

The `xint` source code

JEAN-FRANÇOIS BURNOL

jfbu (at) free (dot) fr

Package version: 1.1 (2014/10/28); documentation date: 2014/10/28.

From source file xint.dtx. Time-stamp: <28-10-2014 at 19:39:20 CET>.

Contents

1	Package <code>xintkernel</code> implementation	1
2	Package <code>xinttools</code> implementation	7
3	Package <code>xintcore</code> implementation	33
4	Package <code>xint</code> implementation	82
5	Package <code>xintbinhex</code> implementation	118
6	Package <code>xintgcd</code> implementation	130
7	Package <code>xintfrac</code> implementation	142
8	Package <code>xintseries</code> implementation	200
9	Package <code>xintcfrc</code> implementation	209
10	Package <code>xintexpr</code> implementation	232

The main changes with Release 1.1 are in `xintexpr`.

1 Package `xintkernel` implementation

.1	Catcodes, ε -TeX and reload detection	1	.7	<code>\xint_dothis</code> , <code>\xint_orthat</code>	5
.2	Package identification	4	.8	<code>\xint_zapspaces</code>	5
.3	Token management utilities	4	.9	Constants	5
.4	gob til macros and UD style fork	5	.10	<code>\odef</code> , <code>\oodef</code> , <code>\fdef</code>	6
.5	<code>\xint_afterfi</code>	5	.11	<code>\xintReverseOrder</code>	6
.6	<code>\xint_bye</code>	5	.12	<code>\xintLength</code>	6

Release 1.09g of 2013/11/22 splits off `xinttools.sty` from `xint.sty`. But it is still loaded automatically by `xint.sty`.

Release 1.1 of 2014/10/28 splits off `xintcore.sty` from `xint.sty` and also `xintkernel.sty` which is the common minimal code base for loading management and catcode control with also a few programming utilities. It is loaded by both `xintcore.sty` and `xinttools.sty` hence by all other packages.

`xinttools.sty` is not loaded anymore by `xint.sty`, nor by `xintfrac.sty`, but only by `xintexpr.sty`.

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.

1 Package *xintkernel* implementation

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 `\roman{numeral-‘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 \TeX 3.

Release 1.09b is more economical: some macros are defined already in `xint.sty` (now `xinttools.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 `xinttools.sty`) and the module (not very probable but...).

```
1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2 \catcode13=5 % ^^M
3 \endlinechar=13 %
4 \catcode123=1 % {
5 \catcode125=2 % }
6 \catcode35=6 % #
7 \catcode44=12 % ,
8 \catcode45=12 % -
9 \catcode46=12 % .
10 \catcode58=12 % :
11 \catcode95=11 % _
12 \expandafter
13 \ifx\csname PackageInfo\endcsname\relax
14 \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
15 \else
16 \def\y#1#2{\PackageInfo{#1}{#2}}%
17 \fi
18 \expandafter
19 \ifx\csname numexpr\endcsname\relax
20 \y{xintkernel}{\numexpr not available, aborting input}%
21 \aftergroup\endinput
22 \else
23 \expandafter
24 \ifx\csname XINTsetupcatcodes\endcsname\relax
25 \else
26 \y{xintkernel}{I was already loaded, aborting input}%
27 \aftergroup\endinput
28 \fi
29 \fi
30 \def\SetCatcodesIfInputNotAborted
31 {%
32 \endgroup
33 \def\XINT_restorecatcodes
34 {% takes care of all, to allow more economical code in modules
35 \catcode59=\the\catcode59 % ; xintexpr
36 \catcode126=\the\catcode126 % ~ xintexpr
37 \catcode39=\the\catcode39 % ' xintexpr
38 \catcode34=\the\catcode34 % " xintbinhex, and xintexpr
39 \catcode63=\the\catcode63 % ? xintexpr
40 \catcode124=\the\catcode124 % | xintexpr
41 \catcode38=\the\catcode38 % & xintexpr
42 \catcode64=\the\catcode64 % @ xintexpr
```

1 Package *xintkernel* implementation

```

43      \catcode33=\the\catcode33    % ! xintexpr
44      \catcode93=\the\catcode93    % ] -, xintfrac, xintseries, xintcfrac
45      \catcode91=\the\catcode91    % [ -, xintfrac, xintseries, xintcfrac
46      \catcode36=\the\catcode36    % $ xintgcd only
47      \catcode94=\the\catcode94    % ^
48      \catcode96=\the\catcode96    % '
49      \catcode47=\the\catcode47    % /
50      \catcode41=\the\catcode41    % )
51      \catcode40=\the\catcode40    % (
52      \catcode42=\the\catcode42    % *
53      \catcode43=\the\catcode43    % +
54      \catcode62=\the\catcode62    % >
55      \catcode60=\the\catcode60    % <
56      \catcode58=\the\catcode58    % :
57      \catcode46=\the\catcode46    % .
58      \catcode45=\the\catcode45    % -
59      \catcode44=\the\catcode44    % ,
60      \catcode35=\the\catcode35    % #
61      \catcode95=\the\catcode95    % _
62      \catcode125=\the\catcode125 % }
63      \catcode123=\the\catcode123 % {
64      \endlinechar=\the\endlinechar
65      \catcode13=\the\catcode13    % ^^M
66      \catcode32=\the\catcode32    %
67      \catcode61=\the\catcode61\relax % =
68  }%
69  \edef\XINT_restorecatcodes_endinput
70  {%
71      \XINT_restorecatcodes\noexpand\endinput %
72  }%
73  \def\XINT_setcatcodes
74  {%
75      \catcode61=12    % =
76      \catcode32=10    % space
77      \catcode13=5     % ^^M
78      \endlinechar=13  %
79      \catcode123=1    % {
80      \catcode125=2    % }
81      \catcode95=11    % _ LETTER
82      \catcode35=6     % #
83      \catcode44=12    % ,
84      \catcode45=12    % -
85      \catcode46=12    % .
86      \catcode58=11    % : LETTER
87      \catcode60=12    % <
88      \catcode62=12    % >
89      \catcode43=12    % +
90      \catcode42=12    % *
91      \catcode40=12    % (
92      \catcode41=12    % )
93      \catcode47=12    % /
94      \catcode96=12    % '

```

1 Package *xintkernel* implementation

```
95      \catcode94=11  % ^ LETTER
96      \catcode36=3   % $
97      \catcode91=12  % [
98      \catcode93=12  % ]
99      \catcode33=11  % ! LETTER
100     \catcode64=11  % @ LETTER
101     \catcode38=12  % &
102     \catcode124=12 % |
103     \catcode63=11  % ? LETTER
104     \catcode34=12  % "
105     \catcode39=12  % '
106     \catcode126=3   % ~
107     \catcode59=12  % ;
108     }%
109     \XINT_setcatcodes
110 }%
111 \SetCatcodesIfInputNotAborted
```

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

```
112 \def\XINTsetupcatcodes {% for use by other modules
113     \edef\XINT_restorecatcodes_endinput
114     {%
115         \XINT_restorecatcodes\noexpand\endinput %
116     }%
117     \XINT_setcatcodes
118 }%
```

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`.

```
119 \ifdefined\ProvidesPackage
120   \let\XINT_providespackage\relax
121 \else
122   \def\XINT_providespackage #1#2[#3]%
123       {\immediate\write-1{Package: #2 #3}%
124        \expandafter\xdef\csname ver@#2.sty\endcsname{#3}}%
125 \fi
126 \XINT_providespackage
127 \ProvidesPackage {xintkernel}%
128 [2014/10/28 v1.1 Paraphernalia for the xint packages (jfb)]%
```

1.3 Token management utilities

Most, but not all, are `\long`.

```
129 \long\def\xint_gobble_   {%
130 \long\def\xint_gobble_i   #1{%
131 \long\def\xint_gobble_ii  #1#2{%
132 \long\def\xint_gobble_iii #1#2#3{%
```

```

133 \long\def\xint_gobble_iv    #1#2#3#4{%
134 \long\def\xint_gobble_v    #1#2#3#4#5{%
135 \long\def\xint_gobble_vi    #1#2#3#4#5#6{%
136 \long\def\xint_gobble_vii   #1#2#3#4#5#6#7{%
137 \long\def\xint_gobble_viii  #1#2#3#4#5#6#7#8{%
138 \long\def\xint_firstofone   #1{%1}%
139 \long\def\xint_firstoftwo   #1#2{%1}%
140 \long\def\xint_secondoftwo  #1#2{%2}%
141 \long\def\xint_firstofone_thenstop #1{ #1}%
142 \long\def\xint_firstoftwo_thenstop #1#2{ #1}%
143 \long\def\xint_secondoftwo_thenstop #1#2{ #2}%

```

1.4 gob til macros and UD style fork

```

144 \def\xint_gob_til_zero #10{%
145 \def\xint_UDzerominusfork #10-#2#3\krof {#2}%
146 \long\def\xint_gob_til_R #1\R {}%
147 \long\def\xint_gob_til_W #1\W {}%
148 \long\def\xint_gob_til_Z #1\Z {}%
149 \let\xint_relax\relax
150 \def\xint_brelax {\xint_relax }%
151 \long\def\xint_gob_til_xint_relax #1\xint_relax {}%

```

1.5 \xint_afterfi

```

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

```

1.6 \xint_bye

```

153 \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.

