
  \XINT_x
  \let\xintifForFirst\xint_secondoftwo\XINT_forfour_d #1{#2}%
}%
\catcode`Z 11
\catcode`j 11
```

2.24 \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-`0#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-`0#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 }%
        \else
        \xint_afterfi {%
              \ifnum ##1>#2
                  \xint_afterfi {\xintError:ArrayIndexBeyondLimit\space }%
              \else\xint_afterfi
       {\expandafter\expandafter\expandafter\space\csname #3##1\endcsname}%
              \fi}%
        \fi
     }%
}%
\let\xintDigitsOf\xintAssignArray
\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 (macros \XINT_SQ, \xintLDg, \xintHalf which are dependencies of \XINT_div_prepare were forgotten and they were added to the package only later with 1.1b). Release 1.1 also added the new macro \xintiiDivRound. The package does not load xinttools.

Since release xint 1.09a the macros doing arithmetic operations apply systematically \xintnum to their arguments; this adds a little overhead but this is more convenient for using count

registers even with infix notation; also this is what xintfrac.sty did all along. It simplifies the discussion in the documentation too.

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}%
  [2015/09/12 v1.1c Expandable arithmetic on big integers (jfB)]%
```

3.3 More token management, constants

```
\def\xint_minus_thenstop { -}%
\def\xint_gob_til_zeros_iii #1000{}%
\def\xint_gob_til_zeros_iv  #10000{}%
\def\xint_gob_til_one   #11{}%
\def\xint_gob_til_G     #1G{}%
\def\xint_gob_til_minus #1-{}%
\def\xint_gob_til_relax #1\relax {}%
\def\xint_exchangetwo_keepbraces          #1#2{{#2}{#1}}%
\def\xint_exchangetwo_keepbraces_thenstop #1#2{ {#2}{#1}}%
\def\xint_UDzerofork       #10#2#3\krof {#2}%
\def\xint_UDsignfork       #1-#2#3\krof {#2}%
\def\xint_UDwfork         #1\W#2#3\krof {#2}%
\def\xint_UDzerosfork     #100#2#3\krof {#2}%
\def\xint_UDonezerofork   #110#2#3\krof {#2}%
\def\xint_UDsignsfork     #1--#2#3\krof {#2}%
\chardef\xint_c_ix     9
\chardef\xint_c_x     10
\chardef\xint_c_ii^v  32 % not used in xint, common to xintfrac and xintbinhex
\chardef\xint_c_ii^vi 64
\mathchardef\xint_c_ixixixix 9999
\mathchardef\xint_c_x^iv    10000
\newcount\xint_c_x^viii  \xint_c_x^viii 100000000
```

3.4 \XINT_RQ

Cette macro renverse et ajoute le nombre minimal de zéros à la fin pour que la longueur soit alors multiple de 4
\romannumeral0\XINT_RQ {}<le truc à renverser>\R\R\R\R\R\R\R\R\Z
Attention, ceci n'est utilisé que pour des chaînes de chiffres, et donc le comportement avec des {..} ou autres espaces n'a fait l'objet d'aucune attention.

```
\def\XINT_RQ #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_R #9\XINT_RQ_end_a\R\XINT_RQ {#9#8#7#6#5#4#3#2#1}%
}%
\def\XINT_RQ_end_a\R\XINT_RQ #1#2\Z
{%
    \XINT_RQ_end_b #1\Z
}%
\def\XINT_RQ_end_b #1#2#3#4#5#6#7#8%
{%
    \xint_gob_til_R
            #8\XINT_RQ_end_viii
            #7\XINT_RQ_end_vii
            #6\XINT_RQ_end_vi
            #5\XINT_RQ_end_v
            #4\XINT_RQ_end_iv
```

```
            #3\XINT_RQ_end_iii
            #2\XINT_RQ_end_ii
            \R\XINT_RQ_end_i
            \Z #2#3#4#5#6#7#8%
}%
\def\XINT_RQ_end_viii #1\Z #2#3#4#5#6#7#8#9\Z { #9}%
\def\XINT_RQ_end_vii  #1\Z #2#3#4#5#6#7#8#9\Z { #8#9000}%
\def\XINT_RQ_end_vi   #1\Z #2#3#4#5#6#7#8#9\Z { #7#8#900}%
\def\XINT_RQ_end_v    #1\Z #2#3#4#5#6#7#8#9\Z { #6#7#8#90}%
\def\XINT_RQ_end_iv   #1\Z #2#3#4#5#6#7#8#9\Z { #5#6#7#8#9}%
\def\XINT_RQ_end_iii  #1\Z #2#3#4#5#6#7#8#9\Z { #4#5#6#7#8#9000}%
\def\XINT_RQ_end_ii   #1\Z #2#3#4#5#6#7#8#9\Z { #3#4#5#6#7#8#900}%
\def\XINT_RQ_end_i      \Z #1#2#3#4#5#6#7#8\Z { #1#2#3#4#5#6#7#80}%
```

3.5 \XINT_OQ

```
\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}%
```

3.6 \XINT_SQ

```
\def\XINT_SQ #1#2#3#4#5#6#7#8%
{%
    \xint_gob_til_R #8\XINT_SQ_end_a\R\XINT_SQ {#8#7#6#5#4#3#2#1}%
}%
\def\XINT_SQ_end_a\R\XINT_SQ #1#2\Z
{%
```

```
    \XINT_SQ_end_b #1\Z
}%
\def\XINT_SQ_end_b #1#2#3#4#5#6#7%
{%
    \xint_gob_til_R
            #7\XINT_SQ_end_vii
            #6\XINT_SQ_end_vi
            #5\XINT_SQ_end_v
            #4\XINT_SQ_end_iv
            #3\XINT_SQ_end_iii
            #2\XINT_SQ_end_ii
            \R\XINT_SQ_end_i
            \Z #2#3#4#5#6#7%
}%
\def\XINT_SQ_end_vii  #1\Z #2#3#4#5#6#7#8\Z { #8}%
\def\XINT_SQ_end_vi   #1\Z #2#3#4#5#6#7#8\Z { #7#8000000}%
\def\XINT_SQ_end_v    #1\Z #2#3#4#5#6#7#8\Z { #6#7#800000}%
\def\XINT_SQ_end_iv   #1\Z #2#3#4#5#6#7#8\Z { #5#6#7#80000}%
\def\XINT_SQ_end_iii  #1\Z #2#3#4#5#6#7#8\Z { #4#5#6#7#8000}%
\def\XINT_SQ_end_ii   #1\Z #2#3#4#5#6#7#8\Z { #3#4#5#6#7#800}%
\def\XINT_SQ_end_i      \Z #1#2#3#4#5#6#7\Z { #1#2#3#4#5#6#70}%
```

3.7 \XINT_cuz

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

routine CleanUpZeros. Utilisée en particulier par la soustraction.
INPUT: longueur **multiple de 4** (<-- ATTENTION)
OUTPUT: on a retiré tous les leading zéros, on n'est **plus* nécessairement de longueur 4n
Délimiteur pour _main: \W\W\W\W\W\W\W\Z avec SEPT \W

```
\def\XINT_cuz #1%
{%
    \XINT_cuz_loop #1\W\W\W\W\W\W\W\Z%
}%
\def\XINT_cuz_loop #1#2#3#4#5#6#7#8%
{%
    \xint_gob_til_W #8\xint_cuz_end_a\W
    \xint_gob_til_Z #8\xint_cuz_end_A\Z
```

```
    \XINT_cuz_check_a {#1#2#3#4#5#6#7#8}%
}%
\def\xint_cuz_end_a #1\XINT_cuz_check_a #2%
{%
    \xint_cuz_end_b #2%
}%
\edef\xint_cuz_end_b #1#2#3#4#5\Z
{%
    \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4\relax
}%
\def\xint_cuz_end_A \Z\XINT_cuz_check_a #1{ 0}%
\def\XINT_cuz_check_a #1%
{%
    \expandafter\XINT_cuz_check_b\the\numexpr #1\relax
}%
\def\XINT_cuz_check_b #1%
{%
    \xint_gob_til_zero #1\xint_cuz_backtoloop 0\XINT_cuz_stop #1%
}%
\def\XINT_cuz_stop #1\W #2\Z{ #1}%
\def\xint_cuz_backtoloop 0\XINT_cuz_stop 0{\XINT_cuz_loop }%
```

3.8 \xintNum

For example \xintNum {----+-+++---+----000000000000003}
1.05 defines \xintiNum, which allows redefinition of \xintNum by xintfrac.sty Slightly modified in 1.06b (\R->\xint_relax) to avoid initial re-scan of input stack (while still allowing empty #1). In versions earlier than 1.09a it was entirely up to the user to apply \xintnum; starting with 1.09a arithmetic macros of xint.sty (like earlier already xintfrac.sty with its own \xintnum) make use of \xintnum. This allows arguments to be count registers, or even \numexpr arbitrary long expressions (with the trick of braces, see the user documentation).

Note (22/06/14): \xintiNum jamais utilisé sous ce nom, le supprimer? \XINT_num maintenant utilisé par le parseur de xintexpr.

```
\def\xintiNum {\romannumeral0\xintinum }%
\def\xintinum #1%
{%
    \expandafter\XINT_num_loop
    \romannumeral-`0#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}%
```

3.9 \xintSgn, \xintiiSgn, \XINT_Sgn, \XINT_cntSgn

Changed in 1.05. Earlier code was unnecessarily strange. 1.09a with \xintnum 1.09i defines \XINT_Sgn and \XINT_cntSgn (was \XINT__Sgn in 1.09i) for reasons of internal optimizations.
xintfrac.sty will overwrite \xintsgn with use of \xintraw rather than \xintnum, naturally.

```
\def\xintiiSgn {\romannumeral0\xintiisgn }%
\def\xintiisgn #1%
{%
    \expandafter\XINT_sgn \romannumeral-`0#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\m@ne % I will not allocate a count only for -1?
       0-\xint_c_i
    \krof
}%
```

3.10 \xintiOpp

\xintnum added in 1.09a

```
\def\xintiiOpp {\romannumeral0\xintiiopp }%
\def\xintiiopp #1%
{%
    \expandafter\XINT_opp \romannumeral-`0#1%
}%
\def\xintiOpp {\romannumeral0\xintiopp }%
\def\xintiopp #1%
{%
    \expandafter\XINT_opp \romannumeral0\xintnum{#1}%
}%
\let\xintOpp\xintiOpp \let\xintopp\xintiopp
\def\XINT_Opp #1{\romannumeral0\XINT_opp #1}%
\def\XINT_opp #1%
{%
    \xint_UDzerominusfork
      #1-{ 0}%      zero
      0#1{ }%     negative
       0-{ -#1}%  positive
    \krof
}%
```

3.11 \xintiAbs, \xintiiAbs

Release 1.09a has now \xintiabs which does \xintnum and this is inherited by DecSplit, by Sqr, and macros of xintgcd.sty. Attention, car ces macros de toute façon doivent passer à la valeur absolue et donc en profite pour faire le \xintnum, mais pour optimisation sans overhead il vaut mieux utiliser \xintiiAbs ou autre point d'accès.

```
\def\xintiiAbs {\romannumeral0\xintiiabs }%
\def\xintiiabs #1%
{%
    \expandafter\XINT_abs \romannumeral-`0#1%
}%
\def\xintiAbs {\romannumeral0\xintiabs }%
\def\xintiabs #1%
{%
    \expandafter\XINT_abs \romannumeral0\xintnum{#1}%
}%
\let\xintAbs\xintiAbs \let\xintabs\xintiabs
\def\XINT_Abs #1{\romannumeral0\XINT_abs #1}%
```

```
\def\XINT_abs #1%
{%
    \xint_UDsignfork
      #1{ }%
       -{ #1}%
    \krof
}%
```

3.12 \xintFDg, \xintiiFDg

FIRST DIGIT. Code simplified in 1.05. And prepared for redefinition by xintfrac to parse through \xintNum. Version 1.09a inserts the \xintnum already here.

```
\def\xintiiFDg {\romannumeral0\xintiifdg }%
\def\xintiifdg #1%
{%
    \expandafter\XINT_fdg \romannumeral-`0#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
}%
```

3.13 \xintLDg, \xintiiLDg

LAST DIGIT. Simplified in 1.05. And prepared for extension by xintfrac to parse through \xintNum. Release 1.09a adds the \xintnum already here, and this propagates to \xintOdd, etc... 1.09e The \xintiiLDg is for defining \xintiiOdd which is used once (currently) elsewhere .
  bug fix (1.1b): \xintiiLDg is needed by the division macros next, thus it needs to be in the xintcore.sty

```
\def\xintiiLDg {\romannumeral0\xintiildg }%
\def\xintiildg #1%
{%
    \expandafter\XINT_ldg\expandafter {\romannumeral-`0#1}%
}%
\def\xintLDg {\romannumeral0\xintldg }%
\def\xintldg #1%
{%
    \expandafter\XINT_ldg\expandafter {\romannumeral0\xintnum{#1}}%
}%
\def\XINT_LDg #1{\romannumeral0\XINT_ldg {#1}}%
\def\XINT_ldg #1%
```

```
{%
    \expandafter\XINT_ldg_\romannumeral0\xintreverseorder {#1}\Z
}%
\def\XINT_ldg_ #1#2\Z{ #1}%
```

3.14 \xintDouble

v1.08

```
\def\xintDouble {\romannumeral0\xintdouble }%
\def\xintdouble #1%
{%
     \expandafter\XINT_dbl\romannumeral-`0#1%
     \R\R\R\R\R\R\Z \W\W\W\W\W\W\W
}%
\def\XINT_dbl #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_dbl_zero
      0#1\XINT_dbl_neg
       0-{\XINT_dbl_pos #1}%
    \krof
}%
\def\XINT_dbl_zero #1\Z \W\W\W\W\W\W\W { 0}%
\def\XINT_dbl_neg
   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_dbl_pos }%
\def\XINT_dbl_pos
{%
    \expandafter\XINT_dbl_a \expandafter{\expandafter}\expandafter 0%
    \romannumeral0\XINT_SQ {}%
}%
\def\XINT_dbl_a #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_W #9\XINT_dbl_end_a\W
    \expandafter\XINT_dbl_b
    \the\numexpr \xint_c_x^viii+#2+\xint_c_ii*#9#8#7#6#5#4#3\relax {#1}%
}%
\def\XINT_dbl_b 1#1#2#3#4#5#6#7#8#9%
{%
    \XINT_dbl_a {#2#3#4#5#6#7#8#9}{#1}%
}%
\def\XINT_dbl_end_a #1+#2+#3\relax #4%
{%
    \expandafter\XINT_dbl_end_b #2#4%
}%
\edef\XINT_dbl_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
}%
```

3.15 \xintHalf

v1.08. Relase 1.1 left it in xint.sty, but it is needed by the division routines included in xintcore.sty. Thus moved here for bugfix release 1.1c. Also \XINT_SQ which it uses. Moved here \xintDouble as well by sympathy.

```
\def\xintHalf {\romannumeral0\xinthalf }%
\def\xinthalf #1%
{%
     \expandafter\XINT_half\romannumeral-`0#1%
     \R\R\R\R\R\R\Z \W\W\W\W\W\W\W
}%
\def\XINT_half #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_half_zero
      0#1\XINT_half_neg
       0-{\XINT_half_pos #1}%
    \krof
}%
\def\XINT_half_zero #1\Z \W\W\W\W\W\W\W { 0}%
\def\XINT_half_neg {\expandafter\XINT_opp\romannumeral0\XINT_half_pos }%
\def\XINT_half_pos {\expandafter\XINT_half_a\romannumeral0\XINT_SQ {}}%
\def\XINT_half_a #1#2#3#4#5#6#7#8%
{%
    \xint_gob_til_W #8\XINT_half_dont\W
    \expandafter\XINT_half_b
    \the\numexpr \xint_c_x^viii+\xint_c_v*#7#6#5#4#3#2#1\relax #8%
}%
\edef\XINT_half_dont\W\expandafter\XINT_half_b
    \the\numexpr \xint_c_x^viii+\xint_c_v*#1#2#3#4#5#6#7\relax \W\W\W\W\W\W\W
{%
    \noexpand\expandafter\space
    \noexpand\the\numexpr (#1#2#3#4#5#6#7+\xint_c_i)/\xint_c_ii-\xint_c_i \relax
}%
\def\XINT_half_b 1#1#2#3#4#5#6#7#8%
{%
    \XINT_half_c {#2#3#4#5#6#7}{#1}%
}%
\def\XINT_half_c #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_W #3\XINT_half_end_a #2\W
    \expandafter\XINT_half_d
    \the\numexpr \xint_c_x^viii+\xint_c_v*#9#8#7#6#5#4#3+#2\relax {#1}%
}%
\def\XINT_half_d 1#1#2#3#4#5#6#7#8#9%
{%
    \XINT_half_c {#2#3#4#5#6#7#8#9}{#1}%
}%
\def\XINT_half_end_a #1\W #2\relax #3%
{%
    \xint_gob_til_zero #1\XINT_half_end_b 0\space #1#3%
}%
```

```
\edef\XINT_half_end_b 0\space 0#1#2#3#4#5#6#7%
{%
    \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7\relax
}%
```

3.16 \xintDec

v1.08

```
\def\xintDec {\romannumeral0\xintdec }%
\def\xintdec #1%
{%
     \expandafter\XINT_dec\romannumeral-`0#1%
     \R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W\W
}%
\def\XINT_dec #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_dec_zero
      0#1\XINT_dec_neg
       0-{\XINT_dec_pos #1}%
    \krof
}%
\def\XINT_dec_zero #1\W\W\W\W\W\W\W\W { -1}%
\def\XINT_dec_neg
   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_inc_pos }%
\def\XINT_dec_pos
{%
    \expandafter\XINT_dec_a \expandafter{\expandafter}%
    \romannumeral0\XINT_OQ {}%
}%
\def\XINT_dec_a #1#2#3#4#5#6#7#8#9%
{%
    \expandafter\XINT_dec_b
    \the\numexpr 11#9#8#7#6#5#4#3#2-\xint_c_i\relax {#1}%
}%
\def\XINT_dec_b 1#1%
{%
    \xint_gob_til_one #1\XINT_dec_A 1\XINT_dec_c
}%
\def\XINT_dec_c #1#2#3#4#5#6#7#8#9{\XINT_dec_a {#1#2#3#4#5#6#7#8#9}}%
\def\XINT_dec_A 1\XINT_dec_c #1#2#3#4#5#6#7#8#9%
   {\XINT_dec_B {#1#2#3#4#5#6#7#8#9}}%
\def\XINT_dec_B #1#2\W\W\W\W\W\W\W\W
{%
    \expandafter\XINT_dec_cleanup
    \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%
}%
```

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

3.17 \xintInc

v1.08

```
\def\xintInc {\romannumeral0\xintinc }%
\def\xintinc #1%
{%
     \expandafter\XINT_inc\romannumeral-`0#1%
     \R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W\W
}%
\def\XINT_inc #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_inc_zero
      0#1\XINT_inc_neg
       0-{\XINT_inc_pos #1}%
    \krof
}%
\def\XINT_inc_zero #1\W\W\W\W\W\W\W\W { 1}%
\def\XINT_inc_neg {\expandafter\XINT_opp\romannumeral0\XINT_dec_pos }%
\def\XINT_inc_pos
{%
    \expandafter\XINT_inc_a \expandafter{\expandafter}%
    \romannumeral0\XINT_OQ {}%
}%
\def\XINT_inc_a #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_W #9\XINT_inc_end\W
    \expandafter\XINT_inc_b
    \the\numexpr 10#9#8#7#6#5#4#3#2+\xint_c_i\relax {#1}%
}%
\def\XINT_inc_b 1#1%
{%
    \xint_gob_til_zero #1\XINT_inc_A 0\XINT_inc_c
}%
\def\XINT_inc_c #1#2#3#4#5#6#7#8#9{\XINT_inc_a {#1#2#3#4#5#6#7#8#9}}%
\def\XINT_inc_A 0\XINT_inc_c #1#2#3#4#5#6#7#8#9%
               {\XINT_dec_B {#1#2#3#4#5#6#7#8#9}}%
\def\XINT_inc_end\W #1\relax #2{ 1#2}%
```

3.18 Variants for addition sub-routines

Release 1.03 re-organizes sub-routines to facilitate future developments: the diverse variants of addition, with diverse conditions on inputs and output are first listed; they will be used in multiplication, or in the summation, or in the power routines. I am aware that the commenting is close to non-existent, sorry about that.

Addition and multiplication each have multiple implementations corresponding to slightly differing formats on input and on output.

3.18.1 Addition vI: \XINT_add_A

```
INPUT:
\romannumeral0\XINT_add_A 0{}<N1>\W\X\Y\Z <N2>\W\X\Y\Z
1. <N1> et <N2> renversés
2. de longueur 4n (avec des leading zéros éventuels)
3. l'un des deux ne doit pas se terminer par 0000
[Donc on peut avoir 0000 comme input si l'autre est >0 et ne se termine pas en 0000 bien sûr]. On
peut avoir l'un des deux vides. Mais alors l'autre ne doit être ni vide ni 0000.
  OUTPUT: la somme <N1>+<N2>, ordre normal, plus sur 4n, pas de leading zeros La procédure est plus
rapide lorsque <N1> est le plus court des deux.
Nota bene: (30 avril 2013). J'ai une version qui est deux fois plus rapide sur des nombres d'environ
1000 chiffres chacun, et qui commence à être avantageuse pour des nombres d'au moins 200 chiffres.
Cependant il serait vraiment compliqué d'en étendre l'utilisation aux emplois de l'addition dans
les autres routines, comme celle de multiplication ou celle de division; et son implémentation
ajouterait au minimum la mesure de la longueur des summands.
```

```
\def\XINT_add_A #1#2#3#4#5#6%
{%
    \xint_gob_til_W #3\xint_add_az\W
    \XINT_add_AB #1{#3#4#5#6}{#2}%
}%
\def\xint_add_az\W\XINT_add_AB #1#2%
{%
    \XINT_add_AC_checkcarry #1%
}%
```

```
ici #2 est prévu pour l'addition, mais attention il devra être renversé pour \numexpr. #3 = résul-
tat partiel. #4 = chiffres qui restent. On vérifie si le deuxième nombre s'arrête.
```

```
\def\XINT_add_AB #1#2#3#4\W\X\Y\Z #5#6#7#8%
{%
    \xint_gob_til_W #5\xint_add_bz\W
    \XINT_add_ABE #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
}%
\def\XINT_add_ABE #1#2#3#4#5#6%
{%
    \expandafter\XINT_add_ABEA\the\numexpr #1+10#5#4#3#2+#6.%
}%
\def\XINT_add_ABEA #1#2#3.#4%
{%
    \XINT_add_A  #2{#3#4}%
}%
```

```
ici le deuxième nombre est fini #6 part à la poubelle, #2#3#4#5 est le #2 dans \XINT_add_AB on ne
vérifie pas la retenue cette fois, mais les fois suivantes
```

```
\def\xint_add_bz\W\XINT_add_ABE #1#2#3#4#5#6%
{%
    \expandafter\XINT_add_CC\the\numexpr #1+10#5#4#3#2.%
}%
\def\XINT_add_CC #1#2#3.#4%
{%
    \XINT_add_AC_checkcarry  #2{#3#4}% on va examiner et \'eliminer #2
}%
```

retenue plus chiffres qui restent de l'un des deux nombres. #2 = résultat partiel #3#4#5#6 = summand, avec plus significatif à droite

```
\def\XINT_add_AC_checkcarry #1%
{%
    \xint_gob_til_zero #1\xint_add_AC_nocarry 0\XINT_add_C
}%
\def\xint_add_AC_nocarry 0\XINT_add_C #1#2\W\X\Y\Z
{%
    \expandafter
    \xint_cleanupzeros_andstop
    \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%
}%
\def\XINT_add_C #1#2#3#4#5%
{%
    \xint_gob_til_W #2\xint_add_cz\W
    \XINT_add_CD {#5#4#3#2}{#1}%
}%
\def\XINT_add_CD #1%
{%
    \expandafter\XINT_add_CC\the\numexpr 1+10#1.%
}%
\def\xint_add_cz\W\XINT_add_CD #1#2{ 1#2}%
```

3.18.2 Addition vII: \XINT_addr_A

INPUT: \romannumeral0\XINT_addr_A 0{}<N1>\W\X\Y\Z <N2>\W\X\Y\Z
  Comme \XINT_add_A, la différence principale c'est qu'elle donne son résultat aussi *sur 4n*, renversé. De plus cette variante accepte que l'un ou même les deux inputs soient vides. Utilisé par la sommation et par la division (pour les quotients). Et aussi par la multiplication d'ailleurs.
INPUT: comme pour \XINT_add_A
1. <N1> et <N2> renversés
2. de longueur 4n (avec des leading zéros éventuels)
3. l'un des deux ne doit pas se terminer par 0000
OUTPUT: la somme <N1>+<N2>, *aussi renversée* et *sur 4n*

```
\def\XINT_addr_A #1#2#3#4#5#6%
{%
    \xint_gob_til_W #3\xint_addr_az\W
    \XINT_addr_B #1{#3#4#5#6}{#2}%
}%
\def\xint_addr_az\W\XINT_addr_B #1#2%
{%
    \XINT_addr_AC_checkcarry #1%
}%
\def\XINT_addr_B #1#2#3#4\W\X\Y\Z #5#6#7#8%
{%
```

```
    \xint_gob_til_W #5\xint_addr_bz\W
    \XINT_addr_E #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
}%
\def\XINT_addr_E #1#2#3#4#5#6%
{%
    \expandafter\XINT_addr_ABEA\the\numexpr #1+10#5#4#3#2+#6\relax
}%
\def\XINT_addr_ABEA #1#2#3#4#5#6#7%
{%
    \XINT_addr_A  #2{#7#6#5#4#3}%
}%
\def\xint_addr_bz\W\XINT_addr_E #1#2#3#4#5#6%
{%
    \expandafter\XINT_addr_CC\the\numexpr #1+10#5#4#3#2\relax
}%
\def\XINT_addr_CC #1#2#3#4#5#6#7%
{%
    \XINT_addr_AC_checkcarry  #2{#7#6#5#4#3}%
}%
\def\XINT_addr_AC_checkcarry #1%
{%
    \xint_gob_til_zero #1\xint_addr_AC_nocarry 0\XINT_addr_C
}%
\def\xint_addr_AC_nocarry 0\XINT_addr_C #1#2\W\X\Y\Z { #1#2}%
\def\XINT_addr_C #1#2#3#4#5%
{%
    \xint_gob_til_W #2\xint_addr_cz\W
    \XINT_addr_D {#5#4#3#2}{#1}%
}%
\def\XINT_addr_D #1%
{%
    \expandafter\XINT_addr_CC\the\numexpr 1+10#1\relax
}%
\def\xint_addr_cz\W\XINT_addr_D #1#2{ #21000}%
```

3.18.3 Addition vIII: \XINT_addm_A

```
INPUT:\romannumeral0\XINT_addm_A 0{}<N1>\W\X\Y\Z <N2>\W\X\Y\Z
1. <N1> et <N2> renversés
2. <N1> de longueur 4n ; <N2> non
3. <N2> est *garanti au moins aussi long* que <N1>
OUTPUT: la somme <N1>+<N2>, ordre normal, pas sur 4n, leading zeros retirés. Utilisé par la mul-
tiplication.
```

```
\def\XINT_addm_A #1#2#3#4#5#6%
{%
    \xint_gob_til_W #3\xint_addm_az\W
    \XINT_addm_AB #1{#3#4#5#6}{#2}%
}%
\def\xint_addm_az\W\XINT_addm_AB #1#2%
{%
    \XINT_addm_AC_checkcarry #1%
}%
```

```
\def\XINT_addm_AB #1#2#3#4\W\X\Y\Z #5#6#7#8%
{%
    \XINT_addm_ABE #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
}%
\def\XINT_addm_ABE #1#2#3#4#5#6%
{%
    \expandafter\XINT_addm_ABEA\the\numexpr #1+10#5#4#3#2+#6.%
}%
\def\XINT_addm_ABEA #1#2#3.#4%
{%
    \XINT_addm_A  #2{#3#4}%
}%
\def\XINT_addm_AC_checkcarry #1%
{%
    \xint_gob_til_zero #1\xint_addm_AC_nocarry 0\XINT_addm_C
}%
\def\xint_addm_AC_nocarry 0\XINT_addm_C #1#2\W\X\Y\Z
{%
    \expandafter
    \xint_cleanupzeros_andstop
    \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%
}%
\def\XINT_addm_C #1#2#3#4#5%
{%
    \xint_gob_til_W
    #5\xint_addm_cw
    #4\xint_addm_cx
    #3\xint_addm_cy
    #2\xint_addm_cz
    \W\XINT_addm_CD {#5#4#3#2}{#1}%
}%
\def\XINT_addm_CD #1%
{%
    \expandafter\XINT_addm_CC\the\numexpr 1+10#1.%
}%
\def\XINT_addm_CC #1#2#3.#4%
{%
    \XINT_addm_AC_checkcarry  #2{#3#4}%
}%
\def\xint_addm_cw
    #1\xint_addm_cx
    #2\xint_addm_cy
    #3\xint_addm_cz
    \W\XINT_addm_CD
{%
    \expandafter\XINT_addm_CDw\the\numexpr 1+#1#2#3.%
```

```
}%
\def\XINT_addm_CDw #1.#2#3\X\Y\Z
{%
    \XINT_addm_end #1#3%
}%
\def\xint_addm_cx
    #1\xint_addm_cy
    #2\xint_addm_cz
    \W\XINT_addm_CD
{%
    \expandafter\XINT_addm_CDx\the\numexpr 1+#1#2.%
}%
\def\XINT_addm_CDx #1.#2#3\Y\Z
{%
    \XINT_addm_end #1#3%
}%
\def\xint_addm_cy
    #1\xint_addm_cz
    \W\XINT_addm_CD
{%
    \expandafter\XINT_addm_CDy\the\numexpr 1+#1.%
}%
\def\XINT_addm_CDy  #1.#2#3\Z
{%
    \XINT_addm_end #1#3%
}%
\def\xint_addm_cz\W\XINT_addm_CD #1#2#3{\XINT_addm_end #1#3}%
\edef\XINT_addm_end #1#2#3#4#5%
    {\noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5\relax}%
```

3.18.4 Addition vIV: \XINT_addp_A

```
INPUT: \romannumeral0\XINT_addp_A 0{}<N1>\W\X\Y\Z <N2>\W\X\Y\Z
1. <N1> et <N2> renversés
2. <N1> de longueur 4n ; <N2> non
3. <N2> est *garanti au moins aussi long* que <N1>
OUTPUT: la somme <N1>+<N2>, dans l'ordre renversé, sur 4n, et en faisant attention de ne pas ter-
miner en 0000. Utilisé par la multiplication servant pour le calcul des puissances.
```

```
\def\XINT_addp_A #1#2#3#4#5#6%
{%
    \xint_gob_til_W #3\xint_addp_az\W
    \XINT_addp_AB #1{#3#4#5#6}{#2}%
}%
\def\xint_addp_az\W\XINT_addp_AB #1#2%
{%
    \XINT_addp_AC_checkcarry #1%
}%
\def\XINT_addp_AC_checkcarry #1%
{%
    \xint_gob_til_zero #1\xint_addp_AC_nocarry 0\XINT_addp_C
}%
\def\xint_addp_AC_nocarry 0\XINT_addp_C
```

```
{%
    \XINT_addp_F
}%
\def\XINT_addp_AB #1#2#3#4\W\X\Y\Z #5#6#7#8%
{%
    \XINT_addp_ABE #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
}%
\def\XINT_addp_ABE #1#2#3#4#5#6%
{%
    \expandafter\XINT_addp_ABEA\the\numexpr #1+10#5#4#3#2+#6\relax
}%
\def\XINT_addp_ABEA #1#2#3#4#5#6#7%
{%
   \XINT_addp_A  #2{#7#6#5#4#3}%<-- attention on met donc \`a droite
}%
\def\XINT_addp_C #1#2#3#4#5%
{%
    \xint_gob_til_W
    #5\xint_addp_cw
    #4\xint_addp_cx
    #3\xint_addp_cy
    #2\xint_addp_cz
    \W\XINT_addp_CD   {#5#4#3#2}{#1}%
}%
\def\XINT_addp_CD #1%
{%
    \expandafter\XINT_addp_CC\the\numexpr 1+10#1\relax
}%
\def\XINT_addp_CC #1#2#3#4#5#6#7%
{%
    \XINT_addp_AC_checkcarry  #2{#7#6#5#4#3}%
}%
\def\xint_addp_cw
    #1\xint_addp_cx
    #2\xint_addp_cy
    #3\xint_addp_cz
    \W\XINT_addp_CD
{%
    \expandafter\XINT_addp_CDw\the\numexpr \xint_c_i+10#1#2#3\relax
}%
\def\XINT_addp_CDw #1#2#3#4#5#6%
{%
    \xint_gob_til_zeros_iv #2#3#4#5\XINT_addp_endDw_zeros
                           0000\XINT_addp_endDw #2#3#4#5%
}%
\def\XINT_addp_endDw_zeros 0000\XINT_addp_endDw 0000#1\X\Y\Z{ #1}%
\def\XINT_addp_endDw #1#2#3#4#5\X\Y\Z{ #5#4#3#2#1}%
\def\xint_addp_cx
    #1\xint_addp_cy
    #2\xint_addp_cz
    \W\XINT_addp_CD
{%
```

```
    \expandafter\XINT_addp_CDx\the\numexpr \xint_c_i+100#1#2\relax
}%
\def\XINT_addp_CDx #1#2#3#4#5#6%
{%
    \xint_gob_til_zeros_iv #2#3#4#5\XINT_addp_endDx_zeros
                           0000\XINT_addp_endDx #2#3#4#5%
}%
\def\XINT_addp_endDx_zeros 0000\XINT_addp_endDx 0000#1\Y\Z{ #1}%
\def\XINT_addp_endDx #1#2#3#4#5\Y\Z{ #5#4#3#2#1}%
\def\xint_addp_cy #1\xint_addp_cz\W\XINT_addp_CD
{%
    \expandafter\XINT_addp_CDy\the\numexpr \xint_c_i+1000#1\relax
}%
\def\XINT_addp_CDy #1#2#3#4#5#6%
{%
    \xint_gob_til_zeros_iv #2#3#4#5\XINT_addp_endDy_zeros
                           0000\XINT_addp_endDy #2#3#4#5%
}%
\def\XINT_addp_endDy_zeros 0000\XINT_addp_endDy 0000#1\Z{ #1}%
\def\XINT_addp_endDy #1#2#3#4#5\Z{ #5#4#3#2#1}%
\def\xint_addp_cz\W\XINT_addp_CD #1#2{ #21000}%
\def\XINT_addp_F #1#2#3#4#5%
{%
    \xint_gob_til_W
    #5\xint_addp_Gw
    #4\xint_addp_Gx
    #3\xint_addp_Gy
    #2\xint_addp_Gz
    \W\XINT_addp_G   {#2#3#4#5}{#1}%
}%
\def\XINT_addp_G #1#2%
{%
    \XINT_addp_F {#2#1}%
}%
\def\xint_addp_Gw
    #1\xint_addp_Gx
    #2\xint_addp_Gy
    #3\xint_addp_Gz
    \W\XINT_addp_G #4%
{%
    \xint_gob_til_zeros_iv  #3#2#10\XINT_addp_endGw_zeros
                           0000\XINT_addp_endGw #3#2#10%
}%
\def\XINT_addp_endGw_zeros 0000\XINT_addp_endGw 0000#1\X\Y\Z{ #1}%
\def\XINT_addp_endGw #1#2#3#4#5\X\Y\Z{ #5#1#2#3#4}%
\def\xint_addp_Gx
    #1\xint_addp_Gy
    #2\xint_addp_Gz
    \W\XINT_addp_G #3%
{%
    \xint_gob_til_zeros_iv  #2#100\XINT_addp_endGx_zeros
                           0000\XINT_addp_endGx #2#100%
```

```
}%
\def\XINT_addp_endGx_zeros 0000\XINT_addp_endGx 0000#1\Y\Z{ #1}%
\def\XINT_addp_endGx #1#2#3#4#5\Y\Z{ #5#1#2#3#4}%
\def\xint_addp_Gy
    #1\xint_addp_Gz
    \W\XINT_addp_G #2%
{%
    \xint_gob_til_zeros_iv   #1000\XINT_addp_endGy_zeros
                           0000\XINT_addp_endGy #1000%
}%
\def\XINT_addp_endGy_zeros 0000\XINT_addp_endGy 0000#1\Z{ #1}%
\def\XINT_addp_endGy #1#2#3#4#5\Z{ #5#1#2#3#4}%
\def\xint_addp_Gz\W\XINT_addp_G #1#2{ #2}%
```

3.19 \xintiAdd, \xintiiAdd

ADDITION [algo plus efficace lorsque le premier argument plus long que le second]
  Note (octobre 2014, pendant la préparation de la sortie de 1.1)
  Je n'aurais pas dû l'appeler \xintAdd, mais seulement \xintiAdd. Le format de sortie de \xintAdd est modifié par xintfrac.sty, celui de \xintiAdd ne bouge pas, et \xintiiAdd reste la version stricte.

```
\def\xintiiAdd {\romannumeral0\xintiiadd }%
\def\xintiiadd #1{\expandafter\xint_iiadd\romannumeral-`0#1\Z  }%
\def\xint_iiadd #1#2\Z #3%
{%
    \expandafter\XINT_add_fork\expandafter #1\romannumeral-`0#3\Z #2\Z
}%
\def\xintiAdd {\romannumeral0\xintiadd }%
\def\xintiadd #1%
{%
    \expandafter\xint_add\romannumeral0\xintnum{#1}\Z
}%
\def\xint_add #1#2\Z #3%
{%
    \expandafter\XINT_add_fork\expandafter #1\romannumeral0\xintnum{#3}\Z #2\Z
}%
\let\xintAdd\xintiAdd \let\xintadd\xintiadd
\def\XINT_add_fork #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 #2#3\Z #4\Z { #3}%
```

```
\def\XINT_add_secondiszero #1\krof #2#3\Z #4\Z { #2#4}%
\def\XINT_add_plusplus     #1#2#3\Z #4\Z {\XINT_add_pre {#1#4}{#2#3}}%
\def\XINT_add_minusminus   #1#2#3\Z #4\Z
   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_add_pre {#4}{#3}}%
\def\XINT_add_minusplus    #1#2#3\Z #4\Z {\XINT_sub_pre {#2#3}{#4}}%
\def\XINT_add_plusminus    #1#2#3\Z #4\Z {\XINT_sub_pre {#1#4}{#3}}%
```

positive summands

```
\def\XINT_add_pre #1%
{%
  \expandafter\XINT_add_pre_b\expandafter
  {\romannumeral0\XINT_RQ {}#1\R\R\R\R\R\R\R\Z }%
}%
\def\XINT_add_pre_b #1#2%
{%
    \expandafter\XINT_add_A
        \expandafter0\expandafter{\expandafter}%
    \romannumeral0\XINT_RQ {}#2\R\R\R\R\R\R\R\Z
        \W\X\Y\Z #1\W\X\Y\Z
}%
```