```

154 \long\def\xint_dothis #1#2\xint_orthat #3{\fi #1}% v1.1
155 \let\xint_orthat \xint_firstofone

```

1.8 \xint_zapspace

New with 1.1 `\xint_zapspace` is to be used (usually within an `\edef` or a `\csname...\endcsname`) as `\xint_zapspace foo<space>\xint_bye\xint_bye`. Will strip some brace pairs. By the way the `\zap@space` of LaTeX chokes on things such as `\zap@space 1 {22} 3 4 \@empty`

```

156 \def\xint_zapspace #1 #2{#1#2\xint_zapspace }% v1.1

```

1.9 Constants

```

157 \chardef\xint_c_    0
158 \chardef\xint_c_i   1
159 \chardef\xint_c_ii  2
160 \chardef\xint_c_iii 3
161 \chardef\xint_c_iv   4
162 \chardef\xint_c_v    5

```

```
163 \chardef\xint_c_vi    6
164 \chardef\xint_c_vii   7
165 \chardef\xint_c_viii  8
```

1.10 \odef, \oodef, \fdef

May be prefixed with \global. No parameter text.

```
166 \def\xintodef #1{\expandafter\def\expandafter#1\expandafter }%
167 \def\xintoodef #1{\expandafter\expandafter\expandafter\def
168     \expandafter\expandafter\expandafter#1%
169     \expandafter\expandafter\expandafter }%
170 \def\xintfdef #1#2{\expandafter\def\expandafter#1\expandafter
171     {\romannumeral-'\0#2}}%
172 \ifdefined\odef\else\let\odef\xintodef\fi
173 \ifdefined\oodef\else\let\oodef\xintoodef\fi
174 \ifdefined\fdef\else\let\fdef\xintfdef\fi
```

1.11 \xintReverseOrder

\xintReverseOrder: does NOT expand its argument.

```
175 \def\xintReverseOrder {\romannumeral0\xintreverseorder }%
176 \long\def\xintreverseorder #1%
177 {%
178     \XINT_rord_main {}#1%
179     \xint_relax
180     \xint_bye\xint_bye\xint_bye\xint_bye
181     \xint_bye\xint_bye\xint_bye\xint_bye
182     \xint_relax
183 }%
184 \long\def\XINT_rord_main #1#2#3#4#5#6#7#8#9%
185 {%
186     \xint_bye #9\XINT_rord_cleanup\xint_bye
187     \XINT_rord_main {#9#8#7#6#5#4#3#2#1}%
188 }%
189 \long\edef\XINT_rord_cleanup\xint_bye\XINT_rord_main #1#2\xint_relax
190 {%
191     \noexpand\expandafter\space\noexpand\xint_gob_til_xint_relax #1%
192 }%
```

1.12 \xintLength

\xintLength does NOT expand its argument.

```
193 \def\xintLength {\romannumeral0\xintlenth }%
194 \long\def\xintlenth #1%
195 {%
196     \XINT_length_loop
197     0.#1\xint_relax\xint_relax\xint_relax\xint_relax
198     \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
199 }%
200 \long\def\XINT_length_loop #1.#2#3#4#5#6#7#8#9%
```

```

201 {%
202   \xint_gob_til_xint_relax #9\XINT_length_finish_a\xint_relax
203   \expandafter\XINT_length_loop\the\numexpr #1+\xint_c_viii.%
204 }%
205 \def\XINT_length_finish_a\xint_relax\expandafter\XINT_length_loop
206   \the\numexpr #1+\xint_c_viii.#2\xint_bye
207 {%
208   \XINT_length_finish_b #2\W\W\W\W\W\W\W\Z {#1}%
209 }%
210 \def\XINT_length_finish_b #1#2#3#4#5#6#7#8\Z
211 {%
212   \xint_gob_til_W
213     #1\XINT_length_finish_c \xint_c_
214     #2\XINT_length_finish_c \xint_c_i
215     #3\XINT_length_finish_c \xint_c_ii
216     #4\XINT_length_finish_c \xint_c_iii
217     #5\XINT_length_finish_c \xint_c_iv
218     #6\XINT_length_finish_c \xint_c_v
219     #7\XINT_length_finish_c \xint_c_vi
220     \W\XINT_length_finish_c \xint_c_vii\Z
221 }%
222 \edef\XINT_length_finish_c #1#2\Z #3%
223   {\noexpand\expandafter\space\noexpand\the\numexpr #3+#1\relax}%
224 \XINT_restorecatcodes_endinput%

```

2 Package *xinttools* implementation

.1	Catcodes, ε -T _E X and reload detection	7	.17	\xintloop, \xintbreakloop, \xintbreak-	
.2	Package identification	8		loopanddo, \xintloopskiptonext	22
.3	\xintgodef, \xintgoodef, \xintgfdef	8	.18	\xintiloop, \xintiloopindex,	
.4	\xintRevWithBraces	9		\xintouterloopindex, \xintbreakilop,	
.5	\xintZapFirstSpaces	10		\xintbreakilopanddo, \xintloopskip-	
.6	\xintZapLastSpaces	10		tonext, \xintloopskipandredo	22
.7	\xintZapSpaces	11	.19	\XINT_xflet	22
.8	\xintZapSpacesB	11	.20	\xintApplyInline	23
.9	\xintCSVtoList, \xintCSVtoListNon-		.21	\xintFor, \xintFor*, \xintBreakFor,	
	Stripped	11		\xintBreakForAndDo	24
.10	\xintListWithSep	13	.22	\XINT_forever, \xintintegers, \xintdi-	
.11	\xintNthElt	13		mensions, \xinrationals	27
.12	\xintKeep	15	.23	\xintForpair, \xintForthree, \xintFor-	
.13	\xintTrim	17		four	29
.14	\xintApply	18	.24	\xintAssign, \xintAssignArray, \xint-	
.15	\xintApplyUnbraced	19		DigitsOf	30
.16	\xintSeq	19			

2.1 Catcodes, ε -T_EX and reload detection

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

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2   \catcode13=5    % ^^M
3   \endlinechar=13 %

```

2 Package *xinttools* implementation

```
4 \catcode123=1 % {
5 \catcode125=2 % }
6 \catcode64=11 % @
7 \catcode35=6 % #
8 \catcode44=12 % ,
9 \catcode45=12 % -
10 \catcode46=12 % .
11 \catcode58=12 % :
12 \let\z\endgroup
13 \expandafter\let\expandafter\x\csname ver@xinttools.sty\endcsname
14 \expandafter\let\expandafter\w\csname ver@xintkernel.sty\endcsname
15 \expandafter
16 \ifx\csname PackageInfo\endcsname\relax
17 \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
18 \else
19 \def\y#1#2{\PackageInfo{#1}{#2}}%
20 \fi
21 \expandafter
22 \ifx\csname numexpr\endcsname\relax
23 \y{xinttools}{\numexpr not available, aborting input}%
24 \aftergroup\endinput
25 \else
26 \ifx\x\relax % plain-TeX, first loading of xinttools.sty
27 \ifx\w\relax % but xintkernel.sty not yet loaded.
28 \def\z{\endgroup\input xintkernel.sty\relax}%
29 \fi
30 \else
31 \def\empty {}%
32 \ifx\x\empty % LaTeX, first loading,
33 % variable is initialized, but \ProvidesPackage not yet seen
34 \ifx\w\relax % xintkernel.sty not yet loaded.
35 \def\z{\endgroup\RequirePackage{xintkernel}}%
36 \fi
37 \else
38 \aftergroup\endinput % xinttools already loaded.
39 \fi
40 \fi
41 \fi
42 \z%
43 \XINTsetupcatcodes% defined in xintkernel.sty
```

2.2 Package identification

```
44 \XINT_providespackage
45 \ProvidesPackage{xinttools}%
46 [2014/10/28 v1.1 Expandable and non-expandable utilities (jfb)]%
```

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

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

2.3 *\xintgodef*, *\xintgoodef*, *\xintgfdef*

1.09i. For use in *\xintAssign*.


```

49 \def\xintgodef {\global\xintodef }%
50 \def\xintgoodef {\global\xintoodef }%
51 \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).

As in some other places, 1.09e replaces \Z by \xint_bye, although here it is just for coherence of notation as \Z would be perfectly safe. The reason for \xint_relax, here and in other locations, is in case #1 expands to nothing, the \romannumeral-‘0 must be stopped

```

52 \def\xintRevWithBraces {\romannumeral0\xintrevwithbraces }%
53 \def\xintRevWithBracesNoExpand {\romannumeral0\xintrevwithbracesnoexpand }%
54 \long\def\xintrevwithbraces #1%
55 {%
56   \expandafter\XINT_revwbr_loop\expandafter{\expandafter}%
57   \romannumeral-‘0#1\xint_relax\xint_relax\xint_relax\xint_relax
58   \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
59 }%
60 \long\def\xintrevwithbracesnoexpand #1%
61 {%
62   \XINT_revwbr_loop {}%
63   #1\xint_relax\xint_relax\xint_relax\xint_relax
64   \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
65 }%
66 \long\def\XINT_revwbr_loop #1#2#3#4#5#6#7#8#9%
67 {%
68   \xint_gob_til_xint_relax #9\XINT_revwbr_finish_a\xint_relax
69   \XINT_revwbr_loop {{#9}{#8}{#7}{#6}{#5}{#4}{#3}{#2}{#1}}%
70 }%
71 \long\def\XINT_revwbr_finish_a\xint_relax\XINT_revwbr_loop #1#2\xint_bye
72 {%
73   \XINT_revwbr_finish_b #2\R\R\R\R\R\R\R\Z #1%
74 }%
75 \def\XINT_revwbr_finish_b #1#2#3#4#5#6#7#8\Z
76 {%
77   \xint_gob_til_R
78   #1\XINT_revwbr_finish_c 8%
79   #2\XINT_revwbr_finish_c 7%
80   #3\XINT_revwbr_finish_c 6%
81   #4\XINT_revwbr_finish_c 5%
82   #5\XINT_revwbr_finish_c 4%
83   #6\XINT_revwbr_finish_c 3%
84   #7\XINT_revwbr_finish_c 2%
85   \R\XINT_revwbr_finish_c 1\Z
86 }%
87 \def\XINT_revwbr_finish_c #1#2\Z
88 {%
89   \expandafter\expandafter\expandafter
90   \space
91   \csname xint_gobble_\romannumeral #1\endcsname
92 }%

```

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.

```
93 \def\xintZapFirstSpaces {\romannumeral0\xintzapfirstspaces }%
```

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

```
94 \long\edef\xintzapfirstspaces #1%
95   {\noexpand\XINT_zapbsp_a \space #1\xint_relax \space\space\xint_relax }%
96 \xint_firstofone {\long\edef\XINT_zapbsp_a #1 } %<- space token here
97 {%
```

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`

```
98   \noexpand\XINT_zapbsp_again? #1\noexpand\xint_bye\noexpand\XINT_zapbsp_b #1\space\space
99 }%
100 \long\def\XINT_zapbsp_again? #1{\xint_bye #1\XINT_zapbsp_again }%
101 \xint_firstofone{\def\XINT_zapbsp_again\XINT_zapbsp_b} {\XINT_zapbsp_a }%
102 \long\def\XINT_zapbsp_b #1\xint_relax #2\xint_relax {#1}%
```

2.6 `\xintZapLastSpaces`

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

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

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

```
104 \long\edef\xintzaplastspaces #1{\noexpand\XINT_zapbsp_a {} \noexpand\empty#1%
105   \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.

```
106 \xint_firstofone {\long\def\XINT_zapbsp_a #1#2 } %<- second space here
107   {\expandafter\XINT_zapbsp_b\expandafter{#2}{#1}}%
108 \long\def\XINT_zapbsp_b #1#2#3\xint_relax
109   {\XINT_zapbsp_end? #3\XINT_zapbsp_e {#2#1}\empty #3\xint_relax }%
```

When we have reached the ending space tokens, #3 is a bunch of spaces followed by `\xint_bye`. So the #1 below will be `\xint_bye`. In all other cases #1 can not be `\xint_bye` nor can it give birth to it via brace stripping.

```
110 \long\def\XINT_zapbsp_end? #1{\xint_bye #1\XINT_zapbsp_end }%
```

We are done. The #1 here has accumulated all the previous material.

```
111 \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.

```
112 \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.

```
113 \def\xintZapSpaces {\romannumeral0\xintzapspaces }%
114 \long\edef\xintzapspaces #1% like \xintZapFirstSpaces.
115     {\noexpand\XINT_zapsp_a \space #1\xint_relax \space\space\xint_relax }%
116 \xint_firstofone {\long\edef\XINT_zapsp_a #1 } %
117     {\noexpand\XINT_zapsp_again? #1\noexpand\xint_bye\noexpand\XINT_zapsp_b #1\space\space}%
118 \long\def\XINT_zapsp_again? #1{\xint_bye #1\XINT_zapsp_again }%
119 \xint_firstofone{\def\XINT_zapsp_again\XINT_zapsp_b} {\XINT_zapsp_a }%
120 \xint_firstofone{\def\XINT_zapsp_b} {\XINT_zapsp_c }%
121 \long\edef\XINT_zapsp_c #1\xint_relax #2\xint_relax {\noexpand\XINT_zapesp_a
122     {} \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).

```
123 \def\xintZapSpacesB {\romannumeral0\xintzapspacesb }%
124 \long\def\xintzapspacesb #1{\XINT_zapspb_one? #1\xint_relax\xint_relax
125     \xint_bye\xintzapspaces {#1}}%
126 \long\def\XINT_zapspb_one? #1#2%
127     {\xint_gob_til_xint_relax #1\XINT_zapspb_onlyspaces\xint_relax
128     \xint_gob_til_xint_relax #2\XINT_zapspb_bracedorone\xint_relax
129     \xint_bye {#1}}%
130 \def\XINT_zapspb_onlyspaces\xint_relax
131     \xint_gob_til_xint_relax\xint_relax\XINT_zapspb_bracedorone\xint_relax
132     \xint_bye #1\xint_bye\xintzapspaces #2{ }%
133 \long\def\XINT_zapspb_bracedorone\xint_relax
134     \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 expanded (protect the first item with a space if it is not to be 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 through \xintZapSpacesB to strip off 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.

```
135 \def\xintCSVtoList {\romannumeral0\xintcsvtolist }%
```

2 Package *xinttools* implementation

```

136 \long\def\xintcsvtolist #1{\expandafter\xintApply
137     \expandafter\xintzapspacesb
138     \expandafter{\romannumeral0\xintcsvtolistnonstripped{#1}}}%
139 \def\xintCSVtoListNoExpand {\romannumeral0\xintcsvtolistnoexpand}%
140 \long\def\xintcsvtolistnoexpand #1{\expandafter\xintApply
141     \expandafter\xintzapspacesb
142     \expandafter{\romannumeral0\xintcsvtolistnonstrippednoexpand{#1}}}%
143 \def\xintCSVtoListNonStripped {\romannumeral0\xintcsvtolistnonstripped}%
144 \def\xintCSVtoListNonStrippedNoExpand
145     {\romannumeral0\xintcsvtolistnonstrippednoexpand}%
146 \long\def\xintcsvtolistnonstripped #1%
147 {%
148     \expandafter\XINT_csvtol_loop_a\expandafter
149     {\expandafter}\romannumeral-‘0#1%
150     ,\xint_bye,\xint_bye,\xint_bye,\xint_bye
151     ,\xint_bye,\xint_bye,\xint_bye,\xint_bye,\Z
152 }%
153 \long\def\xintcsvtolistnonstrippednoexpand #1%
154 {%
155     \XINT_csvtol_loop_a
156     {#1,\xint_bye,\xint_bye,\xint_bye,\xint_bye
157     ,\xint_bye,\xint_bye,\xint_bye,\xint_bye,\Z
158 }%
159 \long\def\XINT_csvtol_loop_a #1#2,#3,#4,#5,#6,#7,#8,#9,%
160 {%
161     \xint_bye #9\XINT_csvtol_finish_a\xint_bye
162     \XINT_csvtol_loop_b {#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}}%
163 }%
164 \long\def\XINT_csvtol_loop_b #1#2{\XINT_csvtol_loop_a {#1#2}}%
165 \long\def\XINT_csvtol_finish_a\xint_bye\XINT_csvtol_loop_b #1#2#3\Z
166 {%
167     \XINT_csvtol_finish_b #3\R,\R,\R,\R,\R,\R,\R,\Z #2{#1}%
168 }%
169 \def\XINT_csvtol_finish_b #1,#2,#3,#4,#5,#6,#7,#8\Z
170 {%
171     \xint_gob_til_R
172     #1\XINT_csvtol_finish_c 8%
173     #2\XINT_csvtol_finish_c 7%
174     #3\XINT_csvtol_finish_c 6%
175     #4\XINT_csvtol_finish_c 5%
176     #5\XINT_csvtol_finish_c 4%
177     #6\XINT_csvtol_finish_c 3%
178     #7\XINT_csvtol_finish_c 2%
179     \R\XINT_csvtol_finish_c 1\Z
180 }%
181 \def\XINT_csvtol_finish_c #1#2\Z
182 {%
183     \csname XINT_csvtol_finish_d\romannumeral #1\endcsname
184 }%
185 \long\def\XINT_csvtol_finish_dviii #1#2#3#4#5#6#7#8#9{ #9}%
186 \long\def\XINT_csvtol_finish_dvii #1#2#3#4#5#6#7#8#9{ #9{#1}}%
187 \long\def\XINT_csvtol_finish_dvi #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}}%

```

```

188 \long\def\XINT_csvtol_finish_dv    #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}}%
189 \long\def\XINT_csvtol_finish_div  #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}{#4}}%
190 \long\def\XINT_csvtol_finish_diii #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}{#4}{#5}}%
191 \long\def\XINT_csvtol_finish_dii  #1#2#3#4#5#6#7#8#9%
192                                { #9{#1}{#2}{#3}{#4}{#5}{#6}}%
193 \long\def\XINT_csvtol_finish_di   #1#2#3#4#5#6#7#8#9%
194                                { #9{#1}{#2}{#3}{#4}{#5}{#6}{#7}}%

```

2.10 `\xintListWithSep`

`\xintListWithSep {sep}{a}{b}...{z}` returns a `\sep b \sep ... \sep z`

Included in release 1.04. The 'sep' can be `\par`'s: the macro `xintlistwithsep` etc... are all declared long. 'sep' does not have to be a single token. It is not expanded. The list may be a macro and it is expanded. 1.06 modifies the 'feature' of returning sep if the list is empty: the output is now empty in that case. (sep was not used for a one element list, but strangely it was for a zero-element list).

Use of `\Z` as delimiter was objectively an error, which I fix here in 1.09e, now the code uses `\xint_bye`.

```

195 \def\xintListWithSep {\romannumeral0\xintlistwithsep}%
196 \def\xintListWithSepNoExpand {\romannumeral0\xintlistwithsepnoexpand}%
197 \long\def\xintlistwithsep #1#2%
198     {\expandafter\XINT_lws\expandafter {\romannumeral-0#2}{#1}}%
199 \long\def\XINT_lws #1#2{\XINT_lws_start {#2}#1\xint_bye}%
200 \long\def\xintlistwithsepnoexpand #1#2{\XINT_lws_start {#1}#2\xint_bye}%
201 \long\def\XINT_lws_start #1#2%
202 {%
203     \xint_bye #2\XINT_lws_dont\xint_bye
204     \XINT_lws_loop_a {#2}{#1}%
205}%
206 \long\def\XINT_lws_dont\xint_bye\XINT_lws_loop_a #1#2{}%
207 \long\def\XINT_lws_loop_a #1#2#3%
208 {%
209     \xint_bye #3\XINT_lws_end\xint_bye
210     \XINT_lws_loop_b {#1}{#2#3}{#2}%
211}%
212 \long\def\XINT_lws_loop_b #1#2{\XINT_lws_loop_a {#1#2}}%
213 \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 expanding to {a}{b}...{z}}` (or 'tokens' `abcd...z`) returns the *i*th element (one pair of braces removed). The list is first 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 first 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), some 'improvements' made earlier in 1.09c were quite sub-efficient. Now uses `\xint_UDzerominusfork`, moved from `xint.sty`.

A bug in `\XINT_nthelt_finish` was introduced in 1.09i (2013/12/18): in order to pre-expand `\space`, I used an `\edef` as in quite a few other places of the code. But I didn't pay attention

2 Package *xinttools* implementation

that the earlier `\xint_gobble_iii` had to be transformed into a `\xint_gobble_ii`, as the `\space` was now already expanded to a `<space token>`. This bug meant that when the index `N` was `>` the length of the list the macro would gobble one token `:`. Fixed in 1.09n.