3.20 \xintiSub, \xintiiSub

Release 1.09a has \xintnum added into \xintiSub.

```
\def\xintiiSub   {\romannumeral0\xintiisub }%
\def\xintiisub #1{\expandafter\xint_iisub\romannumeral-`0#1\Z  }%
\def\xint_iisub #1#2\Z #3%
{%
    \expandafter\XINT_sub_fork\expandafter #1\romannumeral-`0#3\Z #2\Z
}%
\def\xintiSub {\romannumeral0\xintisub }%
\def\xintisub #1%
{%
    \expandafter\xint_sub\romannumeral0\xintnum{#1}\Z
}%
\def\xint_sub #1#2\Z #3%
{%
    \expandafter\XINT_sub_fork\expandafter #1\romannumeral0\xintnum{#3}\Z #2\Z
}%
\let\xintSub\xintiSub \let\xintsub\xintisub
\def\XINT_sub_fork #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 #2#3\Z #4\Z {\XINT_opp #3}%
\def\XINT_sub_secondiszero #1\krof #2#3\Z #4\Z { #2#4}%
\def\XINT_sub_plusplus     #1#2#3\Z #4\Z {\XINT_sub_pre {#1#4}{#2#3}}%
\def\XINT_sub_minusminus   #1#2#3\Z #4\Z {\XINT_sub_pre {#3}{#4}}%
\def\XINT_sub_minusplus    #1#2#3\Z #4\Z
   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_add_pre {#4}{#2#3}}%
\def\XINT_sub_plusminus    #1#2#3\Z #4\Z {\XINT_add_pre {#1#4}{#3}}%
```

```
SOUSTRACTION A-B avec A premier argument, B second argument de \xintSub et ensuite \XINT_sub_pre
ici
```

```
\def\XINT_sub_pre #1%
{%
  \expandafter\XINT_sub_pre_b\expandafter
  {\romannumeral0\XINT_RQ {}#1\R\R\R\R\R\R\R\Z }%
}%
\def\XINT_sub_pre_b #1#2%
{%
    \expandafter\XINT_sub_A
        \expandafter1\expandafter{\expandafter}%
    \romannumeral0\XINT_RQ {}#2\R\R\R\R\R\R\R\Z
        \W\X\Y\Z #1 \W\X\Y\Z
}%
```

```
\romannumeral0\XINT_sub_A 1{}<N1>\W\X\Y\Z<N2>\W\X\Y\Z
N1 et N2 sont présentés à l'envers ET ON A RAJOUTÉ DES ZÉROS POUR QUE LEURS LONGUEURS À CHACUN
SOIENT MULTIPLES DE 4, MAIS AUCUN NE SE TERMINE EN 0000.
output: N2 - N1
Elle donne le résultat dans le **bon ordre**, avec le bon signe, et sans zéros superflus.
```

```
\def\XINT_sub_A #1#2#3\W\X\Y\Z #4#5#6#7%
{%
    \xint_gob_til_W
    #4\xint_sub_az
    \W\XINT_sub_B #1{#4#5#6#7}{#2}#3\W\X\Y\Z
}%
\def\XINT_sub_B #1#2#3#4#5#6#7%
{%
    \xint_gob_til_W
    #4\xint_sub_bz
    \W\XINT_sub_onestep #1#2{#7#6#5#4}{#3}%
}%
```

```
d'abord la branche principale #6 = 4 chiffres de N1, plus significatif en *premier*, #2#3#4#5
chiffres de N2, plus significatif en *dernier* On veut N2 - N1.
```

```
\def\XINT_sub_onestep #1#2#3#4#5#6%
{%
    \expandafter\XINT_sub_backtoA\the\numexpr 11#5#4#3#2-#6+#1-\xint_c_i.%
}%
```

```
ON PRODUIT LE RÉSULTAT DANS LE BON ORDRE

\def\XINT_sub_backtoA #1#2#3.#4%
{%
    \XINT_sub_A #2{#3#4}%
}%
\def\xint_sub_bz
    \W\XINT_sub_onestep #1#2#3#4#5#6#7%
{%
    \xint_UDzerofork
      #1\XINT_sub_C   % une retenue
       0\XINT_sub_D   % pas de retenue
    \krof
    {#7}#2#3#4#5%
}%
\def\XINT_sub_D #1#2\W\X\Y\Z
{%
    \expandafter
    \xint_cleanupzeros_andstop
    \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%
}%
\def\XINT_sub_C #1#2#3#4#5%
{%
    \xint_gob_til_W
    #2\xint_sub_cz
    \W\XINT_sub_AC_onestep {#5#4#3#2}{#1}%
}%
\def\XINT_sub_AC_onestep #1%
{%
    \expandafter\XINT_sub_backtoC\the\numexpr 11#1-\xint_c_i.%
}%
\def\XINT_sub_backtoC #1#2#3.#4%
{%
    \XINT_sub_AC_checkcarry #2{#3#4}% la retenue va \^etre examin\'ee
}%
\def\XINT_sub_AC_checkcarry #1%
{%
    \xint_gob_til_one #1\xint_sub_AC_nocarry 1\XINT_sub_C
}%
\def\xint_sub_AC_nocarry 1\XINT_sub_C #1#2\W\X\Y\Z
{%
    \expandafter
    \XINT_cuz_loop
    \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\W\W\W\W\W\W\W\Z
}%
\def\xint_sub_cz\W\XINT_sub_AC_onestep #1%
{%
    \XINT_cuz
}%
\def\xint_sub_az\W\XINT_sub_B #1#2#3#4#5#6#7%
{%
    \xint_gob_til_W
    #4\xint_sub_ez
    \W\XINT_sub_Eenter #1{#3}#4#5#6#7%
}%
```

le premier nombre continue, le résultat sera < 0.

```
\def\XINT_sub_Eenter #1#2%
{%
    \expandafter
    \XINT_sub_E\expandafter1\expandafter{\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
    \W\X\Y\Z #1%
}%
\def\XINT_sub_E #1#2#3#4#5#6%
{%
    \xint_gob_til_W #3\xint_sub_F\W
    \XINT_sub_Eonestep #1{#6#5#4#3}{#2}%
}%
\def\XINT_sub_Eonestep #1#2%
{%
    \expandafter\XINT_sub_backtoE\the\numexpr 109999-#2+#1.%
}%
\def\XINT_sub_backtoE #1#2#3.#4%
{%
    \XINT_sub_E #2{#3#4}%
}%
\def\xint_sub_F\W\XINT_sub_Eonestep #1#2#3#4%
{%
    \xint_UDonezerofork
      #4#1{\XINT_sub_Fdec 0}% soustraire  1. Et faire signe -
      #1#4{\XINT_sub_Finc 1}% additionner 1. Et faire signe -
        10\XINT_sub_DD     % terminer. Mais avec signe -
    \krof
    {#3}%
}%
\def\XINT_sub_DD {\expandafter\xint_minus_thenstop\romannumeral0\XINT_sub_D }%
\def\XINT_sub_Fdec #1#2#3#4#5#6%
```

```
{%
    \xint_gob_til_W #3\xint_sub_Fdec_finish\W
    \XINT_sub_Fdec_onestep #1{#6#5#4#3}{#2}%
}%
\def\XINT_sub_Fdec_onestep #1#2%
{%
    \expandafter\XINT_sub_backtoFdec\the\numexpr 11#2+#1-\xint_c_i.%
}%
\def\XINT_sub_backtoFdec #1#2#3.#4%
{%
    \XINT_sub_Fdec #2{#3#4}%
}%
\def\xint_sub_Fdec_finish\W\XINT_sub_Fdec_onestep #1#2%
{%
    \expandafter\xint_minus_thenstop\romannumeral0\XINT_cuz
}%
\def\XINT_sub_Finc #1#2#3#4#5#6%
{%
    \xint_gob_til_W #3\xint_sub_Finc_finish\W
    \XINT_sub_Finc_onestep #1{#6#5#4#3}{#2}%
}%
\def\XINT_sub_Finc_onestep #1#2%
{%
    \expandafter\XINT_sub_backtoFinc\the\numexpr 10#2+#1.%
}%
\def\XINT_sub_backtoFinc #1#2#3.#4%
{%
    \XINT_sub_Finc #2{#3#4}%
}%
\def\xint_sub_Finc_finish\W\XINT_sub_Finc_onestep #1#2#3%
{%
    \xint_UDzerofork
     #1{\expandafter\expandafter\expandafter
        \xint_minus_thenstop\xint_cleanupzeros_nostop}%
      0{ -1}%
    \krof
    #3%
}%
\def\xint_sub_ez\W\XINT_sub_Eenter #1%
{%
    \xint_UDzerofork
      #1\XINT_sub_K % il y a une retenue
       0\XINT_sub_L % pas de retenue
    \krof
}%
\def\XINT_sub_L #1\W\X\Y\Z {\XINT_cuz_loop #1\W\W\W\W\W\W\W\Z }%
\def\XINT_sub_K #1%
{%
    \expandafter
    \XINT_sub_KK\expandafter1\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
}%
\def\XINT_sub_KK #1#2#3#4#5#6%
{%
    \xint_gob_til_W #3\xint_sub_KK_finish\W
    \XINT_sub_KK_onestep #1{#6#5#4#3}{#2}%
}%
\def\XINT_sub_KK_onestep #1#2%
{%
    \expandafter\XINT_sub_backtoKK\the\numexpr 109999-#2+#1.%
}%
\def\XINT_sub_backtoKK #1#2#3.#4%
{%
    \XINT_sub_KK #2{#3#4}%
}%
\def\xint_sub_KK_finish\W\XINT_sub_KK_onestep #1#2#3%
{%
    \expandafter\xint_minus_thenstop
    \romannumeral0\XINT_cuz_loop #3\W\W\W\W\W\W\W\Z
}%
```

3.21 \xintiMul, \xintiiMul

1.09a adds \xintnum

```
\def\xintiiMul {\romannumeral0\xintiimul }%
\def\xintiimul #1%
{%
    \expandafter\xint_iimul\expandafter {\romannumeral-`0#1}%
}%
\def\xint_iimul #1#2%
{%
    \expandafter\XINT_mul_fork \romannumeral-`0#2\Z #1\Z
}%
\def\xintiMul {\romannumeral0\xintimul }%
\def\xintimul #1%
{%
    \expandafter\xint_mul\expandafter {\romannumeral0\xintnum{#1}}%
}%
\def\xint_mul #1#2%
{%
    \expandafter\XINT_mul_fork \romannumeral0\xintnum{#2}\Z #1\Z
}%
\let\xintMul\xintiMul \let\xintmul\xintimul
\def\XINT_Mul #1#2{\romannumeral0\XINT_mul_fork #2\Z #1\Z }%
```

MULTIPLICATION
Ici #1#2 = 2e input et #3#4 = 1er input
Release 1.03 adds some overhead to first compute and compare the lengths of the two inputs. The algorithm is asymmetrical and whether the first input is the longest or the shortest sometimes

has a strong impact. 50 digits times 1000 digits used to be 5 times faster than 1000 digits times 50 digits. With the new code, the user input order does not matter as it is decided by the routine what is best. This is important for the extension to fractions, as there is no way then to generally control or guess the most frequent sizes of the inputs besides actually computing their lengths.

```
\def\XINT_mul_fork #1#2\Z #3#4\Z
{%
    \xint_UDzerofork
      #1\XINT_mul_zero
      #3\XINT_mul_zero
       0{}%
    \krof
    \xint_UDsignsfork
          #1#3\XINT_mul_minusminus          % #1 = #3 = -
           #1-{\XINT_mul_minusplus #3}%          % #1 = -
           #3-{\XINT_mul_plusminus #1}%          % #3 = -
            --{\XINT_mul_plusplus #1#3}%
    \krof
    {#2}{#4}%
}%
\def\XINT_mul_zero #1\krof #2#3{ 0}%
\def\XINT_mul_minusminus #1#2%
{%
      \expandafter\XINT_mul_choice_a
      \expandafter{\romannumeral0\xintlength {#2}}%
      {\romannumeral0\xintlength {#1}}{#1}{#2}%
}%
\def\XINT_mul_minusplus #1#2#3%
{%
    \expandafter\xint_minus_thenstop\romannumeral0\expandafter
    \XINT_mul_choice_a
    \expandafter{\romannumeral0\xintlength {#1#3}}%
    {\romannumeral0\xintlength {#2}}{#2}{#1#3}%
}%
\def\XINT_mul_plusminus #1#2#3%
{%
    \expandafter\xint_minus_thenstop\romannumeral0\expandafter
    \XINT_mul_choice_a
    \expandafter{\romannumeral0\xintlength {#3}}%
    {\romannumeral0\xintlength {#1#2}}{#1#2}{#3}%
}%
\def\XINT_mul_plusplus #1#2#3#4%
{%
    \expandafter\XINT_mul_choice_a
    \expandafter{\romannumeral0\xintlength {#2#4}}%
    {\romannumeral0\xintlength {#1#3}}{#1#3}{#2#4}%
}%
\def\XINT_mul_choice_a #1#2%
{%
    \expandafter\XINT_mul_choice_b\expandafter{#2}{#1}%
}%
\def\XINT_mul_choice_b #1#2%
{%
```

```
    \ifnum #1<\xint_c_v
       \expandafter\XINT_mul_choice_littlebyfirst
    \else
    \ifnum #2<\xint_c_v
      \expandafter\expandafter\expandafter\XINT_mul_choice_littlebysecond
      \else
      \expandafter\expandafter\expandafter\XINT_mul_choice_compare
      \fi
    \fi
    {#1}{#2}%
}%
\def\XINT_mul_choice_littlebyfirst #1#2#3#4%
{%
    \expandafter\XINT_mul_M
    \expandafter{\the\numexpr #3\expandafter}%
    \romannumeral0\XINT_RQ {}#4\R\R\R\R\R\R\R\Z \Z\Z\Z\Z
}%
\def\XINT_mul_choice_littlebysecond #1#2#3#4%
{%
    \expandafter\XINT_mul_M
    \expandafter{\the\numexpr #4\expandafter}%
    \romannumeral0\XINT_RQ {}#3\R\R\R\R\R\R\R\Z \Z\Z\Z\Z
}%
\def\XINT_mul_choice_compare #1#2%
{%
    \ifnum #1>#2
        \expandafter \XINT_mul_choice_i
    \else
        \expandafter \XINT_mul_choice_ii
    \fi
    {#1}{#2}%
}%
\def\XINT_mul_choice_i #1#2%
{%
   \ifnum #1<\numexpr\ifcase \numexpr (#2-\xint_c_iii)/\xint_c_iv\relax
                      \or 330\or 168\or 109\or 80\or 66\or 52\else 0\fi\relax
       \expandafter\XINT_mul_choice_same
   \else
       \expandafter\XINT_mul_choice_permute
   \fi
}%
\def\XINT_mul_choice_ii #1#2%
{%
   \ifnum #2<\numexpr\ifcase \numexpr (#1-\xint_c_iii)/\xint_c_iv\relax
                      \or 330\or 168\or 109\or 80\or 66\or 52\else 0\fi\relax
       \expandafter\XINT_mul_choice_permute
   \else
       \expandafter\XINT_mul_choice_same
   \fi
}%
\def\XINT_mul_choice_same #1#2%
{%
```

```
    \expandafter\XINT_mul_enter
    \romannumeral0\XINT_RQ {}#1\R\R\R\R\R\R\R\Z
    \Z\Z\Z\Z #2\W\W\W\W
}%
\def\XINT_mul_choice_permute #1#2%
{%
    \expandafter\XINT_mul_enter
    \romannumeral0\XINT_RQ {}#2\R\R\R\R\R\R\R\Z
    \Z\Z\Z\Z #1\W\W\W\W
}%
```

Cette portion de routine d'addition se branche directement sur _addr_ lorsque le premier nombre est épuisé, ce qui est garanti arriver avant le second nombre. Elle produit son résultat toujours sur 4n, renversé. Ses deux inputs sont garantis sur 4n.

```
\def\XINT_mul_Ar #1#2#3#4#5#6%
{%
    \xint_gob_til_Z #6\xint_mul_br\Z\XINT_mul_Br #1{#6#5#4#3}{#2}%
}%
\def\xint_mul_br\Z\XINT_mul_Br #1#2%
{%
     \XINT_addr_AC_checkcarry #1%
}%
\def\XINT_mul_Br #1#2#3#4\W\X\Y\Z #5#6#7#8%
{%
    \expandafter\XINT_mul_ABEAr
    \the\numexpr #1+10#2+#8#7#6#5.{#3}#4\W\X\Y\Z
}%
\def\XINT_mul_ABEAr #1#2#3#4#5#6.#7%
{%
    \XINT_mul_Ar #2{#7#6#5#4#3}%
}%
```

3.21.1 "Small" multiplication: \XINT_mul_Mr

<< Petite >> multiplication. mul_Mr renvoie le résultat *à l'envers*, sur *4n*
\romannumeral0\XINT_mul_Mr {<n>}<N>\Z\Z\Z\Z
Fait la multiplication de <N> par <n>, qui est < 10000. <N> est présenté *à l'envers*, sur *4n*.
Lorsque <n> vaut 0, donne 0000.

```
\def\XINT_mul_Mr #1%
{%
    \expandafter\XINT_mul_Mr_checkifzeroorone\expandafter{\the\numexpr #1}%
}%
\def\XINT_mul_Mr_checkifzeroorone #1%
{%
    \ifcase #1
      \expandafter\XINT_mul_Mr_zero
    \or
      \expandafter\XINT_mul_Mr_one
    \else
      \expandafter\XINT_mul_Nr
    \fi
```

```
    {0000}{}{#1}%
}%
\def\XINT_mul_Mr_zero #1\Z\Z\Z\Z { 0000}%
\def\XINT_mul_Mr_one #1#2#3#4\Z\Z\Z\Z { #4}%
\def\XINT_mul_Nr #1#2#3#4#5#6#7%
{%
    \xint_gob_til_Z #4\xint_mul_pr\Z\XINT_mul_Pr {#1}{#3}{#7#6#5#4}{#2}{#3}%
}%
\def\XINT_mul_Pr #1#2#3%
{%
    \expandafter\XINT_mul_Lr\the\numexpr \xint_c_x^viii+#1+#2*#3\relax
}%
\def\XINT_mul_Lr 1#1#2#3#4#5#6#7#8#9%
{%
    \XINT_mul_Nr {#1#2#3#4}{#9#8#7#6#5}%
}%
\def\xint_mul_pr\Z\XINT_mul_Pr #1#2#3#4#5%
{%
    \xint_gob_til_zeros_iv #1\XINT_mul_Mr_end_nocarry 0000%
    \XINT_mul_Mr_end_carry #1{#4}%
}%
\def\XINT_mul_Mr_end_nocarry 0000\XINT_mul_Mr_end_carry 0000#1{ #1}%
\def\XINT_mul_Mr_end_carry #1#2#3#4#5{ #5#4#3#2#1}%
```

3.21.2 "Small" multiplication variant: \XINT_mul_M

```
<< Petite >> multiplication. renvoie le résultat *à l'endroit*, avec *nettoyage des leading
zéros*.
\romannumeral0\XINT_mul_M {<n>}<N>\Z\Z\Z\Z
Fait la multiplication de <N> par <n>, qui est < 10000. <N> est présenté *à l'envers*, sur *4n*.
```

```
\def\XINT_mul_M #1%
{%
    \expandafter\XINT_mul_M_checkifzeroorone\expandafter{\the\numexpr #1}%
}%
\def\XINT_mul_M_checkifzeroorone #1%
{%
    \ifcase #1
      \expandafter\XINT_mul_M_zero
    \or
      \expandafter\XINT_mul_M_one
    \else
      \expandafter\XINT_mul_N
    \fi
    {0000}{}{#1}%
}%
\def\XINT_mul_M_zero #1\Z\Z\Z\Z { 0}%
\def\XINT_mul_M_one #1#2#3#4\Z\Z\Z\Z
{%
    \expandafter\xint_cleanupzeros_andstop\romannumeral0\xintreverseorder{#4}%
}%
\def\XINT_mul_N #1#2#3#4#5#6#7%
{%
```

```
    \xint_gob_til_Z #4\xint_mul_p\Z\XINT_mul_P {#1}{#3}{#7#6#5#4}{#2}{#3}%
}%
\def\XINT_mul_P #1#2#3%
{%
    \expandafter\XINT_mul_L\the\numexpr \xint_c_x^viii+#1+#2*#3\relax
}%
\def\XINT_mul_L 1#1#2#3#4#5#6#7#8#9%
{%
    \XINT_mul_N {#1#2#3#4}{#5#6#7#8#9}%
}%
\def\xint_mul_p\Z\XINT_mul_P #1#2#3#4#5%
{%
    \XINT_mul_M_end #1#4%
}%
\edef\XINT_mul_M_end #1#2#3#4#5#6#7#8%
{%
    \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7#8\relax
}%
```

3.21.3 Main routine: \XINT_mul_enter

```
Routine de multiplication principale (attention délimiteurs modifiés pour 1.08)
Le résultat partiel est toujours maintenu avec significatif à droite et il a un nombre multiple de
4 de chiffres
\romannumeral0\XINT_mul_enter <N1>\Z\Z\Z\Z <N2>\W\W\W\W
avec <N1> *renversé*, *longueur 4n* (zéros éventuellement ajoutés au-delà du chiffre le plus sig-
nificatif) et <N2> dans l'ordre *normal*, et pas forcément longueur 4n. pas de signes.
Pour 1.08: dans \XINT_mul_enter et les modifs de 1.03 qui filtrent les courts, on pourrait croire
que le second opérande a au moins quatre chiffres; mais le problème c'est que ceci est appelé par
\XINT_sqr. Et de plus \XINT_sqr est utilisé dans la nouvelle routine d'extraction de racine car-
rée: je ne veux pas rajouter l'overhead à \XINT_sqr de voir si a longueur est au moins 4. Dilemme
donc. Il ne semble pas y avoir d'autres accès directs (celui de big fac n'est pas un problème). J'ai
presque été tenté de faire du 5x4, mais si on veut maintenir les résultats intermédiaires sur 4n,
il y a des complications. Par ailleurs, je modifie aussi un petit peu la façon de coder la suite,
compte tenu du style que j'ai développé ultérieurement. Attention terminaison modifiée pour le
deuxième opérande.
```

```
\def\XINT_mul_enter #1\Z\Z\Z\Z #2#3#4#5%
{%
    \xint_gob_til_W #5\XINT_mul_exit_a\W
    \XINT_mul_start {#2#3#4#5}#1\Z\Z\Z\Z
}%
\def\XINT_mul_exit_a\W\XINT_mul_start #1%
{%
    \XINT_mul_exit_b #1%
}%
\def\XINT_mul_exit_b #1#2#3#4%
{%
    \xint_gob_til_W
      #2\XINT_mul_exit_ci
      #3\XINT_mul_exit_cii
      \W\XINT_mul_exit_ciii #1#2#3#4%
}%
```

```
\def\XINT_mul_exit_ciii #1\W #2\Z\Z\Z\Z \W\W\W
{%
    \XINT_mul_M {#1}#2\Z\Z\Z\Z
}%
\def\XINT_mul_exit_cii\W\XINT_mul_exit_ciii #1\W\W #2\Z\Z\Z\Z \W\W
{%
    \XINT_mul_M {#1}#2\Z\Z\Z\Z
}%
\def\XINT_mul_exit_ci\W\XINT_mul_exit_cii
                      \W\XINT_mul_exit_ciii #1\W\W\W #2\Z\Z\Z\Z \W
{%
    \XINT_mul_M {#1}#2\Z\Z\Z\Z
}%
\def\XINT_mul_start #1#2\Z\Z\Z\Z
{%
    \expandafter\XINT_mul_main\expandafter
    {\romannumeral0\XINT_mul_Mr {#1}#2\Z\Z\Z\Z}#2\Z\Z\Z\Z
}%
\def\XINT_mul_main #1#2\Z\Z\Z\Z #3#4#5#6%
{%
    \xint_gob_til_W #6\XINT_mul_finish_a\W
    \XINT_mul_compute {#3#4#5#6}{#1}#2\Z\Z\Z\Z
}%
\def\XINT_mul_compute #1#2#3\Z\Z\Z\Z
{%
    \expandafter\XINT_mul_main\expandafter
    {\romannumeral0\expandafter
     \XINT_mul_Ar\expandafter0\expandafter{\expandafter}%
     \romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z
     \W\X\Y\Z 0000#2\W\X\Y\Z }#3\Z\Z\Z\Z
}%
```

Ici, le deuxième nombre se termine. Fin du calcul. On utilise la variante \XINT_addm_A de l'addition car on sait que le deuxième terme est au moins aussi long que le premier. Lorsque le multiplicateur avait longueur 4n, la dernière addition a fourni le résultat à l'envers, il faut donc encore le renverser.

```
\def\XINT_mul_finish_a\W\XINT_mul_compute #1%
{%
    \XINT_mul_finish_b #1%
}%
\def\XINT_mul_finish_b #1#2#3#4%
{%
    \xint_gob_til_W
      #1\XINT_mul_finish_c
      #2\XINT_mul_finish_ci
      #3\XINT_mul_finish_cii
      \W\XINT_mul_finish_ciii #1#2#3#4%
}%
\def\XINT_mul_finish_ciii #1\W #2#3\Z\Z\Z\Z \W\W\W
{%
    \expandafter\XINT_addm_A\expandafter0\expandafter{\expandafter}%
    \romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z \W\X\Y\Z 000#2\W\X\Y\Z
```

```
}%
\def\XINT_mul_finish_cii
    \W\XINT_mul_finish_ciii #1\W\W #2#3\Z\Z\Z\Z \W\W
{%
    \expandafter\XINT_addm_A\expandafter0\expandafter{\expandafter}%
    \romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z \W\X\Y\Z 00#2\W\X\Y\Z
}%
\def\XINT_mul_finish_ci #1\XINT_mul_finish_ciii #2\W\W\W #3#4\Z\Z\Z\Z \W
{%
    \expandafter\XINT_addm_A\expandafter0\expandafter{\expandafter}%
    \romannumeral0\XINT_mul_Mr {#2}#4\Z\Z\Z\Z \W\X\Y\Z 0#3\W\X\Y\Z
}%
\def\XINT_mul_finish_c #1\XINT_mul_finish_ciii \W\W\W\W #2#3\Z\Z\Z\Z
{%
    \expandafter\xint_cleanupzeros_andstop\romannumeral0\xintreverseorder{#2}%
}%
```

3.21.4 Variant: \XINT_mulr_enter

```
\romannumeral0\XINT_mulr_enter <N1>\Z\Z\Z\Z <N2>\W\W\W\W
Ici <N1> est à l'envers sur 4n, et <N2> est à l'endroit, pas sur 4n, comme dans \XINT_mul_enter,
mais le résultat est lui-même fourni *à l'envers*, sur *4n* (en faisant attention de ne pas avoir
0000 à la fin).
Utilisé par le calcul des puissances. J'ai modifié dans 1.08 sur le modèle de la nouvelle ver-
sion de \XINT_mul_enter. Je pourrais économiser des macros et fusionner \XINT_mul_enter et
\XINT_mulr_enter. Une autre fois.
```

```
\def\XINT_mulr_enter #1\Z\Z\Z\Z #2#3#4#5%
{%
    \xint_gob_til_W #5\XINT_mulr_exit_a\W
    \XINT_mulr_start {#2#3#4#5}#1\Z\Z\Z\Z
}%
\def\XINT_mulr_exit_a\W\XINT_mulr_start #1%
{%
    \XINT_mulr_exit_b #1%
}%
\def\XINT_mulr_exit_b #1#2#3#4%
{%
    \xint_gob_til_W
      #2\XINT_mulr_exit_ci
      #3\XINT_mulr_exit_cii
      \W\XINT_mulr_exit_ciii #1#2#3#4%
}%
\def\XINT_mulr_exit_ciii #1\W #2\Z\Z\Z\Z \W\W\W
{%
    \XINT_mul_Mr {#1}#2\Z\Z\Z\Z
}%
\def\XINT_mulr_exit_cii\W\XINT_mulr_exit_ciii #1\W\W #2\Z\Z\Z\Z \W\W
{%
    \XINT_mul_Mr {#1}#2\Z\Z\Z\Z
}%
\def\XINT_mulr_exit_ci\W\XINT_mulr_exit_cii
                      \W\XINT_mulr_exit_ciii #1\W\W\W #2\Z\Z\Z\Z \W
```

```
{%
    \XINT_mul_Mr {#1}#2\Z\Z\Z\Z
}%
\def\XINT_mulr_start #1#2\Z\Z\Z\Z
{%
    \expandafter\XINT_mulr_main\expandafter
    {\romannumeral0\XINT_mul_Mr {#1}#2\Z\Z\Z\Z}#2\Z\Z\Z\Z
}%
\def\XINT_mulr_main #1#2\Z\Z\Z\Z #3#4#5#6%
{%
    \xint_gob_til_W #6\XINT_mulr_finish_a\W
    \XINT_mulr_compute {#3#4#5#6}{#1}#2\Z\Z\Z\Z
}%
\def\XINT_mulr_compute #1#2#3\Z\Z\Z\Z
{%
    \expandafter\XINT_mulr_main\expandafter
    {\romannumeral0\expandafter
     \XINT_mul_Ar\expandafter0\expandafter{\expandafter}%
     \romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z
     \W\X\Y\Z 0000#2\W\X\Y\Z }#3\Z\Z\Z\Z
}%
\def\XINT_mulr_finish_a\W\XINT_mulr_compute #1%
{%
    \XINT_mulr_finish_b #1%
}%
\def\XINT_mulr_finish_b #1#2#3#4%
{%
    \xint_gob_til_W
      #1\XINT_mulr_finish_c
      #2\XINT_mulr_finish_ci
      #3\XINT_mulr_finish_cii
      \W\XINT_mulr_finish_ciii #1#2#3#4%
}%
\def\XINT_mulr_finish_ciii #1\W #2#3\Z\Z\Z\Z \W\W\W
{%
    \expandafter\XINT_addp_A\expandafter0\expandafter{\expandafter}%
    \romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z \W\X\Y\Z 000#2\W\X\Y\Z
}%
\def\XINT_mulr_finish_cii
    \W\XINT_mulr_finish_ciii #1\W\W #2#3\Z\Z\Z\Z \W\W
{%
    \expandafter\XINT_addp_A\expandafter0\expandafter{\expandafter}%
    \romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z \W\X\Y\Z 00#2\W\X\Y\Z
}%
\def\XINT_mulr_finish_ci #1\XINT_mulr_finish_ciii #2\W\W\W #3#4\Z\Z\Z\Z \W
{%
    \expandafter\XINT_addp_A\expandafter0\expandafter{\expandafter}%
    \romannumeral0\XINT_mul_Mr {#2}#4\Z\Z\Z\Z \W\X\Y\Z 0#3\W\X\Y\Z
}%
\def\XINT_mulr_finish_c #1\XINT_mulr_finish_ciii \W\W\W\W #2#3\Z\Z\Z\Z { #2}%
```

3.22 \xintiSqr, \xintiiSqr

```
\def\xintiiSqr {\romannumeral0\xintiisqr }%
\def\xintiisqr #1%
{%
    \expandafter\XINT_sqr\expandafter {\romannumeral0\xintiiabs{#1}}%
}%
\def\xintiSqr {\romannumeral0\xintisqr }%
\def\xintisqr #1%
{%
    \expandafter\XINT_sqr\expandafter {\romannumeral0\xintiabs{#1}}%
}%
\let\xintSqr\xintiSqr \let\xintsqr\xintisqr
\def\XINT_sqr #1%
{%
    \expandafter\XINT_mul_enter
          \romannumeral0%
          \XINT_RQ {}#1\R\R\R\R\R\R\R\Z
          \Z\Z\Z\Z #1\W\W\W\W
}%
```

3.23 \xintiPow, \xintiiPow

1.02 modified the \XINT_posprod routine, the was renamed \XINT_pow_posprod and moved here, as it was well adapted for computing powers. Then 1.03 moved the special variants of multiplication (hence of addition) which were needed to earlier in this style file.

Modified in 1.06, the exponent is given to a \numexpr rather than twice expanded. \xintnum added in 1.09a.

\XINT_pow_posprod: Routine de produit servant pour le calcul des puissances. Chaque nouveau terme est plus grand que ce qui a déjà été calculé. Par conséquent on a intérêt à le conserver en second dans la routine de multiplication, donc le précédent calcul a intérêt à avoir été donné sur 4n, à l'envers. Il faut donc modifier la multiplication pour qu'elle fasse cela. Ce qui oblige à utiliser une version spéciale de l'addition également.

1.09j has reorganized the main loop, the described above \XINT_pow_posprod routine has been removed, intermediate multiplications are done immediately. Also, the maximal accepted exponent is now 100000 (no such restriction in \xintFloatPow, which accepts any exponent less than 2^31, and in \xintFloatPower which accepts long integers as exponent).

2^100000=9.990020930143845e30102 and multiplication of two numbers with 30000 digits would take hours on my laptop (seconds for 1000 digits).

```
\def\xintiiPow {\romannumeral0\xintiipow }%
\def\xintiipow #1%
{%
    \expandafter\xint_pow\romannumeral-`0#1\Z%
}%
\def\xintiPow {\romannumeral0\xintipow }%
\def\xintipow #1%
{%
    \expandafter\xint_pow\romannumeral0\xintnum{#1}\Z%
}%
\let\xintPow\xintiPow \let\xintpow\xintipow
\def\xint_pow #1#2\Z
{%
    \xint_UDsignfork
```

```
      #1\XINT_pow_Aneg
       -\XINT_pow_Anonneg
    \krof
       #1{#2}%
}%
\def\XINT_pow_Aneg #1#2#3%
{%
   \expandafter\XINT_pow_Aneg_\expandafter{\the\numexpr #3}#2\Z
}%
\def\XINT_pow_Aneg_ #1%
{%
   \ifodd #1
       \expandafter\XINT_pow_Aneg_Bodd
   \fi
   \XINT_pow_Anonneg_ {#1}%
}%
\def\XINT_pow_Aneg_Bodd #1%
{%
    \expandafter\XINT_opp\romannumeral0\XINT_pow_Anonneg_
}%
```

B = #3, faire le xpxp. Modified with 1.06: use of \numexpr.

```
\def\XINT_pow_Anonneg #1#2#3%
{%
   \expandafter\XINT_pow_Anonneg_\expandafter {\the\numexpr #3}#1#2\Z
}%
```

#1 = B, #2 = |A|. Modifié pour v1.1, car utilisait \XINT_Cmp, ce qui d'ailleurs n'était sans doute pas super efficace, et m'obligeait à mettre \xintCmp dans xintcore. Donc ici A est déjà #2#3 et il y a un \Z après.

```
\def\XINT_pow_Anonneg_ #1#2#3\Z
{%
    \if\relax #3\relax\xint_dothis
                       {\ifcase #2 \expandafter\XINT_pow_AisZero
                                \or\expandafter\XINT_pow_AisOne
                              \else\expandafter\XINT_pow_AatleastTwo
                        \fi }\fi
    \xint_orthat \XINT_pow_AatleastTwo {#1}{#2#3}%
}%
\def\XINT_pow_AisOne #1#2{ 1}%
```

#1 = B

```
\def\XINT_pow_AisZero #1#2%
{%
     \ifcase\XINT_cntSgn #1\Z
         \xint_afterfi { 1}%
     \or
         \xint_afterfi { 0}%
     \else
         \xint_afterfi {\xintError:DivisionByZero\space 0}%
     \fi
```

```
}%
\def\XINT_pow_AatleastTwo #1%
{%
    \ifcase\XINT_cntSgn #1\Z
        \expandafter\XINT_pow_BisZero
    \or
        \expandafter\XINT_pow_checkBsize
    \else
        \expandafter\XINT_pow_BisNegative
    \fi
    {#1}%
}%
\edef\XINT_pow_BisNegative #1#2%
    {\noexpand\xintError:FractionRoundedToZero\space 0}%
\def\XINT_pow_BisZero #1#2{ 1}%
```

B = #1 > 0, A = #2 > 1. With 1.05, I replace \xintiLen{#1}>9 by direct use of \numexpr [to generate an error message if the exponent is too large] 1.06: \numexpr was already used above.

```
\def\XINT_pow_checkBsize #1%
{%
    \ifnum #1>100000
        \expandafter\XINT_pow_BtooBig
    \else
        \expandafter\XINT_pow_loopI
    \fi
    {#1}%
}%
\edef\XINT_pow_BtooBig #1#2{\noexpand\xintError:ExponentTooBig\space 0}%
\def\XINT_pow_loopI #1%
{%
    \ifnum #1=\xint_c_i\XINT_pow_Iend\fi
    \ifodd #1
       \expandafter\XINT_pow_loopI_odd
    \else
       \expandafter\XINT_pow_loopI_even
    \fi
    {#1}%
}%
\edef\XINT_pow_Iend\fi #1\fi #2#3{\noexpand\fi\space #3}%
\def\XINT_pow_loopI_even #1#2%
{%
    \expandafter\XINT_pow_loopI\expandafter
    {\the\numexpr #1/\xint_c_ii\expandafter}\expandafter
    {\romannumeral0\xintiisqr {#2}}%
}%
\def\XINT_pow_loopI_odd #1#2%
{%
    \expandafter\XINT_pow_loopI_odda\expandafter
    {\romannumeral0\XINT_RQ {}#2\R\R\R\R\R\R\R\Z }{#1}{#2}%
}%
\def\XINT_pow_loopI_odda #1#2#3%
{%
```

```
    \expandafter\XINT_pow_loopII\expandafter
    {\the\numexpr #2/\xint_c_ii-\xint_c_i\expandafter}\expandafter
    {\romannumeral0\xintiisqr {#3}}{#1}%
}%
\def\XINT_pow_loopII #1%
{%
    \ifnum #1 = \xint_c_i\XINT_pow_IIend\fi
    \ifodd #1
       \expandafter\XINT_pow_loopII_odd
    \else
       \expandafter\XINT_pow_loopII_even
    \fi
    {#1}%
}%
\def\XINT_pow_loopII_even #1#2%
{%
    \expandafter\XINT_pow_loopII\expandafter
    {\the\numexpr #1/\xint_c_ii\expandafter}\expandafter
    {\romannumeral0\xintiisqr {#2}}%
}%
\def\XINT_pow_loopII_odd #1#2#3%
{%
    \expandafter\XINT_pow_loopII_odda\expandafter
    {\romannumeral0\XINT_mulr_enter #3\Z\Z\Z\Z #2\W\W\W\W}{#1}{#2}%
}%
\def\XINT_pow_loopII_odda #1#2#3%
{%
    \expandafter\XINT_pow_loopII\expandafter
    {\the\numexpr #2/\xint_c_ii-\xint_c_i\expandafter}\expandafter
    {\romannumeral0\xintiisqr {#3}}{#1}%
}%
\def\XINT_pow_IIend\fi #1\fi #2#3#4%
{%
    \fi\XINT_mul_enter #4\Z\Z\Z\Z #3\W\W\W\W
}%
```

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

The 1.09a release inserted the use of \xintnum. The \xintiiDivision etc... are the ones which do only \romannumeral-`0.

January 5, 2014: Naturally, addition, subtraction, multiplication and division are the first things I did and since then I had left the division untouched. So in preparation of release 1.09j, I started revisiting the division, I did various minor improvements obtaining roughly 10% efficiency gain. Then I decided I should deliberately impact the input save stack, with the hope to gain more speed from removing tokens and leaving them upstream.

For this however I had to modify the underlying mathematical algorithm. The initial one is a bit unusual I guess, and, I trust, rather efficient, but it does not produce the quotient digits (in base 10000) one by one; at any given time it is possible that some correction will be made, which means it is not an appropriate algorithm for a TeX implementation which will abandon the quotient upstream. Thus I now have with 1.09j a new underlying mathematical algorithm, presumably much more standard. It is a bit complicated to implement expandably these things, but in the end I

had regained the already mentioned 10% efficiency and even more for small to medium sized inputs (up to 30% perhaps). And in passing I did a special routine for divisors < 10000, which is 5 to 10 times faster still.