```
214 \def\xintNthElt          {\romannumeral0\xintnthelt }%
215 \def\xintNthEltNoExpand {\romannumeral0\xintntheltnoexpand }%
216 \def\xintnthelt #1#2%
217 {%
218   \expandafter\XINT_nthelt_a\the\numexpr #1\expandafter.%
219   \expandafter{\romannumeral-'\0#2}%
220 }%
221 \def\xintntheltnoexpand #1%
222 {%
223   \expandafter\XINT_nthelt_a\the\numexpr #1.%
224 }%
225 \def\XINT_nthelt_a #1#2.%
226 {%
227   \xint_UDzerominusfork
228     #1-{\XINT_nthelt_bzero}%
229     0#1{\XINT_nthelt_bneg {#2}}%
230     0-{\XINT_nthelt_bpos {#1#2}}%
231   \krof
232 }%
233 \long\def\XINT_nthelt_bzero #1%
234 {%
235   \XINT_length_loop 0.#1\xint_relax\xint_relax\xint_relax\xint_relax
236   \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
237 }%
238 \long\def\XINT_nthelt_bneg #1#2%
239 {%
240   \expandafter\XINT_nthelt_loop_a\expandafter {\the\numexpr #1\expandafter}%
241   \romannumeral0\xintrevwithbracesnoexpand {#2}%
242   \xint_relax\xint_relax\xint_relax\xint_relax
243   \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
244 }%
245 \long\def\XINT_nthelt_bpos #1#2%
246 {%
247   \XINT_nthelt_loop_a {#1}#2\xint_relax\xint_relax\xint_relax\xint_relax
248   \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
249 }%
250 \def\XINT_nthelt_loop_a #1%
251 {%
252   \ifnum #1>\xint_c_viii
253     \expandafter\XINT_nthelt_loop_b
254   \else
255     \XINT_nthelt_getit
256   \fi
257   {#1}%
258 }%
259 \long\def\XINT_nthelt_loop_b #1#2#3#4#5#6#7#8#9%
260 {%
261   \xint_gob_til_xint_relax #9\XINT_nthelt_silentend\xint_relax
262   \expandafter\XINT_nthelt_loop_a\expandafter{\the\numexpr #1-\xint_c_viii}%

```

```

263 }%
264 \def\XINT_nthelt_silentend #1\xint_bye { }%
265 \def\XINT_nthelt_getit\fi #1%
266 {%
267   \fi\expandafter\expandafter\expandafter\XINT_nthelt_finish
268   \csname xint_gobble_\romannumeral\numexpr#1-\xint_c_i\endcsname
269 }%
270 \long\edef\XINT_nthelt_finish #1#2\xint_bye
271   {\noexpand\xint_gob_til_xint_relax #1\noexpand\expandafter\space
272     \noexpand\xint_gobble_ii\xint_relax\space #1}%

```

2.12 \xintKeep

First included in release 1.09m.

\xintKeep {i}{stuff 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. 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.

```

273 \def\xintKeep          {\romannumeral0\xintkeep }%
274 \def\xintKeepNoExpand {\romannumeral0\xintkeepnoexpand }%
275 \def\xintkeep #1#2%
276 {%
277   \expandafter\XINT_keep_a\the\numexpr #1\expandafter.%
278   \expandafter{\romannumeral-‘0#2}%
279 }%
280 \def\xintkeepnoexpand #1%
281 {%
282   \expandafter\XINT_keep_a\the\numexpr #1.%
283 }%
284 \def\XINT_keep_a #1#2.%
285 {%
286   \xint_UDzerominusfork
287     #1-{\expandafter\space\xint_gobble_i }%
288     0#1{\XINT_keep_bneg_a {#2}}%
289     0-{\XINT_keep_bpos {#1#2}}%
290   \krof
291 }%
292 \long\def\XINT_keep_bneg_a #1#2%
293 {%
294   \expandafter\XINT_keep_bneg_b \the\numexpr \xintLength{#2}-#1.{#2}%
295 }%
296 \def\XINT_keep_bneg_b #1#2.%
297 {%
298   \xint_UDzerominusfork
299     #1-{\xint_firstofone_thenstop }%
300     0#1{\xint_firstofone_thenstop }%
301     0-{\XINT_trim_bpos {#1#2}}%
302   \krof
303 }%

```

2 Package `xinttools` implementation

```
304 \long\def\XINT_keep_bpos #1#2%
305 {%
306     \XINT_keep_loop_a {#1}{#2\xint_relax\xint_relax\xint_relax\xint_relax
307         \xint_relax\xint_relax\xint_relax\xint_bye
308 }%
309 \def\XINT_keep_loop_a #1%
310 {%
311     \ifnum #1>\xint_c_vi
312         \expandafter\XINT_keep_loop_b
313     \else
314         \XINT_keep_finish
315     \fi
316     {#1}%
317 }%
318 \long\def\XINT_keep_loop_b #1#2#3#4#5#6#7#8#9%
319 {%
320     \xint_gob_til_xint_relax #9\XINT_keep_enda\xint_relax
321     \expandafter\XINT_keep_loop_c\expandafter{\the\numexpr #1-\xint_c_vii}%
322     {{#3}{#4}{#5}{#6}{#7}{#8}{#9}}{#2}%
323 }%
324 \long\def\XINT_keep_loop_c #1#2#3{\XINT_keep_loop_a {#1}{#3#2}}%
325 \long\def\XINT_keep_enda\xint_relax
326     \expandafter\XINT_keep_loop_c\expandafter #1#2#3#4\xint_bye
327 {%
328     \XINT_keep_endb #4\W\W\W\W\W\W\Z #2{#3}%
329 }%
330 \def\XINT_keep_endb #1#2#3#4#5#6#7\Z
331 {%
332     \xint_gob_til_W
333     #1\XINT_keep_endc_
334     #2\XINT_keep_endc_i
335     #3\XINT_keep_endc_ii
336     #4\XINT_keep_endc_iii
337     #5\XINT_keep_endc_iv
338     #6\XINT_keep_endc_v
339     \W\XINT_keep_endc_vi\Z
340 }%
341 \long\def\XINT_keep_endc_ #1\Z #2#3#4#5#6#7#8#9{ #9}%
342 \long\def\XINT_keep_endc_i #1\Z #2#3#4#5#6#7#8#9{ #9{#2}}%
343 \long\def\XINT_keep_endc_ii #1\Z #2#3#4#5#6#7#8#9{ #9{#2}{#3}}%
344 \long\def\XINT_keep_endc_iii #1\Z #2#3#4#5#6#7#8#9{ #9{#2}{#3}{#4}}%
345 \long\def\XINT_keep_endc_iv #1\Z #2#3#4#5#6#7#8#9{ #9{#2}{#3}{#4}{#5}}%
346 \long\def\XINT_keep_endc_v #1\Z #2#3#4#5#6#7#8#9{ #9{#2}{#3}{#4}{#5}{#6}}%
347 \long\def\XINT_keep_endc_vi\Z #1#2#3#4#5#6#7#8{ #8{#1}{#2}{#3}{#4}{#5}{#6}}%
348 \long\def\XINT_keep_finish\fi #1#2#3#4#5#6#7#8#9\xint_bye
349 {%
350     \fi\XINT_keep_finish_loop_a {#1}{#3}{#4}{#5}{#6}{#7}{#8}\Z {#2}%
351 }%
352 \def\XINT_keep_finish_loop_a #1%
353 {%
354     \xint_gob_til_zero #1\XINT_keep_finish_z0%
355     \expandafter\XINT_keep_finish_loop_b\expandafter
```



```

356      {\the\numexpr #1-\xint_c_i}%
357 }%
358 \long\def\xINT_keep_finish_z0%
359      \expandafter\xINT_keep_finish_loop_b\expandafter #1#2#3\Z #4{ #4#2}%
360 \long\def\xINT_keep_finish_loop_b #1#2#3%
361 {%
362      \xint_gob_til_xint_relax #3\xINT_keep_finish_exit\xint_relax
363      \XINT_keep_finish_loop_c {#1}{#2}{#3}%
364 }%
365 \long\def\xINT_keep_finish_exit\xint_relax
366      \XINT_keep_finish_loop_c #1#2#3\Z #4{ #4#2}%
367 \long\def\xINT_keep_finish_loop_c #1#2#3%
368      {\XINT_keep_finish_loop_a {#1}{#2}{#3}}}%

```

2.13 `\xintTrim`

First included in release 1.09m.

`\xintTrim {i}{stuff 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.