But, I then tested a variant of my new implementation which again did not impact the input save stack and, for sizes of up to 200 digits, it is not much worse, indeed it is perhaps actually better than the one abandoning the quotient digits upstream (and in the end putting them in the correct order). So, finally, I re-incorporated the produced quotient digits within a tail recursion. Hence \xintiDivision, like all other routines in xint (except \xintSeq without optional parameter) does not impact the input save stack. One can have a produced quotient longer than 4x5000=20000 digits, and no need to worry about consequences propagating to \xintTrunc, \xintRound, \xintFloat, \xintFloatSqrt, etc... and all other places using the division. See also \xintXTrunc in this context.

```
\def\xintiiQuo {\romannumeral0\xintiiquo }%
\def\xintiiRem {\romannumeral0\xintiirem }%
\def\xintiiquo {\expandafter\xint_firstoftwo_thenstop\romannumeral0\xintiidivision }%
\def\xintiirem {\expandafter\xint_secondoftwo_thenstop\romannumeral0\xintiidivision }%
\def\xintiQuo {\romannumeral0\xintiquo }%
\def\xintiRem {\romannumeral0\xintirem }%
\def\xintiquo {\expandafter\xint_firstoftwo_thenstop\romannumeral0\xintidivision }%
\def\xintirem {\expandafter\xint_secondoftwo_thenstop\romannumeral0\xintidivision }%
\let\xintQuo\xintiQuo\let\xintquo\xintiquo % deprecated (1.1)
\let\xintRem\xintiRem\let\xintrem\xintirem % deprecated (1.1)
```

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

```
\def\xintiDivision {\romannumeral0\xintidivision }%
\def\xintidivision #1{\expandafter\XINT_division\romannumeral0\xintnum{#1}\Z }%
\let\xintDivision\xintiDivision \let\xintdivision\xintidivision % deprecated
\def\XINT_division #1#2\Z #3{\expandafter\XINT_iidivision_a\expandafter #1%
                             \romannumeral0\xintnum{#3}\Z #2\Z }%
\def\xintiiDivision   {\romannumeral0\xintiidivision }%
\def\xintiidivision  #1{\expandafter\XINT_iidivision \romannumeral-`0#1\Z }%
\def\XINT_iidivision #1#2\Z #3{\expandafter\XINT_iidivision_a\expandafter #1%
                             \romannumeral-`0#3\Z #2\Z }%
\def\XINT_iidivision_a #1#2% #1 de A, #2 de B.
{%
    \if0#2\xint_dothis\XINT_iidivision_divbyzero\fi
    \if0#1\xint_dothis\XINT_iidivision_aiszero\fi
    \if-#2\xint_dothis{\expandafter\XINT_iidivision_bneg
                       \romannumeral0\XINT_iidivision_bpos #1}\fi
    \xint_orthat{\XINT_iidivision_bpos #1#2}%
}%
\def\XINT_iidivision_divbyzero #1\Z #2\Z {\xintError:DivisionByZero\space {0}{0}}%
\def\XINT_iidivision_aiszero #1\Z #2\Z   { {0}{0}}%
\def\XINT_iidivision_bneg #1% q->-q, r unchanged
   {\expandafter\space\expandafter{\romannumeral0\XINT_opp #1}}%
\def\XINT_iidivision_bpos #1%
{%
    \xint_UDsignfork
            #1\XINT_iidivision_aneg
             -{\XINT_iidivision_apos #1}%
```

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

Pour la suite A et B sont > 0. #1 = B. Pour le moment à l'endroit. Calcul du plus petit K = 4n >= longueur de B

```
\def\XINT_div_prepare #1%
{%
    \expandafter \XINT_div_prepareB_aa \expandafter
        {\romannumeral0\xintlength {#1}}{#1}% B > 0 ici
}%
\def\XINT_div_prepareB_aa #1%
{%
    \ifnum #1=\xint_c_i
      \expandafter\XINT_div_prepareB_onedigit
    \else
      \expandafter\XINT_div_prepareB_a
    \fi
    {#1}%
}%
\def\XINT_div_prepareB_a #1%
{%
  \expandafter\XINT_div_prepareB_c\expandafter
  {\the\numexpr \xint_c_iv*((#1+\xint_c_i)/\xint_c_iv)}{#1}%
}%
```

B=1 and B=2 treated specially.

```
\def\XINT_div_prepareB_onedigit #1#2%
{%
    \ifcase#2
    \or\expandafter\XINT_div_BisOne
    \or\expandafter\XINT_div_BisTwo
    \else\expandafter\XINT_div_prepareB_e
```

```
    \fi {000}{0}{4}{#2}%
}%
\def\XINT_div_BisOne #1#2#3#4#5{ {#5}{0}}%
\def\XINT_div_BisTwo #1#2#3#4#5%
{%
    \expandafter\expandafter\expandafter\XINT_div_BisTwo_a
    \ifodd\xintiiLDg{#5} \expandafter1\else \expandafter0\fi {#5}%
}%
\edef\XINT_div_BisTwo_a #1#2%
{%
    \noexpand\expandafter\space\noexpand\expandafter
    {\noexpand\romannumeral0\noexpand\xinthalf {#2}}{#1}%
}%
```

#1 = K. 1.09j uses \csname, earlier versions did it with \ifcase.

```
\def\XINT_div_prepareB_c #1#2%
{%
    \csname XINT_div_prepareB_d\romannumeral\numexpr#1-#2\endcsname
    {#1}%
}%
\def\XINT_div_prepareB_d    {\XINT_div_prepareB_e {}{0000}}%
\def\XINT_div_prepareB_di   {\XINT_div_prepareB_e {0}{000}}%
\def\XINT_div_prepareB_dii  {\XINT_div_prepareB_e {00}{00}}%
\def\XINT_div_prepareB_diii {\XINT_div_prepareB_e {000}{0}}%
\def\XINT_div_cleanR        #10000.{{#1}}%
```

#1 = zéros à rajouter à B, #2=c [modifié dans 1.09j, ce sont maintenant des zéros explicites en nombre 4 - ancien c, et on utilisera \XINT_div_cleanR et non plus \XINT_dsh_checksignx pour nettoyer à la fin des zéros en excès dans le Reste; in all comments next, «c» stands now {0} or {00} or {000} or {0000} rather than a digit as in earlier versions], #3=K, #4 = B

```
\def\XINT_div_prepareB_e #1#2#3#4%
{%
    \ifnum#3=\xint_c_iv\expandafter\XINT_div_prepareLittleB_f
                    \else\expandafter\XINT_div_prepareB_f
    \fi
    #4#1{#3}{#2}{#1}%
}%
```

x = #1#2#3#4 = 4 premiers chiffres de B. #1 est non nul. B is reversed. With 1.09j or latter x+1 and (x+1)/2 are pre-computed. Si K=4 on ne renverse pas B, et donc B=x dans la suite. De plus pour K=4 on ne travaille pas avec x+1 et (x+1)/2 mais avec x et x/2.

```
\def\XINT_div_prepareB_f #1#2#3#4#5#{%
    \expandafter\XINT_div_prepareB_g
     \the\numexpr  #1#2#3#4+\xint_c_i\expandafter
    .\the\numexpr (#1#2#3#4+\xint_c_i)/\xint_c_ii\expandafter
    .\romannumeral0\xintreverseorder {#1#2#3#4#5}.{#1#2#3#4}%
}%
\def\XINT_div_prepareLittleB_f #1#{%
    \expandafter\XINT_div_prepareB_g \the\numexpr  #1/\xint_c_ii.{}.{}.{#1}%
}%
```

```
 #1 = x' = x+1= 1+quatre premiers chiffres de B, #2 = y = (x+1)/2 précalculé #3 = B préparé et
 maintenant renversé, #4=x, #5 = K, #6 = «c», #7= {} ou {0} ou {00} ou {000}, #8 = A initial On
 multiplie aussi A par 10^c. -> AK{x'yx}B«c». Par contre dans le cas little on a #1=y=(x/2), #2={},
 #3={}, #4=x, donc cela donne ->AK{y{}x}{}«c», il n'y a pas de B.
```

```
\def\XINT_div_prepareB_g #1.#2.#3.#4#5#6#7#8%
{%
    \XINT_div_prepareA_a {#8#7}{#5}{{#1}{#2}{#4}}{#3}{#6}%
}%
```

```
 A, K, {x'yx}, B«c»
```

```
\def\XINT_div_prepareA_a #1%
{%
    \expandafter\XINT_div_prepareA_b\expandafter
       {\romannumeral0\xintlength {#1}}{#1}%
}%
```

```
 L0, A, K, {x'yx}, B«c»
```

```
\def\XINT_div_prepareA_b #1%
{%
  \expandafter\XINT_div_prepareA_c\expandafter
    {\the\numexpr \xint_c_iv*((#1+\xint_c_i)/\xint_c_iv)}{#1}%
}%
```

```
 L, L0, A, K, {x'yx}, B, «c»
```

```
\def\XINT_div_prepareA_c #1#2%
{%
    \csname XINT_div_prepareA_d\romannumeral\numexpr #1-#2\endcsname
    {#1}%
}%
\def\XINT_div_prepareA_d     {\XINT_div_prepareA_e {}}%
\def\XINT_div_prepareA_di    {\XINT_div_prepareA_e {0}}%
\def\XINT_div_prepareA_dii   {\XINT_div_prepareA_e {00}}%
\def\XINT_div_prepareA_diii  {\XINT_div_prepareA_e {000}}%
```

```
 #1#3 = A préparé, #2 = longueur de ce A préparé, #4=K, #5={x'yx}-> LKAx'yxB«c»
```

```
\def\XINT_div_prepareA_e #1#2#3#4#5%
{%
    \XINT_div_start_a {#2}{#4}{#1#3}#5%
}%
```

```
 L, K, A, x',y,x, B, «c» (avec y{}x{} au lieu de x'yxB dans la variante little)
```

```
\def\XINT_div_start_a #1#2%
{%
    \ifnum #2=\xint_c_iv \expandafter\XINT_div_little_b
    \else
      \ifnum #1 < #2
        \expandafter\expandafter\expandafter\XINT_div_III_aa
      \else
        \expandafter\expandafter\expandafter\XINT_div_start_b
```

```
      \fi
    \fi
    {#1}{#2}%
}%
```

L, K, A, x',y,x, B, «c».

```
\def\XINT_div_III_aa #1#2#3#4#5#6#7%
{%
    \expandafter\expandafter\expandafter
    \XINT_div_III_b\xint_cleanupzeros_nostop #3.{0000}%
}%
```

R.Q«c».

```
\def\XINT_div_III_b #1%
{%
    \if0#1%
       \expandafter\XINT_div_III_bRzero
    \else
       \expandafter\XINT_div_III_bRpos
    \fi
    #1%
}%
\def\XINT_div_III_bRzero 0.#1#2%
{%
    \expandafter\space\expandafter
   {\romannumeral0\XINT_cuz_loop #1\W\W\W\W\W\W\W\Z}{0}%
}%
\def\XINT_div_III_bRpos #1.#2#3%
{%
    \expandafter\XINT_div_III_c \XINT_div_cleanR #1#3.{#2}%
}%
\def\XINT_div_III_c #1#2%
{%
    \expandafter\space\expandafter
   {\romannumeral0\XINT_cuz_loop #2\W\W\W\W\W\W\W\Z}{#1}%
}%
```

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%
{%
    \XINT_div_start_c {#2}.#3.{#6}{{#1}{#2}{{#4}{#5}}{#6}}%
}%
```

Kalpha.A.x{LK{x'y}x}, B, «c», au début #2=alpha est vide

```
\def\XINT_div_start_c #1#2.#3#4#5#6%
{%
    \ifnum #1=\xint_c_iv\XINT_div_start_ca\fi
    \expandafter\XINT_div_start_c\expandafter
         {\the\numexpr #1-\xint_c_iv}#2#3#4#5#6.%
}%
\def\XINT_div_start_ca\fi\expandafter\XINT_div_start_c\expandafter
    #1#2#3#4#5{\fi\XINT_div_start_d {#2#3#4#5}#2#3#4#5}%
```

```
  #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, {0000}, L, K, {x'y},x, alpha'=reste de A, B{}«c». Pour K=4 on a en fait B=x, faudra revoir
  après.

\def\XINT_div_start_d #1#2.#3.#4#5#6%
{%
    \XINT_div_I_a {#1}{#4}{#2}{#6}{0000}#5{#3}{#6}{}%
}%

  Ceci est le point de retour de la boucle principale. a, x, alpha, B, q0, L, K, {x'y}, x, alpha',
  BQ«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 quotient nul: #1=a, x, alpha, B, #5=q0, L, K, {x'y}, x, alpha', BQ«c» -> q{alpha} L,
  K, {x'y}, x, alpha', BQ«c»

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

  r.q.alpha, B, q0, L, K, {x'y}, x, alpha', BQ«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
}%
\def\XINT_div_I_dN #1.%
{%
    \expandafter\XINT_div_I_dP\the\numexpr #1-\xint_c_i.%
}%
\def\XINT_div_I_db #1.#2#3% #1=q=un chiffre, #2=alpha, #3=B
{%
    \expandafter\XINT_div_I_dc\expandafter
    {\romannumeral0\expandafter\XINT_div_sub_xpxp\expandafter
       {\romannumeral0\xintreverseorder{#2}}%
```

```
1908        {\romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z }}%
1909     #1{#2}{#3}%
1910 }%
1911 \def\XINT_div_I_dc #1#2%
1912 {%
1913     \if-#1% s'arranger pour que si n\'egatif on ait renvoy\'e alpha=-.
1914          \expandafter\xint_firstoftwo
1915     \else\expandafter\xint_secondoftwo\fi
1916     {\expandafter\XINT_div_I_dP\the\numexpr #2-\xint_c_i.}%
1917     {\XINT_div_I_e {#1}#2}%
1918 }%
```

alpha,q,ancien alpha,B, q0->1nouveauq.alpha, L, K, {x'y},x, alpha', BQ«c»

```
1919 \def\XINT_div_I_e #1#2#3#4#5%
1920 {%
1921     \expandafter\XINT_div_I_f \the\numexpr \xint_c_x^iv+#2+#5{#1}%
1922 }%
```

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

```
1923 \def\XINT_div_I_dP #1.#2#3#4%
1924 {%
1925     \expandafter \XINT_div_I_f \the\numexpr \xint_c_x^iv+#1+#4\expandafter
1926     {\romannumeral0\expandafter\XINT_div_sub_xpxp\expandafter
1927       {\romannumeral0\xintreverseorder{#2}}%
1928       {\romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z }}%
1929 }%
```

1#1#2#3#4=nouveau q, nouvel alpha, L, K, {x'y},x,alpha', BQ«c»

```
1930 \def\XINT_div_I_f 1#1#2#3#4{\XINT_div_I_g {#1#2#3#4}}%
```

#1=q,#2=nouvel alpha,#3=L, #4=K, #5={x'y}, #6=x, #7= alpha',#8=B, #9=Q«c» -> {x'y}alpha.alpha'.{{x'y}xKL}B{Qq}

```
1931 \def\XINT_div_I_g #1#2#3#4#5#6#7#8#9%
1932 {%
1933      \ifnum#3=#4
1934           \expandafter\XINT_div_III_ab
1935      \else
1936           \expandafter\XINT_div_I_h
1937      \fi
1938      {#5}#2.#7.{{#5}{#6}{#4}{#3}}{#8}{#9#1}%
1939 }%
```

{x'y}alpha.alpha'.{{x'y}xKL}B{Qq}«c» -> R sans leading zeros.{Qq}«c»

```
1940 \def\XINT_div_III_ab #1#2.#3.#4#5%
1941 {%
1942     \expandafter\XINT_div_III_b
1943     \romannumeral0\XINT_cuz_loop #2#3\W\W\W\W\W\W\W\Z.%
1944 }%
```

```
  #1={x'y}alpha.#2#3#4#5#6=reste de A. #7={{x'y},x,K,L},#8=B,nouveauQ«c» devient {x'y},alpha sur
  K+4 chiffres.B, {{x'y},x,K,L}, #6= nouvel alpha',B,nouveauQ«c»

1945 \def\XINT_div_I_h #1.#2#3#4#5#6.#7#8%
1946 {%
1947     \XINT_div_II_b #1#2#3#4#5.{#8}{#7}{#6}{#8}%
1948 }%

  {x'y}alpha.B, {{x'y},x,K,L}, nouveau alpha',B, Q«c» On intercepte la situation avec alpha débu-
  tant par 0000 qui est la seule qui pourrait donner un q1 nul. Donc q1 est non nul et la soustraction
  spéciale recevra un q1*B de longueur K ou K+4 et jamais 0000. Ensuite un q2 éventuel s'il est cal-
  culé est nécessairement non nul lui aussi. Comme dans la phase I on a aussi intercepté un q nul,
  la soustraction spéciale ne reçoit donc jamais un qB nul. Note: j'ai testé plusieurs fois que ma
  technique de gob_til_zeros est plus rapide que d'utiliser un \ifnum

1949 \def\XINT_div_II_b #1#2#3#4#5#6#7#8#9%
1950 {%
1951     \xint_gob_til_zeros_iv #2#3#4#5\XINT_div_II_skipc 0000%
1952     \XINT_div_II_c #1{#2#3#4#5}{#6#7#8#9}%
1953 }%

  x'y{0000}{4chiffres}reste de alpha.#6=B,#7={{x'y},x,K,L}, alpha',B, Q«c» -> {x'y}x,K,L (à dimin-
  uer de 4), {alpha sur K}B{q1=0000}{alpha'}B,Q«c»

1954 \def\XINT_div_II_skipc 0000\XINT_div_II_c #1#2#3#4#5.#6#7%
1955 {%
1956     \XINT_div_II_k #7{#4#5}{#6}{0000}%
1957 }%

  x'ya->1qx'yalpha.B, {{x'y},x,K,L}, nouveau alpha',B, Q«c»

1958 \def\XINT_div_II_c #1#2#3#4%
1959 {%
1960      \expandafter\XINT_div_II_d\the\numexpr (#3#4+#2)/#1+\xint_c_ixixixix\relax
1961      {#1}{#2}#3#4%
1962 }%

  1 suivi de q1 sur quatre chiffres, #5=x', #6=y, #7=alpha.#8=B, {{x'y},x,K,L}, alpha', B, Q«c» -->
  nouvel alpha.x',y,B,q1,{{x'y},x,K,L}, alpha', B, Q«c»

1963 \def\XINT_div_II_d 1#1#2#3#4#5#6#7.#8%
1964 {%
1965     \expandafter\XINT_div_II_e
1966     \romannumeral0\expandafter\XINT_div_sub_xpxp\expandafter
1967       {\romannumeral0\xintreverseorder{#7}}%
1968       {\romannumeral0\XINT_mul_Mr {#1#2#3#4}#8\Z\Z\Z\Z }.%
1969     {#5}{#6}{#8}{#1#2#3#4}%
1970 }%

  alpha.x',y,B,q1, {{x'y},x,K,L}, alpha', B, Q«c»

1971 \def\XINT_div_II_e #1#2#3#4%
1972 {%
1973     \xint_gob_til_zeros_iv #1#2#3#4\XINT_div_II_skipf 0000%
1974     \XINT_div_II_f #1#2#3#4%
1975 }%
```

```
  0000alpha sur K chiffres.#2=x',#3=y,#4=B,#5=q1, #6={{x'y},x,K,L}, #7=alpha',BQ«c» -> {x'y}x,K,L
  (à diminuer de 4), {alpha sur K}B{q1}{alpha'}BQ«c»

\def\XINT_div_II_skipf 0000\XINT_div_II_f 0000#1.#2#3#4#5#6%
{%
    \XINT_div_II_k #6{#1}{#4}{#5}%
}%

  a1 (huit chiffres), alpha (sur K+4), x', y, B, q1, {{x'y},x,K,L}, alpha', B,Q«c»

\def\XINT_div_II_f #1#2#3#4#5#6#7#8#9.%
{%
    \XINT_div_II_fa {#1#2#3#4#5#6#7#8}{#1#2#3#4#5#6#7#8#9}%
}%
\def\XINT_div_II_fa #1#2#3#4%
{%
    \expandafter\XINT_div_II_g\expandafter
                                  {\the\numexpr (#1+#4)/#3-\xint_c_i}{#2}%
}%

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

\def\XINT_div_II_g #1#2#3#4%
{%
    \expandafter \XINT_div_II_h
    \the\numexpr #4+#1+\xint_c_x^iv\expandafter\expandafter\expandafter
    {\expandafter\xint_gobble_iv
    \romannumeral0\expandafter\XINT_div_sub_xpxp\expandafter
       {\romannumeral0\xintreverseorder{#2}}%
       {\romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z }}{#3}%
}%

  1 puis nouveau q sur 4 chiffres, #5=nouvel alpha sur K chiffres, #6=B, #7={{x'y},x,K,L} avec L à
  ajuster, alpha', BQ«c» -> {x'y}x,K,L à diminuer de 4, {alpha}B{q}, alpha', BQ«c»

\def\XINT_div_II_h 1#1#2#3#4#5#6#7%
{%
    \XINT_div_II_k #7{#5}{#6}{#1#2#3#4}%
}%

  {x'y}x,K,L à diminuer de 4, alpha, B{q}alpha',BQ«c» ->nouveau L.K,x',y,x,alpha.B,q,alpha',B,Q«c»
  ->{LK{x'y}x},x,a,alpha.B,q,alpha',B,Q«c»

\def\XINT_div_II_k #1#2#3#4#5%
{%
    \expandafter\XINT_div_II_l \the\numexpr #4-\xint_c_iv.{#3}#1{#2}#5.%
}%
\def\XINT_div_II_l #1.#2#3#4#5#6#7#8#9%
{%
    \XINT_div_II_m {{#1}{#2}{{#3}{#4}}{#5}}{#5}{#6#7#8#9}#6#7#8#9%
}%
```

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

\def\XINT_div_II_m #1#2#3#4.#5#6%
{%
     \XINT_div_I_a {#3}{#2}{#4}{#5}{#6}#1%
}%

L, K, A, y,{},x, {},«c»->A.{yx}L{}«c» Comme ici K=4, dans la phase I on n'a pas besoin de alpha,
car a = alpha. De plus on a maintenu B dans l'ordre qui est donc la même chose que x. Par ailleurs
la phase I est simplifiée, il s'agit simplement de la division euclidienne de a par x, et de plus
on n'a à la faire qu'une unique fois et ensuite la phase II peut boucler sur elle-même au lieu
de revenir en phase I, par conséquent il n'y a pas non plus de q0 ici. Enfin, le y est (x/2) pas
((x+1)/2) il n'y a pas de x'=x+1

\def\XINT_div_little_b #1#2#3#4#5#6#7%
{%
    \XINT_div_little_c #3.{{#4}{#6}}{#1}%
}%

#1#2#3#4=a, #5=alpha'=reste de A.#6={yx}, #7=L, «c» -> a, y, x, L, alpha'=reste de A, «c».

\def\XINT_div_little_c #1#2#3#4#5.#6#7%
{%
    \XINT_div_littleI_a {#1#2#3#4}#6{#7}{#5}%
}%

a, y, x, L, alpha',«c» On calcule ici (contrairement à la phase I générale) le vrai quotient eu-
clidien de a par x=B, c'est donc un chiffre de 0 à 9. De plus on n'a à faire cela qu'une unique
fois.

\def\XINT_div_littleI_a #1#2#3%
{%
    \expandafter\XINT_div_littleI_b
    \the\numexpr (#1+#2)/#3-\xint_c_i{#1}{#2}{#3}%
}%

On intercepte quotient nul: [est-ce vraiment utile? ou n'est-ce pas plutôt une perte de temps en
moyenne? il faudrait tester] q=0#1=a, #2=y, x, L, alpha', «c» -> II_a avec L{alpha}alpha'.{yx}{0000}«c».
Et en cas de quotient non nul on procède avec littleI_c avec #1=q, #2=a, #3=y, #4=x -> {nouvel alpha
sur 4 chiffres}q{yx},L,alpha',«c».

\def\XINT_div_littleI_b #1%
{%
    \xint_gob_til_zero #1\XINT_div_littleI_skip 0\XINT_div_littleI_c #1%
}%
\def\XINT_div_littleI_skip 0\XINT_div_littleI_c 0#1#2#3#4#5%
   {\XINT_div_littleII_a {#4}{#1}#5.{{#2}{#3}}{0000}}%
\def\XINT_div_littleI_c #1#2#3#4%
{%
    \expandafter\expandafter\expandafter\XINT_div_littleI_e
    \expandafter\expandafter\expandafter
    {\expandafter\xint_gobble_i\the\numexpr \xint_c_x^iv+#2-#1*#4}#1{{#3}{#4}}%
}%
```

```
#1=nouvel alpha sur 4 chiffres#2=q,#3={yx}, #4=L, #5=alpha',«c» -> L{alpha}alpha'.{yx}{000q}«c»
point d'entrée de la boucle principale

\def\XINT_div_littleI_e #1#2#3#4#5%
   {\XINT_div_littleII_a {#4}{#1}#5.{#3}{000#2}}%

L{alpha}alpha'.{yx}Q«c» et c'est là qu'on boucle

\def\XINT_div_littleII_a #1%
{%
     \ifnum#1=\xint_c_iv
          \expandafter\XINT_div_littleIII_ab
     \else
          \expandafter\XINT_div_littleII_b
     \fi {#1}%
}%

L{alpha}alpha'.{yx}Q«c» -> (en fait #3 est vide normalement ici) R sans leading zeros.Q«c»

\def\XINT_div_littleIII_ab #1#2#3.#4%
{%
    \expandafter\XINT_div_III_b\the\numexpr #2#3.%
}%

L{alpha}alpha'.{yx}Q«c». On diminue L de quatre, comme cela c'est fait.

\def\XINT_div_littleII_b #1%
{%
   \expandafter\XINT_div_littleII_c\expandafter {\the\numexpr #1-\xint_c_iv}%
}%

{nouveauL}{alpha}alpha'.{yx}Q«c». On prélève 4 chiffres de alpha' -> {nouvel alpha sur huit
chiffres}yx{nouveau L}{nouvel alpha'}Q«c». Regarder si l'ancien alpha était 0000 n'avancerait
à rien car obligerait à refaire une chose comme la phase I, donc on ne perd pas de temps avec ça,
on reste en permanence en phase II.

\def\XINT_div_littleII_c #1#2#3#4#5#6#7.#8%
{%
    \XINT_div_littleII_d {#2#3#4#5#6}#8{#1}{#7}%
}%
\def\XINT_div_littleII_d #1#2#3%
{%
    \expandafter\XINT_div_littleII_e\the\numexpr (#1+#2)/#3+\xint_c_ixixixix.%
    {#1}{#2}{#3}%
}%

1 suivi de #1=q1 sur quatre chiffres.#2=alpha, #3=y, #4=x, L, alpha', Q«c» --> nouvel alpha sur
4.{q1}{yx},L,alpha', Q«c»

\def\XINT_div_littleII_e 1#1.#2#3#4%
{%
    \expandafter\expandafter\expandafter\XINT_div_littleII_f
    \expandafter\xint_gobble_i\the\numexpr \xint_c_x^iv+#2-#1*#4.%
    {#1}{{#3}{#4}}%
}%
```

```
alpha.q,{yx},L,alpha',Q«c»->L{alpha}alpha'.{yx}{Qq}«c»
```

```
\def\XINT_div_littleII_f #1.#2#3#4#5#6%
{%
    \XINT_div_littleII_a {#4}{#1}#5.{#3}{#6#2}%
}%
```

```
La soustraction spéciale. Dans 1.09j, elle fait A-qB, pour A (en fait alpha dans mes dénominations
des commentaires du code) et qB chacun de longueur K ou K+4, avec K au moins huit multiple de
quatre, qB a ses quatre chiffres significatifs (qui sont à droite) non nuls. Si A-qB<0 il suffit
de renvoyer -, le résultat n'importe pas. On est sûr que qB est non nul. On le met dans cette version
en premier pour tester plus facilement le cas avec qB de longueur K+4 et A de longueur seulement
K. Lorsque la longueur de qB est inférieure ou égale à celle de A, on va jusqu'à la fin de A et donc
c'est la retenue finale qui décide du cas négatif éventuel. Le résultat non négatif est toujours
donc renvoyé avec la même longueur que A, et il est dans l'ordre. J'ai fait une implémentation des
phases I et II en maintenant alpha toujours à l'envers afin d'éviter le reverse order systématique
fait sur A (ou plutôt alpha), mais alors il fallait que la soustraction ici s'arrange pour repérer
les huit chiffres les plus significatifs, au final ce n'était pas plus rapide, et même pénalisant
pour de gros inputs. Dans les versions 1.09i et antérieures (en fait je pense qu'ici rien quasiment
n'avait bougé depuis la première implémentation), la soustraction spéciale n'était pratiquée que
dans des cas avec certainement A-qB positif ou nul. De plus on n'excluait pas q=0, donc il fallait
aussi faire un éventuel reverseorder sur ce qui était encore non traité. Les cas avec q=0 sont
maintenant interceptés en amont et comme A et qB ont toujours quasiment la même longueur on ne
s'embarrasse pas de complications pour la fin.
```

```
\def\XINT_div_sub_xpxp #1#2% #1=alpha d\'ej\`a renvers\'e, #2 se d\'eveloppe en qB
{%
    \expandafter\XINT_div_sub_xpxp_b #2\W\X\Y\Z #1\W\X\Y\Z
}%
\def\XINT_div_sub_xpxp_b
{%
    \XINT_div_sub_A 1{}%
}%
\def\XINT_div_sub_A #1#2#3#4#5#6%
{%
    \xint_gob_til_W #3\xint_div_sub_az\W
    \XINT_div_sub_B #1{#3#4#5#6}{#2}%
}%
\def\XINT_div_sub_B #1#2#3#4\W\X\Y\Z #5#6#7#8%
{%
    \xint_gob_til_W #5\xint_div_sub_bz\W
    \XINT_div_sub_onestep #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
}%
\def\XINT_div_sub_onestep #1#2#3#4#5#6%
{%
    \expandafter\XINT_div_sub_backtoA
    \the\numexpr 11#6-#5#4#3#2+#1-\xint_c_i.%
}%
\def\XINT_div_sub_backtoA #1#2#3.#4%
{%
    \XINT_div_sub_A #2{#3#4}%
}%
```

si on arrive en sub_bz c'est que qB était de longueur K+4 et A seulement de longueur K, le résultat est donc < 0, renvoyer juste -

```
\def\xint_div_sub_bz\W\XINT_div_sub_onestep #1\Z { -}%
```

si on arrive en sub_az c'est que qB était de longueur inférieure ou égale à celle de A, donc on continue jusqu'à la fin de A, et on vérifiera la retenue à la fin.

```
\def\xint_div_sub_az\W\XINT_div_sub_B #1#2{\XINT_div_sub_C #1}%
\def\XINT_div_sub_C #1#2#3#4#5#6%
{%
    \xint_gob_til_W #3\xint_div_sub_cz\W
    \XINT_div_sub_C_onestep #1{#6#5#4#3}{#2}%
}%
\def\XINT_div_sub_C_onestep #1#2%
{%
    \expandafter\XINT_div_sub_backtoC \the\numexpr 11#2+#1-\xint_c_i.%
}%
\def\XINT_div_sub_backtoC #1#2#3.#4%
{%
    \XINT_div_sub_C #2{#3#4}%
}%
```

une fois arrivé en sub_cz on teste la retenue pour voir si le résultat final est en fait négatif, dans ce cas on renvoie seulement -

```
\def\xint_div_sub_cz\W\XINT_div_sub_C_onestep #1#2%
{%
    \if#10% retenue
         \expandafter\xint_div_sub_neg
    \else\expandafter\xint_div_sub_ok
    \fi
}%
\def\xint_div_sub_neg #1{ -}%
\def\xint_div_sub_ok  #1{ #1}%
```

3.25 \xintiDivRound, \xintiiDivRound

v1.1, transferred from first release of bnumexpr.

```
\def\xintiDivRound    {\romannumeral0\xintidivround }%
\def\xintidivround  #1{\expandafter\XINT_iidivround\romannumeral0\xintnum{#1}\Z }%
\def\xintiiDivRound   {\romannumeral0\xintiidivround }%
\def\xintiidivround #1{\expandafter\XINT_iidivround \romannumeral-`0#1\Z }%
\def\XINT_iidivround #1#2\Z #3{\expandafter\XINT_iidivround_a\expandafter #1%
                               \romannumeral-`0#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\XINT_iidivround_pos_a
                           \romannumeral0\XINT_div_prepare {#2}{#1#30}}%
\def\XINT_iidivround_pos_a #1#2{\xintReverseOrder {#1\XINT_iidivround_pos_b}\Z }%
\def\XINT_iidivround_pos_b #1#2{\xint_gob_til_Z #2\XINT_iidivround_pos_small\Z
                         \XINT_iidivround_pos_c #1#2}%
\def\XINT_iidivround_pos_c #1#2\Z {\ifnum #1>\xint_c_iv
                                \expandafter\XINT_iidivround_pos_up
                       \else   \expandafter\xintreverseorder
                       \fi {#2}}%
\def\XINT_iidivround_pos_up #1{\xintinc {\xintReverseOrder{#1}}}%
\def\XINT_iidivround_pos_small\Z\XINT_iidivround_pos_c #1#2%
    {\ifnum #1>\xint_c_iv\expandafter\xint_secondoftwo\else\expandafter
                                    \xint_firstoftwo\fi { 0}{ 1}}%
```

3.26 \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-`0#1\Z }%
\def\XINT_iidivtrunc #1#2\Z #3{\expandafter\XINT_iidivtrunc_a\expandafter #1%
                               \romannumeral-`0#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}}%
```

3.27 \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-`0#1\Z }%
\def\XINT_iimod #1#2\Z #3{\expandafter\XINT_iimod_a\expandafter #1%
                            \romannumeral-`0#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}}%
\XINT_restorecatcodes_endinput%
```

4 Package xint implementation

The basic arithmetic routines \xintiiAdd, \xintiiSub, \xintiiMul, \xintiiQuo and \xintiiPow have been moved to new package 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}%
  [2015/09/12 v1.1c 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}%
```

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.

```
\def\xintiiIsOne   {\romannumeral0\xintiiisone }%
\def\xintiiisone #1{\expandafter\XINT_isone\romannumeral-`0#1\W\Z }%
\def\xintIsOne   {\romannumeral0\xintisone }%
\def\xintisone #1{\expandafter\XINT_isone\romannumeral0\xintnum{#1}\W\Z }%
\def\XINT_isOne #1{\romannumeral0\XINT_isone #1\W\Z }%
\def\XINT_isone #1#2%
{%
    \xint_gob_til_one #1\XINT_isone_b 1%
    \expandafter\space\expandafter 0\xint_gob_til_Z #2%
}%
\def\XINT_isone_b #1\xint_gob_til_Z #2%
{%
```

```
    \xint_gob_til_W #2\XINT_isone_yes \W
    \expandafter\space\expandafter 0\xint_gob_til_Z
}%
\def\XINT_isone_yes #1\Z { 1}%
```

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.

```
\def\xintRev {\romannumeral0\xintrev }%
\def\xintrev #1%
{%
    \expandafter\XINT_rev_fork
    \romannumeral-`0#1\xint_relax % empty #1 ok, \xint_relax stops expansion
        \xint_bye\xint_bye\xint_bye\xint_bye
        \xint_bye\xint_bye\xint_bye\xint_bye
    \xint_relax
}%
\def\XINT_rev_fork #1%
{%
    \xint_UDsignfork
    #1{\expandafter\xint_minus_thenstop\romannumeral0\XINT_rord_main {}}%
     -{\XINT_rord_main {}#1}%
    \krof
}%
```

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_bye
}%
\def\XINT_Len #1% variant which does not expand via \xintnum.
{%
    \romannumeral0\XINT_len_fork
    #1\xint_relax\xint_relax\xint_relax\xint_relax
      \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
}%
\def\XINT_len_fork #1%
{%
    \expandafter\XINT_length_loop
    \xint_UDsignfork
      #1{0.}%
       -{0.#1}%
    \krof
```

```
}%
```

4.7 \xintBool, \xintToggle

1.09c

```
\def\xintBool #1{\romannumeral-`0%
                 \csname if#1\endcsname\expandafter1\else\expandafter0\fi }%
\def\xintToggle #1{\romannumeral-`0\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. Warning, \xintifTrueFalse, \xintifTrue deprecated, to be removed

```
\let\xintifTrueAelseB\xintifNotZero
\let\xintifFalseAelseB\xintifZero
\let\xintifTrue\xintifNotZero
\let\xintifTrueFalse\xintifNotZero
```

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

Release 1.09a has \xintnum inserted into \xintCmp. Unnecessary \xintiCmp suppressed in 1.09f. And 1.1a does \xintiiCmp, for optimization in \xintiiexpr. (not needed before, because \XINT_cmp_fork was directly used, or \XINT_Cmp)

```
\def\xintCmp {\romannumeral0\xintcmp }%
\def\xintcmp #1%
{%
    \expandafter\xint_cmp\expandafter{\romannumeral0\xintnum{#1}}%
}%
\def\xint_cmp #1#2%
{%
    \expandafter\XINT_cmp_fork \romannumeral0\xintnum{#2}\Z #1\Z
}%
\def\xintiiCmp {\romannumeral0\xintiicmp }%
\def\xintiicmp #1%
{%
    \expandafter\xint_iicmp\expandafter{\romannumeral-`0#1}%
}%
\def\xint_iicmp #1#2%
{%
    \expandafter\XINT_cmp_fork \romannumeral-`0#2\Z #1\Z
}%
\def\XINT_Cmp #1#2{\romannumeral0\XINT_cmp_fork #2\Z #1\Z }%
```

```
COMPARAISON
1 si #3#4>#1#2, 0 si #3#4=#1#2, -1 si #3#4<#1#2
#3#4 vient du *premier*, #1#2 vient du *second*
```

```
\def\XINT_cmp_fork #1#2\Z #3#4\Z
{%
    \xint_UDsignsfork
          #1#3\XINT_cmp_minusminus
           #1-\XINT_cmp_minusplus
           #3-\XINT_cmp_plusminus
            --{\xint_UDzerosfork
                       #1#3\XINT_cmp_zerozero
                        #10\XINT_cmp_zeroplus
                        #30\XINT_cmp_pluszero
                         00\XINT_cmp_plusplus
                       \krof }%
    \krof
    {#2}{#4}#1#3%
}%
```

```
\def\XINT_cmp_minusplus #1#2#3#4{ 1}%
\def\XINT_cmp_plusminus #1#2#3#4{ -1}%
\def\XINT_cmp_zerozero  #1#2#3#4{ 0}%
\def\XINT_cmp_zeroplus  #1#2#3#4{ 1}%
\def\XINT_cmp_pluszero  #1#2#3#4{ -1}%
\def\XINT_cmp_plusplus #1#2#3#4%
{%
    \XINT_cmp_pre {#4#2}{#3#1}%
}%
\def\XINT_cmp_minusminus #1#2#3#4%
{%
    \XINT_cmp_pre {#1}{#2}%
}%
\def\XINT_cmp_pre #1%
{%
  \expandafter\XINT_cmp_pre_b\expandafter
  {\romannumeral0\XINT_RQ {}#1\R\R\R\R\R\R\R\Z }%
}%
\def\XINT_cmp_pre_b #1#2%
{%
    \expandafter\XINT_cmp_A
    \expandafter1\expandafter{\expandafter}%
    \romannumeral0\XINT_RQ {}#2\R\R\R\R\R\R\R\Z
        \W\X\Y\Z #1\W\X\Y\Z
}%
```

```
COMPARAISON
N1 et N2 sont présentés à l'envers ET ON A RAJOUTÉ DES ZÉROS POUR QUE LEUR LONGUEURS À CHACUN SOIENT
MULTIPLES DE 4, MAIS AUCUN NE SE TERMINE EN 0000. routine appelée via
\XINT_cmp_A 1{}<N1>\W\X\Y\Z<N2>\W\X\Y\Z
ATTENTION RENVOIE 1 SI N1 < N2, 0 si N1 = N2, -1 si N1 > N2
```

```
\def\XINT_cmp_A #1#2#3\W\X\Y\Z #4#5#6#7%
{%
    \xint_gob_til_W #4\xint_cmp_az\W
    \XINT_cmp_B #1{#4#5#6#7}{#2}#3\W\X\Y\Z
}%
\def\XINT_cmp_B #1#2#3#4#5#6#7%
{%
    \xint_gob_til_W#4\xint_cmp_bz\W
    \XINT_cmp_onestep #1#2{#7#6#5#4}{#3}%
}%
\def\XINT_cmp_onestep #1#2#3#4#5#6%
{%
    \expandafter\XINT_cmp_backtoA\the\numexpr 11#5#4#3#2-#6+#1-\xint_c_i.%
}%
\def\XINT_cmp_backtoA #1#2#3.#4%
{%
    \XINT_cmp_A #2{#3#4}%
}%
\def\xint_cmp_bz\W\XINT_cmp_onestep #1\Z { 1}%
\def\xint_cmp_az\W\XINT_cmp_B #1#2#3#4#5#6#7%
{%
```

```
    \xint_gob_til_W #4\xint_cmp_ez\W
    \XINT_cmp_Eenter #1{#3}#4#5#6#7%
}%
\def\XINT_cmp_Eenter #1\Z { -1}%
\def\xint_cmp_ez\W\XINT_cmp_Eenter #1%
{%
    \xint_UDzerofork
      #1\XINT_cmp_K             %     il y a une retenue
       0\XINT_cmp_L             %     pas de retenue
    \krof
}%
\def\XINT_cmp_K #1\Z { -1}%
\def\XINT_cmp_L #1{\XINT_OneIfPositive_main #1}%
\def\XINT_OneIfPositive #1%
{%
    \XINT_OneIfPositive_main #1\W\X\Y\Z%
}%
\def\XINT_OneIfPositive_main #1#2#3#4%
{%
    \xint_gob_til_Z #4\xint_OneIfPositive_terminated\Z
    \XINT_OneIfPositive_onestep #1#2#3#4%
}%
\def\xint_OneIfPositive_terminated\Z\XINT_OneIfPositive_onestep\W\X\Y\Z { 0}%
\def\XINT_OneIfPositive_onestep #1#2#3#4%
{%
    \expandafter\XINT_OneIfPositive_check\the\numexpr #1#2#3#4\relax
}%
\def\XINT_OneIfPositive_check #1%
{%
    \xint_gob_til_zero #1\xint_OneIfPositive_backtomain 0%
    \XINT_OneIfPositive_finish #1%
}%
\def\XINT_OneIfPositive_finish #1\W\X\Y\Z{ 1}%
\def\xint_OneIfPositive_backtomain 0\XINT_OneIfPositive_finish 0%
                    {\XINT_OneIfPositive_main }%
```

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-`0#1\relax }%
\def\XINT_andof_a #1{\expandafter\XINT_andof_b\romannumeral-`0#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-`0#1\relax }%
\def\XINT_orof_a #1{\expandafter\XINT_orof_b\romannumeral-`0#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-`0#1\relax }%
\def\XINT_xorof_a #1#2{\expandafter\XINT_xorof_b\romannumeral-`0#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

Release 1.09a has \xintnum added into \xintGeq. PLUS GRAND OU ÉGAL attention compare les **valeurs absolues**

```
\def\xintGeq {\romannumeral0\xintgeq }%
\def\xintgeq #1%
{%
```

```
    \expandafter\xint_geq\expandafter {\romannumeral0\xintnum{#1}}%
}%
\def\xint_geq #1#2%
{%
    \expandafter\XINT_geq_fork \romannumeral0\xintnum{#2}\Z #1\Z
}%
\def\XINT_Geq #1#2{\romannumeral0\XINT_geq_fork #2\Z #1\Z }%
```

PLUS GRAND OU ÉGAL ATTENTION, TESTE les VALEURS ABSOLUES

```
\def\XINT_geq_fork #1#2\Z #3#4\Z
{%
    \xint_UDzerofork
      #1\XINT_geq_secondiszero % |#1#2|=0
      #3\XINT_geq_firstiszero % |#1#2|>0
       0{\xint_UDsignsfork
                  #1#3\XINT_geq_minusminus
                   #1-\XINT_geq_minusplus
                   #3-\XINT_geq_plusminus
                    --\XINT_geq_plusplus
                \krof }%
    \krof
    {#2}{#4}#1#3%
}%
\def\XINT_geq_secondiszero     #1#2#3#4{ 1}%
\def\XINT_geq_firstiszero      #1#2#3#4{ 0}%
\def\XINT_geq_plusplus   #1#2#3#4{\XINT_geq_pre {#4#2}{#3#1}}%
\def\XINT_geq_minusminus #1#2#3#4{\XINT_geq_pre {#2}{#1}}%
\def\XINT_geq_minusplus  #1#2#3#4{\XINT_geq_pre {#4#2}{#1}}%
\def\XINT_geq_plusminus  #1#2#3#4{\XINT_geq_pre {#2}{#3#1}}%
\def\XINT_geq_pre #1%
{%
  \expandafter\XINT_geq_pre_b\expandafter
  {\romannumeral0\XINT_RQ {}#1\R\R\R\R\R\R\R\Z }%
}%
\def\XINT_geq_pre_b #1#2%
{%
    \expandafter\XINT_geq_A
    \expandafter1\expandafter{\expandafter}%
    \romannumeral0\XINT_RQ {}#2\R\R\R\R\R\R\R\Z
        \W\X\Y\Z #1 \W\X\Y\Z
}%
```

PLUS GRAND OU ÉGAL
N1 et N2 sont présentés à l'envers ET ON A RAJOUTÉ DES ZÉROS POUR QUE LEURS LONGUEURS À CHACUN SOIENT MULTIPLES DE 4, MAIS AUCUN NE SE TERMINE EN 0000
routine appelée via
\romannumeral0\XINT_geq_A 1{}<N1>\W\X\Y\Z<N2>\W\X\Y\Z
ATTENTION RENVOIE 1 SI N1 < N2 ou N1 = N2 et 0 si N1 > N2

```
\def\XINT_geq_A #1#2#3\W\X\Y\Z #4#5#6#7%
{%
    \xint_gob_til_W #4\xint_geq_az\W
    \XINT_geq_B #1{#4#5#6#7}{#2}#3\W\X\Y\Z
```

```
}%
\def\XINT_geq_B #1#2#3#4#5#6#7%
{%
    \xint_gob_til_W #4\xint_geq_bz\W
    \XINT_geq_onestep #1#2{#7#6#5#4}{#3}%
}%
\def\XINT_geq_onestep #1#2#3#4#5#6%
{%
    \expandafter\XINT_geq_backtoA\the\numexpr 11#5#4#3#2-#6+#1-\xint_c_i.%
}%
\def\XINT_geq_backtoA #1#2#3.#4%
{%
    \XINT_geq_A #2{#3#4}%
}%
\def\xint_geq_bz\W\XINT_geq_onestep #1\W\X\Y\Z { 1}%
\def\xint_geq_az\W\XINT_geq_B #1#2#3#4#5#6#7%
{%
    \xint_gob_til_W #4\xint_geq_ez\W
    \XINT_geq_Eenter #1%
}%
\def\XINT_geq_Eenter #1\W\X\Y\Z { 0}%
\def\xint_geq_ez\W\XINT_geq_Eenter #1%
{%
    \xint_UDzerofork
      #1{ 0}             %     il y a une retenue
       0{ 1}             %     pas de retenue
    \krof
}%
```

4.29 \xintiMax, \xintiiMax

The rationale is that it is more efficient than using \xintCmp. 1.03 makes the code a tiny bit slower but easier to re-use for fractions. Note: actually since 1.08a code for fractions does not all reduce to these entry points, so perhaps I should revert the changes made in 1.03. Release 1.09a has \xintnum added into \xintiMax.

1.1 adds the missing \xintiiMax. Using \xintMax and not \xintiMax in xint is deprecated.