```

369 \def\xintTrim      {\romannumeral0\xinttrim }%
370 \def\xintTrimNoExpand {\romannumeral0\xinttrimnoexpand }%
371 \def\xinttrim #1#2%
372 {%
373      \expandafter\xINT_trim_a\the\numexpr #1\expandafter.%
374      \expandafter{\romannumeral-‘0#2}%
375 }%
376 \def\xinttrimnoexpand #1%
377 {%
378      \expandafter\xINT_trim_a\the\numexpr #1.%
379 }%
380 \def\xINT_trim_a #1#2.%
381 {%
382      \xint_UDzerominusfork
383      #1-{\xint_firstofone_thenstop }%
384      0#1{\XINT_trim_bneg_a {#2}}%
385      0-{\XINT_trim_bpos {#1#2}}%
386      \krof
387 }%
388 \long\def\xINT_trim_bneg_a #1#2%
389 {%
390      \expandafter\xINT_trim_bneg_b \the\numexpr \xintLength{#2}-#1.{#2}%
391 }%
392 \def\xINT_trim_bneg_b #1#2.%
393 {%
394      \xint_UDzerominusfork
395      #1-{\expandafter\space\xint_gobble_i }%
396      0#1{\expandafter\space\xint_gobble_i }%

```

```

397      0-{\XINT_keep_bpos {#1#2}}%
398      \krof
399 }%
400 \long\def\XINT_trim_bpos #1#2%
401 {%
402     \XINT_trim_loop_a {#1}#2\xint_relax\xint_relax\xint_relax\xint_relax
403         \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
404 }%
405 \def\XINT_trim_loop_a #1%
406 {%
407     \ifnum #1>\xint_c_vii
408         \expandafter\XINT_trim_loop_b
409     \else
410         \XINT_trim_finish
411     \fi
412     {#1}%
413 }%
414 \long\def\XINT_trim_loop_b #1#2#3#4#5#6#7#8#9%
415 {%
416     \xint_gob_til_xint_relax #9\XINT_trim_silentend\xint_relax
417     \expandafter\XINT_trim_loop_a\expandafter{\the\numexpr #1-\xint_c_viii}%
418 }%
419 \def\XINT_trim_silentend #1\xint_bye { }%
420 \def\XINT_trim_finish\fi #1%
421 {%
422     \fi\expandafter\expandafter\expandafter\XINT_trim_finish_a
423     \expandafter\expandafter\expandafter\space % avoids brace removal
424     \csname xint_gobble_\romannumeral\numexpr#1\endcsname
425 }%
426 \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 ff-expanded. The list is first expanded and may thus be a macro. Introduced with release 1.04. Modified in 1.09e to not use `\Z` but rather `\xint_bye`.

```

427 \def\xintApply          {\romannumeral0\xintapply }%
428 \def\xintApplyNoExpand {\romannumeral0\xintapplynoexpand }%
429 \long\def\xintapply #1#2%
430 {%
431     \expandafter\XINT_apply\expandafter {\romannumeral-‘0#2}%
432     {#1}%
433 }%
434 \long\def\XINT_apply #1#2{\XINT_apply_loop_a {}{#2}#1\xint_bye }%
435 \long\def\xintapplynoexpand #1#2{\XINT_apply_loop_a {}{#1}#2\xint_bye }%
436 \long\def\XINT_apply_loop_a #1#2#3%
437 {%
438     \xint_bye #3\XINT_apply_end\xint_bye
439     \expandafter
440     \XINT_apply_loop_b
441     \expandafter {\romannumeral-‘0#2{#3}}{#1}{#2}%
442 }%

```

```

443 \long\def\XINT_apply_loop_b #1#2{\XINT_apply_loop_a {#2{#1}}}%
444 \long\def\XINT_apply_end\xint_bye\expandafter\XINT_apply_loop_b
445   \expandafter #1#2#3{ #2}%

```

2.15 \xintApplyUnbraced

`\xintApplyUnbraced {\macro}{a}{b}...{z}` returns `\macro{a}...\macro{z}` where each instance of `\macro` is expanded using `\romannumeral-‘0`. The second argument may be a macro as it is first expanded itself (fully). No braces are added: this allows for example a non-expandable `\def` in `\macro`, without having to do `\gdef`. The list is first expanded. Introduced with release 1.06b. Define `\macro` to start with a space if it is not expandable or its execution should be delayed only when all of `\macro{a}...\macro{z}` is ready.

Modified in 1.09e to use `\xint_bye` rather than `\Z`.

```

446 \def\xintApplyUnbraced {\romannumeral0\xintapplyunbraced}%
447 \def\xintApplyUnbracedNoExpand {\romannumeral0\xintapplyunbracednoexpand}%
448 \long\def\xintapplyunbraced #1#2%
449 {%
450   \expandafter\XINT_applyunbr\expandafter {\romannumeral-‘0#2}%
451   {#1}%
452}%
453 \long\def\XINT_applyunbr #1#2{\XINT_applyunbr_loop_a {}{#2}#1\xint_bye}%
454 \long\def\xintapplyunbracednoexpand #1#2%
455   {\XINT_applyunbr_loop_a {}{#1}#2\xint_bye}%
456 \long\def\XINT_applyunbr_loop_a #1#2#3%
457 {%
458   \xint_bye #3\XINT_applyunbr_end\xint_bye
459   \expandafter\XINT_applyunbr_loop_b
460   \expandafter {\romannumeral-‘0#2{#3}}{#1}{#2}%
461}%
462 \long\def\XINT_applyunbr_loop_b #1#2{\XINT_applyunbr_loop_a {#2}#1}%
463 \long\def\XINT_applyunbr_end\xint_bye\expandafter\XINT_applyunbr_loop_b
464   \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 generate thousands of terms then. Here also, let’s use `\xint_bye` rather than `\Z` as delimiter (1.09e; necessary change as `#1` is used prior to being expanded, thus `\Z` might very well arise here as a macro).

```

465 \def\xintSeq {\romannumeral0\xintseq}%
466 \def\xintseq #1{\XINT_seq_chkopt #1\xint_bye}%
467 \def\XINT_seq_chkopt #1%
468 {%
469   \ifx [#1\expandafter\XINT_seq_opt
470     \else\expandafter\XINT_seq_noopt
471   \fi #1%
472}%
473 \def\XINT_seq_noopt #1\xint_bye #2%
474 {%
475   \expandafter\XINT_seq\expandafter
476     {\the\numexpr#1\expandafter}\expandafter{\the\numexpr #2}%

```

```

477 }%
478 \def\XINT_seq #1#2%
479 {%
480   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
481     \expandafter\xint_firstoftwo_thenstop
482   \or
483     \expandafter\XINT_seq_p
484   \else
485     \expandafter\XINT_seq_n
486   \fi
487   {#2}{#1}%
488 }%
489 \def\XINT_seq_p #1#2%
490 {%
491   \ifnum #1>#2
492     \expandafter\expandafter\expandafter\XINT_seq_p
493   \else
494     \expandafter\XINT_seq_e
495   \fi
496   \expandafter{\the\numexpr #1-\xint_c_i}{#2}{#1}%
497 }%
498 \def\XINT_seq_n #1#2%
499 {%
500   \ifnum #1<#2
501     \expandafter\expandafter\expandafter\XINT_seq_n
502   \else
503     \expandafter\XINT_seq_e
504   \fi
505   \expandafter{\the\numexpr #1+\xint_c_i}{#2}{#1}%
506 }%
507 \def\XINT_seq_e #1#2#3{ }%
508 \def\XINT_seq_opt [\xint_bye #1]#2#3%
509 {%
510   \expandafter\XINT_seqo\expandafter
511   {\the\numexpr #2\expandafter}\expandafter
512   {\the\numexpr #3\expandafter}\expandafter
513   {\the\numexpr #1}%
514 }%
515 \def\XINT_seqo #1#2%
516 {%
517   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
518     \expandafter\XINT_seqo_a
519   \or
520     \expandafter\XINT_seqo_pa
521   \else
522     \expandafter\XINT_seqo_na
523   \fi
524   {#1}{#2}%
525 }%
526 \def\XINT_seqo_a #1#2#3{ {#1}}%
527 \def\XINT_seqo_o #1#2#3#4{ #4}%
528 \def\XINT_seqo_pa #1#2#3%

```

2 Package *xinttools* implementation

```

529 {%
530   \ifcase\ifnum #3=\xint_c_ 0\else\ifnum #3>\xint_c_ 1\else -1\fi\fi\space
531     \expandafter\XINT_seqo_o
532   \or
533     \expandafter\XINT_seqo_pb
534   \else
535     \xint_afterfi{\expandafter\space\xint_gobble_iv}%
536   \fi
537   {#1}{#2}{#3}{{#1}}%
538 }%
539 \def\XINT_seqo_pb #1#2#3%
540 {%
541   \expandafter\XINT_seqo_pc\expandafter{\the\numexpr #1+#3}{#2}{#3}%
542 }%
543 \def\XINT_seqo_pc #1#2%
544 {%
545   \ifnum #1>#2
546     \expandafter\XINT_seqo_o
547   \else
548     \expandafter\XINT_seqo_pd
549   \fi
550   {#1}{#2}%
551 }%
552 \def\XINT_seqo_pd #1#2#3#4{\XINT_seqo_pb {#1}{#2}{#3}{#4{#1}}}%
553 \def\XINT_seqo_na #1#2#3%
554 {%
555   \ifcase\ifnum #3=\xint_c_ 0\else\ifnum #3>\xint_c_ 1\else -1\fi\fi\space
556     \expandafter\XINT_seqo_o
557   \or
558     \xint_afterfi{\expandafter\space\xint_gobble_iv}%
559   \else
560     \expandafter\XINT_seqo_nb
561   \fi
562   {#1}{#2}{#3}{{#1}}%
563 }%
564 \def\XINT_seqo_nb #1#2#3%
565 {%
566   \expandafter\XINT_seqo_nc\expandafter{\the\numexpr #1+#3}{#2}{#3}%
567 }%
568 \def\XINT_seqo_nc #1#2%
569 {%
570   \ifnum #1<#2
571     \expandafter\XINT_seqo_o
572   \else
573     \expandafter\XINT_seqo_nd
574   \fi
575   {#1}{#2}%
576 }%
577 \def\XINT_seqo_nd #1#2#3#4{\XINT_seqo_nb {#1}{#2}{#3}{#4{#1}}}%

```

2.17 `\xintloop`, `\xintbreakloop`, `\xintbreakloopaddo`, `\xintloopskiptonext`

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

```

578 \long\def\xintloop #1#2\repeat {#1#2\xintloop_again\fi\xint_gobble_i {#1#2}}%
579 \long\def\xintloop_again\fi\xint_gobble_i #1{\fi
580      #1\xintloop_again\fi\xint_gobble_i {#1}}%
581 \long\def\xintbreakloop #1\xintloop_again\fi\xint_gobble_i #2{}%
582 \long\def\xintbreakloopaddo #1#2\xintloop_again\fi\xint_gobble_i #3{#1}%
583 \long\def\xintloopskiptonext #1\xintloop_again\fi\xint_gobble_i #2{%
584      #2\xintloop_again\fi\xint_gobble_i {#2}}%

```

2.18 `\xintiloop`, `\xintiloopindex`, `\xintouteriloopindex`, `\xintbreakiloop`, `\xintbreakiloopaddo`, `\xintiloopskiptonext`, `\xintiloopskipandredo`

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

```

585 \def\xintiloop [#1+#2]{%
586   \expandafter\xintiloop_a\the\numexpr #1\expandafter.\the\numexpr #2.}%
587 \long\def\xintiloop_a #1.#2.#3#4\repeat{%
588   #3#4\xintiloop_again\fi\xint_gobble_iii {#1}{#2}{#3#4}}%
589 \def\xintiloop_again\fi\xint_gobble_iii #1#2{%
590   \fi\expandafter\xintiloop_again_b\the\numexpr#1+#2.#2.}%
591 \long\def\xintiloop_again_b #1.#2.#3{%
592   #3\xintiloop_again\fi\xint_gobble_iii {#1}{#2}{#3}}%
593 \long\def\xintbreakiloop #1\xintiloop_again\fi\xint_gobble_iii #2#3#4{%
594 \long\def\xintbreakiloopaddo
595   #1.#2\xintiloop_again\fi\xint_gobble_iii #3#4#5{#1}%
596 \long\def\xintiloopindex #1\xintiloop_again\fi\xint_gobble_iii #2%
597   {#2#1\xintiloop_again\fi\xint_gobble_iii {#2}}%
598 \long\def\xintouteriloopindex #1\xintiloop_again
599   #2\xintiloop_again\fi\xint_gobble_iii #3%
600   {#3#1\xintiloop_again #2\xintiloop_again\fi\xint_gobble_iii {#3}}%
601 \long\def\xintiloopskiptonext #1\xintiloop_again\fi\xint_gobble_iii #2#3{%
602   \expandafter\xintiloop_again_b \the\numexpr#2+#3.#3.}%
603 \long\def\xintiloopskipandredo #1\xintiloop_again\fi\xint_gobble_iii #2#3#4{%
604   #4\xintiloop_again\fi\xint_gobble_iii {#2}{#3}{#4}}%

```

2.19 `\XINT_xflet`

1.09e [2013/10/29]: we expand fully unbraced tokens and swallow arising space tokens until the dust settles. For treating cases `{<blank>\x<blank>y...}`, with guaranteed expansion of the `\x` (which may itself give space tokens), a simpler approach is possible with doubled `\romannumeral-0`, this is what I first did, but it had the feature that `<sptoken><sptoken>\x` would not expand the `\x`. At any rate, `<sptoken>`'s before the list terminator `z` were all correctly moved out of the way, hence the stuff was robust for use in (the then current versions of) `\xintApplyInline` and `\xintFor`. Although **two** space tokens would need devilishly prepared input, nevertheless I decided to also survive that, so here the method is a bit more complicated. But it simplifies things on the caller side.

```

605 \def\XINT_xflet #1%
606 {%
607   \def\XINT_xflet_macro {#1}\XINT_xflet_zapsp

```

```

608 }%
609 \def\XINT_xflet_zapsp
610 {%
611   \expandafter\futurelet\expandafter\XINT_token
612   \expandafter\XINT_xflet_sp?\romannumeral-'0%
613 }%
614 \def\XINT_xflet_sp?
615 {%
616   \ifx\XINT_token\XINT_sptoken
617     \expandafter\XINT_xflet_zapsp
618   \else\expandafter\XINT_xflet_zapspB
619   \fi
620 }%
621 \def\XINT_xflet_zapspB
622 {%
623   \expandafter\futurelet\expandafter\XINT_tokenB
624   \expandafter\XINT_xflet_spB?\romannumeral-'0%
625 }%
626 \def\XINT_xflet_spB?
627 {%
628   \ifx\XINT_tokenB\XINT_sptoken
629     \expandafter\XINT_xflet_zapspB
630   \else\expandafter\XINT_xflet_eq?
631   \fi
632 }%
633 \def\XINT_xflet_eq?
634 {%
635   \ifx\XINT_token\XINT_tokenB
636     \expandafter\XINT_xflet_macro
637   \else\expandafter\XINT_xflet_zapsp
638   \fi
639 }%

```

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 expands (fully) its second (list) argument first, which may thus be encapsulated in a macro.

Release 1.09c has a new `\xintApplyInline`: the new version, while not expandable, is designed to survive when the applied macro closes a group, as is the case in `alignemnts` when it contains a `&` or `\\`. It uses catcode 3 Z as list terminator. Don't use it among the list items.

1.09d: the bug which was discovered in `\xintFor*` regarding space tokens at the very end of the item list also was in `\xintApplyInline`. The new version will expand unbraced item elements and this is in fact convenient to simulate insertion of lists in others.

1.09e: the applied macro is allowed to be long, with items (or the first fixed arguments of the macro, passed together with it as #1 to `\xintApplyInline`) containing explicit `\par`'s. (1.09g: some missing `\long`'s added)

1.09f: terminator used to be `z`, now `Z` (still catcode 3).

```

640 \catcode'Z 3
641 \long\def\xintApplyInline #1#Z%
642 {%

```

```

643 \long\expandafter\def\expandafter\XINT_inline_macro
644 \expandafter ##\expandafter 1\expandafter {#1{##1}}%
645 \XINT_xflet\XINT_inline_b #2Z% this Z has catcode 3
646 }%
647 \def\XINT_inline_b
648 {%
649     \ifx\XINT_token Z\expandafter\xint_gobble_i
650     \else\expandafter\XINT_inline_d\fi
651 }%
652 \long\def\XINT_inline_d #1%
653 {%
654     \long\def\XINT_item{#1}\XINT_xflet\XINT_inline_e
655 }%
656 \def\XINT_inline_e
657 {%
658     \ifx\XINT_token Z\expandafter\XINT_inline_w
659     \else\expandafter\XINT_inline_f\fi
660 }%
661 \def\XINT_inline_f
662 {%
663     \expandafter\XINT_inline_g\expandafter{\XINT_inline_macro {##1}}%
664 }%
665 \long\def\XINT_inline_g #1%
666 {%
667     \expandafter\XINT_inline_macro\XINT_item
668     \long\def\XINT_inline_macro ##1{#1}\XINT_inline_d
669 }%
670 \def\XINT_inline_w #1%
671 {%
672     \expandafter\XINT_inline_macro\XINT_item
673 }%

```

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, expanded fully.

1.09d: [2013/10/22] `\xintFor*` crashed when a space token was at the very end of the list. It is crucial in this code to not let the ending Z be picked up as a macro parameter without knowing in advance that it is its turn. So, we conscientiously clean out of the way space tokens, but also we ff-expand with `\romannumeral-‘0` (unbraced) items, a process which may create new space tokens, so it is iterated. As unbraced items are expanded, it is easy to simulate insertion of a list in another. Unbraced items consecutive to an even (non-zero) number of space tokens will not get expanded.

1.09e: [2013/10/29] does this better, no difference between an even or odd number of explicit consecutive space tokens. Normal situations anyhow only create at most one space token, but well. There was a feature in `\xintFor` (not `\xintFor*`) from 1.09c that it treated an empty list as a list with one, empty, item. This feature is kept in 1.09e, knowingly... Also, macros are made long, hence the iterated text may contain `\par` and also the looped over items. I thought about providing

2 Package *xinttools* implementation

some macro expanding to the loop count, but as the `\xintFor` is not expandable anyhow, there is no loss of generality if the iterated commands do themselves the bookkeeping using a count or a LaTeX counter, and deal with nesting or other problems. I can't do *everything*!

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. Compatibility with `\XINT_forever`, the necessity to prevent unwanted brace stripping, and shared code with `\xintFor*`, make this all a delicate balancing act. 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`.

The 1.09f `\xintFor` and `\xintFor*` modified the value of `\count 255` which was silly, 1.09g used `\XINT_count`, but requiring a `\count` only for that was also silly, 1.09h just uses `\numexpr` (all of that was only to get rid simply of a possibly space in #2...).

1.09ka [2014/02/05] corrects the following bug: `\xintBreakFor` and `\xintBreakForAndDo` could not be used in the last iteration.