```
\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-`0#1}%
}%
\def\xint_iimax #1#2%
{%
```

```
    \expandafter\XINT_max_pre\expandafter {\romannumeral-`0#2}{#1}%
}%
\let\xintMax\xintiMax \let\xintmax\xintimax % deprecated, should be only with xintfrac
\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 \xintMaxof

New with 1.09a.

```
\def\xintiMaxof      {\romannumeral0\xintimaxof }%
\def\xintimaxof    #1{\expandafter\XINT_imaxof_a\romannumeral-`0#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-`0#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}%
\let\xintMaxof\xintiMaxof \let\xintmaxof\xintimaxof
```

4.31 \xintiMin, \xintiiMin

```
\xintnum added New with 1.09a. I add \xintiiMin in 1.1 and mark as deprecated \xintMin, renamed
\xintiMin.

\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-`0#1}%
}%
\def\xint_iimin #1#2%
{%
    \expandafter\XINT_min_pre\expandafter {\romannumeral-`0#2}{#1}%
}%
\let\xintMin\xintiMin \let\xintmin\xintimin % deprecated
\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

1.09a

```
\def\xintiMinof      {\romannumeral0\xintiminof }%
\def\xintiminof    #1{\expandafter\XINT_iminof_a\romannumeral-`0#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-`0#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}%
\let\xintMinof\xintiMinof \let\xintminof\xintiminof
```

4.33 \xintiiSum

\xintSum {{a}{b}...{z}}
\xintSumExpr {a}{b}...{z}\relax
1.03 (drastically) simplifies and makes the routines more efficient (for big computations). Also the way \xintSum and \xintSumExpr ...\relax are related. has been modified. Now \xintSumExpr \z

\relax is accepted input when \z expands to a list of braced terms (prior only \xintSum {\z} or \xintSum \z was possible).
  1.09a does NOT add the \xintnum overhead. 1.09h renames \xintiSum to \xintiiSum to correctly reflect this.

```
\def\xintiiSum {\romannumeral0\xintiisum }%
\def\xintiisum #1{\xintiisumexpr #1\relax }%
\def\xintiiSumExpr {\romannumeral0\xintiisumexpr }%
\def\xintiisumexpr {\expandafter\XINT_sumexpr\romannumeral-`0}%
\let\xintSum\xintiiSum \let\xintsum\xintiisum
\let\xintSumExpr\xintiiSumExpr \let\xintsumexpr\xintiisumexpr
\def\XINT_sumexpr {\XINT_sum_loop {0000}{0000}}%
\def\XINT_sum_loop #1#2#3%
{%
    \expandafter\XINT_sum_checksign\romannumeral-`0#3\Z {#1}{#2}%
}%
\def\XINT_sum_checksign #1%
{%
    \xint_gob_til_relax #1\XINT_sum_finished\relax
    \xint_gob_til_zero #1\XINT_sum_skipzeroinput0%
    \xint_UDsignfork
      #1\XINT_sum_N
       -{\XINT_sum_P #1}%
    \krof
}%
\def\XINT_sum_finished #1\Z #2#3%
{%
    \XINT_sub_A 1{}#3\W\X\Y\Z #2\W\X\Y\Z
}%
\def\XINT_sum_skipzeroinput #1\krof #2\Z {\XINT_sum_loop }%
\def\XINT_sum_P #1\Z #2%
{%
    \expandafter\XINT_sum_loop\expandafter
    {\romannumeral0\expandafter
     \XINT_addr_A\expandafter0\expandafter{\expandafter}%
    \romannumeral0\XINT_RQ {}#1\R\R\R\R\R\R\R\Z
    \W\X\Y\Z #2\W\X\Y\Z }%
}%
\def\XINT_sum_N #1\Z #2#3%
{%
    \expandafter\XINT_sum_NN\expandafter
    {\romannumeral0\expandafter
     \XINT_addr_A\expandafter0\expandafter{\expandafter}%
    \romannumeral0\XINT_RQ {}#1\R\R\R\R\R\R\R\Z
    \W\X\Y\Z #3\W\X\Y\Z }{#2}%
}%
\def\XINT_sum_NN #1#2{\XINT_sum_loop {#2}{#1}}%
```

4.34 \xintiiPrd

\xintPrd {{a}...{z}}
\xintPrdExpr {a}...{z}\relax
Release 1.02 modified the product routine. The earlier version was faster in situations where

each new term is bigger than the product of all previous terms, a situation which arises in the algorithm for computing powers. The 1.02 version was changed to be more efficient on big products, where the new term is small compared to what has been computed so far (the power algorithm now has its own product routine).

Finally, the 1.03 version just simplifies everything as the multiplication now decides what is best, with the price of a little overhead. So the code has been dramatically reduced here.

In 1.03 I also modify the way \xintPrd and \xintPrdExpr ...\relax are related. Now \xintPrdExpr \z \relax is accepted input when \z expands to a list of braced terms (prior only \xintPrd {\z} or \xintPrd \z was possible).

In 1.06a I suddenly decide that \xintProductExpr was a silly name, and as the package is new and certainly not used, I decide I may just switch to \xintPrdExpr which I should have used from the beginning.

1.09a does NOT add the \xintnum overhead. 1.09h renames \xintiPrd to \xintiiPrd to correctly reflect this.

```
\def\xintiiPrd {\romannumeral0\xintiiprd }%
\def\xintiiprd #1{\xintiiprdexpr #1\relax }%
\let\xintPrd\xintiiPrd
\let\xintprd\xintiiprd
\def\xintiiPrdExpr {\romannumeral0\xintiiprdexpr }%
\def\xintiiprdexpr {\expandafter\XINT_prdexpr\romannumeral-`0}%
\let\xintPrdExpr\xintiiPrdExpr
\let\xintprdexpr\xintiiprdexpr
\def\XINT_prdexpr {\XINT_prod_loop_a 1\Z }%
\def\XINT_prod_loop_a #1\Z #2%
    {\expandafter\XINT_prod_loop_b \romannumeral-`0#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 #1\Z #2\Z \Z { #2}%
```

4.35 \xintFac

Modified with 1.02 and again in 1.03 for greater efficiency. I am tempted, here and elsewhere, to use \ifcase\XINT_Geq {#1}{1000000000} rather than \ifnum\xintLength {#1}>9 but for the time being I leave things as they stand. With release 1.05, rather than using \xintLength I opt finally for direct use of \numexpr (which will throw a suitable number too big message), and to raise the \xintError:
FactorialOfTooBigNumber for argument larger than 1000000 (rather than 1000000000). With 1.09a, \xintFac uses \xintnum.

1.09j for no special reason, I lower the maximal number from 999999 to 100000. Any how this computation would need more memory than TL2013 standard allows to TeX. And I don't even mention time...

```
\def\xintiFac {\romannumeral0\xintifac }%
\def\xintifac #1%
{%
    \expandafter\XINT_fac_fork\expandafter{\the\numexpr #1}%
}%
\let\xintFac\xintiFac \let\xintfac\xintifac
\def\XINT_fac_fork #1%
{%
```

```
    \ifcase\XINT_cntSgn #1\Z
       \xint_afterfi{\expandafter\space\expandafter 1\xint_gobble_i }%
    \or
       \expandafter\XINT_fac_checklength
    \else
       \xint_afterfi{\expandafter\xintError:FactorialOfNegativeNumber
                \expandafter\space\expandafter 1\xint_gobble_i }%
    \fi
    {#1}%
}%
\def\XINT_fac_checklength #1%
{%
    \ifnum #1>100000
         \xint_afterfi{\expandafter\xintError:FactorialOfTooBigNumber
                       \expandafter\space\expandafter 1\xint_gobble_i }%
    \else
         \xint_afterfi{\ifnum #1>\xint_c_ixixixix
                          \expandafter\XINT_fac_big_loop
                       \else
                          \expandafter\XINT_fac_loop
                       \fi }%
    \fi
    {#1}%
}%
\def\XINT_fac_big_loop #1{\XINT_fac_big_loop_main {10000}{#1}{}}%
\def\XINT_fac_big_loop_main #1#2#3%
{%
    \ifnum #1<#2
        \expandafter
            \XINT_fac_big_loop_main
        \expandafter
           {\the\numexpr #1+1\expandafter }%
    \else
        \expandafter\XINT_fac_big_docomputation
    \fi
    {#2}{#3{#1}}%
}%
\def\XINT_fac_big_docomputation #1#2%
{%
    \expandafter \XINT_fac_bigcompute_loop \expandafter
    {\romannumeral0\XINT_fac_loop {9999}}#2\relax
}%
\def\XINT_fac_bigcompute_loop #1#2%
{%
    \xint_gob_til_relax #2\XINT_fac_bigcompute_end\relax
    \expandafter\XINT_fac_bigcompute_loop\expandafter
    {\expandafter\XINT_mul_enter
     \romannumeral0\XINT_RQ {}#2\R\R\R\R\R\R\R\Z
     \Z\Z\Z\Z #1\W\W\W\W }%
}%
\def\XINT_fac_bigcompute_end #1#2#3#4#5%
{%
```

```
    \XINT_fac_bigcompute_end_ #5%
}%
\def\XINT_fac_bigcompute_end_ #1\R #2\Z \W\X\Y\Z #3\W\X\Y\Z { #3}%
\def\XINT_fac_loop #1{\XINT_fac_loop_main 1{1000}{#1}}%
\def\XINT_fac_loop_main #1#2#3%
{%
    \ifnum #3>#1
    \else
        \expandafter\XINT_fac_loop_exit
    \fi
    \expandafter\XINT_fac_loop_main\expandafter
    {\the\numexpr #1+1\expandafter }\expandafter
    {\romannumeral0\XINT_mul_Mr {#1}#2\Z\Z\Z\Z }%
    {#3}%
}%
\def\XINT_fac_loop_exit #1#2#3#4#5#6#7%
{%
    \XINT_fac_loop_exit_ #6%
}%
\def\XINT_fac_loop_exit_ #1#2#3%
{%
    \XINT_mul_M
}%
```

```
------------------------------------------------------------------
------------------------------------------------------------------
```

DECIMAL OPERATIONS: FIRST DIGIT, LASTDIGIT, (<- moved to xintcore because xintiiLDg need by division macros) ODDNESS, MULTIPLICATION BY TEN, QUOTIENT BY TEN, QUOTIENT OR MULTIPLICATION BY POWER OF TEN, SPLIT OPERATION.

4.36 \xintMON, \xintMMON, \xintiiMON, \xintiiMMON

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

```
\def\xintiiMON {\romannumeral0\xintiimon }%
\def\xintiimon #1%
{%
    \ifodd\xintiiLDg {#1}
        \xint_afterfi{ -1}%
    \else
        \xint_afterfi{ 1}%
    \fi
}%
\def\xintiiMMON {\romannumeral0\xintiimmon }%
\def\xintiimmon #1%
{%
    \ifodd\xintiiLDg {#1}
        \xint_afterfi{ 1}%
    \else
        \xint_afterfi{ -1}%
    \fi
}%
\def\xintMON {\romannumeral0\xintmon }%
```

```
\def\xintmon #1%
{%
    \ifodd\xintLDg {#1}
        \xint_afterfi{ -1}%
    \else
        \xint_afterfi{ 1}%
    \fi
}%
\def\xintMMON {\romannumeral0\xintmmon }%
\def\xintmmon #1%
{%
    \ifodd\xintLDg {#1}
        \xint_afterfi{ 1}%
    \else
        \xint_afterfi{ -1}%
    \fi
}%
```

4.37 \xintOdd, \xintiiOdd, \xintEven, \xintiiEven

1.05 has \xintiOdd, whereas \xintOdd parses through \xintNum. Inadvertently, 1.09a redefined \xintiLDg hence \xintiOdd also parsed through \xintNum. Anyway, having a \xintOdd and a \xintiOdd was silly. Removed in 1.09f, now only \xintOdd and \xintiiOdd. 1.1: \xintEven and \xintiiEven added for \xintiiexpr.

```
\def\xintiiOdd {\romannumeral0\xintiiodd }%
\def\xintiiodd #1%
{%
    \ifodd\xintiiLDg{#1}
        \xint_afterfi{ 1}%
    \else
        \xint_afterfi{ 0}%
    \fi
}%
\def\xintiiEven {\romannumeral0\xintiieven }%
\def\xintiieven #1%
{%
    \ifodd\xintiiLDg{#1}
        \xint_afterfi{ 0}%
    \else
        \xint_afterfi{ 1}%
    \fi
}%
\def\xintOdd {\romannumeral0\xintodd }%
\def\xintodd #1%
{%
    \ifodd\xintLDg{#1}
        \xint_afterfi{ 1}%
    \else
        \xint_afterfi{ 0}%
    \fi
}%
\def\xintEven {\romannumeral0\xinteven }%
```

```
\def\xinteven #1%
{%
    \ifodd\xintLDg{#1}
        \xint_afterfi{ 0}%
    \else
        \xint_afterfi{ 1}%
    \fi
}%
```

4.38 \xintDSL

DECIMAL SHIFT LEFT (=MULTIPLICATION PAR 10)

```
\def\xintDSL {\romannumeral0\xintdsl }%
\def\xintdsl #1%
{%
    \expandafter\XINT_dsl \romannumeral-`0#1\Z
}%
\def\XINT_DSL #1{\romannumeral0\XINT_dsl #1\Z }%
\def\XINT_dsl #1%
{%
    \xint_gob_til_zero #1\xint_dsl_zero 0\XINT_dsl_ #1%
}%
\def\xint_dsl_zero 0\XINT_dsl_ 0#1\Z { 0}%
\def\XINT_dsl_ #1\Z { #10}%
```

4.39 \xintDSR

DECIMAL SHIFT RIGHT (=DIVISION PAR 10). Release 1.06b which replaced all @'s by underscores left undefined the \xint_minus used in \XINT_dsr_b, and this bug was fixed only later in release 1.09b

```
\def\xintDSR {\romannumeral0\xintdsr }%
\def\xintdsr #1%
{%
    \expandafter\XINT_dsr_a\expandafter {\romannumeral-`0#1}\W\Z
}%
\def\XINT_DSR #1{\romannumeral0\XINT_dsr_a {#1}\W\Z }%
\def\XINT_dsr_a
{%
    \expandafter\XINT_dsr_b\romannumeral0\xintreverseorder
}%
\def\XINT_dsr_b #1#2#3\Z
{%
    \xint_gob_til_W #2\xint_dsr_onedigit\W
    \xint_gob_til_minus #2\xint_dsr_onedigit-%
    \expandafter\XINT_dsr_removew
    \romannumeral0\xintreverseorder {#2#3}%
}%
\def\xint_dsr_onedigit #1\xintreverseorder #2{ 0}%
\def\XINT_dsr_removew #1\W { }%
```

4.40 \xintDSH, \xintDSHr

```
DECIMAL SHIFTS \xintDSH {x}{A}
si x <= 0, fait A -> A.10^(|x|). v1.03 corrige l'oversight pour A=0.
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).
  Release 1.06 now feeds x to a \numexpr first. I will have to revise this code at some point.
```

```
\def\xintDSHr {\romannumeral0\xintdshr }%
\def\xintdshr #1%
{%
    \expandafter\XINT_dshr_checkxpositive \the\numexpr #1\relax\Z
}%
\def\XINT_dshr_checkxpositive #1%
{%
    \xint_UDzerominusfork
      0#1\XINT_dshr_xzeroorneg
      #1-\XINT_dshr_xzeroorneg
       0-\XINT_dshr_xpositive
    \krof #1%
}%
\def\XINT_dshr_xzeroorneg #1\Z #2{ 0}%
\def\XINT_dshr_xpositive #1\Z
{%
    \expandafter\xint_secondoftwo_thenstop\romannumeral0\xintdsx {#1}%
}%
\def\xintDSH {\romannumeral0\xintdsh }%
\def\xintdsh #1#2%
{%
    \expandafter\xint_dsh\expandafter {\romannumeral-`0#2}{#1}%
}%
\def\xint_dsh #1#2%
{%
    \expandafter\XINT_dsh_checksignx \the\numexpr #2\relax\Z {#1}%
}%
\def\XINT_dsh_checksignx #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_dsh_xiszero
      0#1\XINT_dsx_xisNeg_checkA     % on passe direct dans DSx
       0-{\XINT_dsh_xisPos #1}%
    \krof
}%
\def\XINT_dsh_xiszero #1\Z #2{ #2}%
\def\XINT_dsh_xisPos #1\Z #2%
{%
    \expandafter\xint_firstoftwo_thenstop
    \romannumeral0\XINT_dsx_checksignA #2\Z {#1}% via DSx
}%
```

4.41 \xintDSx

Je fais cette routine pour la version 1.01, après modification de \xintDecSplit. Dorénavant \xintDSx fera appel à \xintDecSplit et de même \xintDSH fera appel à \xintDSx. J'ai donc supprimé entièrement l'ancien code de \xintDSH et re-écrit entièrement celui de \xintDecSplit pour x positif.
  --> 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.
  Release 1.06 has a faster and more compactly coded \XINT_dsx_zeroloop. Also, x is now given to a \numexpr. The earlier code should be then simplified, but I leave as is for the time being.
  Release 1.07 modified the coding of \XINT_dsx_zeroloop, to avoid impacting the input stack. Indeed the truncating, rounding, and conversion to float routines all use internally \XINT_dsx_zeroloop (via \XINT_dsx_addzerosnofuss), and they were thus roughly limited to generating N = 8 times the input save stack size digits. On TL2012 and TL2013, this means 40000 = 8x5000 digits. Although generating more than 40000 digits is more like a one shot thing, I wanted to open the possibility of outputting tens of thousands of digits to faile, thus I re-organized \XINT_dsx_zeroloop.
  January 5, 2014: but it is only with the new division implementation of 1.09j and also with its special \xintXTrunc routine that the possibility mentioned in the last paragraph has become a concrete one in terms of computation time.

```
\def\xintDSx {\romannumeral0\xintdsx }%
\def\xintdsx #1#2%
{%
    \expandafter\xint_dsx\expandafter {\romannumeral-`0#2}{#1}%
}%
\def\xint_dsx #1#2%
{%
    \expandafter\XINT_dsx_checksignx \the\numexpr #2\relax\Z {#1}%
}%
\def\XINT_DSx #1#2{\romannumeral0\XINT_dsx_checksignx #1\Z {#2}}%
\def\XINT_dsx #1#2{\XINT_dsx_checksignx #1\Z {#2}}%
\def\XINT_dsx_checksignx #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_dsx_xisZero
      0#1\XINT_dsx_xisNeg_checkA
       0-{\XINT_dsx_xisPos #1}%
    \krof
}%
\def\XINT_dsx_xisZero #1\Z #2{ {#2}{0}}% attention comme x > 0
\def\XINT_dsx_xisNeg_checkA #1\Z #2%
{%
    \XINT_dsx_xisNeg_checkA_ #2\Z {#1}%
}%
\def\XINT_dsx_xisNeg_checkA_ #1#2\Z #3%
{%
    \xint_gob_til_zero #1\XINT_dsx_xisNeg_Azero 0%
```

```
    \XINT_dsx_xisNeg_checkx {#3}{#3}{}\Z {#1#2}%
}%
\def\XINT_dsx_xisNeg_Azero #1\Z #2{ 0}%
\def\XINT_dsx_xisNeg_checkx #1%
{%
    \ifnum #1>1000000
       \xint_afterfi
       {\xintError:TooBigDecimalShift
        \expandafter\space\expandafter 0\xint_gobble_iv }%
    \else
       \expandafter \XINT_dsx_zeroloop
    \fi
}%
\def\XINT_dsx_addzerosnofuss #1{\XINT_dsx_zeroloop {#1}{}\Z }%
\def\XINT_dsx_zeroloop #1#2%
{%
    \ifnum #1<\xint_c_ix \XINT_dsx_exita\fi
    \expandafter\XINT_dsx_zeroloop\expandafter
        {\the\numexpr #1-\xint_c_viii}{#2000000000}%
}%
\def\XINT_dsx_exita\fi\expandafter\XINT_dsx_zeroloop
{%
    \fi\expandafter\XINT_dsx_exitb
}%
\def\XINT_dsx_exitb #1#2%
{%
    \expandafter\expandafter\expandafter
    \XINT_dsx_addzeros\csname xint_gobble_\romannumeral -#1\endcsname #2%
}%
\def\XINT_dsx_addzeros #1\Z #2{ #2#1}%
\def\XINT_dsx_xisPos #1\Z #2%
{%
    \XINT_dsx_checksignA #2\Z {#1}%
}%
\def\XINT_dsx_checksignA #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_dsx_AisZero
      0#1\XINT_dsx_AisNeg
       0-{\XINT_dsx_AisPos #1}%
    \krof
}%
\def\XINT_dsx_AisZero #1\Z #2{ {0}{0}}%
\def\XINT_dsx_AisNeg #1\Z #2%
{%
    \expandafter\XINT_dsx_AisNeg_dosplit_andcheckfirst
    \romannumeral0\XINT_split_checksizex {#2}{#1}%
}%
\def\XINT_dsx_AisNeg_dosplit_andcheckfirst #1%
{%
    \XINT_dsx_AisNeg_checkiffirstempty #1\Z
}%
```

```
\def\XINT_dsx_AisNeg_checkiffirstempty #1%
{%
    \xint_gob_til_Z #1\XINT_dsx_AisNeg_finish_zero\Z
    \XINT_dsx_AisNeg_finish_notzero #1%
}%
\def\XINT_dsx_AisNeg_finish_zero\Z
    \XINT_dsx_AisNeg_finish_notzero\Z #1%
{%
    \expandafter\XINT_dsx_end
    \expandafter {\romannumeral0\XINT_num {-#1}}{0}%
}%
\def\XINT_dsx_AisNeg_finish_notzero #1\Z #2%
{%
    \expandafter\XINT_dsx_end
    \expandafter {\romannumeral0\XINT_num {#2}}{-#1}%
}%
\def\XINT_dsx_AisPos #1\Z #2%
{%
    \expandafter\XINT_dsx_AisPos_finish
    \romannumeral0\XINT_split_checksizex {#2}{#1}%
}%
\def\XINT_dsx_AisPos_finish #1#2%
{%
    \expandafter\XINT_dsx_end
    \expandafter {\romannumeral0\XINT_num {#2}}%
                 {\romannumeral0\XINT_num {#1}}%
}%
\edef\XINT_dsx_end #1#2%
{%
    \noexpand\expandafter\space\noexpand\expandafter{#2}{#1}%
}%
```

4.42 \xintDecSplit, \xintDecSplitL, \xintDecSplitR

DECIMAL SPLIT

The macro \xintDecSplit {x}{A} first replaces A with |A| (*) This macro cuts the number into two pieces L and R. The concatenation LR always reproduces |A|, and R may be empty or have leading zeros. 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.

(*) warning: this may change in a future version. Only the behavior for A non-negative is guaranteed to remain the same.

v1.05a: \XINT_split_checksizex does not compute the length anymore, rather the error will be from a \numexpr; but the limit of 999999999 does not make much sense.

v1.06: Improvements in \XINT_split_fromleft_loop, \XINT_split_fromright_loop and related macros. More readable coding, speed gains. Also, I now feed immediately a \numexpr with x. Some simplifications should probably be made to the code, which is kept as is for the time being.

1.09e pays attention to the use of xintiabs which acquired in 1.09a the xintnum overhead. So xintiiabs rather without that overhead.

```
\def\xintDecSplitL {\romannumeral0\xintdecsplitl }%
\def\xintDecSplitR {\romannumeral0\xintdecsplitr }%
```

```
\def\xintdecsplitl
{%
    \expandafter\xint_firstoftwo_thenstop
    \romannumeral0\xintdecsplit
}%
\def\xintdecsplitr
{%
    \expandafter\xint_secondoftwo_thenstop
    \romannumeral0\xintdecsplit
}%
\def\xintDecSplit {\romannumeral0\xintdecsplit }%
\def\xintdecsplit #1#2%
{%
    \expandafter \xint_split \expandafter
    {\romannumeral0\xintiiabs {#2}}{#1}%  fait expansion de A
}%
\def\xint_split #1#2%
{%
    \expandafter\XINT_split_checksizex\expandafter{\the\numexpr #2}{#1}%
}%
\def\XINT_split_checksizex #1% 999999999 is anyhow very big, could be reduced
{%
    \ifnum\numexpr\XINT_Abs{#1}>999999999
       \xint_afterfi {\xintError:TooBigDecimalSplit\XINT_split_bigx }%
    \else
       \expandafter\XINT_split_xfork
    \fi
    #1\Z
}%
\def\XINT_split_bigx  #1\Z #2%
{%
    \ifcase\XINT_cntSgn #1\Z
    \or \xint_afterfi { {}{#2}}% positive big x
    \else
        \xint_afterfi { {#2}{}}% negative big x
    \fi
}%
\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\Z #2{ {#2}{}}%
\def\XINT_split_fromleft  #1\Z #2%
{%
    \XINT_split_fromleft_loop {#1}{}#2\W\W\W\W\W\W\W\W\Z
}%
\def\XINT_split_fromleft_loop #1%
{%
```

```
    \ifnum #1<\xint_c_viii\XINT_split_fromleft_exita\fi
    \expandafter\XINT_split_fromleft_loop_perhaps\expandafter
    {\the\numexpr #1-\xint_c_viii\expandafter}\XINT_split_fromleft_eight
}%
\def\XINT_split_fromleft_eight #1#2#3#4#5#6#7#8#9{#9{#1#2#3#4#5#6#7#8#9}}%
\def\XINT_split_fromleft_loop_perhaps #1#2%
{%
    \xint_gob_til_W #2\XINT_split_fromleft_toofar\W
    \XINT_split_fromleft_loop {#1}%
}%
\def\XINT_split_fromleft_toofar\W\XINT_split_fromleft_loop #1#2#3\Z
{%
    \XINT_split_fromleft_toofar_b #2\Z
}%
\def\XINT_split_fromleft_toofar_b #1\W #2\Z { {#1}{}}%
\def\XINT_split_fromleft_exita\fi
    \expandafter\XINT_split_fromleft_loop_perhaps\expandafter #1#2%
   {\fi \XINT_split_fromleft_exitb #1}%
\def\XINT_split_fromleft_exitb\the\numexpr #1-\xint_c_viii\expandafter
{%
    \csname XINT_split_fromleft_endsplit_\romannumeral #1\endcsname
}%
\def\XINT_split_fromleft_endsplit_ #1#2\W #3\Z { {#1}{#2}}%
\def\XINT_split_fromleft_endsplit_i #1#2%
                {\XINT_split_fromleft_checkiftoofar #2{#1#2}}%
\def\XINT_split_fromleft_endsplit_ii #1#2#3%
                {\XINT_split_fromleft_checkiftoofar #3{#1#2#3}}%
\def\XINT_split_fromleft_endsplit_iii #1#2#3#4%
                {\XINT_split_fromleft_checkiftoofar #4{#1#2#3#4}}%
\def\XINT_split_fromleft_endsplit_iv #1#2#3#4#5%
                {\XINT_split_fromleft_checkiftoofar #5{#1#2#3#4#5}}%
\def\XINT_split_fromleft_endsplit_v #1#2#3#4#5#6%
                {\XINT_split_fromleft_checkiftoofar #6{#1#2#3#4#5#6}}%
\def\XINT_split_fromleft_endsplit_vi #1#2#3#4#5#6#7%
                {\XINT_split_fromleft_checkiftoofar #7{#1#2#3#4#5#6#7}}%
\def\XINT_split_fromleft_endsplit_vii #1#2#3#4#5#6#7#8%
                {\XINT_split_fromleft_checkiftoofar #8{#1#2#3#4#5#6#7#8}}%
\def\XINT_split_fromleft_checkiftoofar #1#2#3\W #4\Z
{%
    \xint_gob_til_W #1\XINT_split_fromleft_wenttoofar\W
    \space {#2}{#3}%
}%
\def\XINT_split_fromleft_wenttoofar\W\space #1%
{%
    \XINT_split_fromleft_wenttoofar_b #1\Z
}%
\def\XINT_split_fromleft_wenttoofar_b #1\W #2\Z { {#1}}%
\def\XINT_split_fromright #1\Z #2%
{%
    \expandafter \XINT_split_fromright_a \expandafter
    {\romannumeral0\xintreverseorder {#2}}{#1}{#2}%
}%
```

```
\def\XINT_split_fromright_a #1#2%
{%
    \XINT_split_fromright_loop {#2}{}#1\W\W\W\W\W\W\W\W\Z
}%
\def\XINT_split_fromright_loop #1%
{%
    \ifnum #1<\xint_c_viii\XINT_split_fromright_exita\fi
    \expandafter\XINT_split_fromright_loop_perhaps\expandafter
    {\the\numexpr #1-\xint_c_viii\expandafter }\XINT_split_fromright_eight
}%
\def\XINT_split_fromright_eight #1#2#3#4#5#6#7#8#9{#9{#9#8#7#6#5#4#3#2#1}}%
\def\XINT_split_fromright_loop_perhaps #1#2%
{%
    \xint_gob_til_W #2\XINT_split_fromright_toofar\W
    \XINT_split_fromright_loop {#1}%
}%
\def\XINT_split_fromright_toofar\W\XINT_split_fromright_loop #1#2#3\Z { {}}%
\def\XINT_split_fromright_exita\fi
    \expandafter\XINT_split_fromright_loop_perhaps\expandafter #1#2%
    {\fi \XINT_split_fromright_exitb #1}%
\def\XINT_split_fromright_exitb\the\numexpr #1-\xint_c_viii\expandafter
{%
    \csname XINT_split_fromright_endsplit_\romannumeral #1\endcsname
}%
\edef\XINT_split_fromright_endsplit_ #1#2\W #3\Z #4%
{%
    \noexpand\expandafter\space\noexpand\expandafter
    {\noexpand\romannumeral0\noexpand\xintreverseorder {#2}}{#1}%
}%
\def\XINT_split_fromright_endsplit_i   #1#2%
            {\XINT_split_fromright_checkiftoofar #2{#2#1}}%
\def\XINT_split_fromright_endsplit_ii  #1#2#3%
            {\XINT_split_fromright_checkiftoofar #3{#3#2#1}}%
\def\XINT_split_fromright_endsplit_iii #1#2#3#4%
            {\XINT_split_fromright_checkiftoofar #4{#4#3#2#1}}%
\def\XINT_split_fromright_endsplit_iv  #1#2#3#4#5%
            {\XINT_split_fromright_checkiftoofar #5{#5#4#3#2#1}}%
\def\XINT_split_fromright_endsplit_v   #1#2#3#4#5#6%
            {\XINT_split_fromright_checkiftoofar #6{#6#5#4#3#2#1}}%
\def\XINT_split_fromright_endsplit_vi  #1#2#3#4#5#6#7%
            {\XINT_split_fromright_checkiftoofar #7{#7#6#5#4#3#2#1}}%
\def\XINT_split_fromright_endsplit_vii #1#2#3#4#5#6#7#8%
            {\XINT_split_fromright_checkiftoofar #8{#8#7#6#5#4#3#2#1}}%
\def\XINT_split_fromright_checkiftoofar #1%
{%
    \xint_gob_til_W #1\XINT_split_fromright_wenttoofar\W
    \XINT_split_fromright_endsplit_
}%
\def\XINT_split_fromright_wenttoofar\W\XINT_split_fromright_endsplit_ #1\Z #2%
    { {}{#2}}%
```

4.43 \xintiiSqrt, \xintiiSqrtR, \xintiiSquareRoot

v1.08. 1.09a uses \xintnum.
Some overhead was added inadvertently in 1.09a to inner routines when \xintiquo and \xintidivision were also promoted to use \xintnum; release 1.09f thus uses \xintiiquo and \xintiidivision which avoid this \xintnum overhead.
1.09j replaced the previous long \ifcase from \XINT_sqrt_c by some nested \ifnum's.
1.1 Ajout de \xintiiSqrt et \xintiiSquareRoot.
1.1a ajoute \xintiiSqrtR, which provides the rounded, not truncated square root.

```
\def\xintiiSqrt  {\romannumeral0\xintiisqrt  }%
\def\xintiiSqrtR {\romannumeral0\xintiisqrtr }%
\def\xintiiSquareRoot {\romannumeral0\xintiisquareroot }%
\def\xintiSqrt        {\romannumeral0\xintisqrt        }%
\def\xintiSquareRoot  {\romannumeral0\xintisquareroot  }%
\def\xintisqrt   {\expandafter\XINT_sqrt_post\romannumeral0\xintisquareroot   }%
\def\xintiisqrt  {\expandafter\XINT_sqrt_post\romannumeral0\xintiisquareroot  }%
\def\xintiisqrtr {\expandafter\XINT_sqrtr_post\romannumeral0\xintiisquareroot }%
\def\XINT_sqrt_post #1#2{\XINT_dec_pos #1\R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W\W }%
```

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

```
\def\XINT_sqrtr_post #1#2{\xintiiifLt {#2}{#1}% Lt <-> a<b
                          { #1}{\XINT_dec_pos #1\R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W\W }}%
\def\xintisquareroot  #1{\expandafter\XINT_sqrt_checkin\romannumeral0\xintnum{#1}\Z }%
\def\xintiisquareroot #1{\expandafter\XINT_sqrt_checkin\romannumeral-`0#1\Z }%
\def\XINT_sqrt_checkin #1%
{%
    \xint_UDzerominusfork
     #1-\XINT_sqrt_iszero
     0#1\XINT_sqrt_isneg
      0-{\XINT_sqrt #1}%
    \krof
}%
\def\XINT_sqrt_iszero #1\Z { 11}%
\edef\XINT_sqrt_isneg #1\Z {\noexpand\xintError:RootOfNegative\space 11}%
\def\XINT_sqrt #1\Z
{%
    \expandafter\XINT_sqrt_start\expandafter {\romannumeral0\xintlength {#1}}{#1}%
}%
\def\XINT_sqrt_start #1%
{%
    \ifnum #1<\xint_c_x
       \expandafter\XINT_sqrt_small_a
    \else
       \expandafter\XINT_sqrt_big_a
    \fi
    {#1}%
}%
\def\XINT_sqrt_small_a #1{\XINT_sqrt_a {#1}\XINT_sqrt_small_d }%
\def\XINT_sqrt_big_a   #1{\XINT_sqrt_a {#1}\XINT_sqrt_big_d   }%
\def\XINT_sqrt_a #1%
```

```
{%
   \ifodd #1
     \expandafter\XINT_sqrt_bB
   \else
     \expandafter\XINT_sqrt_bA
   \fi
   {#1}%
}%
\def\XINT_sqrt_bA #1#2#3%
{%
    \XINT_sqrt_bA_b #3\Z #2{#1}{#3}%
}%
\def\XINT_sqrt_bA_b #1#2#3\Z
{%
    \XINT_sqrt_c {#1#2}%
}%
\def\XINT_sqrt_bB #1#2#3%
{%
    \XINT_sqrt_bB_b #3\Z #2{#1}{#3}%
}%
\def\XINT_sqrt_bB_b #1#2\Z
{%
    \XINT_sqrt_c #1%
}%
\def\XINT_sqrt_c #1#2%
{%
    \expandafter #2\expandafter
    {\the\numexpr\ifnum #1>\xint_c_iii
                  \ifnum #1>\xint_c_viii
                  \ifnum #1>15 \ifnum #1>24 \ifnum #1>35
                  \ifnum #1>48 \ifnum #1>63 \ifnum #1>80
      10\else 9\fi \else 8\fi \else 7\fi \else 6\fi
        \else 5\fi \else 4\fi \else 3\fi \else 2\fi \relax }%
}%
\def\XINT_sqrt_small_d #1#2%
{%
   \expandafter\XINT_sqrt_small_e\expandafter
   {\the\numexpr #1\ifcase \numexpr #2/\xint_c_ii-\xint_c_i\relax
                     \or 0\or 00\or 000\or 0000\fi }%
}%
\def\XINT_sqrt_small_e #1#2%
{%
   \expandafter\XINT_sqrt_small_f\expandafter {\the\numexpr #1*#1-#2}{#1}%
}%
\def\XINT_sqrt_small_f #1#2%
{%
   \expandafter\XINT_sqrt_small_g\expandafter
   {\the\numexpr ((#1+#2)/(\xint_c_ii*#2))-\xint_c_i}{#1}{#2}%
}%
\def\XINT_sqrt_small_g #1%
{%
    \ifnum #1>\xint_c_
```

```
1310        \expandafter\XINT_sqrt_small_h
1311     \else
1312        \expandafter\XINT_sqrt_small_end
1313     \fi
1314     {#1}%
1315 }%
1316 \def\XINT_sqrt_small_h #1#2#3%
1317 {%
1318     \expandafter\XINT_sqrt_small_f\expandafter
1319     {\the\numexpr #2-\xint_c_ii*#1*#3+#1*#1\expandafter}\expandafter
1320     {\the\numexpr #3-#1}%
1321 }%
1322 \def\XINT_sqrt_small_end #1#2#3{ {#3}{#2}}%
1323 \def\XINT_sqrt_big_d #1#2%
1324 {%
1325    \ifodd #2
1326      \expandafter\expandafter\expandafter\XINT_sqrt_big_eB
1327    \else
1328      \expandafter\expandafter\expandafter\XINT_sqrt_big_eA
1329    \fi
1330    \expandafter {\the\numexpr #2/\xint_c_ii }{#1}%
1331 }%
1332 \def\XINT_sqrt_big_eA  #1#2#3%
1333 {%
1334     \XINT_sqrt_big_eA_a #3\Z {#2}{#1}{#3}%
1335 }%
1336 \def\XINT_sqrt_big_eA_a #1#2#3#4#5#6#7#8#9\Z
1337 {%
1338     \XINT_sqrt_big_eA_b {#1#2#3#4#5#6#7#8}%
1339 }%
1340 \def\XINT_sqrt_big_eA_b #1#2%
1341 {%
1342     \expandafter\XINT_sqrt_big_f
1343     \romannumeral0\XINT_sqrt_small_e {#2000}{#1}{#1}%
1344 }%
1345 \def\XINT_sqrt_big_eB #1#2#3%
1346 {%
1347     \XINT_sqrt_big_eB_a #3\Z {#2}{#1}{#3}%
1348 }%
1349 \def\XINT_sqrt_big_eB_a #1#2#3#4#5#6#7#8#9%
1350 {%
1351     \XINT_sqrt_big_eB_b {#1#2#3#4#5#6#7#8#9}%
1352 }%
1353 \def\XINT_sqrt_big_eB_b #1#2\Z #3%
1354 {%
1355     \expandafter\XINT_sqrt_big_f
1356     \romannumeral0\XINT_sqrt_small_e {#30000}{#1}{#1}%
1357 }%
1358 \def\XINT_sqrt_big_f #1#2#3#4%
1359 {%
1360    \expandafter\XINT_sqrt_big_f_a\expandafter
1361    {\the\numexpr #2+#3\expandafter}\expandafter
```

```
    {\romannumeral0\XINT_dsx_addzerosnofuss
                      {\numexpr #4-\xint_c_iv\relax}{#1}}{#4}%
}%
\def\XINT_sqrt_big_f_a #1#2#3#4%
{%
   \expandafter\XINT_sqrt_big_g\expandafter
   {\romannumeral0\xintiisub
       {\XINT_dsx_addzerosnofuss
        {\numexpr \xint_c_ii*#3-\xint_c_viii\relax}{#1}}{#4}}%
   {#2}{#3}%
}%
\def\XINT_sqrt_big_g #1#2%
{%
    \expandafter\XINT_sqrt_big_j
    \romannumeral0\xintiidivision{#1}%
        {\romannumeral0\XINT_dbl_pos #2\R\R\R\R\R\R\Z \W\W\W\W\W\W\W }{#2}%
}%
\def\XINT_sqrt_big_j #1%
{%
    \if0\XINT_Sgn #1\Z
          \expandafter \XINT_sqrt_big_end
    \else \expandafter \XINT_sqrt_big_k
    \fi {#1}%
}%
\def\XINT_sqrt_big_k #1#2#3%
{%
    \expandafter\XINT_sqrt_big_l\expandafter
    {\romannumeral0\xintiisub {#3}{#1}}%
    {\romannumeral0\xintiiadd {#2}{\xintiiSqr {#1}}}%
}%
\def\XINT_sqrt_big_l #1#2%
{%
   \expandafter\XINT_sqrt_big_g\expandafter
   {#2}{#1}%
}%
\def\XINT_sqrt_big_end #1#2#3#4{ {#3}{#2}}%
```

4.44 \xintiiE

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

```
\def\xintiiE {\romannumeral0\xintiie }% used in \xintMod.
\def\xintiie #1#2%
   {\expandafter\XINT_iie\the\numexpr #2\expandafter.\expandafter{\romannumeral-`0#1}}%
\def\XINT_iie #1.#2{\ifnum#1>\xint_c_ \xint_dothis{\xint_dsh {#2}{-#1}}\fi
                    \xint_orthat{ #2}}%
\XINT_restorecatcodes_endinput%
```

5 Package xintbinhex implementation

The commenting is currently (2015/09/17) 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}%
  [2015/09/12 v1.1c Expandable binary and hexadecimal conversions (jfB)]%
```

5.3 Constants, etc...

v1.08

```
\chardef\xint_c_xvi           16
% \chardef\xint_c_ii^v          32 % already in xint.sty
% \chardef\xint_c_ii^vi         64 % already in xint.sty
\chardef\xint_c_ii^vii       128
\mathchardef\xint_c_ii^viii  256
\mathchardef\xint_c_ii^xii  4096
\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
\newcount\xint_c_x^ix   \xint_c_x^ix   1000000000
\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 \xintDecToHex, \xintDecToBin

v1.08

```
\def\xintDecToHex {\romannumeral0\xintdectohex }%
\def\xintdectohex #1%
        {\expandafter\XINT_dth_checkin\romannumeral-`0#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\xint_minus_thenstop\romannumeral0\XINT_dth_P }%
\def\XINT_dth_P {\expandafter\XINT_dth_III\romannumeral-`0\XINT_dtbh_I {0.}}%
\def\xintDecToBin {\romannumeral0\xintdectobin }%
\def\xintdectobin #1%
        {\expandafter\XINT_dtb_checkin\romannumeral-`0#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\xint_minus_thenstop\romannumeral0\XINT_dtb_P }%
\def\XINT_dtb_P {\expandafter\XINT_dtb_III\romannumeral-`0\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-`0\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-`0\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-`0\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-`0\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
}%
```

5.5 \xintHexToDec

v1.08

```
\def\xintHexToDec {\romannumeral0\xinthextodec }%
\def\xinthextodec #1%
        {\expandafter\XINT_htd_checkin\romannumeral-`0#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\xint_minus_thenstop
                   \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
}%
```

5.6 \xintBinToDec

v1.08

```
\def\xintBinToDec {\romannumeral0\xintbintodec }%
\def\xintbintodec #1{\expandafter\XINT_btd_checkin
                     \romannumeral-`0#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\xint_minus_thenstop
                                \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 }%
```

5.7 \xintBinToHex

v1.08

```
\def\xintBinToHex {\romannumeral0\xintbintohex }%
\def\xintbintohex #1%
{%
    \expandafter\XINT_bth_checkin
```

```
                    \romannumeral0\expandafter\XINT_num_loop
                    \romannumeral-`0#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\xint_minus_thenstop\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{ }%
```

5.8 \xintHexToBin

v1.08

```
\def\xintHexToBin {\romannumeral0\xinthextobin }%
\def\xinthextobin #1%
{%
    \expandafter\XINT_htb_checkin\romannumeral-`0#1GGGGGGGG\T
}%
\def\XINT_htb_checkin #1%
{%
    \xint_UDsignfork
       #1\XINT_htb_N
        -{\XINT_htb_P #1}%
     \krof
}%
\def\XINT_htb_N {\expandafter\xint_minus_thenstop\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
}%
```

5.9 \xintCHexToBin

v1.08

```
\def\xintCHexToBin {\romannumeral0\xintchextobin }%
\def\xintchextobin #1%
{%
    \expandafter\XINT_chtb_checkin\romannumeral-`0#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\xint_minus_thenstop\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 (2015/09/17) very sparse. Release 1.09h has modified a bit the \xintTypesetEuclideAlgorithm and \xintTypesetBezoutAlgorithm layout with respect to line indentation in particular. And they use the xinttools \xintloop rather than the Plain TEX or LATEX's \loop.

Since 1.1 the package only loads xintcore, not xint. And for the \xintTypesetEuclideAlgorithm and \xintTypesetBezoutAlgorithm macros to be functional the package xinttools needs to be loaded explicitely by the user.

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}%
  [2015/09/12 v1.1c Euclide algorithm with xint package (jfB)]%
```

6.3 \xintGCD, \xintiiGCD

The macros of 1.09a benefits from the \xintnum which has been inserted inside \xintiabs in \xintnameimp; this is a little overhead but is more convenient for the user and also makes it easier to use into \xintexpr-essions. 1.1a adds \xintiiGCD mainly for \xintiiexpr benefit. Perhaps one should always have had ONLY ii versions from the beginning. And perhaps for sake of consistency, \xintGCD should be named \xintiGCD? too late.

```
\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

New with 1.09a. Inadvertent use of \xintiQuo which was promoted at the same time to add the \xintnum overhead. So with 1.09f \xintiiQuo without the overhead. However \xintiabs has the \xintnum thing. The advantage is that we can thus use lcm in \xintexpr. The disadvantage is that this has overhead in \xintiiexpr. Thus 1.1a has \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

1.09a inserts use of \xintnum

```
\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{#5}{#2}}{#7}}%
        {\romannumeral0\xintiiadd{\XINT_Mul{#6}{#2}}{#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

New with 1.09a. I also tried an optimization (not working two by two) which I thought was clever but it seemed to be less efficient ...

```
\def\xintGCDof      {\romannumeral0\xintgcdof }%
\def\xintgcdof    #1{\expandafter\XINT_gcdof_a\romannumeral-`0#1\relax }%
\def\XINT_gcdof_a #1{\expandafter\XINT_gcdof_b\romannumeral-`0#1\Z }%
\def\XINT_gcdof_b #1\Z #2{\expandafter\XINT_gcdof_c\romannumeral-`0#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-`0#1\relax }%
\def\XINT_lcmof_a #1{\expandafter\XINT_lcmof_b\romannumeral-`0#1\Z }%
\def\XINT_lcmof_b #1\Z #2{\expandafter\XINT_lcmof_c\romannumeral-`0#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}%
```

```
415        \expandafter\xint_gobble_iii
416     \fi
417     \XINT_TypesetBezoutAlgorithm
418 }%
419 \def\XINT_TypesetBezoutAlgorithm #1#2%
420 {%
421   \par
422   \begingroup
423     \xintAssignArray\xintBezoutAlgorithm {#1}{#2}\to\BEZ
424     \edef\A{\BEZ2}\edef\B{\BEZ6}\edef\N{\BEZ1}% A = |#1|, B = |#2|
425     \setbox 0 \vbox{\halign {$##$\cr \A\cr \B \cr}}%
426     \count255 1
427     \xintloop
428       \indent\hbox to \wd 0 {\hfil$\BEZ{4*\count255 - 2}$}%
429       ${} =  \BEZ{4*\count255 + 5}
430       \times \BEZ{4*\count255 + 2}
431           +  \BEZ{4*\count255 + 6}$\hfill\break
432       \hbox to \wd 0 {\hfil$\BEZ{4*\count255 +7}$}%
433       ${} = \BEZ{4*\count255 + 5}
434       \times \BEZ{4*\count255 + 3}
435           +  \BEZ{4*\count255 - 1}$\hfill\break
436       \hbox to \wd 0 {\hfil$\BEZ{4*\count255 +8}$}%
437       ${} =  \BEZ{4*\count255 + 5}
438       \times \BEZ{4*\count255 + 4}
439           +  \BEZ{4*\count255 }$
440       \par
441     \ifnum \count255 < \N
442     \advance \count255 1
443   \repeat
444     \edef\U{\BEZ{4*\N + 4}}%
445     \edef\V{\BEZ{4*\N + 3}}%
446     \edef\D{\BEZ5}%
447     \ifodd\N
448        $\U\times\A  - \V\times \B = -\D$%
449     \else
450        $\U\times\A - \V\times\B = \D$%
451     \fi
452     \par
453   \endgroup
454 }%
455 \XINT_restorecatcodes_endinput%
```

7 Package xintfrac implementation

The commenting is currently (2015/09/17) 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}%
  [2015/09/12 v1.1c Expandable operations on fractions (jfB)]%
\chardef\xint_c_xviii 18
```

7.3 \XINT_cntSgnFork

1.09i. Used internally, #1 must expand to \m@ne, \z@, or \@ne or equivalent. Does not insert a space token to stop a romannumeral0 expansion.

```
\def\XINT_cntSgnFork #1%
{%
    \ifcase #1\expandafter\xint_secondofthree
            \or\expandafter\xint_thirdofthree
          \else\expandafter\xint_firstofthree
    \fi
}%
```

7.4 \xintLen

```
\def\xintLen {\romannumeral0\xintlen }%
\def\xintlen #1%
{%
    \expandafter\XINT_flen\romannumeral0\XINT_infrac {#1}%
}%
\def\XINT_flen #1#2#3%
{%
```

```
    \expandafter\space
    \the\numexpr -1+\XINT_Abs {#1}+\XINT_Len {#2}+\XINT_Len {#3}\relax
}%
```

7.5 \XINT_lenrord_loop

```
\def\XINT_lenrord_loop #1#2#3#4#5#6#7#8#9%
{%  faire \romannumeral-`0\XINT_lenrord_loop 0{}#1\Z\W\W\W\W\W\W\W\Z
    \xint_gob_til_W #9\XINT_lenrord_W\W
    \expandafter\XINT_lenrord_loop\expandafter
    {\the\numexpr #1+7}{#9#8#7#6#5#4#3#2}%
}%
\def\XINT_lenrord_W\W\expandafter\XINT_lenrord_loop\expandafter #1#2#3\Z
{%
    \expandafter\XINT_lenrord_X\expandafter {#1}#2\Z
}%
\def\XINT_lenrord_X #1#2\Z
{%
    \XINT_lenrord_Y #2\R\R\R\R\R\R\T {#1}%
}%
\def\XINT_lenrord_Y #1#2#3#4#5#6#7#8\T
{%
    \xint_gob_til_W
            #7\XINT_lenrord_Z \xint_c_viii
            #6\XINT_lenrord_Z \xint_c_vii
            #5\XINT_lenrord_Z \xint_c_vi
            #4\XINT_lenrord_Z \xint_c_v
            #3\XINT_lenrord_Z \xint_c_iv
            #2\XINT_lenrord_Z \xint_c_iii
            \W\XINT_lenrord_Z \xint_c_ii   \Z
}%
\def\XINT_lenrord_Z #1#2\Z #3% retourne: {longueur}renverse\Z
{%
    \expandafter{\the\numexpr #3-#1\relax}%
}%
```

7.6 \XINT_outfrac

1.06a version now outputs 0/1[0] and not 0[0] in case of zero. More generally all macros have been checked in xintfrac, xintseries, xintcfrac, to make sure the output format for fractions was always A/B[n]. (except \xintIrr, \xintJrr, \xintRawWithZeros)

The problem with statements like those in the previous paragraph is that it is hard to maintain consistencies across relases.

Months later (2014/10/22): perhaps I should document what this macro does before I forget? from {e}{N}{D} it outputs N/D[e], checking in passing if D=0 or if N=0. It also makes sure D is not < 0. I am not sure but I don't think there is any place in the code which could call \XINT_outfrac with a D < 0, but I should check.

```
\def\XINT_outfrac #1#2#3%
{%
    \ifcase\XINT_cntSgn #3\Z
        \expandafter \XINT_outfrac_divisionbyzero
    \or
        \expandafter \XINT_outfrac_P
```

```
    \else
        \expandafter \XINT_outfrac_N
    \fi
    {#2}{#3}[#1]%
}%
\def\XINT_outfrac_divisionbyzero #1#2{\xintError:DivisionByZero\space #1/0}%
\edef\XINT_outfrac_P #1#2%
{%
    \noexpand\if0\noexpand\XINT_Sgn #1\noexpand\Z
        \noexpand\expandafter\noexpand\XINT_outfrac_Zero
    \noexpand\fi
    \space #1/#2%
}%
\def\XINT_outfrac_Zero #1[#2]{ 0/1[0]}%
\def\XINT_outfrac_N #1#2%
{%
    \expandafter\XINT_outfrac_N_a\expandafter
    {\romannumeral0\XINT_opp #2}{\romannumeral0\XINT_opp #1}%
}%
\def\XINT_outfrac_N_a #1#2%
{%
    \expandafter\XINT_outfrac_P\expandafter {#2}{#1}%
}%
```

7.7 \XINT_inFrac

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.

```
\def\XINT_inFrac {\romannumeral0\XINT_infrac }%
\def\XINT_infrac #1%
{%
    \expandafter\XINT_infrac_ \romannumeral-`0#1[\W]\Z\T
}%
\def\XINT_infrac_ #1[#2#3]#4\Z
{%
    \xint_UDwfork
      #2\XINT_infrac_A
      \W\XINT_infrac_B
    \krof
    #1[#2#3]#4%
}%
\def\XINT_infrac_A #1[\W]\T
{%
    \XINT_frac #1/\W\Z
}%
\def\XINT_infrac_B #1%
{%
    \xint_gob_til_zero #1\XINT_infrac_Zero0\XINT_infrac_BB #1%
}%
\def\XINT_infrac_BB #1[\W]\T {\XINT_infrac_BC #1/\W\Z }%
```

```
\def\XINT_infrac_BC #1/#2#3\Z
{%
    \xint_UDwfork
     #2\XINT_infrac_BCa
     \W{\expandafter\XINT_infrac_BCb \romannumeral-`0#2}%
    \krof
    #3\Z #1\Z
}%
\def\XINT_infrac_BCa \Z #1[#2]#3\Z { {#2}{#1}{1}}%
\def\XINT_infrac_BCb #1[#2]/\W\Z #3\Z { {#2}{#3}{#1}}%
\def\XINT_infrac_Zero #1\T { {0}{0}{1}}%
```

7.8 \XINT_frac

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

```
\def\XINT_frac #1/#2#3\Z
{%
    \xint_UDwfork
     #2\XINT_frac_A
     \W{\expandafter\XINT_frac_U \romannumeral-`0#2}%
    \krof
    #3e\W\Z #1e\W\Z
}%
\def\XINT_frac_U #1e#2#3\Z
{%
    \xint_UDwfork
      #2\XINT_frac_Ua
      \W{\XINT_frac_Ub #2}%
    \krof
    #3\Z #1\Z
}%
\def\XINT_frac_Ua      \Z #1/\W\Z  {\XINT_frac_B #1.\W\Z {0}}%
\def\XINT_frac_Ub #1/\W e\W\Z #2\Z {\XINT_frac_B #2.\W\Z {#1}}%
\def\XINT_frac_B #1.#2#3\Z
{%
    \xint_UDwfork
      #2\XINT_frac_Ba
      \W{\XINT_frac_Bb #2}%
    \krof
    #3\Z #1\Z
}%
\def\XINT_frac_Ba \Z #1\Z {\XINT_frac_T {0}{#1}}%
\def\XINT_frac_Bb #1.\W\Z #2\Z
{%
    \expandafter \XINT_frac_T \expandafter
    {\romannumeral0\xintlength {#1}}{#2#1}%
}%
\def\XINT_frac_A e\W\Z {\XINT_frac_T {0}{1}{0}}%
\def\XINT_frac_T #1#2#3#4e#5#6\Z
{%
```

```
    \xint_UDwfork
      #5\XINT_frac_Ta
      \W{\XINT_frac_Tb #5}%
    \krof
    #6\Z #4\Z {#1}{#2}{#3}%
}%
\def\XINT_frac_Ta \Z #1\Z      {\XINT_frac_C #1.\W\Z {0}}%
\def\XINT_frac_Tb #1e\W\Z #2\Z {\XINT_frac_C #2.\W\Z {#1}}%
\def\XINT_frac_C #1.#2#3\Z
{%
    \xint_UDwfork
      #2\XINT_frac_Ca
      \W{\XINT_frac_Cb #2}%
    \krof
    #3\Z #1\Z
}%
\def\XINT_frac_Ca \Z #1\Z {\XINT_frac_D {0}{#1}}%
\def\XINT_frac_Cb #1.\W\Z #2\Z
{%
    \expandafter\XINT_frac_D\expandafter
    {\romannumeral0\xintlength {#1}}{#2#1}%
}%
\def\XINT_frac_D #1#2#3#4#5#6%
{%
    \expandafter \XINT_frac_E \expandafter
    {\the\numexpr -#1+#3+#4-#6\expandafter}\expandafter
    {\romannumeral0\XINT_num_loop #2%
     \xint_relax\xint_relax\xint_relax\xint_relax
     \xint_relax\xint_relax\xint_relax\xint_relax\Z }%
    {\romannumeral0\XINT_num_loop #5%
     \xint_relax\xint_relax\xint_relax\xint_relax
     \xint_relax\xint_relax\xint_relax\xint_relax\Z }%
}%
\def\XINT_frac_E #1#2#3%
{%
   \expandafter \XINT_frac_F  #3\Z {#2}{#1}%
}%
\def\XINT_frac_F #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_frac_Gdivisionbyzero
      0#1\XINT_frac_Gneg
       0-{\XINT_frac_Gpos #1}%
    \krof
}%
\edef\XINT_frac_Gdivisionbyzero #1\Z #2#3%
{%
   \noexpand\xintError:DivisionByZero\space {0}{#2}{0}%
}%
\def\XINT_frac_Gneg #1\Z #2#3%
{%
    \expandafter\XINT_frac_H \expandafter{\romannumeral0\XINT_opp #2}{#3}{#1}%
```

```
}%
\def\XINT_frac_H #1#2{ {#2}{#1}}%
\def\XINT_frac_Gpos #1\Z #2#3{ {#3}{#2}{#1}}%
```

7.9 \XINT_factortens, \XINT_cuz_cnt

```
\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}%
}%
```

7.10 \xintRaw

1.07: this macro simply prints in a user readable form the fraction after its initial scanning. Useful when put inside braces in an \xintexpr, when the input is not yet in the A/B[n] form.

```
\def\xintRaw {\romannumeral0\xintraw }%
\def\xintraw
```

```
{%
    \expandafter\XINT_raw\romannumeral0\XINT_infrac
}%
\def\XINT_raw #1#2#3{ #2/#3[#1]}%
```

7.11 \xintPRaw

1.09b

```
\def\xintPRaw {\romannumeral0\xintpraw }%
\def\xintpraw
{%
    \expandafter\XINT_praw\romannumeral0\XINT_infrac
}%
\def\XINT_praw #1%
{%
    \ifnum #1=\xint_c_ \expandafter\XINT_praw_a\fi \XINT_praw_A {#1}%
}%
\def\XINT_praw_A #1#2#3%
{%
    \if\XINT_isOne{#3}1\expandafter\xint_firstoftwo
                   \else\expandafter\xint_secondoftwo
    \fi { #2[#1]}{ #2/#3[#1]}%
}%
\def\XINT_praw_a\XINT_praw_A #1#2#3%
{%
    \if\XINT_isOne{#3}1\expandafter\xint_firstoftwo
                   \else\expandafter\xint_secondoftwo
    \fi { #2}{ #2/#3}%
}%
```

7.12 \xintRawWithZeros

This was called \xintRaw in versions earlier than 1.07

```
\def\xintRawWithZeros {\romannumeral0\xintrawwithzeros }%
\def\xintrawwithzeros
{%
    \expandafter\XINT_rawz\romannumeral0\XINT_infrac
}%
\def\XINT_rawz #1%
{%
    \ifcase\XINT_cntSgn #1\Z
      \expandafter\XINT_rawz_Ba
    \or
      \expandafter\XINT_rawz_A
    \else
      \expandafter\XINT_rawz_Ba
    \fi
    {#1}%
}%
\def\XINT_rawz_A  #1#2#3{\xint_dsh {#2}{-#1}/#3}%
\def\XINT_rawz_Ba #1#2#3{\expandafter\XINT_rawz_Bb
```

```
                        \expandafter{\romannumeral0\xint_dsh {#3}{#1}}{#2}}%
\def\XINT_rawz_Bb #1#2{ #2/#1}%
```

7.13 \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}}%
```

7.14 \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}}}}%
```

7.15 \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_dsh {#2}{-#1}}%
\def\XINT_numer_B #1#2#3{ #2}%
```

7.16 \xintDenominator

```
\def\xintDenominator {\romannumeral0\xintdenominator }%
\def\xintdenominator
{%
    \expandafter\XINT_denom\romannumeral0\XINT_infrac
}%
\def\XINT_denom #1%
```

```
{%
    \ifcase\XINT_cntSgn #1\Z
      \expandafter\XINT_denom_B
    \or
      \expandafter\XINT_denom_A
    \else
      \expandafter\XINT_denom_B
    \fi
    {#1}%
}%
\def\XINT_denom_A #1#2#3{ #3}%
\def\XINT_denom_B #1#2#3{\xint_dsh {#3}{#1}}%
```

7.17 \xintFrac

```
\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 }%
```

7.18 \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\xint_minus_thenstop\romannumeral0\XINT_sgnfrac_P
}%
```

7.19 \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%
}%
```

7.20 \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\xint_minus_thenstop\romannumeral0\XINT_sgnfwover_P
}%
```

7.21 \xintREZ

```
\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\xint_minus_thenstop\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]}%
```

7.22 \xintE, \xintFloatE, \XINTinFloatE

1.07: The fraction is the first argument contrarily to \xintTrunc and \xintRound.
  \xintfE (1.07) and \xintiE (1.09i) are for \xintexpr and cousins. It is quite annoying that \numexpr does not know how to deal correctly with a minus sign - as prefix: \numexpr -(1)\relax is

illegal! (one can do \numexpr 0-(1)\relax).
  the 1.07 \xintE puts directly its second argument in a \numexpr. The \xintfE first uses \xintNum on it, this is necessary for use in \xintexpr. (but one cannot use directly infix notation in the second argument of \xintfE)
  1.09i also adds \xintFloatE and modifies \XINTinFloatfE, although currently the latter is only used from \xintfloatexpr hence always with \XINTdigits, it comes equipped with its first argument within brackets as the other \XINTinFloat... macros.
  1.09m ceases here and elsewhere, also in \xintcfracname, to use \Z as delimiter in the code for the optional argument, as this is unsafe (it makes impossible to the user to employ \Z as argument to the macro). Replaced by \xint_relax. 1.09e had already done that in \xintSeq, but this should have been systematic.
  1.1 modifies and moves \xintiiE to xint.sty, and cleans up some unneeded stuff, now that expressions implement scientific notation directly at the number parsing level.

```
\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\expandafter{\the\numexpr #1+#4}{#2}{#3}%
}%
\def\XINT_e_end #1#2#3{ #2/#3[#1]}%
\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\expandafter\expandafter\XINT_float_a
    \expandafter\xint_exchangetwo_keepbraces\expandafter
            {\the\numexpr #2+#5}{#1}{#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\expandafter {\the\numexpr #3+#2}{#1}}%
\def\XINT_infloate_end #1#2{ #2[#1]}%
```

7.23 \xintIrr

1.04 fixes a buggy \xintIrr {0}. 1.05 modifies the initial parsing and post-processing to use \xintrawwithzeros and to more quickly deal with an input denominator equal to 1. 1.08 version does not remove a /1 denominator.

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

7.24 \xintifInt

1.09e. xintfrac.sty only. Fixed in 1.1 to not use \xintIrr anymore as it was really stupid overhead.

```
\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
}%
```

7.25 \xintJrr

Modified similarly as \xintIrr in release 1.05. 1.08 version does not remove a /1 denominator.

```
\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 \xint_minus_thenstop }%
\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{#4}{#1}}{#6}}%
        {\romannumeral0\xintiiadd{\XINT_Mul{#5}{#1}}{#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}%
}%
```

7.26 \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}}%
```

7.27 \XINTinFloatFracdigits

1.09i, for frac 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]}%
```

7.28 \xintTrunc, \xintiTrunc

Modified in 1.06 to give the first argument to a \numexpr.

1.09f fixes the overhead added in 1.09a to some inner routines when \xintiquo was redefined to use \xintnum. Now uses \xintiiquo, rather.

1.09j: minor improvements, \XINT_trunc_E was very strange and defined two never occuring branches; also, optimizes the call to the division routine, and the zero loops.

1.1 adds \xintTTrunc as a shortcut to what \xintiTrunc 0 does, and maps \xintNum to it.

```
\def\xintTrunc  {\romannumeral0\xinttrunc }%
\def\xintiTrunc {\romannumeral0\xintitrunc }%
\def\xinttrunc #1%
{%
    \expandafter\XINT_trunc\expandafter {\the\numexpr #1}%
}%
\def\XINT_trunc #1#2%
{%
    \expandafter\XINT_trunc_G
    \romannumeral0\expandafter\XINT_trunc_A
    \romannumeral0\XINT_infrac {#2}{#1}{#1}%
}%
\def\xintitrunc #1%
{%
```

```
    \expandafter\XINT_itrunc\expandafter {\the\numexpr #1}%
}%
\def\XINT_itrunc #1#2%
{%
    \expandafter\XINT_itrunc_G
    \romannumeral0\expandafter\XINT_trunc_A
    \romannumeral0\XINT_infrac {#2}{#1}{#1}%
}%
\def\XINT_trunc_A #1#2#3#4%
{%
    \expandafter\XINT_trunc_checkifzero
    \expandafter{\the\numexpr #1+#4}#2\Z {#3}%
}%
\def\XINT_trunc_checkifzero #1#2#3\Z
{%
    \xint_gob_til_zero #2\XINT_trunc_iszero0\XINT_trunc_B {#1}{#2#3}%
}%
\def\XINT_trunc_iszero0\XINT_trunc_B #1#2#3{ 0\Z 0}%
\def\XINT_trunc_B #1%
{%
    \ifcase\XINT_cntSgn #1\Z
      \expandafter\XINT_trunc_D
    \or
      \expandafter\XINT_trunc_D
    \else
      \expandafter\XINT_trunc_C
    \fi
    {#1}%
}%
\def\XINT_trunc_C #1#2#3%
{%
    \expandafter\XINT_trunc_CE\expandafter
    {\romannumeral0\XINT_dsx_zeroloop {-#1}{}\Z {#3}}{#2}%
}%
\def\XINT_trunc_CE #1#2{\XINT_trunc_E #2.{#1}}%
\def\XINT_trunc_D #1#2%
{%
    \expandafter\XINT_trunc_E
    \romannumeral0\XINT_dsx_zeroloop {#1}{}\Z {#2}.%
}%
\def\XINT_trunc_E #1%
{%
    \xint_UDsignfork
       #1\XINT_trunc_Fneg
        -{\XINT_trunc_Fpos #1}%
    \krof
}%
\def\XINT_trunc_Fneg #1.#2{\expandafter\xint_firstoftwo_thenstop
           \romannumeral0\XINT_div_prepare {#2}{#1}\Z \xint_minus_thenstop}%
\def\XINT_trunc_Fpos #1.#2{\expandafter\xint_firstoftwo_thenstop
           \romannumeral0\XINT_div_prepare {#2}{#1}\Z \space }%
\def\XINT_itrunc_G #1#2\Z #3#4%
```

```
{%
    \xint_gob_til_zero #1\XINT_trunc_zero 0#3#1#2%
}%
\def\XINT_trunc_zero 0#1#20{ 0}%
\def\XINT_trunc_G #1\Z #2#3%
{%
    \xint_gob_til_zero #2\XINT_trunc_zero 0%
    \expandafter\XINT_trunc_H\expandafter
    {\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.%
    \romannumeral0\XINT_dsx_zeroloop {#1}{}\Z {}#2% #1=-0 autoris\'e !
}%
```

7.29 \xintTTrunc

1.1, a tiny bit more efficient than doing \xintiTrunc0. I map \xintNum to it, and I use it in \xintexpr for various things. Faster I guess than the \xintiFloor.

```
\def\xintTTrunc {\romannumeral0\xintttrunc }%
\def\xintttrunc #1%
{%
    \expandafter\XINT_itrunc_G
    \romannumeral0\expandafter\XINT_ttrunc_A
    \romannumeral0\XINT_infrac {#1}0% this last 0 to let \XINT_itrunc_G be happy
}%
\def\XINT_ttrunc_A #1#2#3{\XINT_trunc_checkifzero {#1}#2\Z {#3}}%
```

7.30 \xintNum

This extension of the xint original xintNum is added in 1.05, as a synonym to \xintIrr, but raising an error when the input does not evaluate to an integer. Usable with not too much overhead on integer input as \xintIrr checks quickly for a denominator equal to 1 (which will be put there by the \XINT_infrac called by \xintrawwithzeros). This way, macros such as \xintQuo can be modified with minimal overhead to accept fractional input as long as it evaluates to an integer.

22 june 2014 (dev 1.1) I just don't understand what was the point of going through \xintIrr if to raise an arror afterwards... and raising errors is silly, so let's do it sanely at last. In between I added \xintiFloor, thus, let's just let it to it.
24 october 2014 (final 1.1) (I left it taking dust since June...), I did \xintTTrunc, and will thus map \xintNum to it

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

7.31 \xintRound, \xintiRound

Modified in 1.06 to give the first argument to a \numexpr.

```
\def\xintRound {\romannumeral0\xintround }%
\def\xintiRound {\romannumeral0\xintiround }%
\def\xintround #1%
{%
    \expandafter\XINT_round\expandafter {\the\numexpr #1}%
}%
\def\XINT_round
{%
    \expandafter\XINT_trunc_G\romannumeral0\XINT_round_A
}%
\def\xintiround #1%
{%
    \expandafter\XINT_iround\expandafter {\the\numexpr #1}%
}%
\def\XINT_iround
{%
    \expandafter\XINT_itrunc_G\romannumeral0\XINT_round_A
}%
\def\XINT_round_A #1#2%
{%
    \expandafter\XINT_round_B
    \romannumeral0\expandafter\XINT_trunc_A
    \romannumeral0\XINT_infrac {#2}{\the\numexpr #1+1\relax}{#1}%
}%
\def\XINT_round_B #1\Z
{%
    \expandafter\XINT_round_C
    \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
    \Z
}%
\def\XINT_round_C #1%
{%
    \ifnum #1<5
        \expandafter\XINT_round_Daa
    \else
        \expandafter\XINT_round_Dba
```

```
898     \fi
899 }%
900 \def\XINT_round_Daa #1%
901 {%
902     \xint_gob_til_Z #1\XINT_round_Daz\Z \XINT_round_Da #1%
903 }%
904 \def\XINT_round_Daz\Z \XINT_round_Da \Z { 0\Z }%
905 \def\XINT_round_Da #1\Z
906 {%
907     \XINT_rord_main {}#1%
908       \xint_relax
909         \xint_bye\xint_bye\xint_bye\xint_bye
910         \xint_bye\xint_bye\xint_bye\xint_bye
911       \xint_relax  \Z
912 }%
913 \def\XINT_round_Dba #1%
914 {%
915     \xint_gob_til_Z #1\XINT_round_Dbz\Z \XINT_round_Db #1%
916 }%
917 \def\XINT_round_Dbz\Z \XINT_round_Db \Z { 1\Z }%
918 \def\XINT_round_Db #1\Z
919 {%
920     \XINT_addm_A 0{}1000\W\X\Y\Z #1000\W\X\Y\Z \Z
921 }%
```

7.32 \xintXTrunc

1.09j [2014/01/06] This is completely expandable but not f-expandable. Designed be used inside an \edef or a \write, if one is interested in getting tens of thousands of digits from the decimal expansion of some fraction... it is not worth using it rather than \xintTrunc if for less than *hundreds* of digits. For efficiency it clones part of the preparatory division macros, as the same denominator will be used again and again. The D parameter which says how many digits to keep after decimal mark must be at least 1 (and it is forcefully set to such a value if found negative or zero, to avoid an eternal loop).

For reasons of efficiency I try to use the shortest possible denominator, so if the fraction is A/B[N], I want to use B. For N at least zero, just immediately replace A by A.10^N. The first division then may be a little longish but the next ones will be fast (if B is not too big). For N<0, this is a bit more complicated. I thought somewhat about this, and I would need a rather complicated approach going through a long division algorithm, forcing me to essentially clone the actual division with some differences; a side thing is that as this would use blocks of four digits I would have a hard time allowing a non-multiple of four number of post decimal mark digits.

Thus, for N<0, another method is followed. First the euclidean division A/B=Q+R/B is done. The number of digits of Q is M. If |N|\leq D, we launch inside a \csname the routine for obtaining D-|N| next digits (this may impact TeX's memory if D is very big), call them T. We then need to position the decimal mark D slots from the right of QT, which has length M+D-|N|, hence |N| slots from the right of Q. We thus avoid having to work will the T, as D may be very very big (\xintXTrunc's only goal is to make it possible to learn by hearts decimal expansions with thousands of digits). We can use the \xintDecSplit for that on Q . Computing the length M of Q was a more or less unavoidable step. If |N|>D, the \csname step is skipped we need to remove the D-|N| last digits from Q, etc.. we compare D-|N| with the length M of Q etc... (well in this last, very uncommon, branch, I stopped trying to optimize things and I even do an \xintnum to ensure a 0 if something comes out empty from \xintDecSplit).

```
\def\xintXTrunc #1#2%
{%
    \expandafter\XINT_xtrunc_a\expandafter
    {\the\numexpr #1\expandafter}\romannumeral0\xintraw {#2}%
}%
\def\XINT_xtrunc_a #1%
{%
    \expandafter\XINT_xtrunc_b\expandafter
    {\the\numexpr\ifnum#1<\xint_c_i \xint_c_i-\fi #1}%
}%
\def\XINT_xtrunc_b #1%
{%
    \expandafter\XINT_xtrunc_c\expandafter
    {\the\numexpr (#1+\xint_c_ii^v)/\xint_c_ii^vi-\xint_c_i}{#1}%
}%
\def\XINT_xtrunc_c #1#2%
{%
    \expandafter\XINT_xtrunc_d\expandafter
    {\the\numexpr #2-\xint_c_ii^vi*#1}{#1}{#2}%
}%
\def\XINT_xtrunc_d #1#2#3#4/#5[#6]%
{%
    \XINT_xtrunc_e #4.{#6}{#5}{#3}{#2}{#1}%
}%
```

#1=numerator.#2=N,#3=B,#4=D,#5=Blocs,#6=extra

```
\def\XINT_xtrunc_e #1%
{%
    \xint_UDzerominusfork
        #1-\XINT_xtrunc_zero
        0#1\XINT_xtrunc_N
        0-{\XINT_xtrunc_P #1}%
    \krof
}%
\def\XINT_xtrunc_zero .#1#2#3#4#5%
{%
    0.\romannumeral0\expandafter\XINT_dsx_zeroloop\expandafter
                                  {\the\numexpr #5}{}\Z {}%
    \xintiloop [#4+-1]
    \ifnum \xintiloopindex>\xint_c_
    0000000000000000000000000000000000000000000000000000000000000000%
    \repeat
}%
\def\XINT_xtrunc_N {-\XINT_xtrunc_P }%
\def\XINT_xtrunc_P #1.#2%
{%
    \ifnum #2<\xint_c_
        \expandafter\XINT_xtrunc_negN_Q
    \else
        \expandafter\XINT_xtrunc_Q
    \fi  {#2}{#1}.%
}%
```

```
\def\XINT_xtrunc_negN_Q #1#2.#3#4#5#6%
{%
    \expandafter\XINT_xtrunc_negN_R
    \romannumeral0\XINT_div_prepare {#3}{#2}{#3}{#1}{#4}%
}%
```

#1=Q, #2=R, #3=B, #4=N<0, #5=D

```
\def\XINT_xtrunc_negN_R #1#2#3#4#5%
{%
    \expandafter\XINT_xtrunc_negN_S\expandafter
    {\the\numexpr -#4}{#5}{#2}{#3}{#1}%
}%
\def\XINT_xtrunc_negN_S #1#2%
{%
    \expandafter\XINT_xtrunc_negN_T\expandafter
    {\the\numexpr #2-#1}{#1}{#2}%
}%
\def\XINT_xtrunc_negN_T #1%
{%
    \ifnum \xint_c_<#1
      \expandafter\XINT_xtrunc_negNA
    \else
      \expandafter\XINT_xtrunc_negNW
    \fi {#1}%
}%
```

#1=D-|N|>0, #2=|N|, #3=D, #4=R, #5=B, #6=Q

```
\def\XINT_xtrunc_unlock #10.{ }%
\def\XINT_xtrunc_negNA #1#2#3#4#5#6%
{%
   \expandafter\XINT_xtrunc_negNB\expandafter
   {\romannumeral0\expandafter\expandafter\expandafter
    \XINT_xtrunc_unlock\expandafter\string
    \csname\XINT_xtrunc_b {#1}#4/#5[0]\expandafter\endcsname
    \expandafter}\expandafter
    {\the\numexpr\xintLength{#6}-#2}{#6}%
}%
\def\XINT_xtrunc_negNB #1#2#3{\XINT_xtrunc_negNC {#2}{#3}#1}%
\def\XINT_xtrunc_negNC #1%
{%
    \ifnum \xint_c_ < #1
      \expandafter\XINT_xtrunc_negNDa
    \else
      \expandafter\XINT_xtrunc_negNE
    \fi {#1}%
}%
\def\XINT_xtrunc_negNDa #1#2%
{%
    \expandafter\XINT_xtrunc_negNDb%
    \romannumeral0\XINT_split_fromleft_loop {#1}{}#2\W\W\W\W\W\W\W\W\Z
}%
\def\XINT_xtrunc_negNDb #1#2{#1.#2}%
```

```
\def\XINT_xtrunc_negNE #1#2%
{%
    0.\romannumeral0\XINT_dsx_zeroloop {-#1}{}\Z {}#2%
}%
```

#1=D-|N|<=0, #2=|N|, #3=D, #4=R, #5=B, #6=Q

```
\def\XINT_xtrunc_negNW #1#2#3#4#5#6%
{%
    \expandafter\XINT_xtrunc_negNX\expandafter
    {\romannumeral0\xintnum{\xintDecSplitL {-#1}{#6}}}{#3}%
}%
\def\XINT_xtrunc_negNX #1#2%
{%
    \expandafter\XINT_xtrunc_negNC\expandafter
    {\the\numexpr\xintLength {#1}-#2}{#1}%
}%
\def\XINT_xtrunc_Q #1%
{%
    \expandafter\XINT_xtrunc_prepare_I
    \romannumeral0\XINT_dsx_zeroloop {#1}{}\Z
}%
\def\XINT_xtrunc_prepare_I #1.#2#3%
{%
    \expandafter\XINT_xtrunc_prepareB_aa\expandafter
    {\romannumeral0\xintlength {#2}}{#2}{#1}%
}%
\def\XINT_xtrunc_prepareB_aa #1%
{%
    \ifnum #1=\xint_c_i
      \expandafter\XINT_xtrunc_prepareB_onedigit
    \else
      \expandafter\XINT_xtrunc_prepareB_PaBa
    \fi
    {#1}%
}%
\def\XINT_xtrunc_prepareB_onedigit #1#2%
{%
    \ifcase#2
    \or\expandafter\XINT_xtrunc_BisOne
    \or\expandafter\XINT_xtrunc_BisTwo
    \else\expandafter\XINT_xtrunc_prepareB_PaBe
    \fi {000}{0}{4}{#2}%
}%
\def\XINT_xtrunc_BisOne #1#2#3#4#5#6#7%
{%
    #5.\romannumeral0\expandafter\XINT_dsx_zeroloop\expandafter
                                    {\the\numexpr #7}{}\Z {}%
    \xintiloop [#6+-1]
    \ifnum \xintiloopindex>\xint_c_
    0000000000000000000000000000000000000000000000000000000000000000%
    \repeat
}%
```

```
\def\XINT_xtrunc_BisTwo #1#2#3#4#5#6#7%
{%
    \xintHalf {#5}.\ifodd\xintiiLDg{#5} 5\else 0\fi
    \romannumeral0\expandafter\XINT_dsx_zeroloop\expandafter
                                  {\the\numexpr #7-\xint_c_i}{}\Z {}%
    \xintiloop [#6+-1]
    \ifnum \xintiloopindex>\xint_c_
    0000000000000000000000000000000000000000000000000000000000000000%
    \repeat
}%
\def\XINT_xtrunc_prepareB_PaBa #1#2%
{%
    \expandafter\XINT_xtrunc_Pa\expandafter
    {\romannumeral0\XINT_xtrunc_prepareB_a {#1}{#2}}%
}%
\def\XINT_xtrunc_prepareB_a #1%
{%
  \expandafter\XINT_xtrunc_prepareB_c\expandafter
  {\the\numexpr \xint_c_iv*((#1+\xint_c_i)/\xint_c_iv)}{#1}%
}%
\def\XINT_xtrunc_prepareB_c #1#2%
{%
    \csname XINT_xtrunc_prepareB_d\romannumeral\numexpr#1-#2\endcsname
    {#1}%
}%
\def\XINT_xtrunc_prepareB_d    {\XINT_xtrunc_prepareB_e {}{0000}}%
\def\XINT_xtrunc_prepareB_di   {\XINT_xtrunc_prepareB_e {0}{000}}%
\def\XINT_xtrunc_prepareB_dii  {\XINT_xtrunc_prepareB_e {00}{00}}%
\def\XINT_xtrunc_prepareB_diii {\XINT_xtrunc_prepareB_e {000}{0}}%
\def\XINT_xtrunc_prepareB_PaBe #1#2#3#4%
{%
    \expandafter\XINT_xtrunc_Pa\expandafter
    {\romannumeral0\XINT_xtrunc_prepareB_e {#1}{#2}{#3}{#4}}%
}%
\def\XINT_xtrunc_prepareB_e #1#2#3#4%
{%
    \ifnum#3=\xint_c_iv\expandafter\XINT_xtrunc_prepareLittleB_f
                    \else\expandafter\XINT_xtrunc_prepareB_f
    \fi
    #4#1{#3}{#2}{#1}%
}%
\def\XINT_xtrunc_prepareB_f #1#2#3#4#5#{%
    \expandafter\space
    \expandafter\XINT_div_prepareB_g
     \the\numexpr  #1#2#3#4+\xint_c_i\expandafter
    .\the\numexpr (#1#2#3#4+\xint_c_i)/\xint_c_ii\expandafter
    .\romannumeral0\xintreverseorder {#1#2#3#4#5}.{#1#2#3#4}%
}%
\def\XINT_xtrunc_prepareLittleB_f #1#{%
    \expandafter\space\expandafter
    \XINT_div_prepareB_g \the\numexpr  #1/\xint_c_ii.{}.{}.{#1}%
}%
```

```
\def\XINT_xtrunc_Pa #1#2%
{%
    \expandafter\XINT_xtrunc_Pb\romannumeral0#1{#2}{#1}%
}%
\def\XINT_xtrunc_Pb #1#2#3#4{#1.\XINT_xtrunc_A {#4}{#2}{#3}}%
\def\XINT_xtrunc_A #1%
{%
    \unless\ifnum #1>\xint_c_ \XINT_xtrunc_transition\fi
    \expandafter\XINT_xtrunc_B\expandafter{\the\numexpr #1-\xint_c_i}%
}%
\def\XINT_xtrunc_B #1#2#3%
{%
    \expandafter\XINT_xtrunc_D\romannumeral0#3%
    {#200000000000000000000000000000000000000000000000000000000000000000}%
    {#1}{#3}%
}%
\def\XINT_xtrunc_D #1#2#3%
{%
    \romannumeral0\expandafter\XINT_dsx_zeroloop\expandafter
                  {\the\numexpr \xint_c_ii^vi-\xintLength{#1}}{}\Z {}#1%
    \XINT_xtrunc_A {#3}{#2}%
}%
\def\XINT_xtrunc_transition\fi
    \expandafter\XINT_xtrunc_B\expandafter #1#2#3#4%
{%
    \fi
    \ifnum #4=\xint_c_ \XINT_xtrunc_abort\fi
    \expandafter\XINT_xtrunc_x\expandafter
    {\romannumeral0\XINT_dsx_zeroloop {#4}{}\Z {#2}}{#3}{#4}%
}%
\def\XINT_xtrunc_x #1#2%
{%
    \expandafter\XINT_xtrunc_y\romannumeral0#2{#1}%
}%
\def\XINT_xtrunc_y #1#2#3%
{%
    \romannumeral0\expandafter\XINT_dsx_zeroloop\expandafter
                  {\the\numexpr #3-\xintLength{#1}}{}\Z {}#1%
}%
\def\XINT_xtrunc_abort\fi\expandafter\XINT_xtrunc_x\expandafter #1#2#3{\fi}%
```

7.33 \xintDigits

The mathchardef used to be called \XINT_digits, but for reasons originating in \xintNewExpr (and now obsolete), release 1.09a uses \XINTdigits without underscore.

```
\mathchardef\XINTdigits 16
\def\xintDigits #1#2%
   {\afterassignment \xint_gobble_i \mathchardef\XINTdigits=}%
\def\xinttheDigits {\number\XINTdigits }%
```

7.34 \xintFloat

1.07. Completely re-written in 1.08a, with spectacular speed gains. 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.