```
674 \def\XINT_tmpa #1#2{\ifnum #2<#1 \xint_afterfi {{#####2}}\fi}%
675 \def\XINT_tmpb #1#2{\ifnum #1<#2 \xint_afterfi {{#####2}}\fi}%
676 \def\XINT_tmpc #1%
677 {%
678   \expandafter\edef \csname XINT_for_left#1\endcsname
679     {\xintApplyUnbraced {\XINT_tmpa #1}{123456789}}%
680   \expandafter\edef \csname XINT_for_right#1\endcsname
681     {\xintApplyUnbraced {\XINT_tmpb #1}{123456789}}%
682 }%
683 \xintApplyInline \XINT_tmpc {123456789}%
684 \long\def\xintBreakFor      #1Z{%}%
685 \long\def\xintBreakForAndDo #1#2Z{#1}%
686 \def\xintFor {\let\xintifForFirst\xint_firstoftwo
687   \futurelet\XINT_token\XINT_for_ifstar }%
688 \def\XINT_for_ifstar {\ifx\XINT_token*\expandafter\XINT_forx
689   \else\expandafter\XINT_for \fi }%
690 \catcode\U 3 % with numexpr
691 \catcode\V 3 % with xintfrac.sty (xint.sty not enough)
692 \catcode\D 3 % with dimexpr
693 % \def\XINT_flet #1%
694 % {%
695 %   \def\XINT_flet_macro {#1}\XINT_flet_zapsp
696 % }%
697 \def\XINT_flet_zapsp
698 {%
699   \futurelet\XINT_token\XINT_flet_sp?
700 }%
701 \def\XINT_flet_sp?
702 {%
703   \ifx\XINT_token\XINT_sptoken
704     \xint_afterfi{\expandafter\XINT_flet_zapsp\romannumeral0}%
705   \else\expandafter\XINT_flet_macro
706   \fi
```

2 Package *xinttools* implementation

```

707 }%
708 \long\def\XINT_for #1#2in#3#4#5%
709 {%
710   \expandafter\XINT_toks\expandafter
711     {\expandafter\XINT_for_d\the\numexpr #2\relax {#5}}%
712   \def\XINT_flet_macro {\expandafter\XINT_for_forever?\space}%
713   \expandafter\XINT_flet_zapsp #3Z%
714 }%
715 \def\XINT_for_forever? #1Z%
716 {%
717   \ifx\XINT_token U\XINT_to_forever\fi
718   \ifx\XINT_token V\XINT_to_forever\fi
719   \ifx\XINT_token D\XINT_to_forever\fi
720   \expandafter\the\expandafter\XINT_toks\romannumeral0\xintcsvtolist {#1}Z%
721 }%
722 \def\XINT_to_forever\fi #1\xintcsvtolist #2{\fi \XINT_forever #2}%
723 \long\def\XINT_forx *#1#2in#3#4#5%
724 {%
725   \expandafter\XINT_toks\expandafter
726     {\expandafter\XINT_forx_d\the\numexpr #2\relax {#5}}%
727   \XINT_xflet\XINT_forx_forever? #3Z%
728 }%
729 \def\XINT_forx_forever?
730 {%
731   \ifx\XINT_token U\XINT_to_forxever\fi
732   \ifx\XINT_token V\XINT_to_forxever\fi
733   \ifx\XINT_token D\XINT_to_forxever\fi
734   \XINT_forx_empty?
735 }%
736 \def\XINT_to_forxever\fi #1\XINT_forx_empty? {\fi \XINT_forever }%
737 \catcode\U 11
738 \catcode\D 11
739 \catcode\V 11
740 \def\XINT_forx_empty?
741 {%
742   \ifx\XINT_token Z\expandafter\xintBreakFor\fi
743   \the\XINT_toks
744 }%
745 \long\def\XINT_for_d #1#2#3%
746 {%
747   \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
748   \XINT_toks {{#3}}%
749   \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
750     \the\XINT_toks \csname XINT_for_right#1\endcsname }%
751   \XINT_toks {\XINT_x\let\xintifForFirst\xint_secondoftwo\XINT_for_d #1{#2}}%
752   \futurelet\XINT_token\XINT_for_last?
753 }%
754 \long\def\XINT_forx_d #1#2#3%
755 {%
756   \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
757   \XINT_toks {{#3}}%
758   \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname

```

```

759          \the\XINT_toks \csname XINT_for_right#1\endcsname }%
760 \XINT_toks {\XINT_x\let\xintifForFirst\xint_secondoftwo\XINT_forx_d #1{#2}}%
761 \XINT_xflet\XINT_for_last?
762 }%
763 \def\XINT_for_last?
764 {%
765   \let\xintifForLast\xint_secondoftwo
766   \ifx\XINT_token Z\let\xintifForLast\xint_firstoftwo
767   \xint_afterfi{\xintBreakForAndDo{\XINT_x\xint_gobble_i Z}}\fi
768   \the\XINT_toks
769 }%

```

2.22 \XINT_forever, \xintintegers, \xintdimensions, \xinrationals

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 has \xintZapSpacesB which helps getting rid of spaces for the \xinrationals case (the other cases end up inside a \numexpr, or \dimexpr, so not necessary).

```

770 \catcode'U 3
771 \catcode'D 3
772 \catcode'V 3
773 \let\xintegers      U%
774 \let\xintintegers  U%
775 \let\xintdimensions D%
776 \let\xinrationals  V%
777 \def\XINT_forever #1%
778 {%
779   \expandafter\XINT_forever_a
780   \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi a\expandafter\endcsname
781   \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi i\expandafter\endcsname
782   \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi \endcsname
783 }%
784 \catcode'U 11
785 \catcode'D 11
786 \catcode'V 11
787 \def\XINT_?expr_Ua #1#2%
788   {\expandafter{\expandafter\numexpr\the\numexpr #1\expandafter\relax
789     \expandafter\relax\expandafter}%
790   \expandafter{\the\numexpr #2}}%
791 \def\XINT_?expr_Da #1#2%
792   {\expandafter{\expandafter\dimexpr\number\dimexpr #1\expandafter\relax
793     \expandafter s\expandafter p\expandafter\relax\expandafter}%
794   \expandafter{\number\dimexpr #2}}%
795 \catcode'Z 11
796 \def\XINT_?expr_Va #1#2%
797 {%
798   \expandafter\XINT_?expr_Vb\expandafter
799   {\romannumeral-'0\xintraawithzeros{\xintZapSpacesB{#2}}}%
800   {\romannumeral-'0\xintraawithzeros{\xintZapSpacesB{#1}}}%
801 }%
802 \catcode'Z 3
803 \def\XINT_?expr_Vb #1#2{\expandafter\XINT_?expr_Vc #2.#1}%

```

2 Package *xinttools* implementation

```

804 \def\XINT_?expr_Vc #1/#2.#3/#4.%
805 {%
806     \xintifEq {#2}{#4}%
807     {\XINT_?expr_Vf {#3}{#1}{#2}}%
808     {\expandafter\XINT_?expr_Vd\expandafter
809     {\romannumeral0\xintiimul {#2}{#4}}%
810     {\romannumeral0\xintiimul {#1}{#4}}%
811     {\romannumeral0\xintiimul {#2}{#3}}%
812     }%
813 }%
814 \def\XINT_?expr_Vd #1#2#3{\expandafter\XINT_?expr_Ve\expandafter {#2}{#3}{#1}}%
815 \def\XINT_?expr_Ve #1#2{\expandafter\XINT_?expr_Vf\expandafter {#2}{#1}}%
816 \def\XINT_?expr_Vf #1#2#3{{#2/#3}{0}{#1}{#2}{#3}}%
817 \def\XINT_?expr_Ui {\numexpr 1\relax}{1}}%
818 \def\XINT_?expr_Di {\dimexpr 0pt\relax}{65536}}%
819 \def\XINT_?expr_Vi {{1/1}{0111}}%
820 \def\XINT_?expr_U #1#2%
821     {\expandafter{\expandafter\numexpr\the\numexpr #1+#2\relax\relax}{#2}}%
822 \def\XINT_?expr_D #1#2%
823     {\expandafter{\expandafter\dimexpr\the\numexpr #1+#2\relax sp\relax}{#2}}%
824 \def\XINT_?expr_V #1#2{\XINT_?expr_Vx #2}%
825 \def\XINT_?expr_Vx #1#2%
826 {%
827     \expandafter\XINT_?expr_Vy\expandafter
828     {\romannumeral0\xintiiadd {#1}{#2}{#2}}%
829 }%
830 \def\XINT_?expr_Vy #1#2#3#4%
831 {%
832     \expandafter{\romannumeral0\xintiiadd {#3}{#1}/#4}{#1}{#2}{#3}{#4}}%
833 }%
834 \def\XINT_forever_a #1#2#3#4%
835 {%
836     \ifx #4[\expandafter\XINT_forever_opt_a
837     \else\expandafter\XINT_forever_b
838     \fi #1#2#3#4%
839 }%
840 \def\XINT_forever_b #1#2#3Z{\expandafter\XINT_forever_c\the\XINT_toks #2#3}%
841 \long\def\XINT_forever_c #1#2#3#4#5%
842     {\expandafter\XINT_forever_d\expandafter #2#4#5{#3}Z}%
843 \def\XINT_forever_opt_a #1#2#3[#4+#5]#6Z%
844 {%
845     \expandafter\expandafter\expandafter
846     \XINT_forever_opt_c\expandafter\the\expandafter\XINT_toks
847     \romannumeral-‘0#1{#4}{#5}#3%
848 }%
849 \long\def\XINT_forever_opt_c #1#2#3#4#5#6{\XINT_forever_d #2{#4}{#5}#6{#3}Z}%
850 \long\def\XINT_forever_d #1#2#3#4#5%
851 {%
852     \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#5}%
853     \XINT_toks {{#2}}%
854     \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
855         \the\XINT_toks \csname XINT_for_right#1\endcsname }%

```

```

856 \XINT_x
857 \let\xintifForFirst\xint_secondoftwo
858 \expandafter\XINT_forever_d\expandafter #1\romannumeral-‘0#4{#2}{#3}#4{#5}%
859 }%

```

2.23 \xintForpair, \xintForthree, \xintForfour

1.09c: I don’t know yet if {a}{b} is better for the user or worse than (a,b). I prefer the former. I am not very motivated to deal with spaces in the (a,b) approach which is the one (currently) followed here.

[2013/11/02] 1.09f: I may not have been very motivated in 1.09c, but since then I developed the \xintZapSpaces/\xintZapSpacesB tools (much to my satisfaction). Based on this, and better parameter texts, \xintForpair and its cousins now handle spaces very satisfactorily (this relies partly on the new \xintCSVtoList which makes use of \xintZapSpacesB). 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.