Here again some inner macros used the \xintiquo with extra \xintnum overhead in 1.09a, 1.09f reinstalled use of \xintiiquo without this overhead.

```
\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
    \romannumeral0\XINT_infrac {#1}\XINT_float_Q
}%
\def\XINT_float_opt [\xint_relax #1]#2%
{%
    \expandafter\XINT_float_a\expandafter
    {\the\numexpr #1\expandafter}%
    \romannumeral0\XINT_infrac {#2}\XINT_float_Q
}%
\def\XINT_float_a #1#2#3% #1=P, #2=n, #3=A, #4=B
{%
    \XINT_float_fork #3\Z {#1}{#2}% #1 = precision, #2=n
}%
\def\XINT_float_fork #1%
{%
    \xint_UDzerominusfork
     #1-\XINT_float_zero
     0#1\XINT_float_J
      0-{\XINT_float_K #1}%
    \krof
}%
\def\XINT_float_zero #1\Z #2#3#4#5{ 0.e0}%
\def\XINT_float_J {\expandafter\xint_minus_thenstop\romannumeral0\XINT_float_K }%
\def\XINT_float_K #1\Z #2% #1=A, #2=P, #3=n, #4=B
{%
    \expandafter\XINT_float_L\expandafter
    {\the\numexpr\xintLength{#1}\expandafter}\expandafter
    {\the\numexpr #2+\xint_c_ii}{#1}{#2}%
}%
\def\XINT_float_L #1#2%
{%
    \ifnum #1>#2
      \expandafter\XINT_float_Ma
    \else
      \expandafter\XINT_float_Mc
```

```
    \fi {#1}{#2}%
}%
\def\XINT_float_Ma #1#2#3%
{%
    \expandafter\XINT_float_Mb\expandafter
    {\the\numexpr #1-#2\expandafter\expandafter\expandafter}%
    \expandafter\expandafter\expandafter
    {\expandafter\xint_firstoftwo
     \romannumeral0\XINT_split_fromleft_loop {#2}{}#3\W\W\W\W\W\W\W\W\Z
     }{#2}%
}%
\def\XINT_float_Mb #1#2#3#4#5#6% #2=A', #3=P+2, #4=P, #5=n, #6=B
{%
   \expandafter\XINT_float_N\expandafter
   {\the\numexpr\xintLength{#6}\expandafter}\expandafter
   {\the\numexpr #3\expandafter}\expandafter
   {\the\numexpr #1+#5}%
   {#6}{#3}{#2}{#4}%
}% long de B, P+2, n', B, |A'|=P+2, A', P
\def\XINT_float_Mc #1#2#3#4#5#6%
{%
   \expandafter\XINT_float_N\expandafter
   {\romannumeral0\xintlength{#6}}{#2}{#5}{#6}{#1}{#3}{#4}%
}% long de B, P+2, n, B, |A|, A, P
\def\XINT_float_N #1#2%
{%
    \ifnum #1>#2
      \expandafter\XINT_float_O
    \else
      \expandafter\XINT_float_P
    \fi {#1}{#2}%
}%
\def\XINT_float_O #1#2#3#4%
{%
    \expandafter\XINT_float_P\expandafter
    {\the\numexpr #2\expandafter}\expandafter
    {\the\numexpr #2\expandafter}\expandafter
    {\the\numexpr #3-#1+#2\expandafter\expandafter\expandafter}%
    \expandafter\expandafter\expandafter
    {\expandafter\xint_firstoftwo
     \romannumeral0\XINT_split_fromleft_loop {#2}{}#4\W\W\W\W\W\W\W\W\Z
     }%
}% |B|,P+2,n,B,|A|,A,P
\def\XINT_float_P #1#2#3#4#5#6#7#8%
{%
    \expandafter #8\expandafter {\the\numexpr #1-#5+#2-\xint_c_i}%
    {#6}{#4}{#7}{#3}%
}% |B|-|A|+P+1,A,B,P,n
\def\XINT_float_Q #1%
{%
    \ifnum #1<\xint_c_
      \expandafter\XINT_float_Ri
```

```
    \else
      \expandafter\XINT_float_Rii
    \fi {#1}%
}%
\def\XINT_float_Ri #1#2#3%
{%
    \expandafter\XINT_float_Sa
    \romannumeral0\xintiiquo {#2}%
         {\XINT_dsx_addzerosnofuss {-#1}{#3}}\Z {#1}%
}%
\def\XINT_float_Rii #1#2#3%
{%
    \expandafter\XINT_float_Sa
    \romannumeral0\xintiiquo
         {\XINT_dsx_addzerosnofuss {#1}{#2}}{#3}\Z {#1}%
}%
\def\XINT_float_Sa #1%
{%
    \if #19%
        \xint_afterfi {\XINT_float_Sb\XINT_float_Wb }%
    \else
        \xint_afterfi {\XINT_float_Sb\XINT_float_Wa }%
    \fi #1%
}%
\def\XINT_float_Sb #1#2\Z #3#4%
{%
    \expandafter\XINT_float_T\expandafter
    {\the\numexpr #4+\xint_c_i\expandafter}%
    \romannumeral-`0\XINT_lenrord_loop 0{}#2\Z\W\W\W\W\W\W\Z #1{#3}{#4}%
}%
\def\XINT_float_T #1#2#3%
{%
    \ifnum #2>#1
      \xint_afterfi{\XINT_float_U\XINT_float_Xb}%
    \else
      \xint_afterfi{\XINT_float_U\XINT_float_Xa #3}%
    \fi
}%
\def\XINT_float_U #1#2%
{%
    \ifnum #2<\xint_c_v
      \expandafter\XINT_float_Va
    \else
      \expandafter\XINT_float_Vb
    \fi #1%
}%
\def\XINT_float_Va #1#2\Z #3%
{%
    \expandafter#1%
    \romannumeral0\expandafter\XINT_float_Wa
    \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 \Z
}%
\def\XINT_float_Vb #1#2\Z #3%
{%
    \expandafter #1%
    \romannumeral0\expandafter #3%
    \romannumeral0\XINT_addm_A 0{}1000\W\X\Y\Z #2000\W\X\Y\Z \Z
}%
\def\XINT_float_Wa #1{ #1.}%
\def\XINT_float_Wb #1#2%
    {\if #11\xint_afterfi{ 10.}\else\xint_afterfi{ #1.#2}\fi }%
\def\XINT_float_Xa #1\Z #2#3#4%
{%
    \expandafter\XINT_float_Y\expandafter
    {\the\numexpr #3+#4-#2}{#1}%
}%
\def\XINT_float_Xb #1\Z #2#3#4%
{%
    \expandafter\XINT_float_Y\expandafter
    {\the\numexpr #3+#4+\xint_c_i-#2}{#1}%
}%
\def\XINT_float_Y #1#2{ #2e#1}%
```

7.35 \xintPFloat

1.1

```
\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\expandafter\XINTdigits
    \romannumeral0\XINTinfloat [\XINTdigits]{#1}%
}%
\def\XINT_pfloat_opt [\xint_relax #1]%#2%
{%
    \expandafter\XINT_pfloat_a\expandafter {\the\numexpr #1\expandafter}%
    \romannumeral0\XINTinfloat [\numexpr #1\relax]%{#2}%
}%
\def\XINT_pfloat_a #1#2%
{%
    \xint_UDzerominusfork
                            #2-\XINT_pfloat_zero
                            0#2\XINT_pfloat_neg
                             0-{\XINT_pfloat_pos #2}%
```

```
    \krof {#1}%
}%
\def\XINT_pfloat_zero #1[#2]{ 0}%
\def\XINT_pfloat_neg
   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_pfloat_pos {}}%
\def\XINT_pfloat_pos #1#2#3[#4]%
{%
    \ifnum#4>0 \xint_dothis\XINT_pfloat_no\fi
    \ifnum#4>\numexpr-#2\relax \xint_dothis\XINT_pfloat_b\fi
    \ifnum#4>\numexpr-#2-\xint_c_v\relax \xint_dothis\XINT_pfloat_B\fi
    \xint_orthat\XINT_pfloat_no {#2}{#4}{#1#3}%
}%
\def\XINT_pfloat_no #1#2%
{%
 \expandafter\XINT_pfloat_no_b\expandafter{\the\numexpr #2+#1-\xint_c_i\relax}%
}%
\def\XINT_pfloat_no_b #1#2{\XINT_pfloat_no_c #2e#1}%
\def\XINT_pfloat_no_c #1{ #1.}%
\def\XINT_pfloat_b #1#2#3%
   {\expandafter\XINT_pfloat_c
    \romannumeral0\expandafter\XINT_split_fromleft_loop
    \expandafter {\the\numexpr #1+#2-\xint_c_i}#3\W\W\W\W\W\W\W\W\Z }%
\def\XINT_pfloat_c #1#2{ #1.#2}% #2 peut \^etre vide
\def\XINT_pfloat_B #1#2#3%
   {\expandafter\XINT_pfloat_C
    \romannumeral0\XINT_dsx_zeroloop {\numexpr -#1-#2}{}\Z {}#3}%
\def\XINT_pfloat_C { 0.}%
```

7.36 \XINTinFloat

1.07. Completely rewritten in 1.08a for immensely greater efficiency when the power of ten is big: previous version had some very serious bottlenecks arising from the creation of long strings of zeros, which made things such as 2^999999 completely impossible, but now even 2^999999999 with 24 significant digits is no problem! Again (slightly) improved in 1.08b.

I decide in 1.09a not to use anymore \romannumeral`-0 mais \romannumeral0 also in the float routines, for consistency of style.

Here again some inner macros used the \xintiquo with extra \xintnum overhead in 1.09a, 1.09f fixed that to use \xintiiquo for example.

1.09i added a stupid bug to \XINT_infloat_zero when it changed 0[0] to a silly 0/1[0], breaking in particular \xintFloatAdd when one of the argument is zero :(((

1.09j fixes this. Besides, for notational coherence \XINT_inFloat and \XINT_infloat have been renamed respectively \XINTinFloat and \XINTinfloat in release 1.09j.

```
\def\XINTinFloat {\romannumeral0\XINTinfloat }%
\def\XINTinfloat [#1]#2%
{%
    \expandafter\XINT_infloat_a\expandafter
    {\the\numexpr #1\expandafter}%
    \romannumeral0\XINT_infrac {#2}\XINT_infloat_Q
}%
\def\XINT_infloat_a #1#2#3% #1=P, #2=n, #3=A, #4=B
{%
    \XINT_infloat_fork #3\Z {#1}{#2}% #1 = precision, #2=n
```

```
}%
\def\XINT_infloat_fork #1%
{%
    \xint_UDzerominusfork
     #1-\XINT_infloat_zero
     0#1\XINT_infloat_J
      0-{\XINT_float_K #1}%
    \krof
}%
\def\XINT_infloat_zero #1\Z #2#3#4#5{ 0[0]}%
```

the 0[0] was stupidly changed to 0/1[0] in 1.09i, with the result that the Float addition would crash when an operand was zero.

```
\def\XINT_infloat_J {\expandafter-\romannumeral0\XINT_float_K }%
\def\XINT_infloat_Q #1%
{%
    \ifnum #1<\xint_c_
      \expandafter\XINT_infloat_Ri
    \else
      \expandafter\XINT_infloat_Rii
    \fi {#1}%
}%
\def\XINT_infloat_Ri #1#2#3%
{%
    \expandafter\XINT_infloat_S\expandafter
    {\romannumeral0\xintiiquo {#2}%
         {\XINT_dsx_addzerosnofuss {-#1}{#3}}}{#1}%
}%
\def\XINT_infloat_Rii #1#2#3%
{%
    \expandafter\XINT_infloat_S\expandafter
    {\romannumeral0\xintiiquo
         {\XINT_dsx_addzerosnofuss {#1}{#2}}{#3}}{#1}%
}%
\def\XINT_infloat_S #1#2#3%
{%
    \expandafter\XINT_infloat_T\expandafter
    {\the\numexpr #3+\xint_c_i\expandafter}%
    \romannumeral-`0\XINT_lenrord_loop 0{}#1\Z\W\W\W\W\W\W\Z
    {#2}%
}%
\def\XINT_infloat_T #1#2#3%
{%
    \ifnum #2>#1
      \xint_afterfi{\XINT_infloat_U\XINT_infloat_Wb}%
    \else
      \xint_afterfi{\XINT_infloat_U\XINT_infloat_Wa #3}%
    \fi
}%
\def\XINT_infloat_U #1#2%
{%
    \ifnum #2<\xint_c_v
```

```
      \expandafter\XINT_infloat_Va
    \else
      \expandafter\XINT_infloat_Vb
    \fi #1%
}%
\def\XINT_infloat_Va #1#2\Z
{%
    \expandafter#1%
    \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 \Z
}%
\def\XINT_infloat_Vb #1#2\Z
{%
    \expandafter #1%
    \romannumeral0\XINT_addm_A 0{}1000\W\X\Y\Z #2000\W\X\Y\Z \Z
}%
\def\XINT_infloat_Wa #1\Z #2#3%
{%
    \expandafter\XINT_infloat_X\expandafter
    {\the\numexpr #3+\xint_c_i-#2}{#1}%
}%
\def\XINT_infloat_Wb #1\Z #2#3%
{%
    \expandafter\XINT_infloat_X\expandafter
    {\the\numexpr #3+\xint_c_ii-#2}{#1}%
}%
\def\XINT_infloat_X #1#2{ #2[#1]}%
```

7.37 \xintAdd

modified in v1.1. Et aussi 25 juin pour intercepter summand nul.

```
\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\expandafter{\the\numexpr #4-#3}{#3}{#4}{#1}{#2}%
}%
\def\XINT_fadd_Aa #1%
{%
    \ifcase\XINT_cntSgn #1\Z
       \expandafter\XINT_fadd_B
    \or
       \expandafter \XINT_fadd_Ba
```

```
    \else
       \expandafter \XINT_fadd_Bb
    \fi {#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_zeroloop {#1}{}\Z {#6}}%
    {#7}{#5}{#4}[#2]%
}%
\def\XINT_fadd_Bb #1#2#3#4#5#6#7%
{%
    \expandafter\XINT_fadd_C\expandafter
        {\romannumeral0\XINT_dsx_zeroloop {-#1}{}\Z {#4}}%
    {#5}{#7}{#6}[#3]%
}%
\def\XINT_fadd_C #1#2#3%
{%
   \ifcase\romannumeral0\XINT_cmp_pre {#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?
```

7.38 \xintSub

refait dans 1.1 pour vérifier si summands nuls.

```
\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 }%
```

7.39 \xintSum

```
\def\xintSum {\romannumeral0\xintsum }%
\def\xintsum #1{\xintsumexpr #1\relax }%
\def\xintSumExpr {\romannumeral0\xintsumexpr }%
\def\xintsumexpr {\expandafter\XINT_fsumexpr\romannumeral-`0}%
\def\XINT_fsumexpr {\XINT_fsum_loop_a {0/1[0]}}%
\def\XINT_fsum_loop_a #1#2%
{%
    \expandafter\XINT_fsum_loop_b \romannumeral-`0#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}%
```

7.40 \xintMul

modif 1.1 25-juin-14 pour vérifier plus tôt si nul

```
\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]}%
```

7.41 \xintSqr

1.1 modifs comme xintMul

```
\def\xintSqr {\romannumeral0\xintsqr }%
\def\xintsqr #1{\expandafter\xint_fsqr\romannumeral0\xintraw {#1}}%
\def\xint_fsqr #1{\xint_gob_til_zero #1\XINT_fsqr_zero 0\XINT_fsqr_a #1}%
\def\xint_fsqr_a #1/#2[#3]%
{%
    \expandafter\XINT_fsqr_b
    \expandafter{\the\numexpr #3+#3\expandafter}%
    \expandafter{\romannumeral0\xintiisqr {#2}}%
    {\romannumeral0\xintiisqr {#1}}%
}%
\def\XINT_fsqr_b #1#2#3{\expandafter \XINT_fmul_e \expandafter{#3}{#1}{#2}}%
\def\XINT_fsqr_zero #1]{ 0/1[0]}%
```

7.42 \xintPow

Modified in 1.06 to give the exponent to a \numexpr.

With 1.07 and for use within the \xintexpr parser, we must allow fractions (which are integers in disguise) as input to the exponent, so we must have a variant which uses \xintNum and not only \numexpr for normalizing the input. Hence the \xintfPow here.

1.08b: well actually I think that with xintfrac.sty loaded the exponent should always be allowed to be a fraction giving an integer. So I do as for \xintFac, and remove here the duplicated. Then \xintexpr can use the \xintPow as defined here.

```
\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}}%
```

7.43 \xintFac

1.07: to be used by the \xintexpr scanner which needs to be able to apply \xintFac to a fraction which is an integer in disguise; so we use \xintNum and not only \numexpr. Je modifie cela dans 1.08b, au lieu d'avoir un \xintfFac spécialement pour \xintexpr, tout simplement j'étends \xintFac comme les autres macros, pour qu'elle utilise \xintNum.

```
\def\xintFac {\romannumeral0\xintfac }%
\def\xintfac #1%
{%
    \expandafter\XINT_fac_fork\expandafter{\the\numexpr \xintNum{#1}}%
}%
```

7.44 \xintPrd

```
\def\xintPrd {\romannumeral0\xintprd }%
\def\xintprd #1{\xintprdexpr #1\relax }%
\def\xintPrdExpr {\romannumeral0\xintprdexpr }%
\def\xintprdexpr {\expandafter\XINT_fprdexpr \romannumeral-`0}%
\def\XINT_fprdexpr {\XINT_fprod_loop_a {1/1[0]}}%
\def\XINT_fprod_loop_a #1#2%
{%
    \expandafter\XINT_fprod_loop_b \romannumeral-`0#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}%
```

7.45 \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}}%
```

7.46 \xintDivFloor

1.1

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

7.47 \xintDivTrunc

1.1. \xintttrunc rather than \xintitrunc0 in 1.1a

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

7.48 \xintDivRound

1.1

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

7.49 \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.

```
\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]}%
```

7.50 \XINTinFloatMod

Pour emploi dans xintexpr 1.1

```
\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}}}%
```

7.51 \xintIsOne

New with 1.09a. Could be more efficient. For fractions with big powers of tens, it is better to use \xintCmp{f}{1}. Restyled in 1.09i.

```
\def\xintIsOne   {\romannumeral0\xintisone }%
\def\xintisone #1{\expandafter\XINT_fracisone
                  \romannumeral0\xintrawwithzeros{#1}\Z }%
\def\XINT_fracisone #1/#2\Z
    {\if0\XINT_Cmp {#1}{#2}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%
```

7.52 \xintGeq

Rewritten completely in 1.08a to be less dumb when comparing fractions having big powers of tens.

```
\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-`0\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_addzerosnofuss {#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_addzerosnofuss {#1}{#2}}{#3}%
}%
```

7.53 \xintMax

Rewritten completely in 1.08a.

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

7.54 \xintMaxof

```
\def\xintMaxof      {\romannumeral0\xintmaxof }%
\def\xintmaxof    #1{\expandafter\XINT_maxof_a\romannumeral-`0#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-`0#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}%
```

7.55 \xintMin

Rewritten completely in 1.08a.

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

7.56 \xintMinof

```
\def\xintMinof      {\romannumeral0\xintminof }%
\def\xintminof    #1{\expandafter\XINT_minof_a\romannumeral-`0#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-`0#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}%
```

7.57 \xintCmp

Rewritten completely in 1.08a to be less dumb when comparing fractions having big powers of tens.

```
%\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-`0\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_addzerosnofuss {#1}{#3}}{#2}%
}%
\def\XINT_fcmp_Fe #1#2{\XINT_cmp_pre {#2}{#1}}%
\def\XINT_fcmp_Fn #1\Z #2#3%
{%
```

```
    \expandafter\XINT_cmp_pre\expandafter
    {\romannumeral0\XINT_dsx_addzerosnofuss {#1}{#2}}{#3}%
}%
```

7.58 \xintAbs

Simplified in 1.09i. (original macro had been written before \xintRaw)

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

7.59 \xintOpp

caution that -#1 would not be ok if #1 has [n] stuff. Simplified in 1.09i. (original macro had been written before \xintRaw)

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

7.60 \xintSgn

Simplified in 1.09i. (original macro had been written before \xintRaw)

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

7.61 \xintFloatAdd, \XINTinFloatAdd

1.07; 1.09ka improves a bit the efficieny of the coding of \XINT_FL_Add_d.

```
\def\xintFloatAdd      {\romannumeral0\xintfloatadd }%
\def\xintfloatadd    #1{\XINT_fladd_chkopt \xintfloat #1\xint_relax }%
\def\XINTinFloatAdd    {\romannumeral0\XINTinfloatadd }%
\def\XINTinfloatadd  #1{\XINT_fladd_chkopt \XINTinfloat #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]{\XINT_FL_Add {\XINTdigits+\xint_c_ii}{#2}{#3}}%
}%
\def\XINT_fladd_opt #1[\xint_relax #2]#3#4%
{%
    #1[#2]{\XINT_FL_Add {#2+\xint_c_ii}{#3}{#4}}%
}%
\def\XINT_FL_Add #1#2%
{%
    \expandafter\XINT_FL_Add_a\expandafter{\the\numexpr #1\expandafter}%
    \expandafter{\romannumeral0\XINTinfloat [#1]{#2}}%
}%
```

```
\def\XINT_FL_Add_a #1#2#3%
{%
    \expandafter\XINT_FL_Add_b\romannumeral0\XINTinfloat [#1]{#3}#2{#1}%
}%
\def\XINT_FL_Add_b #1%
{%
    \xint_gob_til_zero #1\XINT_FL_Add_zero 0\XINT_FL_Add_c #1%
}%
\def\XINT_FL_Add_c #1[#2]#3%
{%
    \xint_gob_til_zero #3\XINT_FL_Add_zerobis 0\XINT_FL_Add_d #1[#2]#3%
}%
\def\XINT_FL_Add_d #1[#2]#3[#4]#5%
{%
    \ifnum \numexpr #2-#4-#5>\xint_c_i
       \expandafter \xint_secondofthree_thenstop
    \else
       \ifnum \numexpr #4-#2-#5>\xint_c_i
              \expandafter\expandafter\expandafter\xint_thirdofthree_thenstop
       \fi
    \fi
    \xintadd {#1[#2]}{#3[#4]}%
}%
\def\XINT_FL_Add_zero 0\XINT_FL_Add_c 0[0]#1[#2]#3{#1[#2]}%
\def\XINT_FL_Add_zerobis 0\XINT_FL_Add_d #1[#2]0[0]#3{#1[#2]}%
```

7.62 \xintFloatSub, \XINTinFloatSub

1.07

```
\def\xintFloatSub {\romannumeral0\xintfloatsub }%
\def\xintfloatsub    #1{\XINT_flsub_chkopt \xintfloat #1\xint_relax }%
\def\XINTinFloatSub {\romannumeral0\XINTinfloatsub }%
\def\XINTinfloatsub #1{\XINT_flsub_chkopt \XINTinfloat #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]{\XINT_FL_Add {\XINTdigits+\xint_c_ii}{#2}{\xintOpp{#3}}}%
}%
\def\XINT_flsub_opt #1[\xint_relax #2]#3#4%
{%
    #1[#2]{\XINT_FL_Add {#2+\xint_c_ii}{#3}{\xintOpp{#4}}}%
}%
```

7.63 \xintFloatMul, \XINTinFloatMul

1.07

```
\def\xintFloatMul    {\romannumeral0\xintfloatmul}%
\def\xintfloatmul    #1{\XINT_flmul_chkopt \xintfloat #1\xint_relax }%
\def\XINTinFloatMul {\romannumeral0\XINTinfloatmul }%
\def\XINTinfloatmul #1{\XINT_flmul_chkopt \XINTinfloat #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]{\XINT_FL_Mul {\XINTdigits+\xint_c_ii}{#2}{#3}}%
}%
\def\XINT_flmul_opt #1[\xint_relax #2]#3#4%
{%
    #1[#2]{\XINT_FL_Mul {#2+\xint_c_ii}{#3}{#4}}%
}%
\def\XINT_FL_Mul #1#2%
{%
    \expandafter\XINT_FL_Mul_a\expandafter{\the\numexpr #1\expandafter}%
    \expandafter{\romannumeral0\XINTinfloat [#1]{#2}}%
}%
\def\XINT_FL_Mul_a #1#2#3%
{%
    \expandafter\XINT_FL_Mul_b\romannumeral0\XINTinfloat [#1]{#3}#2%
}%
\def\XINT_FL_Mul_b #1[#2]#3[#4]{\xintE{\xintiiMul {#1}{#3}}{#2+#4}}%
```

7.64 \xintFloatDiv, \XINTinFloatDiv

1.07

```
\def\xintFloatDiv    {\romannumeral0\xintfloatdiv}%
\def\xintfloatdiv    #1{\XINT_fldiv_chkopt \xintfloat #1\xint_relax }%
\def\XINTinFloatDiv  {\romannumeral0\XINTinfloatdiv }%
\def\XINTinfloatdiv  #1{\XINT_fldiv_chkopt \XINTinfloat #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]{\XINT_FL_Div {\XINTdigits+\xint_c_ii}{#2}{#3}}%
}%
\def\XINT_fldiv_opt #1[\xint_relax #2]#3#4%
{%
    #1[#2]{\XINT_FL_Div {#2+\xint_c_ii}{#3}{#4}}%
}%
\def\XINT_FL_Div #1#2%
{%
```

```
    \expandafter\XINT_FL_Div_a\expandafter{\the\numexpr #1\expandafter}%
    \expandafter{\romannumeral0\XINTinfloat [#1]{#2}}%
}%
\def\XINT_FL_Div_a #1#2#3%
{%
    \expandafter\XINT_FL_Div_b\romannumeral0\XINTinfloat [#1]{#3}#2%
}%
\def\XINT_FL_Div_b #1[#2]#3[#4]{\xintE{#3/#1}{#4-#2}}%
```

7.65 \xintFloatPow, \XINTinFloatPow

1.07. Release 1.09j has re-organized the core loop, and \XINT_flpow_prd sub-routine has been removed.

```
\def\xintFloatPow {\romannumeral0\xintfloatpow}%
\def\xintfloatpow #1{\XINT_flpow_chkopt \xintfloat #1\xint_relax }%
\def\XINTinFloatPow {\romannumeral0\XINTinfloatpow }%
\def\XINTinfloatpow #1{\XINT_flpow_chkopt \XINTinfloat #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_start\expandafter
                {\the\numexpr #3\expandafter}\expandafter
                {\the\numexpr \XINTdigits}{#2}{#1[\XINTdigits]}%
}%
\def\XINT_flpow_opt #1[\xint_relax #2]#3#4%
{%
   \expandafter\XINT_flpow_checkB_start\expandafter
               {\the\numexpr #4\expandafter}\expandafter
               {\the\numexpr #2}{#3}{#1[#2]}%
}%
\def\XINT_flpow_checkB_start #1{\XINT_flpow_checkB_a #1\Z }%
\def\XINT_flpow_checkB_a #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_flpow_BisZero
      0#1{\XINT_flpow_checkB_b 1}%
       0-{\XINT_flpow_checkB_b 0#1}%
    \krof
}%
\def\XINT_flpow_BisZero \Z #1#2#3{#3{1/1[0]}}%
\def\XINT_flpow_checkB_b #1#2\Z #3%
{%
    \expandafter\XINT_flpow_checkB_c \expandafter
    {\romannumeral0\xintlength{#2}}{#3}{#2}#1%
}%
\def\XINT_flpow_checkB_c #1#2%
```

```
{%
    \expandafter\XINT_flpow_checkB_d \expandafter
    {\the\numexpr \expandafter\xintLength\expandafter
                  {\the\numexpr #1*20/\xint_c_iii }+#1+#2+\xint_c_i }%
}%
\def\XINT_flpow_checkB_d #1#2#3#4%
{%
    \expandafter \XINT_flpow_a
    \romannumeral0\XINTinfloat [#1]{#4}{#1}{#2}#3%
}%
\def\XINT_flpow_a #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_flpow_zero
      0#1{\XINT_flpow_b 1}%
       0-{\XINT_flpow_b 0#1}%
    \krof
}%
\def\XINT_flpow_b #1#2[#3]#4#5%
{%
    \XINT_flpow_loopI {#5}{#2[#3]}{\romannumeral0\XINTinfloatmul [#4]}%
    {#1*\ifodd #5 1\else 0\fi}%
}%
\def\XINT_flpow_zero [#1]#2#3#4#5%
```

xint is not equipped to signal infinity, the 2^31 will provoke deliberately a number too big and arithmetic overflow in \XINT_float_Xb

```
{%
    \if #41\xint_afterfi {\xintError:DivisionByZero #5{1[2147483648]}}%
    \else \xint_afterfi {#5{0[0]}}\fi
}%
\def\XINT_flpow_loopI #1%
{%
    \ifnum #1=\xint_c_i\XINT_flpow_ItoIII\fi
    \ifodd #1
       \expandafter\XINT_flpow_loopI_odd
    \else
       \expandafter\XINT_flpow_loopI_even
    \fi
    {#1}%
}%
\def\XINT_flpow_ItoIII\fi #1\fi #2#3#4#5%
{%
    \fi\expandafter\XINT_flpow_III\the\numexpr #5\relax #3%
}%
\def\XINT_flpow_loopI_even #1#2#3%
{%
    \expandafter\XINT_flpow_loopI\expandafter
    {\the\numexpr #1/\xint_c_ii\expandafter}\expandafter
    {#3{#2}{#2}}{#3}%
}%
\def\XINT_flpow_loopI_odd #1#2#3%
```

```
{%
    \expandafter\XINT_flpow_loopII\expandafter
    {\the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter}\expandafter
    {#3{#2}{#2}}{#3}{#2}%
}%
\def\XINT_flpow_loopII #1%
{%
    \ifnum #1 = \xint_c_i\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%
{%
    \expandafter\XINT_flpow_loopII\expandafter
    {\the\numexpr #1/\xint_c_ii\expandafter}\expandafter
    {#3{#2}{#2}}{#3}%
}%
\def\XINT_flpow_loopII_odd #1#2#3#4%
{%
    \expandafter\XINT_flpow_loopII_odda\expandafter
    {#3{#2}{#4}}{#1}{#2}{#3}%
}%
\def\XINT_flpow_loopII_odda #1#2#3#4%
{%
    \expandafter\XINT_flpow_loopII\expandafter
    {\the\numexpr #2/\xint_c_ii-\xint_c_i\expandafter}\expandafter
    {#4{#3}{#3}}{#4}{#1}%
}%
\def\XINT_flpow_IItoIII\fi #1\fi #2#3#4#5#6%
{%
    \fi\expandafter\XINT_flpow_III\the\numexpr #6\expandafter\relax
    #4{#3}{#5}%
}%
\def\XINT_flpow_III #1#2[#3]#4%
{%
    \expandafter\XINT_flpow_IIIend\expandafter
    {\the\numexpr\if #41-\fi#3\expandafter}%
    \xint_UDzerofork
        #4{{#2}}%
         0{{1/#2}}%
    \krof #1%
}%
\def\XINT_flpow_IIIend #1#2#3#4%
{%
    \xint_UDzerofork
    #3{#4{#2[#1]}}%
     0{#4{-#2[#1]}}%
    \krof
```

```
}%
```

7.66 \xintFloatPower, \XINTinFloatPower

1.07. The core loop has been re-organized in 1.09j for some slight efficiency gain.

```
\def\xintFloatPower {\romannumeral0\xintfloatpower}%
\def\xintfloatpower #1{\XINT_flpower_chkopt \xintfloat #1\xint_relax }%
\def\XINTinFloatPower {\romannumeral0\XINTinfloatpower}%
\def\XINTinfloatpower #1{\XINT_flpower_chkopt \XINTinfloat #1\xint_relax }%
\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_start\expandafter
                {\the\numexpr \XINTdigits\expandafter}\expandafter
                {\romannumeral0\xintnum{#3}}{#2}{#1[\XINTdigits]}%
}%
\def\XINT_flpower_opt #1[\xint_relax #2]#3#4%
{%
   \expandafter\XINT_flpower_checkB_start\expandafter
               {\the\numexpr #2\expandafter}\expandafter
               {\romannumeral0\xintnum{#4}}{#3}{#1[#2]}%
}%
\def\XINT_flpower_checkB_start #1#2{\XINT_flpower_checkB_a #2\Z {#1}}%
\def\XINT_flpower_checkB_a #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_flpower_BisZero
      0#1{\XINT_flpower_checkB_b 1}%
       0-{\XINT_flpower_checkB_b 0#1}%
    \krof
}%
\def\XINT_flpower_BisZero \Z #1#2#3{#3{1/1[0]}}%
\def\XINT_flpower_checkB_b #1#2\Z #3%
{%
    \expandafter\XINT_flpower_checkB_c \expandafter
    {\romannumeral0\xintlength{#2}}{#3}{#2}#1%
}%
\def\XINT_flpower_checkB_c #1#2%
{%
    \expandafter\XINT_flpower_checkB_d \expandafter
    {\the\numexpr \expandafter\xintLength\expandafter
                  {\the\numexpr #1*20/\xint_c_iii }+#1+#2+\xint_c_i }%
}%
\def\XINT_flpower_checkB_d #1#2#3#4%
{%
    \expandafter \XINT_flpower_a
```

```
    \romannumeral0\XINTinfloat [#1]{#4}{#1}{#2}#3%
}%
\def\XINT_flpower_a #1%
{%
    \xint_UDzerominusfork
      #1-\XINT_flpow_zero
      0#1{\XINT_flpower_b 1}%
       0-{\XINT_flpower_b 0#1}%
    \krof
}%
\def\XINT_flpower_b #1#2[#3]#4#5%
{%
    \XINT_flpower_loopI {#5}{#2[#3]}{\romannumeral0\XINTinfloatmul [#4]}%
    {#1*\xintiiOdd {#5}}%
}%
\def\XINT_flpower_loopI #1%
{%
    \if1\XINT_isOne {#1}\XINT_flpower_ItoIII\fi
    \if1\xintiiOdd{#1}%
       \expandafter\expandafter\expandafter\XINT_flpower_loopI_odd
    \else
       \expandafter\expandafter\expandafter\XINT_flpower_loopI_even
    \fi
    \expandafter {\romannumeral0\xinthalf{#1}}%
}%
\def\XINT_flpower_ItoIII\fi #1\fi\expandafter #2#3#4#5%
{%
    \fi\expandafter\XINT_flpow_III \the\numexpr #5\relax #3%
}%
\def\XINT_flpower_loopI_even #1#2#3%
{%
    \expandafter\XINT_flpower_toI\expandafter {#3{#2}{#2}}{#1}{#3}%
}%
\def\XINT_flpower_loopI_odd #1#2#3%
{%
    \expandafter\XINT_flpower_toII\expandafter {#3{#2}{#2}}{#1}{#3}{#2}%
}%
\def\XINT_flpower_toI  #1#2{\XINT_flpower_loopI  {#2}{#1}}%
\def\XINT_flpower_toII #1#2{\XINT_flpower_loopII {#2}{#1}}%
\def\XINT_flpower_loopII #1%
{%
    \if1\XINT_isOne {#1}\XINT_flpower_IItoIII\fi
    \if1\xintiiOdd{#1}%
       \expandafter\expandafter\expandafter\XINT_flpower_loopII_odd
    \else
       \expandafter\expandafter\expandafter\XINT_flpower_loopII_even
    \fi
    \expandafter {\romannumeral0\xinthalf{#1}}%
}%
\def\XINT_flpower_loopII_even #1#2#3%
{%
    \expandafter\XINT_flpower_toII\expandafter
```

```
    {#3{#2}{#2}}{#1}{#3}%
}%
\def\XINT_flpower_loopII_odd #1#2#3#4%
{%
    \expandafter\XINT_flpower_loopII_odda\expandafter
    {#3{#2}{#4}}{#2}{#3}{#1}%
}%
\def\XINT_flpower_loopII_odda #1#2#3#4%
{%
    \expandafter\XINT_flpower_toII\expandafter
    {#3{#2}{#2}}{#4}{#3}{#1}%
}%
\def\XINT_flpower_IItoIII\fi #1\fi\expandafter #2#3#4#5#6%
{%
    \fi\expandafter\XINT_flpow_III\the\numexpr #6\expandafter\relax
    #4{#3}{#5}%
}%
```

7.67 \xintFloatSqrt, \XINTinFloatSqrt

1.08

```
\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
{%
    #1[\XINTdigits]{\XINT_FL_sqrt \XINTdigits {#2}}%
}%
\def\XINT_flsqrt_opt #1[\xint_relax #2]#3%
{%
    #1[#2]{\XINT_FL_sqrt {#2}{#3}}%
}%
\def\XINT_FL_sqrt #1%
{%
    \ifnum\numexpr #1<\xint_c_xviii
        \xint_afterfi {\XINT_FL_sqrt_a\xint_c_xviii}%
    \else
        \xint_afterfi {\XINT_FL_sqrt_a {#1+\xint_c_i}}%
    \fi
}%
\def\XINT_FL_sqrt_a #1#2%
{%
    \expandafter\XINT_FL_sqrt_checkifzeroorneg
    \romannumeral0\XINTinfloat [#1]{#2}%
}%
```

```
\def\XINT_FL_sqrt_checkifzeroorneg #1%
{%
    \xint_UDzerominusfork
     #1-\XINT_FL_sqrt_iszero
     0#1\XINT_FL_sqrt_isneg
      0-{\XINT_FL_sqrt_b #1}%
    \krof
}%
\def\XINT_FL_sqrt_iszero #1[#2]{0[0]}%
\def\XINT_FL_sqrt_isneg  #1[#2]{\xintError:RootOfNegative 0[0]}%
\def\XINT_FL_sqrt_b #1[#2]%
{%
    \ifodd #2
        \xint_afterfi{\XINT_FL_sqrt_c 01}%
    \else
        \xint_afterfi{\XINT_FL_sqrt_c {}0}%
    \fi
    {#1}{#2}%
}%
\def\XINT_FL_sqrt_c #1#2#3#4%
{%
    \expandafter\XINT_flsqrt\expandafter {\the\numexpr #4-#2}{#3#1}%
}%
\def\XINT_flsqrt #1#2%
{%
    \expandafter\XINT_sqrt_a
    \expandafter{\romannumeral0\xintlength {#2}}\XINT_flsqrt_big_d {#2}{#1}%
}%
\def\XINT_flsqrt_big_d #1#2%
{%
   \ifodd #2
     \expandafter\expandafter\expandafter\XINT_flsqrt_big_eB
   \else
     \expandafter\expandafter\expandafter\XINT_flsqrt_big_eA
   \fi
   \expandafter {\the\numexpr (#2-\xint_c_i)/\xint_c_ii }{#1}%
}%
\def\XINT_flsqrt_big_eA  #1#2#3%
{%
    \XINT_flsqrt_big_eA_a #3\Z {#2}{#1}{#3}%
}%
\def\XINT_flsqrt_big_eA_a #1#2#3#4#5#6#7#8#9\Z
{%
    \XINT_flsqrt_big_eA_b {#1#2#3#4#5#6#7#8}%
}%
\def\XINT_flsqrt_big_eA_b #1#2%
{%
    \expandafter\XINT_flsqrt_big_f
    \romannumeral0\XINT_flsqrt_small_e {#2001}{#1}%
}%
\def\XINT_flsqrt_big_eB #1#2#3%
{%
```

```
    \XINT_flsqrt_big_eB_a #3\Z {#2}{#1}{#3}%
}%
\def\XINT_flsqrt_big_eB_a #1#2#3#4#5#6#7#8#9%
{%
    \XINT_flsqrt_big_eB_b {#1#2#3#4#5#6#7#8#9}%
}%
\def\XINT_flsqrt_big_eB_b #1#2\Z #3%
{%
    \expandafter\XINT_flsqrt_big_f
    \romannumeral0\XINT_flsqrt_small_e {#30001}{#1}%
}%
\def\XINT_flsqrt_small_e #1#2%
{%
   \expandafter\XINT_flsqrt_small_f\expandafter
   {\the\numexpr #1*#1-#2-\xint_c_i}{#1}%
}%
\def\XINT_flsqrt_small_f #1#2%
{%
   \expandafter\XINT_flsqrt_small_g\expandafter
   {\the\numexpr (#1+#2)/(2*#2)-\xint_c_i }{#1}{#2}%
}%
\def\XINT_flsqrt_small_g #1%
{%
    \ifnum #1>\xint_c_
       \expandafter\XINT_flsqrt_small_h
    \else
       \expandafter\XINT_flsqrt_small_end
    \fi
    {#1}%
}%
\def\XINT_flsqrt_small_h #1#2#3%
{%
    \expandafter\XINT_flsqrt_small_f\expandafter
    {\the\numexpr #2-\xint_c_ii*#1*#3+#1*#1\expandafter}\expandafter
    {\the\numexpr #3-#1}%
}%
\def\XINT_flsqrt_small_end #1#2#3%
{%
    \expandafter\space\expandafter
    {\the\numexpr \xint_c_i+#3*\xint_c_x^iv-
                        (#2*\xint_c_x^iv+#3)/(\xint_c_ii*#3)}%
}%
\def\XINT_flsqrt_big_f #1%
{%
    \expandafter\XINT_flsqrt_big_fa\expandafter
    {\romannumeral0\xintiisqr {#1}}{#1}%
}%
\def\XINT_flsqrt_big_fa #1#2#3#4%
{%
    \expandafter\XINT_flsqrt_big_fb\expandafter
    {\romannumeral0\XINT_dsx_addzerosnofuss
                       {\numexpr #3-\xint_c_viii\relax}{#2}}%
```

```
    {\romannumeral0\xintiisub
      {\XINT_dsx_addzerosnofuss
           {\numexpr \xint_c_ii*(#3-\xint_c_viii)\relax}{#1}}{#4}}%
    {#3}%
}%
\def\XINT_flsqrt_big_fb #1#2%
{%
    \expandafter\XINT_flsqrt_big_g\expandafter {#2}{#1}%
}%
\def\XINT_flsqrt_big_g #1#2%
{%
    \expandafter\XINT_flsqrt_big_j
    \romannumeral0\xintiidivision
    {#1}{\romannumeral0\XINT_dbl_pos #2\R\R\R\R\R\R\Z \W\W\W\W\W\W\W }{#2}%
}%
\def\XINT_flsqrt_big_j #1%
{%
    \if0\XINT_Sgn #1\Z
        \expandafter \XINT_flsqrt_big_end_a
    \else \expandafter \XINT_flsqrt_big_k
    \fi {#1}%
}%
\def\XINT_flsqrt_big_k #1#2#3%
{%
    \expandafter\XINT_flsqrt_big_l\expandafter
    {\romannumeral0\XINT_sub_pre {#3}{#1}}%
    {\romannumeral0\xintiiadd {#2}{\romannumeral0\XINT_sqr {#1}}}%
}%
\def\XINT_flsqrt_big_l #1#2%
{%
   \expandafter\XINT_flsqrt_big_g\expandafter
   {#2}{#1}%
}%
\def\XINT_flsqrt_big_end_a #1#2#3#4#5%
{%
   \expandafter\XINT_flsqrt_big_end_b\expandafter
   {\the\numexpr -#4+#5/\xint_c_ii\expandafter}\expandafter
   {\romannumeral0\xintiisub
    {\XINT_dsx_addzerosnofuss {#4}{#3}}%
    {\xintHalf{\xintiiQuo{\XINT_dsx_addzerosnofuss {#4}{#2}}{#3}}}}%
}%
\def\XINT_flsqrt_big_end_b #1#2{#2[#1]}%
\XINT_restorecatcodes_endinput%
```

8 Package xintseries implementation

The commenting is currently (2015/09/17) very sparse.

8.1 Catcodes, ε-TEX and reload detection

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

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5    % ^^M
  \endlinechar=13 %
  \catcode123=1   % {
  \catcode125=2   % }
  \catcode64=11   % @
  \catcode35=6    % #
  \catcode44=12   % ,
  \catcode45=12   % -
  \catcode46=12   % .
  \catcode58=12   % :
  \let\z\endgroup
  \expandafter\let\expandafter\x\csname ver@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}%
  [2015/09/12 v1.1c Expandable partial sums with xint package (jfB)]%
```

8.3 \xintSeries

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\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

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\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-`0#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-`0#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-`0#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-`0#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}%
}%
```

```
386 \XINT_restorecatcodes_endinput%
```

9 Package xintcfrac implementation

The commenting is currently (2015/09/17) very sparse. Release 1.09m (2014/02/26) has modified a few things: \xintFtoCs and \xintCntoCs insert spaces after the commas, \xintCstoF and \xintCstoCv authorize spaces in the input also before the commas, \xintCntoCs does not brace the produced coefficients, new macros \xintFtoC, \xintCtoF, \xintCtoCv, \xintFGtoC, and \xintGGCFrac.

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}%
  [2015/09/12 v1.1c 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-`0%
}%
\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-`0%
}%
\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-`0#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-`0#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 {#2}{#6}}{\XINT_Mul {#1}{#4}}}%
    {\romannumeral0\xintiiadd {\XINT_Mul {#2}{#5}}{\XINT_Mul {#1}{#3}}}%
}%
\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-`0#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-`0#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 {#1}{#3}}}%
    {\romannumeral0\xintiiadd {#4}{\XINT_Mul {#1}{#2}}}%
    {#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-`0#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 {#2}{#6}}{\XINT_Mul {#1}{#4}}}%
    {\romannumeral0\xintiiadd {\XINT_Mul {#2}{#5}}{\XINT_Mul {#1}{#3}}}%
}%
\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-`0#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-`0#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 {#1}{#3}}}%
    {\romannumeral0\xintiiadd {#4}{\XINT_Mul {#1}{#2}}}%
    {#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-`0#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-`0#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 {#2}{#6}}{\XINT_Mul {#1}{#4}}}%
    {\romannumeral0\xintiiadd {\XINT_Mul {#2}{#5}}{\XINT_Mul {#1}{#3}}}%
}%
\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-`0#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-`0#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 {#1}{#3}}}%
    {\romannumeral0\xintiiadd {#4}{\XINT_Mul {#1}{#2}}}%
    {{#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-`0#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 {#2}{#6}}{\XINT_Mul {#1}{#4}}}%
    {\romannumeral0\xintiiadd {\XINT_Mul {#2}{#5}}{\XINT_Mul {#1}{#3}}}%
}%
\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-`0#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-`0#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 {#1}{#3}}}%
    {\romannumeral0\xintiiadd {#4}{\XINT_Mul {#1}{#2}}}%
    {{#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-`0#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-`0#2{#1}}{#2}}%
   \else
      \xint_afterfi
         {\ifnum #1=\xint_c_
              \xint_afterfi {\expandafter\space \romannumeral-`0#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-`0#2{#1}}{#2}{#3}}%
   \else
      \xint_afterfi
         {\ifnum #1=\xint_c_
              \xint_afterfi {\expandafter\space\romannumeral-`0#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-`0#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-`0#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-`0#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-`0#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-`0#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-`0#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-`0#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-`0#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-`0#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-`0#2}{#3}{#1}%
}%
\def\XINT_gctgc_loop_b #1#2%
{%
    \expandafter\XINT_gctgc_loop_c\expandafter
    {\romannumeral-`0#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

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].

Release 1.1 [2014/10/28] has made many extensions, some bug fixes, and some breaking changes:

bug fixes
- \xintiiexpr did not strip leading zeroes,
- \xinttheexpr \xintiexpr 1.23\relax\relax should have produced 1, but it produced 1.23
- the catcode of ; was not set at package launching time.

breaking changes
- in \xintiiexpr, / does *rounded* division, rather than the Euclidean division (for positive arguments, this is truncated division). The new // operator does truncated division,
- the : operator for three-way branching is gone, replaced with ??,
- 1e(3+5) is now illegal. The number parser identifies e and E in the same way it does for the decimal mark, earlier versions treated e as E rather as postfix operators,
- the add and mul have a new syntax, old syntax is with `+` and `*` (quotes mandatory), sum and prd are gone,
- no more special treatment for encountered brace pairs {..} by the number scanner, a/b[N] notation can be used without use of braces (the N will end up as is in a \numexpr, it is not parsed by the \xintexpr-ession scanner).
- although & and | are still available as Boolean operators the use of && and || is strongly recommended. The single letter operators might be assigned some other meaning in later releases (bitwise operations, perhaps). Do not use them.
- place holders for \xintNewExpr could be denoted #1, #2, ... or also, for special purposes $1, $2, ... Only the first form is now accepted and the special cases previously treated via the second form are now managed via a protect(...) function.

novelties They are quite a few.
- \xintiexpr, \xinttheiexpr admit an optional argument within brackets [d], they round the computation result (or results, if comma separated) to d digits after decimal mark, (the whole computation is done exactly, as in xintexpr),

- \xintfloatexpr, \xintthefloatexpr similarly admit an optional argument which serves to keep only d digits of precision, getting rid of cumulated uncertainties in the last digits (the whole computation is done according to the precision set via \xintDigits),
- \xinttheexpr and \xintthefloatexpr ''pretty-print'' if possible, the former removing unit denominator or [0] brackets, the latter avoiding scientific notation if decimal notation is practical,
- the // does truncated division and /: is the associated modulo,
- multi-character operators &&, ||, ==, <=, >=, !=, **,
- multi-letter infix binary words 'and', 'or', 'xor', 'mod' (quotes mandatory),
- functions even, odd,
- \xintdefvar A3:=3.1415; for variable definitions (non expandable, naturally), usable in subsequent expressions; variable names may contain letters, digits, underscores. They should not start with a digit, the @ is reserved, and single lowercase and uppercase Latin letters are predefined to work as dummy variables (see next),
- generation of comma separated lists a..b, a..[d]..b,
- Python syntax-like list extractors [list][n:], [list][:n], [list][a:b] allowing negative indices, but no optional step argument, and [list][n] (n=0 for the number of items in the list),
- functions first, last, reversed,
- itemwise operations on comma separated lists a*[list], etc.., possible on both sides a*[list]^b, an obeying the same precedence rules as with numbers,
- add and mul must use a dummy variable: add(x(x+1)(x-1), x=-10..10),
- variable substitutions with subs: subs(subs(add(x^2+y^2,x=1..y),y=t),t=20),
- sequence generation using seq with a dummy variable: seq(x^3, x=-10..10),
- simple recursive lists with rseq, with @ given the last value, rseq(1;2@+1,i=1..10),
- higher recursion with rrseq, @1, @2, @3, @4, and @@(n) for earlier values, up to n=K where K is the number of terms of the initial stretch rrseq(0,1;@1+@2,i=2..100),
- iteration with iter which is like rrseq but outputs only the last K terms, where K was the number of initial terms,
- inside seq, rseq, rrseq, iter, possibility to use omit, abort and break to control termination,
- n++ potentially infinite index generation for seq, rseq, rrseq, and iter, it is advised to use abort or break(..) at some point,
- the add, mul, seq, ... are nestable,
- \xintthecoords converts a comma separated list of an even number of items to the format as expected by the TikZ coordinates syntax,
- completely rewritten \xintNewExpr, protect function to handle external macros. However not all constructs are compatible with \xintNewExpr.

Comments dating back to earlier releases:

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.

1.08b [2013/06/14] corrected a problem originating in the attempt to attribute a special rôle to braces: expansion could be stopped by space tokens, as various macros tried to expand without grabbing what came next. They now have a doubled \romannumeral-`0.

1.09a [2013/09/24] has a better mechanism regarding \xintthe, more commenting and better organization of the code, and most importantly it implements functions, comparison operators, logic operators, conditionals. The code was reorganized and expansion proceeds a bit differently in order to have the _getnext and _getop codes entirely shared by \xintexpr and \xintfloatexpr. \xintNewExpr was rewritten in order to work with the standard macro parameter character #, to be catcode protected and to also allow comma separated expressions.

1.09c [2013/10/09] added the bool and togl operators, \xintboolexpr, and \xintNewNumExpr, \xintNewBoolExpr. The code for \xintNewExpr is shared with float, num, and bool-expressions. Also the precedence level of the postfix operators !, ? and : has been made lower than the one of functions.

1.09i [2013/12/18] unpacks count and dimen registers and control squences, with tacit multiplication. It has also made small improvements. (speed gains in macro expansions in quite a few places.)

Also, 1.09i implements \xintiiexpr, \xinttheiiexpr. New function frac. And encapsulation in \csname..\endcsname is done with .= as first tokens, so unpacking with \string can be done in a completely escape char agnostic way.

1.09j [2014/01/09] extends the tacit multiplication to the case of a sub \xintexpr-essions. Also, it now \xint_protects the result of the \xintexpr full expansions, thus, an \xintexpr without \xintthe prefix can be used not only as the first item within an ``\fdef'' as previously but also now anywhere within an \edef. Five tokens are used to pack the computation result rather than the possibly hundreds or thousands of digits of an \xintthe unlocked result. I deliberately omit a second \xint_protect which, however would be necessary if some macro \.=digits/digits[digits] had acquired some expandable meaning elsewhere. But this seems not that probable, and adding the protection would mean impacting everything only to allow some crazy user which has loaded something else than xint to do an \edef... the \xintexpr computations are otherwise in no way affected if such control sequences have a meaning.

1.09k [2014/01/21] does tacit multiplication also for an opening parenthesis encountered during the scanning of a number, or at a time when the parser expects an infix operator.

And it adds to the syntax recognition of hexadecimal numbers starting with a ", and having possibly a fractional part (except in \xintiiexpr, naturally).

1.09kb [2014/02/13] fixes the bug introduced in \xintNewExpr in 1.09i of December 2013: an \endlinechar -1 was removed, but without it there is a spurious trailing space token in the outputs of the created macros, and nesting is then impossible.

This is release 1.1c of [2015/09/12].

10.1 Catcodes, ε-TEX and reload detection

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

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

10.2 Package identification

```
\XINT_providespackage
\ProvidesPackage{xintexpr}%
  [2015/09/12 v1.1c Expandable expression parser (jfB)]%
```

10.3 Locking and unlocking

je dois réfléchir si je dois bloquer expansion après unlock_a, à cause de nil.

```
\def\xint_gob_til_! #1!{}% this ! has catcode 11
\edef\XINT_expr_lockscan#1!{\noexpand\expandafter\space\noexpand\csname .=#1\endcsname }%
\edef\XINT_expr_lockit   #1{\noexpand\expandafter\space\noexpand\csname .=#1\endcsname }%
\def\XINT_expr_inintpart  #1!{\XINT_num{#1}}%
\def\XINT_expr_infracpart #1e#2!{#1![\the\numexpr#2-\xintLength{#1}]!}%
\def\XINT_expr_inexppart  e#1!{![\the\numexpr #1]!}%
\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
```

10.4 \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 }%
```

10.5 \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!}%
```

10.6 \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}}%
```

10.7 \xintexpr, \xintiexpr, \xintfloatexpr, \xintiiexpr, \xinttheexpr, etc...

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

```
\def\xintiiexpr     {\romannumeral0\xintiieval    }%
\def\xinttheexpr
   {\romannumeral-`0\expandafter\XINT_expr_print\romannumeral0\xintbareeval  }%
\def\xintthеiexpr     {\romannumeral-`0\xintthe\xintiexpr }%
\def\xintthefloatexpr {\romannumeral-`0\xintthe\xintfloatexpr }%
\def\xintthеiiexpr
   {\romannumeral-`0\expandafter\XINT_iiexpr_print\romannumeral0\xintbareiieval }%
```

10.8 \xintthe

```
\def\xintthe #1{\romannumeral-`0\expandafter\xint_gobble_iii\romannumeral-`0#1}%
```

10.9 \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 accustome users to too loose requirements!

```
\def\xintthecoords  #1{\romannumeral-`0\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-`0#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^{}%
```

10.10 \xintbareeval, \xintbarefloateval, \xintbareiieval

```
\def\xintbareeval
   {\expandafter\XINT_expr_until_end_a\romannumeral-`0\XINT_expr_getnext }%
\def\xintbarefloateval
   {\expandafter\XINT_flexpr_until_end_a\romannumeral-`0\XINT_expr_getnext }%
\def\xintbareiieval
   {\expandafter\XINT_iiexpr_until_end_a\romannumeral-`0\XINT_expr_getnext }%
```

10.11 \xinteval, \xintiieval

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

10.12 \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
}%
```

10.13 \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-`0\expandafter\XINT_expr_unlock_sp\string #1!%
}%
\catcode`: 12
    \def\XINT_expr_unlock_sp #1.;#2.=#3!{{#2}{#3}}%
\catcode`: 11
```

10.14 \xintboolexpr, \xinttheboolexpr

```
\def\xintboolexpr      {\romannumeral0\expandafter\expandafter\expandafter
    \XINT_boolexpr_done \expandafter\xint_gobble_iv\romannumeral0\xinteval }%
\def\xinttheboolexpr   {\romannumeral-`0\expandafter\expandafter\expandafter
    \XINT_boolexpr_print\expandafter\xint_gobble_iv\romannumeral0\xinteval }%
```

```
\def\XINT_boolexpr_done { !\XINT_expr_usethe\XINT_protectii\XINT_boolexpr_print }%
```

10.15 \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}}%
```

10.16 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.

10.16.1 \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}%
```

10.16.2 \xintCSV::csv

pour \xinttheiiexpr. 1.1a adds the \romannumeral-`0 for each item, which have no use for \xintiiexpr etc..., but are necessary for \xintNewExpr to be able to handle comma separated inputs. I am not sure but I think I had them just prior to releasing 1.1 but removed them foolishsly.

```
\def\xintCSV::csv #1{\expandafter\XINT_csv::_a\romannumeral-`0#1,^,}%
\def\XINT_csv::_a {\XINT_csv::_b {}}%
\def\XINT_csv::_b #1#2,{\expandafter\XINT_csv::_c \romannumeral-`0#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.
```

10.16.3 \xintSPRaw, \xintSPRaw::csv

Pour \xinttheexpr. J'avais voulu optimiser en testant si présence ou non de [N], cependant reduce() produit résultat sans, et du coup, le /1 peut ne pas être retiré. Bon je rajoute un [0] dans reduce. 14/10/25 au moment de boucler.
  Same added \romannumeral-`0 in 1.1a for \xintNewExpr purposes.

```
\def\xintSPRaw    {\romannumeral0\xintspraw }%
\def\xintspraw  #1{\expandafter\XINT_spraw\romannumeral-`0#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-`0#1,^,}%
\def\XINT_spraw::_a {\XINT_spraw::_b {}}%
\def\XINT_spraw::_b #1#2,{\expandafter\XINT_spraw::_c \romannumeral-`0#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}}%
```

10.16.4 \xintIsTrue::csv

```
\def\xintIsTrue::csv #1{\romannumeral0\expandafter\XINT_istrue::_a\romannumeral-`0#1,^,}%
\def\XINT_istrue::_a {\XINT_istrue::_b {}}%
\def\XINT_istrue::_b #1#2,{\expandafter\XINT_istrue::_c \romannumeral-`0#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}}%
```

10.16.5 \xintRound::csv

Pour \xintiexpr avec argument optionnel (finalement, malgré un certain overhead lors de l'exécution, pour économiser du code je ne distingue plus les deux cas). Reason for annoying expansion bridge is related to \xintNewExpr. Attention utilise \XINT_:::_end.

```
\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-`0#2,^,}%
\def\XINT_round::_b #1#2#3,{\expandafter\XINT_round::_c \romannumeral-`0#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}}%
```

10.16.6 \XINTinFloat::csv

Pour \xintfloatexpr. Attention, prépare sous la forme digits[N] pour traitement par les macros. Pas utilisé en sortie. Utilise \XINT_:::_end.
  1.1a I believe this is not needed for \xintNewExpr, as it is removed by re-defined by \XINT_flexpr_wrap code, hence no need to add the extra \romannumeral-`0. Sub-expressions in \xintNewExpr are not supported.
  I didn't start and don't want now to think about it at all.

```
\def\XINTinFloat::csv #1#2{\romannumeral0\expandafter\XINT_infloat::_b\expandafter
   {\the\numexpr #1\expandafter}\expandafter{\expandafter}\romannumeral-`0#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}}%
```

10.16.7 \xintPFloat::csv

Expansion à cause de \xintNewExpr. Attention à l'ordre, pas le même que pour \XINTinFloat::csv. Donc c'est cette routine qui imprime. Utilise \XINT_:::_end

```
\def\xintPFloat::csv #1#2{\romannumeral0\expandafter\XINT_pfloat::_b\expandafter
```

```
   {\the\numexpr #1\expandafter}\expandafter{\expandafter}\romannumeral-`0#2,^,}%
\def\XINT_pfloat::_b #1#2#3,{\expandafter\XINT_pfloat::_c \romannumeral-`0#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}}%
```

10.17 \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-`0#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 }%
\def\XINT_expr_countetc #1%
{%
    \ifx\count#1\else\ifx#1\dimen\else\ifx#1\numexpr\else\ifx#1\dimexpr\else
    \ifx\skip#1\else\ifx\glueexpr#1\else\ifx\fontdimen#1\else
      \XINT_expr_unpackvar
    \fi\fi\fi\fi\fi\fi\fi
    \expandafter\XINT_expr_getnext\number #1%
}%
\def\XINT_expr_unpackvar\fi\fi\fi\fi\fi\fi\fi\expandafter\XINT_expr_getnext\number #1%
    {\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_scandec_II\XINT_expr_infracpart}\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 }%
```

10.18 The integer or decimal number or hexa-decimal number or function name or variable name or special hacky things big parser

```
\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_scandec_I\fi
    \xint_orthat \XINT_expr_scanfunc #1%
}%
\catcode96 12 % `
\def\XINT_expr_scandec_I
{%
    \expandafter\XINT_expr_getop\romannumeral-`0\expandafter
    \XINT_expr_lockscan\romannumeral0\expandafter\XINT_expr_inintpart
    \romannumeral-`0\XINT_expr_scanintpart_b
}%
\def\XINT_expr_scandec_II
{%
    \expandafter\XINT_expr_getop\romannumeral-`0\expandafter
    \XINT_expr_lockscan\romannumeral0\expandafter\XINT_expr_inintpart
    \romannumeral-`0\XINT_expr_scanfracpart_b
}%
```

10.18.1 Integral part

```
\def\XINT_expr_scanintpart_a #1%
{% careful that ! has catcode letter here
    \ifcat \relax #1\xint_dothis{!!#1}\fi % stops the scan
    \if    e#1\xint_dothis{\expandafter\XINT_expr_inexppart
                  \romannumeral-`0\XINT_expr_scanexppart_a e}\fi
    \if    E#1\xint_dothis{\expandafter\XINT_expr_inexppart
                  \romannumeral-`0\XINT_expr_scanexppart_a e}\fi
```

```
\if @#1\xint_dothis{!*#1}\fi tacit multiplication later

\if _#1\xint_dothis{!*#1}\fi tacit multiplication for variables
```

```
    \ifcat a#1\xint_dothis{!!*#1}\fi % includes subexpressions (#1=! letter)
    \xint_orthat {\expandafter\XINT_expr_scanintpart_aa\string #1}%
}%
\def\XINT_expr_scanintpart_aa #1%
{%
    \if .#1\xint_dothis\XINT_expr_scandec_transition\fi
    \ifnum \xint_c_ix<1#1 \xint_dothis\XINT_expr_scanintpart_b\fi
    \xint_orthat {!!}#1%
}%
\def\XINT_expr_scanintpart_b #1#2%
{%
    \expandafter #1\romannumeral-`0\expandafter
    \XINT_expr_scanintpart_a\romannumeral-`0#2%
}%
\def\XINT_expr_scandec_transition .#1%
{%
```

```
    \expandafter\XINT_expr_scandec_trans_a\romannumeral-`0#1%
}%
\def\XINT_expr_scandec_trans_a #1%
{%
    \if .#1\xint_dothis{!!..}\fi
    \xint_orthat {\expandafter\XINT_expr_infracpart
                  \romannumeral-`0\XINT_expr_scanfracpart_a #1}%
}%
```

10.18.2 Fractional part

```
\def\XINT_expr_scanfracpart_a #1%
{%
    \ifcat \relax #1\xint_dothis{e!#1}\fi % stops the scan
    \if    e#1\xint_dothis{\XINT_expr_scanexppart_a e}\fi
    \if    E#1\xint_dothis{\XINT_expr_scanexppart_a e}\fi
    \ifcat a#1\xint_dothis{e!*#1}\fi % and also the case of subexpressions (!)
    \xint_orthat {\expandafter\XINT_expr_scanfracpart_aa\string #1}%
}%
\def\XINT_expr_scanfracpart_aa #1%
{%
    \ifnum \xint_c_ix<1#1
       \expandafter\XINT_expr_scanfracpart_b
    \else
       \xint_afterfi {e!}%
    \fi
    #1%
}%
\def\XINT_expr_scanfracpart_b #1#2%
{%
    \expandafter #1\romannumeral-`0\expandafter
    \XINT_expr_scanfracpart_a\romannumeral-`0#2%
}%
```

10.18.3 Scientific notation

```
\def\XINT_expr_scanexppart_a #1#2%
{%
    \expandafter #1\romannumeral-`0\expandafter
    \XINT_expr_scanexppart_b\romannumeral-`0#2%
}%
\def\XINT_expr_scanexppart_b #1%
{%
    \ifcat \relax #1\xint_dothis{0!#1}\fi % stops the scan (incorrect syntax)
    \ifcat a#1\xint_dothis{0!*#1}\fi % idem
    \if    +#1\xint_dothis {\XINT_expr_scanexppart_a +}\fi
    \if    -#1\xint_dothis {\XINT_expr_scanexppart_a -}\fi
    \xint_orthat {\expandafter\XINT_expr_scanexppart_c\string #1}%
}%
\def\XINT_expr_scanexppart_c #1%
{%
    \ifnum \xint_c_ix<1#1
       \expandafter\XINT_expr_scanexppart_d
    \else
       \expandafter !%
```

```
    \fi
    #1%
}%
\def\XINT_expr_scanexppart_d #1#2%
{%
    \expandafter #1\romannumeral-`0\expandafter
    \XINT_expr_scanexppart_e\romannumeral-`0#2%
}%
\def\XINT_expr_scanexppart_e #1%
{%
    \ifcat \relax #1\xint_dothis{!#1}\fi % stops the scan
    \ifcat a#1\xint_dothis{!*#1}\fi % idem
    \xint_orthat {\expandafter\XINT_expr_scanexppart_f\string #1}%
}%
\def\XINT_expr_scanexppart_f #1%
{%
    \ifnum \xint_c_ix<1#1
       \expandafter\XINT_expr_scanexppart_d
    \else
       \expandafter !%
    \fi
    #1%
}%
```

10.18.4 Hexadecimal numbers

```
\def\XINT_expr_scanhex_I #1%
{%
    \expandafter\XINT_expr_getop\romannumeral-`0\expandafter
    \XINT_expr_lockscan\expandafter\XINT_expr_inhex
    \romannumeral-`0\XINT_expr_scanhexI_a
}%
\def\XINT_expr_inhex #1.#2#3;% expanded inside \csname..\endcsname
{%
    \if#2I\xintHexToDec{#1}%
    \else
      \xintiiMul{\xintiiPow{625}{\xintLength{#3}}}{\xintHexToDec{#1#3}}%
      [\the\numexpr-4*\xintLength{#3}]%
    \fi
}%
\def\XINT_expr_scanhexI_a #1%
{%
    \ifcat #1\relax\xint_dothis{.I;!#1}\fi
    \ifx        !#1\xint_dothis{.I;!*!}\fi % tacit multiplication
    \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 {.I;!}}%
    \fi
    #1%
}%
\def\XINT_expr_scanhexI_b #1#2%
{%
    \expandafter #1\romannumeral-`0\expandafter
    \XINT_expr_scanhexI_a\romannumeral-`0#2%
}%
\def\XINT_expr_scanhex_transition .#1%
{%
    \expandafter.\expandafter.\romannumeral-`0\expandafter
    \XINT_expr_scanhexII_a\romannumeral-`0#1%
}%
\def\XINT_expr_scanhexII_a #1%
{%
    \ifcat #1\relax\xint_dothis{;!#1}\fi
    \ifx        !#1\xint_dothis{;!*!}\fi % tacit multiplication
    \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 {;!}%
    \fi
    #1%
}%
\def\XINT_expr_scanhexII_b #1#2%
{%
    \expandafter #1\romannumeral-`0\expandafter
    \XINT_expr_scanhexII_a\romannumeral-`0#2%
}%
```

10.18.5 Function and variable names

```
\def\XINT_expr_scanfunc
{%
    \expandafter\XINT_expr_func\romannumeral-`0\XINT_expr_scanfunc_a
}%
\def\XINT_expr_scanfunc_a #1#2%
{%
```

```
    \expandafter #1\romannumeral-`0\expandafter\XINT_expr_scanfunc_b\romannumeral-`0#2%
}%
\def\XINT_expr_scanfunc_b #1%
{%
  \ifx !#1\xint_dothis{\xint_firstoftwo{(_*!}}\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%
}%
\def\XINT_expr_func #1(#2%
{% #2=` pour une fonction, #2=_ pour une variable
   \if #2`\ifcsname XINT_expr_var_#1\endcsname
          \expandafter\expandafter\expandafter\xint_thirdofthree
   \fi\fi
   \xint_firstoftwo {\xint_c_xviii #2{#1}}{\xint_c_xviii _{#1}*(}%
}%
```

10.19 \XINT_expr_getop: finding the next operator or closing parenthesis or end of expression

Release 1.1 implements multi-character operators.

```
\def\XINT_expr_getop #1#2% this #1 is the current locked computed value
{%
    \expandafter\XINT_expr_getop_a\expandafter #1\romannumeral-`0#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
    \ifx   !#2\xint_dothis      \xint_secondofthree\fi
    \xint_orthat \xint_thirdofthree
    {\XINT_expr_foundend #1}%
    {\XINT_expr_precedence_* *#1#2}% tacit multiplication
    {\XINT_expr_getop_b #2#1}%
}%
\catcode`* 12
\def\XINT_expr_foundend {\xint_c_ \relax }% \relax is a place holder here.
```

? and : a special syntax in \xintexpr as they are followed by braced arguments, and thus we must intercept them here. I wanted to change this but now I don't have time to think about it. 1.1 removes : as logic operator. Replaced by ??.

```
\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-`0#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-`0#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}%
```

10.20 Opening and closing parentheses, square brackets for lists, the ^C for omit and abort within seq or rseq

```
\catcode`) 11
\def\XINT_tmpa #1#2#3#4% (avant #4#5)
{%
    \def#1##1%
    {%
        \xint_UDsignfork
                     ##1{\expandafter#1\romannumeral-`0#3}%
                       -{#2##1}%
        \krof
```

```
    }%
    \def#2##1##2%
    {%
        \ifcase ##1\xint_afterfi
        {\ifx\XINT_expr_itself_^C ##2\xint_dothis
        {\expandafter#1\romannumeral-`0\expandafter\XINT_expr_getnext\xint_gobble_i}\fi
         \xint_orthat \XINT_expr_done }%
        \or\xint_afterfi{\XINT_expr_extra_)
                         \expandafter #1\romannumeral-`0\XINT_expr_getop }%
        \else
        \xint_afterfi{\expandafter#1\romannumeral-`0\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-`0\XINT_expr_getnext }%
    \def #2{\expandafter #3\romannumeral-`0\XINT_expr_getnext }%
    \def #3##1{\xint_UDsignfork
                ##1{\expandafter #3\romannumeral-`0#5}%
                  -{#4##1}%
               \krof }%
    \def #4##1##2{\ifcase ##1%
      \xint_afterfi{\ifx\XINT_expr_itself_^C ##2\xint_dothis{\xint_c_ ##2}\fi
                    \xint_orthat\XINT_expr_missing_) }%
      \or   \csname XINT_#6_op_##2\expandafter\endcsname
      \else
      \xint_afterfi{\expandafter #3\romannumeral-`0\csname XINT_#6_op_##2\endcsname }%
      \fi
    }%
}%
\def\XINT_expr_missing_) {\xintError:inserted \xint_c_ \XINT_expr_done }%
\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
\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
\expandafter\let\csname XINT_expr_precedence_^C\endcsname \xint_c_
\expandafter\let\csname XINT_expr_precedence_++)\endcsname \xint_c_i
\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_getop ##1a}%
    \expandafter\let\csname XINT_#1_op_a\endcsname \XINT_expr_getop
    \expandafter\def\csname XINT_#1_op_++)\endcsname ##1##2\relax
  {\expandafter\XINT_expr_foundend \expandafter
                                  {\expandafter\.=+\xintiCeil{\XINT_expr_unlock ##1}}}%
}%
\catcode`. 12 \catcode`= 12 \catcode`+ 12
```

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

```
\xintFor* #1 in {{==}{<=}{>=}{!=}{&&}{||}{**}{//}{/:}{..}{..[}{].}{]..}%
                 {+[}{-[}{*[}{/[}{**[}{^[}{a+}{a-}{a*}{a/}{a**}{a^}%
                 {][}{][:}{:]}{^C}{++}{++)}}
    \do {\expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}}%
```

10.21.1 The |, &, xor, <, >, =, <=, >=, !=, //, /:, .., ..[, and].. operators

```
\def\XINT_tmpc #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-`0\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-`0#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>#5%
    \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral-`0%
      \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_tmpb #1#2#3#4#5#6%
{%
  \expandafter\XINT_tmpc
  \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_#1_op_-#5\expandafter\endcsname
  \csname xint_c_#4\expandafter\endcsname
  \csname #2#6\expandafter\endcsname
```

```
  \csname XINT_expr_precedence_#3\endcsname {#1}%
}%
\xintFor #1 in {expr, flexpr} \do {%
  \def\XINT_tmpa ##1{\XINT_tmpb {#1}{xint}##1}%
  \xintApplyInline {\XINT_tmpa }{%
    {|{iii}{vi}{OR}}%
    {&{iv}{vi}{AND}}%
    {{xor}{iii}{vi}{XOR}}%
    {<{v}{vi}{Lt}}%
    {>{v}{vi}{Gt}}%
    {={v}{vi}{Eq}}%
    {{<=}{v}{vi}{LtorEq}}%
    {{>=}{v}{vi}{GtorEq}}%
    {{!=}{v}{vi}{Neq}}%
    {{..}{iii}{vi}{Seq::csv}}%
    {{//}{vii}{vii}{DivTrunc}}%
    {{/:}{vii}{vii}{Mod}}%
  }%
}%
\def\XINT_tmpa #1{\XINT_tmpb {expr}{xint}#1}%
\xintApplyInline {\XINT_tmpa }{%
  {+{vi}{vi}{Add}}%
  {-{vi}{vi}{Sub}}%
  {*{vii}{vii}{Mul}}%
  {/{vii}{vii}{Div}}%
  {^{viii}{viii}{Pow}}%
  {{..[}{iii}{vi}{SeqA::csv}}%
  {{]..}{iii}{vi}{SeqB::csv}}%
}%
\def\XINT_tmpa #1{\XINT_tmpb {flexpr}{XINTinFloat}#1}%
\xintApplyInline {\XINT_tmpa }{%
  {+{vi}{vi}{Add}}%
  {-{vi}{vi}{Sub}}%
  {*{vii}{vii}{Mul}}%
  {/{vii}{vii}{Div}}%
  {^{viii}{viii}{Power}}%
  {{..[}{iii}{vi}{SeqA::csv}}%
  {{]..}{iii}{vi}{SeqB::csv}}%
}%
\def\XINT_tmpa #1{\XINT_tmpb {iiexpr}{xint}#1}%
\xintApplyInline {\XINT_tmpa }{%
    {|{iii}{vi}{OR}}%
    {&{iv}{vi}{AND}}%
    {{xor}{iii}{vi}{XOR}}%
    {<{v}{vi}{iiLt}}%
    {>{v}{vi}{iiGt}}%
    {={v}{vi}{iiEq}}%
    {{<=}{v}{vi}{iiLtorEq}}%
    {{>=}{v}{vi}{iiGtorEq}}%
    {{!=}{v}{vi}{iiNeq}}%
  {+{vi}{vi}{iiAdd}}%
  {-{vi}{vi}{iiSub}}%
```

```
  {*{vii}{vii}{iiMul}}%
  {/{vii}{vii}{iiDivRound}}% CHANGED IN 1.1! PREVIOUSLY DID EUCLIDEAN QUOTIENT
  {^{viii}{viii}{iiPow}}%
  {{..[}{iii}{vi}{iiSeqA::csv}}%
  {{]..}{iii}{vi}{iiSeqB::csv}}%
  {{..}{iii}{vi}{iiSeq::csv}}%
  {{//}{vii}{vii}{iiDivTrunc}}%
  {{/:}{vii}{vii}{iiMod}}%
}%
```

10.21.2 The]+,]–,]*,]/,]^, +[, –[, *[, /[, and ^[list operators

\XINT_expr_binop_inline_b

```
\def\XINT_expr_binop_inline_a
   {\expandafter\xint_gobble_i\romannumeral-`0\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_tmpc #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-`0\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-`0#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>#5%
    \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral-`0%
      \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-`0\XINT_expr_unlock ##1,^,\endcsname }%
    \fi }%
  \let #7#5%
}%
\def\XINT_tmpb #1#2#3#4%
{%
  \expandafter\XINT_tmpc
  \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}%
}%
\xintApplyInline {\expandafter\XINT_tmpb \xint_firstofone}{%
  {{expr}{a+}{vi}{xintAdd}}%
  {{expr}{a-}{vi}{xintSub}}%
  {{expr}{a*}{vii}{xintMul}}%
  {{expr}{a/}{vii}{xintDiv}}%
  {{expr}{a^}{viii}{xintPow}}%
  {{iiexpr}{a+}{vi}{xintiiAdd}}%
  {{iiexpr}{a-}{vi}{xintiiSub}}%
  {{iiexpr}{a*}{vii}{xintiiMul}}%
  {{iiexpr}{a/}{vii}{xintiiDivRound}}%
  {{iiexpr}{a^}{viii}{xintiiPow}}%
  {{flexpr}{a+}{vi}{XINTinFloatAdd}}%
  {{flexpr}{a-}{vi}{XINTinFloatSub}}%
  {{flexpr}{a*}{vii}{XINTinFloatMul}}%
  {{flexpr}{a/}{vii}{XINTinFloatDiv}}%
  {{flexpr}{a^}{viii}{XINTinFloatPower}}%
}%
\def\XINT_tmpc #1#2#3#4#5#6#7%
{%
  \def #1##1{\expandafter#2\expandafter##1\romannumeral-`0%
             \expandafter #3\romannumeral-`0\XINT_expr_getnext }%
  \def #2##1##2##3##4%
  {% either execute next operation now, or first do next (possibly unary)
    \ifnum ##2>#4%
    \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral-`0%
      \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-`0\XINT_expr_unlock ##4,^,\endcsname }%
    \fi }%
  \let #6#4%
}%
\def\XINT_tmpb #1#2#3#4%
{%
  \expandafter\XINT_tmpc
  \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}%
}%
\xintApplyInline {\expandafter\XINT_tmpb\xint_firstofone }{%
  {{expr}{+[}{vi}{xintAdd}}%
  {{expr}{-[}{vi}{xintSub}}%
  {{expr}{*[}{vii}{xintMul}}%
```

```
  {{expr}{/[]}{vii}{xintDiv}}%
  {{expr}{^[]}{viii}{xintPow}}%
  {{iiexpr}{+[]}{vi}{xintiiAdd}}%
  {{iiexpr}{-[]}{vi}{xintiiSub}}%
  {{iiexpr}{*[]}{vii}{xintiiMul}}%
  {{iiexpr}{/[]}{vii}{xintiiDivRound}}%
  {{iiexpr}{^[]}{viii}{xintiiPow}}%
  {{flexpr}{+[]}{vi}{XINTinFloatAdd}}%
  {{flexpr}{-[]}{vi}{XINTinFloatSub}}%
  {{flexpr}{*[]}{vii}{XINTinFloatMul}}%
  {{flexpr}{/[]}{vii}{XINTinFloatDiv}}%
  {{flexpr}{^[]}{viii}{XINTinFloatPower}}%
}%
```

10.21.3 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
}%
```

10.21.4 The ||, &&, **, **[,]** operators as synonyms

```
\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_||\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_&&\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
}%
```

10.21.5 List selectors: [list][N], [list][:b], [list][a:], [list][a:b]

1.1 (27 octobre 2014) I implement Python syntax, see http://stackoverflow.com/a/13005464/4184837. Do not implement third argument giving the step. Also, I gather that [5:2] selector returns empty and not, as I could have been tempted to do, (list[5], list[4], list[3]). Anyway, it is simpler not to go that way. For reversing I could implement [::-1] but this would get confusing, better to do function "reversed".

This gets the job done, but I would definitely need \xintTrim::csv, \xintKeep::csv, \xintNthElt::csv for better efficiency. Not for 1.1.

```
\def\XINT_tmpa #1#2#3#4#5#6%
{%
    \def #1##1% \XINT_expr_op_][
    {%
        \expandafter #2\expandafter ##1\romannumeral-`0\XINT_expr_getnext
    }%
    \def #2##1##2% \XINT_expr_until_][_a
    {\xint_UDsignfork
        ##2{\expandafter #2\expandafter ##1\romannumeral-`0#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-`0%
                       \csname XINT_#6_op_##3\endcsname {##4}}%
      \else
        \xint_afterfi
        {\expandafter ##2\expandafter ##3\csname
              .=\expandafter\xintListSel:csv \romannumeral-`0\XINT_expr_unlock ##4;%
                \XINT_expr_unlock ##1;\endcsname % unlock 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-`0\XINT_expr_getnext
    }%
    \def #2##1##2% \XINT_expr_until_:_a
    {\xint_UDsignfork
        ##2{\expandafter #2\expandafter ##1\romannumeral-`0#4}%
          -{#3##1##2}%
     \krof }%
    \def #3##1##2##3##4% \XINT_expr_until_:_b
    {%
      \ifnum ##2>\xint_c_iii
        \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral-`0%
                       \csname XINT_#6_op_##3\endcsname {##4}}%
      \else
        \xint_afterfi
        {\expandafter ##2\expandafter ##3\csname
              .=:\xintiiifSgn{\XINT_expr_unlock ##1}NPP.%
                \xintiiifSgn{\XINT_expr_unlock ##4}NPP.%
                \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 .=]\xintiiifSgn{\XINT_expr_unlock #1}npp\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
\edef\XINT_expr_op_][: #1{\xint_c_ii \expandafter\noexpand
                          \csname XINT_expr_itself_][\endcsname #10\string :}%
```

: must be catcode 12, else will be mistaken for start of variable by expression parser. In \xintListSel:csv some complications are due to \xintNewExpr matters.

```
\let\XINT_flexpr_op_][: \XINT_expr_op_][:
\let\XINT_iiexpr_op_][: \XINT_expr_op_][:
\catcode`[ 12 \catcode`] 12
\def\xintListSel:csv #1{%
    \if ]\noexpand#1\xint_dothis{\expandafter\XINT_listsel:_s\romannumeral-`0}\fi
    \if :\noexpand#1\xint_dothis{\XINT_listsel:_:}\fi
    \xint_orthat {\XINT_listsel:_nth #1}%
```

```
}%
\def\XINT_listsel:_s #1{\if p#1\expandafter\XINT_listsel:_trim\else
                              \expandafter\XINT_listsel:_keep\fi }%
\def\XINT_listsel:_: #1.#2.{\csname XINT_listsel:_#1#2\endcsname }%
\def\XINT_listsel:_trim #1;#2;%
   {\xintListWithSep,{\xintTrim {\xintNum{#1}}{\xintCSVtoListNonStripped{#2}}}}%
\def\XINT_listsel:_keep #1;#2;%
   {\xintListWithSep,{\xintKeep {\xintNum{#1}}{\xintCSVtoListNonStripped{#2}}}}%
\def\XINT_listsel:_nth#1;#2;%
   {\xintNthElt {\xintNum{#1}}{\xintCSVtoListNonStripped{#2}}}%
\def\XINT_listsel:_PP #1;#2;#3;%
   {\xintListWithSep,%
    {\xintTrim {\xintNum{#1}}%
               {\xintKeep {\xintNum{#2}}%
                          {\xintCSVtoListNonStripped{#3}}%
               }%
    }%
   }%
\def\XINT_listsel:_NN #1;#2;#3;%
   {\xintListWithSep,%
    {\xintTrim {\xintNum{#2}}%
               {\xintKeep {\xintNum{#1}}%
                          {\xintCSVtoListNonStripped{#3}}%
               }%
    }%
   }%
\def\XINT_listsel:_NP #1;#2;#3;%
   {\expandafter\XINT_listsel:_NP_a \the\numexpr #1+%
             \xintNthElt{0}{\xintCSVtoListNonStripped{#3}};#2;#3;}%
\def\XINT_listsel:_NP_a #1#2;{\if -#1\expandafter\XINT_listsel:_OP\fi
                              \XINT_listsel:_PP #1#2;}%
\def\XINT_listsel:_OP\XINT_listsel:_PP #1;{\XINT_listsel:_PP 0;}%
\def\XINT_listsel:_PN #1;#2;#3;%
   {\expandafter\XINT_listsel:_PN_a \the\numexpr #2+%
             \xintNthElt{0}{\xintCSVtoListNonStripped{#3}};#1;#3;}%
\def\XINT_listsel:_PN_a #1#2;#3;{\if -#1\expandafter\XINT_listsel:_PO\fi
                              \XINT_listsel:_PP #3;#1#2;}%
\def\XINT_listsel:_PO\XINT_listsel:_PP #1;#2;{\XINT_listsel:_PP #1;0;}%
```

10.22 Macros for a..b list generation

Attention, ne produit que des listes de petits entiers!

10.22.1 \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.

```
\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,{ }%
```

10.22.2 \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,{ }%
```

10.23 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.

10.23.1 \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#2{\XINT_iiseqa::csv #1#2}%
\def\XINT_iiseqa::csv #1#2#3#4{\expandafter\XINT_seqa::csv_a
    \romannumeral-`0\expandafter \XINT_expr_unlock\expandafter#4%
    \expandafter;\romannumeral-`0\XINT_expr_unlock #2;}%
\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}%
```

10.23.2 \xintSeqB::csv

```
\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-`0#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}%
```

10.23.3 `\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-`0\expandafter \XINT_expr_unlock\expandafter#2%
    \romannumeral-`0\XINT_expr_unlock #4!}%
\def\XINT_iiseqb::csv_a #1#2;#3;#4!{\expandafter\XINT_expr_seq_empty?
      \romannumeral-`0\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}%
```

10.23.4 `\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-`0#2#1!}%
\def\XINT_flseqb::csv_a #1#2;#3;#4!{\expandafter\XINT_expr_seq_empty?
      \romannumeral-`0\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}%
```

10.24 The comma as binary operator

New with 1.09a.

```
\def\XINT_tmpa #1#2#3#4#5#6%
{%
    \def #1##1% \XINT_expr_op_,
    {%
        \expandafter #2\expandafter ##1\romannumeral-`0\XINT_expr_getnext
    }%
    \def #2##1##2% \XINT_expr_until_,_a
    {\xint_UDsignfork
        ##2{\expandafter #2\expandafter ##1\romannumeral-`0#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-`0%
                       \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}%
}%
```

10.25 The minus as prefix operator of variable precedence level

```
\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-`0\XINT_expr_getnext
    }%
    \def #2##1% \XINT_expr_until_-<l>_a
    {\xint_UDsignfork
        ##1{\expandafter #2\romannumeral-`0#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-`0%
                        \csname XINT_#5_op_##2\endcsname {##3}}%
        \else
         \xint_afterfi {\expandafter ##1\expandafter ##2%
                        \csname .=#6{\XINT_expr_unlock ##3}\endcsname }%
        \fi
    }%
}%
\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}}%
```

10.26 ? 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. ?{yes}{no} and ??{<0}{=0}{>0}

```
\let\XINT_expr_precedence_? \xint_c_x
\def\XINT_expr_op_? #1#2{\if ?#2\expandafter \XINT_expr_op_??\fi
                          \XINT_expr_op_?a #1{#2}}%
\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_?? #1#2#3#4#5#6%
{%
     \xintiiifSgn {\XINT_expr_unlock  #2}{\XINT_expr_getnext #4}{\XINT_expr_getnext #5}%
                  {\XINT_expr_getnext #6}%
}%
```

10.27 ! as postfix factorial operator

As of 2014/11/07, not yet a float version of factorial. I must do it!

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

```
\def\XINT_iiexpr_op_! #1{\expandafter\XINT_expr_getop
                                    \csname .=\xintiFac{\XINT_expr_unlock #1}\endcsname }%
```

10.28 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* 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 CRAZYNESS OF NUMEXPR: \the\numexpr 3 + 7 9 \relax !! Hence we have to do the job ourselves.

```
\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
```

10.29 For variables

```
\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\expandafter
         \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__
```

10.29.1 Defining variables

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

It is not recommended to overwrite single Latin letters which are pre-defined to serve as dummy variables. Variable names may contains letters, digits, underscores, and must not start with a digit.

```
\catcode`: 12
```

```
\def\xintdefvar #1:=#2;{\expandafter\odef
   \csname XINT_expr_var_\xint_zapspaces #1 \xint_gobble_i\endcsname
                           {\expandafter\empty\romannumeral0\xintbareeval #2\relax }}%
\def\xintdefiivar #1:=#2;{\expandafter\odef
   \csname XINT_expr_var_\xint_zapspaces #1 \xint_gobble_i\endcsname
  {\expandafter\empty\romannumeral0\xintbareiieval #2\relax }%
}%
\def\xintdeffloatvar #1:=#2;{\expandafter\odef
   \csname XINT_expr_var_\xint_zapspaces #1 \xint_gobble_i\endcsname
  {\expandafter\empty\romannumeral0\xintbarefloateval #2\relax }%
}%
\catcode`: 11
```

10.29.2 Letters as dummy variables; the nil list

```
\def\XINT_tmpa #1%
{%
   \expandafter\def\csname XINT_expr_var_#1\endcsname ##1\relax !#1##2%
   {\romannumeral0\XINT_expr_lockscan ##2!##1\relax !#1{##2}}%
}%
\xintApplyUnbraced \XINT_tmpa {abcdefghijklmnopqrstuvwxyz}%
\xintApplyUnbraced \XINT_tmpa {ABCDEFGHIJKLMNOPQRSTUVWXYZ}%
\expandafter\def\expandafter\XINT_expr_var_nil\expandafter
               {\expandafter\empty\csname .= \endcsname}%
```

10.29.3 The omit and abort constructs

```
\catcode`. 11 \catcode`= 11
\def\XINT_expr_var_omit  #1\relax !{1^C!{}{}{}\.=!\relax !}% 24 juin
\def\XINT_expr_var_abort #1\relax !{1^C!{}{}{}\.=^\relax !}% 25 juin
\catcode`. 12 \catcode`= 12
```

10.29.4 The @, @1, @2, @3, @4, @@, @@(1), ..., @@@, @@@(1), ... for recursion

I had completely forgotten what the @@@ etc... stuff were supposed to do: this is for nesting recursions! (I was mad back in June 2014). @@(N) gives the Nth back, @@@(N) gives the Nth back of the higher recursion!

```
\catcode`? 3
\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_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
```

10.30 For functions

```
\def\XINT_tmpa #1#2#3{%
    \def #1##1% \XINT_expr_op_`, #2=\XINT_expr_oparen
    {%
        \ifcsname XINT_expr_onlitteral_##1\endcsname
          \xint_dothis{\csname XINT_expr_onlitteral_##1\endcsname}\fi
        \ifcsname XINT_#3_func_##1\endcsname
          \xint_dothis{\expandafter\expandafter
                       \csname XINT_#3_func_##1\endcsname\romannumeral-`0#2}\fi
        \xint_orthat{\XINT_expr_unknown_function {##1}%
           \expandafter\XINT_expr_func_unknown\romannumeral-`0#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}%
}%
\expandafter\def\csname XINT_expr_onlitteral_`\endcsname #1#2#3({\xint_c_xviii `{#2}}%
```

10.31 The bool, togl, protect, unknown, and break "functions"

bool, togl and protect use delimited macros. Only unknown and break are true functions with a more flexible parsing of the opening and closing parentheses, which may possibly arise from expansion itself.

```
\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 }%
\def\XINT_expr_func_unknown #1#2#3{\expandafter #1\expandafter #2\csname .=0\endcsname }%
\def\XINT_expr_func_break #1#2#3%
{\expandafter #1\expandafter #2\csname.=?\romannumeral-`0\XINT_expr_unlock #3\endcsname }%
\let\XINT_flexpr_func_break \XINT_expr_func_break
\let\XINT_iiexpr_func_break \XINT_expr_func_break
```

10.32 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.

10.32.1 \XINT_expr_onlitteral_seq

```
\def\XINT_expr_onlitteral_seq
 {\expandafter\XINT_expr_onlitteral_seq_f\romannumeral-`0\XINT_expr_onlitteral_seq_a {}}%
\def\XINT_expr_onlitteral_seq_f #1#2{\xint_c_xviii `{seqx}#2)\relax #1}%
```

10.32.2 \XINT_expr_onlitteral_seq_a

```
\def\XINT_expr_onlitteral_seq_a #1#2,%
{% checks balancing of parentheses
    \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)}}%
```

10.32.3 \XINT_isbalanced_a for \XINT_expr_onlitteral_seq_a

Expands to \m@ne 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 {\m@ne}%
```

#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 }%
```

10.32.4 \XINT_allexpr_func_seqx, \XINT_allexpr_func_subx

```
\def\XINT_expr_func_seqx   #1#2{\XINT_allexpr_seqx \xintbareeval }%
\def\XINT_flexpr_func_seqx #1#2{\XINT_allexpr_seqx \xintbarefloateval}%
\def\XINT_iiexpr_func_seqx #1#2{\XINT_allexpr_seqx \xintbareiieval }%
\def\XINT_allexpr_seqx #1#2#3#4% #2 is the index list, fully evaluated and encapsulated
{% #3 is the Latin letter serving as dummy variable, #4 is the expression to evaluate
    \expandafter \XINT_expr_getop
    \csname .=\expandafter\XINT_expr_seq:_aa
           \romannumeral-`0\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}%
```

10.32.5 break, abort, omit within seq

when evaluation is done in seq:_d, after the ! we find: the Latin letter, the braced evaluated value to which it will be assigned, a saved copy of the the \xintexpr stuff, the braced accumulated comma separated list of previous computations, and the rest of the list of comma separated values to assign to the dummy letter and at the very end there is ^ and the final comma.

```
\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,{\XINT_expr_seq:_c #2,{#1}}%
\def\XINT_expr_seq:_c #1{\if ,#1\xint_dothis\XINT_expr_seq:_noop\fi
                         \if ^#1\xint_dothis\XINT_expr_seq:_end\fi
                         \xint_orthat\XINT_expr_seq:_d #1}%
\def\XINT_expr_seq:_d #1,#2{\expandafter\XINT_expr_seq:_e
    \romannumeral-`0\expandafter\XINT_expr_unlock\romannumeral0#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:_goon  #1!#2#3#4{,#1\XINT_expr_seq:_b {#4}}%
\def\XINT_expr_seq:_omit  #1!#2#3#4{\XINT_expr_seq:_b {#4}}%
\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:_noop ,#1{\XINT_expr_seq:_b {#1}}%
\def\XINT_expr_seq:_end ^,#1{}% if all is omit, _empty? constructs "nil"
\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? { }%
```

10.32.6 \XINT_expr_seq:_A

This is for index lists generated by ++. The starting point will have been replaced by its ceil. For efficiency I use \numexpr rather than \xintInc, hence the indexing is limited to small integers.

```
\def\XINT_expr_seq:_A +#1!#2,^,%
   {\expandafter\XINT_expr_seq_empty?\romannumeral0\XINT_expr_seq:_D {#1}{#2}}%
\def\XINT_expr_seq:_D #1#2{\expandafter\XINT_expr_seq:_E
    \romannumeral-`0\expandafter\XINT_expr_unlock\romannumeral0#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:_Goon  #1!#2#3#4%
    {,#1\expandafter\XINT_expr_seq:_D\expandafter{\the\numexpr #3+\xint_c_i}{#4}}%
\def\XINT_expr_seq:_Omit  #1!#2#3#4%
    {\expandafter\XINT_expr_seq:_D\expandafter{\the\numexpr #3+\xint_c_i}{#4}}%
\def\XINT_expr_seq:_Abort #1!#2#3#4{}%
\def\XINT_expr_seq:_Break #1!#2#3#4{,#1}%
```

10.32.7 add and mul, \XINT_expr_onlitteral_add, \XINT_expr_onlitteral_mul

```
\def\XINT_expr_onlitteral_add
 {\expandafter\XINT_expr_onlitteral_add_f\romannumeral-`0\XINT_expr_onlitteral_seq_a {}}%
```

```
\def\XINT_expr_onlitteral_add_f #1#2{\xint_c_xviii `{opx}#2)\relax #1+}%
\def\XINT_expr_onlitteral_mul
 {\expandafter\XINT_expr_onlitteral_mul_f\romannumeral-`0\XINT_expr_onlitteral_seq_a {}}%
\def\XINT_expr_onlitteral_mul_f #1#2{\xint_c_xviii `{opx}#2)\relax #1*}%
```

10.32.8 \XINT_expr_func_opx, \XINT_flexpr_func_opx, \XINT_iiexpr_func_opx

```
\expandafter\edef\csname XINT_expr_op:_+\endcsname
     {\noexpand\xint_gobble_v {}{}{}\expandafter\noexpand\csname .=0\endcsname}%
\expandafter\edef\csname XINT_expr_op:_*\endcsname
     {\noexpand\xint_gobble_v {}{}{}\expandafter\noexpand\csname .=1\endcsname}%
\def\XINT_expr_func_opx   #1#2{\XINT_allexpr_opx \xintexpr }%
\def\XINT_flexpr_func_opx #1#2{\XINT_allexpr_opx \xintfloatexpr }%
\def\XINT_iiexpr_func_opx #1#2{\XINT_allexpr_opx \xintiiexpr }%
\def\XINT_allexpr_opx #1#2#3#4#5%
```

```
 au d\'epart on avait op(#4,#3=#2 (\'evalu\'e ici)) #3=la variable, #4=expression, #5=+ ou*.
```

```
{%
    \expandafter\XINT_expr_getop\romannumeral0\expandafter\XINT_expr_op:_a
    \csname XINT_expr_op:_#5\expandafter\endcsname
    \romannumeral-`0\XINT_expr_unlock #2!#5#1#3{#4}%
}%
```

10.32.9 \XINT_expr_op:_a, ...

```
\def\XINT_expr_op:_a #1#2!#3#4#5#6{\XINT_expr_op:_b {#1#4#3{#6\relax\relax !#5}}#2,^,}%
```

```
 #1=op_+ ou op_*, #2=liste, #3=+ou*,#4=\xintexpr, etc, #5=la var,#6=expression
```

```
\def\XINT_expr_op:_b #1#2,{\XINT_expr_op:_c #2,#1}%
\def\XINT_expr_op:_c #1{\if ,#1\xint_dothis\XINT_expr_op:_noop\fi
                        \if ^#1\xint_dothis\XINT_expr_op:_end\fi
                        \xint_orthat\XINT_expr_op:_d #1}%
\def\XINT_expr_op:_noop #1,#2#3#4#5{\XINT_expr_op:_b {{#2}#3#4{#5}}}%
\def\XINT_expr_op:_d #1,#2#3#4#5%
```

```
 #1=valeur, #2=partiel, #3=\xintexpr #4=+ ou *, #5 = expression
```

```
  {\expandafter\expandafter\expandafter\XINT_expr_op:_e #3#2#4#3#5{#1}{#3#4{#5}}}%
```

```
 #2=nom de la variable, #3=ancienne valeur variable
```

```
\def\XINT_expr_op:_e   !#1!#2#3#4{\XINT_expr_op:_b {{!#1}#4}}%
\def\XINT_expr_op:_end ^,#1#2#3#4{\expandafter\expandafter\expandafter\space
                                  \expandafter\xint_gobble_iv #1}%
```

10.32.10 subs, \XINT_expr_onlitteral_subs

```
\def\XINT_expr_onlitteral_subs
 {\expandafter\XINT_expr_onlitteral_subs_f\romannumeral-`0\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 \XINT_expr_getop
    \csname .=\expandafter\XINT_expr_subx:_a
```

```
            \romannumeral-`0\XINT_expr_unlock #2!{#1#4\relax !#3}\endcsname
}%
\def\XINT_expr_subx:_a #1!#2%
   {\expandafter\XINT_expr_subx:_end \romannumeral0#2{#1}}%
```

attention, if one day I add a space in unlock, will need \romannumeral-`0

```
\def\XINT_expr_subx:_end #1!#2#3{\XINT_expr_unlock #1}%
```

10.33 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.

```
\def\XINT_expr_func_rseq   {\XINT_allexpr_rseq \xintbareeval      }%
\def\XINT_flexpr_func_rseq {\XINT_allexpr_rseq \xintbarefloateval }%
\def\XINT_iiexpr_func_rseq {\XINT_allexpr_rseq \xintbareiieval    }%
\def\XINT_allexpr_rseq #1#2%
{%
    \expandafter\XINT_expr_rseqx\expandafter #1\expandafter
    #2\romannumeral-`0\XINT_expr_onlitteral_seq_a {}%
}%
```

10.33.1 \XINT_expr_rseqx

The (#4) is for ++ mechanism which must have its closing parenthesis.

```
\def\XINT_expr_rseqx #1#2#3#4%
{%
    \expandafter\XINT_expr_rseqy\romannumeral0#1(#4)\relax
    #2#3#1%
}%
```

10.33.2 \XINT_expr_rseqy

#1=valeurs pour variable (locked), #2=toutes les valeurs initiales (csv,locked), #3=variable, #4=expr, #5=\xintbareeval ou \xintbarefloateval ou \xintbareiieval

```
\def\XINT_expr_rseqy #1#2#3#4#5%
{%
    \expandafter \XINT_expr_getop
    \csname .=\XINT_expr_unlock #2%
    \expandafter\XINT_expr_rseq:_aa
                \romannumeral-`0\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}%
```

10.33.3 \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,{\XINT_expr_rseq:_c #3,~#1{#2}}%
\def\XINT_expr_rseq:_c #1{\if ,#1\xint_dothis\XINT_expr_rseq:_noop\fi
                          \if ^#1\xint_dothis\XINT_expr_rseq:_end\fi
                          \xint_orthat\XINT_expr_rseq:_d #1}%
```

```
\def\XINT_expr_rseq:_d #1,~#2#3{\expandafter\XINT_expr_rseq:_e
       \romannumeral-`0\expandafter\XINT_expr_unlock\romannumeral0#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:_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}%
\def\XINT_expr_rseq:_noop ,~#1#2{\XINT_expr_rseq:_b #1{#2}}%
\def\XINT_expr_rseq:_end ^,~#1#2{}% no nil for rseq
```

10.33.4 \XINT_expr_rseq:_A etc...

n++ for rseq

```
\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-`0\expandafter\XINT_expr_unlock\romannumeral0#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\expandafter{\the\numexpr #3+\xint_c_i\expandafter}%
    \romannumeral0\XINT_expr_lockit{#1}{#5}}%
\def\XINT_expr_rseq:_Omit  #1!#2#3~%#4#5%
    {\expandafter\XINT_expr_rseq:_D\expandafter{\the\numexpr #3+\xint_c_i}}%
\def\XINT_expr_rseq:_Abort #1!#2#3~#4#5{}%
\def\XINT_expr_rseq:_Break #1!#2#3~#4#5{,#1}%
```

10.34 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      }%
\def\XINT_flexpr_func_rrseq {\XINT_allexpr_rrseq \xintbarefloateval }%
\def\XINT_iiexpr_func_rrseq {\XINT_allexpr_rrseq \xintbareiieval    }%
\def\XINT_allexpr_rrseq #1#2%
{%
    \expandafter\XINT_expr_rrseqx\expandafter #1\expandafter
    #2\romannumeral-`0\XINT_expr_onlitteral_seq_a {}%
}%
```

10.34.1 \XINT_expr_rrseqx

The (#4) is for ++ mechanism which must have its closing parenthesis.

```
\def\XINT_expr_rrseqx #1#2#3#4%
{%
```

```
    \expandafter\XINT_expr_rrseqy\romannumeral0#1(#4)\expandafter\relax
    \expandafter{\romannumeral0\xintapply \XINT_expr_lockit
       {\xintRevWithBraces{\xintCSVtoListNonStripped{\XINT_expr_unlock #2}}}}%
    #2#3#1%
}%
```

10.34.2 \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=\xintbareeval ou \xintbarefloateval ou \xintbareiieval

```
\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-`0\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}%
```

10.34.3 \XINT_expr_rrseq:_a etc...

```
\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,{\XINT_expr_rrseq:_c #3,~#1?{#2}}%
\def\XINT_expr_rrseq:_c #1{\if ,#1\xint_dothis\XINT_expr_rrseq:_noop\fi
                           \if ^#1\xint_dothis\XINT_expr_rrseq:_end\fi
                           \xint_orthat\XINT_expr_rrseq:_d #1}%
\def\XINT_expr_rrseq:_d #1,~#2?#3{\expandafter\XINT_expr_rrseq:_e
       \romannumeral-`0\expandafter\XINT_expr_unlock\romannumeral0#3{#1}~#2?{#3}}%
\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}%
\def\XINT_expr_rrseq:_noop ,~#1?#2{\XINT_expr_rrseq:_b {#1}{#2}}%
\def\XINT_expr_rrseq:_end ^,~#1?#2{}% No nil for rrseq.
\catcode`? 11
\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}%
}%
```

10.34.4 \XINT_expr_rrseq:_A etc...

n++ for rrseq

```
\catcode`? 3
\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-`0\expandafter\XINT_expr_unlock\romannumeral0#3{#1}~#2?{#3}}%
\def\XINT_expr_rrseq:_Goon  #1!#2#3~#4?#5%
   {,#1\expandafter\XINT_expr_rrseq:_D\expandafter{\the\numexpr #3+\xint_c_i\expandafter}%
    \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\expandafter{\the\numexpr #3+\xint_c_i}}%
\def\XINT_expr_rrseq:_Abort #1!#2#3~#4?#5{}%
\def\XINT_expr_rrseq:_Break #1!#2#3~#4?#5{,#1}%
\catcode`? 11
\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}}%
```

10.35 iter

```
\def\XINT_expr_func_iter   {\XINT_allexpr_iter \xintbareeval      }%
\def\XINT_flexpr_func_iter {\XINT_allexpr_iter \xintbarefloateval }%
\def\XINT_iiexpr_func_iter {\XINT_allexpr_iter \xintbareiieval    }%
\def\XINT_allexpr_iter #1#2%
{%
    \expandafter\XINT_expr_iterx\expandafter #1\expandafter
    #2\romannumeral-`0\XINT_expr_onlitteral_seq_a {}%
}%
```

10.35.1 \XINT_expr_iterx

The (#4) is for ++ mechanism which must have its closing parenthesis.

```
\def\XINT_expr_iterx #1#2#3#4%
{%
    \expandafter\XINT_expr_itery\romannumeral0#1(#4)\expandafter\relax
    \expandafter{\romannumeral0\xintapply \XINT_expr_lockit
       {\xintRevWithBraces{\xintCSVtoListNonStripped{\XINT_expr_unlock #2}}}}%
    #2#3#1%
}%
```

10.35.2 \XINT_expr_itery

#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_itery #1#2#3#4#5#6%
{%
    \expandafter \XINT_expr_getop
    \csname .=%
    \expandafter\XINT_expr_iter:_aa
                \romannumeral-`0\XINT_expr_unlock #1!{#6#5\relax !#4}{#2},^,\endcsname
}%
\def\XINT_expr_iter:_aa #1{\if +#1\expandafter\XINT_expr_iter:_A\else
                                  \expandafter\XINT_expr_iter:_a\fi #1}%
```

10.35.3 \XINT_expr_iter:_a etc...

```
\catcode`? 3
\def\XINT_expr_iter:_a #1!#2#3{\XINT_expr_iter:_b {#3}{#2}#1}%
\def\XINT_expr_iter:_b #1#2#3,{\XINT_expr_iter:_c #3,~#1?{#2}}%
\def\XINT_expr_iter:_c #1{\if ,#1\xint_dothis\XINT_expr_iter:_noop\fi
                          \if ^#1\xint_dothis\XINT_expr_iter:_end\fi
                          \xint_orthat\XINT_expr_iter:_d #1}%
\def\XINT_expr_iter:_d #1,~#2?#3{\expandafter\XINT_expr_iter:_e
       \romannumeral-`0\expandafter\XINT_expr_unlock\romannumeral0#3{#1}~#2?{#3}}%
\def\XINT_expr_iter:_goon  #1!#2#3~#4?#5{\expandafter\XINT_expr_iter:_b\expandafter
        {\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
\def\XINT_expr_iter:_omit  #1!#2#3~{\XINT_expr_iter:_b }%
\def\XINT_expr_iter:_abort #1!#2#3~#4?#5#6^,%
   {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
          {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}}}%
\def\XINT_expr_iter:_break #1!#2#3~#4?#5#6^,%
   {\expandafter\xint_gobble_iv\romannumeral0\xintapplyunbraced
          {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}},#1}%
\def\XINT_expr_iter:_noop ,~#1?#2{\XINT_expr_iter:_b {#1}{#2}}%
\def\XINT_expr_iter:_end ^,~#1?#2%
   {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
          {,\XINT_expr:_unlock}{\xintReverseOrder{#1\space}}}%
\catcode`? 11
\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:_unlock #1{\XINT_expr_unlock #1}%
```

10.35.4 \XINT_expr_iter:_A etc...

n++ for iter

```
\catcode`? 3
\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-`0\expandafter\XINT_expr_unlock\romannumeral0#3{#1}~#2?{#3}}%
\def\XINT_expr_iter:_Goon  #1!#2#3~#4?#5%
   {\expandafter\XINT_expr_iter:_D\expandafter{\the\numexpr #3+\xint_c_i\expandafter}%
    \expandafter{\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
\def\XINT_expr_iter:_Omit  #1!#2#3~%#4?#5%
    {\expandafter\XINT_expr_iter:_D\expandafter{\the\numexpr #3+\xint_c_i}}%
\def\XINT_expr_iter:_Abort #1!#2#3~#4?#5%
   {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
          {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}}}%
\def\XINT_expr_iter:_Break #1!#2#3~#4?#5%
   {\expandafter\xint_gobble_iv\romannumeral0\xintapplyunbraced
          {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}},#1}%
\catcode`? 11
\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}}%
```

10.36 Macros handling csv lists for functions with multiple comma separated arguments in expressions

These 17 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.

10.36.1 \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-`0#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
```

10.36.2 \xintORof:csv

1.09a. For use by \xintexpr.

```
\def\xintORof:csv #1{\expandafter\XINT_orof:_a\romannumeral-`0#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
```

10.36.3 \xintXORof:csv

1.09a. For use by \xintexpr (inside a \csname..\endcsname).

```
\def\xintXORof:csv #1{\expandafter\XINT_xorof:_a\expandafter 0\romannumeral-`0#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)
```

10.36.4 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-`0#2{#3}}#1#2}%
\def\XINT_oncsv:_c #1#2#3#4,{\expandafter\XINT_oncsv:_d \romannumeral-`0#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-`0#3{#4{#1}}{#2}}#3#4}%
```

10.36.5 \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-`0#1,^,{0/1[0]}}%
\def\xintiiMaxof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiimax
                       \expandafter\xint_firstofone\romannumeral-`0#1,^,0}%
```

10.36.6 \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-`0#1,^,{0/1[0]}}%
\def\xintiiMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiimin
                       \expandafter\xint_firstofone\romannumeral-`0#1,^,0}%
```

10.36.7 \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-`0#1,^,{0/1[0]}}%
\def\xintiiSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiadd
                      \expandafter\xint_firstofone\romannumeral-`0#1,^,0}%
```

10.36.8 \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-`0#1,^,{1/1[0]}}%
\def\xintiiPrd:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiimul
                      \expandafter\xint_firstofone\romannumeral-`0#1,^,1}%
```

10.36.9 \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-`0#1,^,1}%
\def\xintLCMof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintlcm
                      \expandafter\xint_firstofone\romannumeral-`0#1,^,0}%
```

10.36.10 \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-`0#1,^,1}%
\def\xintiiLCMof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiilcm
                      \expandafter\xint_firstofone\romannumeral-`0#1,^,0}%
```

10.36.11 \XINTinFloatdigits, \XINTinFloatSqrtdigits

for \xintNewExpr matters, mainly.

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

10.36.12 \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-`0#1,^,{0[0]}}%
\def\XINTinFloatMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmin
                         \expandafter\XINTinFloatdigits\romannumeral-`0#1,^,{0[0]}}%
```

10.36.13 \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-`0#1,^,{0[0]}}%
\def\XINTinFloatPrd:csv #1{\expandafter\XINT_oncsv:_a\expandafter\XINTinfloatmul
                         \expandafter\XINTinFloatdigits\romannumeral-`0#1,^,{1[0]}}%
```

10.37 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, first, last, even, odd, and reversed 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
\def\XINT_iiexpr_func_floor #1#2#3%
{% mais absurde si on ne peut pas avoir quotient comme input
  \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%
{% mais absurde si on ne peut pas avoir quotient comme input
  \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_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-`0\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-`0\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-`0\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-`0\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-`0\XINT_expr_unlock #3,,.\endcsname
}%
\def\XINT_expr_func_float #1#2#3%
{%
    \expandafter #1\expandafter #2\csname .=%
    \expandafter\XINT_expr_argandopt
    \romannumeral-`0\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-`0\XINT_expr_unlock #3,\endcsname
}%
\def\XINT_flexpr_func_mod #1#2#3%
{%
    \expandafter #1\expandafter #2\csname .=%
    \expandafter\XINTinFloatMod
    \romannumeral-`0\expandafter\XINT_expr_twoargs
    \romannumeral-`0\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-`0\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-`0\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-`0\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-`0\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-`0\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-`0\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-`0\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_first #1#2#3%
    {\expandafter #1\expandafter #2\csname.=\expandafter\XINT_expr_func_firsta
     \romannumeral-`0\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-`0\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
\def\XINT_expr_func_reversed #1#2#3%
   {\expandafter #1\expandafter #2\csname .=\xintReversed::csv
                                                    {\XINT_expr_unlock #3}\endcsname }%
\let\XINT_flexpr_func_reversed\XINT_expr_func_reversed
\let\XINT_iiexpr_func_reversed\XINT_expr_func_reversed
\def\xintReversed::csv #1% should be done directly, of course
   {\xintListWithSep,{\xintRevWithBraces {\xintCSVtoListNonStripped{#1}}}}%
```

10.38 f-expandable versions of the SeqB::csv routines, for \xintNewExpr

10.38.1 \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-`0#2#1!}%
\def\XINT_seqb:f:csv_a #1#2;#3;#4!{%
   \expandafter\xint_gobble_i\romannumeral-`0%
   \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}%
```

10.38.2 \xintiiSeqB:f:csv

Produces in f-expandable way. If the step is zero, gives empty result except if start and end coincide.

```
\def\xintiiSeqb:f:csv #1#2%
   {\expandafter\XINT_iiseqb:f:csv \expandafter{\romannumeral-`0#2}{#1}}%
\def\XINT_iiseqb:f:csv #1#2{\expandafter\XINT_iiseqb:f:csv_a\romannumeral-`0#2#1!}%
\def\XINT_iiseqb:f:csv_a #1#2;#3;#4!{%
   \expandafter\xint_gobble_i\romannumeral-`0%
   \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}%
}%
```

10.38.3 \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-`0#2#1!}%
\def\XINT_flseqb:f:csv_a #1#2;#3;#4!{%
```

```
    \expandafter\xint_gobble_i\romannumeral-`0%
    \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}%
}%
```

10.39 \xintNewExpr, \xintNewIExpr, \xintNewFloatExpr, \xintNewIIExpr

10.39.1 \xintApply::csv

Don't ask me what this if for. I wrote it in June, and we are now late October.

```
\def\xintApply::csv #1#2%
   {\expandafter\XINT_applyon::_a\expandafter {\romannumeral-`0#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-`0#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-`0#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}%
```

10.39.2 \xintApply:::csv

```
\def\xintApply:::csv #1#2#3%
   {\expandafter\XINT_applyon:::_a\expandafter{\romannumeral-`0#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-`0#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-`0\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}%
```

10.39.3 \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.

```
\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}}%
```

10.39.4 Mysterious stuff

actually I dimly remember that the whole point is to allow maximal evaluation as long as macro parameters not encountered. Else it would be easier. \xintNewIExpr \f [2]{[12] #1+#2+3*6*1} will correctly compute the 18.

1.1a re-establishes the trick with \toks0\expandafter{\the\toks0\expandafter etc...} with *no* space after 0. I don't know why it was removed at some point before releasing 1.1, but \XINT_expr_redefinemacros is even bigger without the trick.

```
\catcode`~ 12 % by the way, catcode is set to 3 in \XINTsetupcatcodes
\catcode`$ 12 % $
\def\XINT_xptwo_getab_b #1#2!#3%
   {\expandafter\XINT_xptwo_getab_c\romannumeral-`0#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,iiDivTrunc,iiDivRound,Mod,iiMod,iRound,Round,iTrunc,Trunc,%
        Lt,Gt,Eq,LtorEq,GtorEq,Neq,AND,OR,XOR,iQuo,iRem,Add,Sub,Mul,Div,Pow,E,%
        iiAdd,iiSub,iiMul,iiPow,iiQuo,iiRem,iiE,SeqA::csv,iiSeqA::csv}\do
 {\toks0
  \expandafter{\the\toks0% no space! (makes shorter macro in the end)
  \expandafter\let\csname xint#1NE\expandafter\endcsname\csname xint#1\expandafter
  \endcsname\expandafter\def\csname xint#1\endcsname ####1####2{%
```

```
        \expandafter\XINT_NEfork
        \romannumeral-`0\expandafter\XINT_xptwo_getab_b
        \romannumeral-`0####2!{####1}{~xint#1}{xint#1}}%
  }%
}%
\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,Fac,iFac,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-`0####1!{~xint#1}{xint#1}{}{}}%
  }%
}%
\xintFor #1 in {Add,Sub,Mul,Div,Power,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-`0\expandafter\XINT_xptwo_getab_b
        \romannumeral-`0####2!{####1}{~XINTinFloat#1}{XINTinFloat#1}}%
  }%
}%
\xintFor #1 in {XINTinFloatdigits,XINTinFloatFracdigits,XINTinFloatSqrtdigits}\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-`0####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-`0\expandafter\XINT_xptwo_getab_b
        \romannumeral-`0####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-`0\expandafter\XINT_xptwo_getab_b
        \romannumeral-`0####2!{####1}{$noexpand$#1:f:csv}{#1::csv}}%
```

```
2196   }%
2197 }%
2198 \toks0
2199   \expandafter{\the\toks0
2200   \let\XINTinFloatNE\XINTinFloat
2201   \def\XINTinFloat [##1]##2{% not ultimately general, but got tired
2202         \expandafter\XINT_NEfork_one
2203         \romannumeral-`0##2!{~XINTinFloat[##1]}{XINTinFloat}{}{[##1]}}%
2204   \let\XINTinFloatSqrtNE\XINTinFloatSqrt
2205   \def\XINTinFloatSqrt [##1]##2{%
2206         \expandafter\XINT_NEfork_one
2207         \romannumeral-`0##2!{~XINTinFloatSqrt[##1]}{XINTinFloatSqrt}{}{[##1]}}%
2208 }%
2209 \xintFor #1 in {ANDof,ORof,XORof,iiMaxof,iiMinof,iiSum,iiPrd,
2210                 GCDof,LCMof,Sum,Prd,Maxof,Minof}\do
2211 {\toks0
2212   \expandafter{\the\toks0\expandafter\def\csname xint#1:csv\endcsname {~xint#1:csv}}%
2213 }%
2214 \xintFor #1 in {XINTinFloatMaxof,XINTinFloatMinof,XINTinFloatSum,XINTinFloatPrd}\do
2215 {\toks0
2216   \expandafter{\the\toks0\expandafter\def\csname #1:csv\endcsname {~#1:csv}}%
2217 }%
2218 \expandafter\def\expandafter\XINT_expr_redefinemacros\expandafter
2219   {\the\toks0
2220    \def\XINT_flexpr_noopt {\expandafter\XINT_flexpr_withopt_b\expandafter-%
2221                            \romannumeral0\xintbarefloateval }%
2222    \def\XINT_flexpr_withopt_b ##1##2%
2223      {\expandafter\XINT_flexpr_wrap\csname .;##1.=\XINT_expr_unlock  ##2\endcsname }%
2224    \def\XINT_expr_unlock_sp ##1.;##2##3.=##4!{\if -##2\expandafter\xint_firstoftwo
2225          \else\expandafter\xint_secondoftwo\fi \XINTdigits{{##2##3}}{##4}}%
2226    \def\XINT_expr_print     ##1{\expandafter\xintSPRaw::csv\expandafter
2227                              {\romannumeral-`0\XINT_expr_unlock ##1}}%
2228    \def\XINT_iiexpr_print   ##1{\expandafter\xintCSV::csv\expandafter
2229                              {\romannumeral-`0\XINT_expr_unlock ##1}}%
2230    \def\XINT_boolexpr_print ##1{\expandafter\xintIsTrue::csv\expandafter
2231                                 {\romannumeral-`0\XINT_expr_unlock ##1}}%
2232    \def\xintCSV::csv     {~xintCSV::csv }% spaces to separate from possible catcode 11
2233    \def\xintSPRaw::csv   {~xintSPRaw::csv }% stuff after
2234    \def\xintPFloat::csv  {~xintPFloat::csv }%
2235    \def\xintIsTrue::csv  {~xintIsTrue::csv }%
2236    \def\xintRound::csv   {~xintRound::csv }%
```

```
\def\XINTinFloat::csv {~XINTinFloat::csv } should not be needed.
```

```
2237    \def\xintReversed::csv {~xintReversed::csv }%
2238    \def\xintListSel:csv   {~xintListSel:csv }%
2239 }%
2240 \toks0 {}%
2241 \def\xintNewExpr      {\xint_NewExpr\xinttheexpr      }%
2242 \def\xintNewFloatExpr {\xint_NewExpr\xintthefloatexpr }%
2243 \def\xintNewIExpr     {\xint_NewExpr\xinttheiexpr     }%
2244 \def\xintNewIIExpr    {\xint_NewExpr\xinttheiiexpr    }%
2245 \def\xintNewBoolExpr  {\xint_NewExpr\xinttheboolexpr  }%
```

```
\def\XINT_newexpr_finish #1>{\noexpand\romannumeral-`0}%
\def\xint_NewExpr #1#2[#3]%
{%
 \begingroup
    \ifcase #3\relax
        \toks0 {\xdef #2}%
    \or \toks0 {\xdef #2##1}%
    \or \toks0 {\xdef #2##1##2}%
    \or \toks0 {\xdef #2##1##2##3}%
    \or \toks0 {\xdef #2##1##2##3##4}%
    \or \toks0 {\xdef #2##1##2##3##4##5}%
    \or \toks0 {\xdef #2##1##2##3##4##5##6}%
    \or \toks0 {\xdef #2##1##2##3##4##5##6##7}%
    \or \toks0 {\xdef #2##1##2##3##4##5##6##7##8}%
    \or \toks0 {\xdef #2##1##2##3##4##5##6##7##8##9}%
    \fi
    \xintexprSafeCatcodes
    \XINT_NewExpr #1%
}%
\catcode`~ 13 \catcode`@ 14 \catcode`\% 6 \catcode`# 12 \catcode`$ 11 @ $
\def\XINT_NewExpr %1%2@
{@
    \def\XINT_tmpa %%1%%2%%3%%4%%5%%6%%7%%8%%9{%2}@
    \XINT_expr_redefinemacros
    \def~{$noexpand$}@
    \catcode`: 11 \catcode`_ 11
    \catcode`# 12 \catcode`~ 13 \escapechar 126
    \endlinechar -1 \everyeof {\noexpand }@
    \edef\XINT_tmpb
    {\scantokens\expandafter
     {\romannumeral-`0\expandafter%1\XINT_tmpa {#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}\relax}@
    }@
    \escapechar 92 \catcode`# 6 \catcode`$ 0 @ $
    \the\toks0
    {\scantokens\expandafter{\expandafter\XINT_newexpr_finish\meaning\XINT_tmpb}}@
  \endgroup
}@
\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: 224.  Total number of code lines: 12251. Each package starts with
 xinttools:1044.  circa 50 lines dealing with catcodes, package identification
  xintcore:2227.  and reloading management, also for Plain TeX. Version 1.1c of
      xint:1403.  2015/09/12.
xintbinhex: 609.
   xintgcd: 455.
  xintfrac:2545.
xintseries: 386.
 xintcfrac:1021.
  xintexpr:2337.
```