```

860 \catcode‘j 3
861 \long\def\xintForpair #1#2#3in#4#5#6%
862 {%
863   \let\xintifForFirst\xint_firstoftwo
864   \XINT_toks {\XINT_forpair_d #2{#6}}%
865   \expandafter\the\expandafter\XINT_toks #4jZ%
866 }%
867 \long\def\XINT_forpair_d #1#2#3(#4)#5%
868 {%
869   \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
870   \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
871   \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
872     \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_i\endcsname}%
873   \let\xintifForLast\xint_secondoftwo
874   \ifx #5j\expandafter\xint_firstoftwo
875     \else\expandafter\xint_secondoftwo
876   \fi
877   {\let\xintifForLast\xint_firstoftwo
878     \xintBreakForAndDo {\XINT_x \xint_gobble_i Z}}%
879   \XINT_x
880   \let\xintifForFirst\xint_secondoftwo\XINT_forpair_d #1{#2}%
881 }%
882 \long\def\xintForthree #1#2#3in#4#5#6%
883 {%
884   \let\xintifForFirst\xint_firstoftwo
885   \XINT_toks {\XINT_forthree_d #2{#6}}%
886   \expandafter\the\expandafter\XINT_toks #4jZ%
887 }%
888 \long\def\XINT_forthree_d #1#2#3(#4)#5%
889 {%
890   \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
891   \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
892   \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
893     \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_ii\endcsname}%
894   \let\xintifForLast\xint_secondoftwo

```

```

895 \ifx #5j\expandafter\xint_firstoftwo
896   \else\expandafter\xint_secondoftwo
897 \fi
898 {\let\xintifForLast\xint_firstoftwo
899   \xintBreakForAndDo {\XINT_x \xint_gobble_i Z}}%
900 \XINT_x
901 \let\xintifForFirst\xint_secondoftwo\XINT_forthree_d #1{#2}%
902 }%
903 \long\def\xintForfour #1#2#3in#4#5#6%
904 {%
905   \let\xintifForFirst\xint_firstoftwo
906   \XINT_toks {\XINT_forfour_d #2{#6}}%
907   \expandafter\the\expandafter\XINT_toks #4jZ%
908 }%
909 \long\def\XINT_forfour_d #1#2#3(#4)#5%
910 {%
911   \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
912   \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
913   \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
914     \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_iii\endcsname}%
915   \let\xintifForLast\xint_secondoftwo
916   \ifx #5j\expandafter\xint_firstoftwo
917     \else\expandafter\xint_secondoftwo
918 \fi
919 {\let\xintifForLast\xint_firstoftwo
920   \xintBreakForAndDo {\XINT_x \xint_gobble_i Z}}%
921 \XINT_x
922 \let\xintifForFirst\xint_secondoftwo\XINT_forfour_d #1{#2}%
923 }%
924 \catcode'Z 11
925 \catcode'j 11

```

2.24 \xintAssign, \xintAssignArray, \xintDigitsOf

\xintAssign {a}{b}..{z}\to A\B...\Z or \xintAssignArray {a}{b}..{z}\to U

```

926 \def\xintAssign{\def\XINT_flet_macro {\XINT_assign_fork}\XINT_flet_zapsp }%
927 \def\XINT_assign_fork
928 {%
929   \let\XINT_assign_def\def
930   \ifx\XINT_token[\expandafter\XINT_assign_opt
931     \else\expandafter\XINT_assign_a
932   \fi
933 }%
934 \def\XINT_assign_opt [#1]%
935 {%
936   \ifcsname #1def\endcsname
937     \expandafter\let\expandafter\XINT_assign_def \csname #1def\endcsname
938   \else
939     \expandafter\let\expandafter\XINT_assign_def \csname xint#1def\endcsname
940   \fi
941   \XINT_assign_a
942 }%

```

2 Package *xinttools* implementation

```
943 \long\def\XINT_assign_a #1\to
944 {%
945   \expandafter\XINT_assign_b\romannumeral-‘0#1{}\to
946 }%
947 \long\def\XINT_assign_b #1% attention to the # at the beginning of next line
948 #{%
949   \def\xint_temp {#1}%
950   \ifx\empty\xint_temp
951     \expandafter\XINT_assign_c
952   \else
953     \expandafter\XINT_assign_d
954   \fi
955 }%
956 \long\def\XINT_assign_c #1#2\to #3%
957 {%
958   \XINT_assign_def #3{#1}%
959   \def\xint_temp {#2}%
960   \unless\ifx\empty\xint_temp\xint_afterfi{\XINT_assign_b #2\to }\fi
961 }%
962 \def\XINT_assign_d #1\to #2% normally #1 is {} here.
963 {%
964   \expandafter\XINT_assign_def\expandafter #2\expandafter{\xint_temp}%
965 }%
966 \def\xintRelaxArray #1%
967 {%
968   \edef\XINT_restoreescapechar {\escapechar\the\escapechar\relax}%
969   \escapechar -1
970   \expandafter\def\expandafter\xint_arrayname\expandafter {\string #1}%
971   \XINT_restoreescapechar
972   \xintilop [\csname\xint_arrayname 0\endcsname+-1]
973   \global
974   \expandafter\let\csname\xint_arrayname\xintilopindex\endcsname\relax
975   \ifnum \xintilopindex > \xint_c_
976   \repeat
977   \global\expandafter\let\csname\xint_arrayname 00\endcsname\relax
978   \global\let #1\relax
979 }%
980 \def\xintAssignArray{\def\XINT_flet_macro {\XINT_assignarray_fork}%
981   \XINT_flet_zapsp }%
982 \def\XINT_assignarray_fork
983 {%
984   \let\XINT_assignarray_def\def
985   \ifx\XINT_token[\expandafter\XINT_assignarray_opt
986     \else\expandafter\XINT_assignarray
987   \fi
988 }%
989 \def\XINT_assignarray_opt [#1]%
990 {%
991   \ifcsname #1def\endcsname
992     \expandafter\let\expandafter\XINT_assignarray_def \csname #1def\endcsname
993   \else
994     \expandafter\let\expandafter\XINT_assignarray_def
```

2 Package *xinttools* implementation

```

995                                     \csname xint#1def\endcsname
996     \fi
997     \XINT_assignarray
998 }%
999 \long\def\XINT_assignarray #1\to #2%
1000 {%
1001     \edef\XINT_restoreescapechar {\escapechar\the\escapechar\relax }%
1002     \escapechar -1
1003     \expandafter\def\expandafter\xint_arrayname\expandafter {\string #2}%
1004     \XINT_restoreescapechar
1005     \def\xint_itemcount {0}%
1006     \expandafter\XINT_assignarray_loop \romannumeral-‘0#1\xint_relax
1007     \csname\xint_arrayname 00\expandafter\endcsname
1008     \csname\xint_arrayname 0\expandafter\endcsname
1009     \expandafter {\xint_arrayname}#2%
1010 }%
1011 \long\def\XINT_assignarray_loop #1%
1012 {%
1013     \def\xint_temp {#1}%
1014     \ifx\xint_brelax\xint_temp
1015         \expandafter\def\csname\xint_arrayname 0\expandafter\endcsname
1016             \expandafter{\the\numexpr\xint_itemcount}%
1017         \expandafter\expandafter\expandafter\XINT_assignarray_end
1018     \else
1019         \expandafter\def\expandafter\xint_itemcount\expandafter
1020             {\the\numexpr\xint_itemcount+\xint_c_i}%
1021         \expandafter\XINT_assignarray_def
1022             \csname\xint_arrayname\xint_itemcount\expandafter\endcsname
1023             \expandafter{\xint_temp }%
1024         \expandafter\XINT_assignarray_loop
1025     \fi
1026 }%
1027 \def\XINT_assignarray_end #1#2#3#4%
1028 {%
1029     \def #4##1%
1030     {%
1031         \romannumeral0\expandafter #1\expandafter{\the\numexpr ##1}%
1032     }%
1033     \def #1##1%
1034     {%
1035         \ifnum ##1<\xint_c_
1036             \xint_afterfi {\xintError:ArrayIndexIsNegative\space }%
1037         \else
1038             \xint_afterfi {%
1039                 \ifnum ##1>#2
1040                     \xint_afterfi {\xintError:ArrayIndexBeyondLimit\space }%
1041                 \else\xint_afterfi
1042                 {\expandafter\expandafter\expandafter\space\csname #3##1\endcsname}%
1043                 \fi}%
1044             \fi
1045         }%
1046 }%

```



```

1047 \let\xintDigitsOf\xintAssignArray
1048 \let\XINT_tmpa\relax \let\XINT_tmpb\relax \let\XINT_tmpc\relax
1049 \XINT_restorecatcodes_endinput%

```

3 Package *xintcore* implementation

.1	Catcodes, ε -TeX and reload detection	33	.11	<code>\xintiAdd</code>	47
.2	Package identification	34	.12	<code>\xintiSub</code>	48
.3	More token management, constants	34	.13	<code>\xintiMul</code>	53
.4	<code>\XINT_RQ</code>	34	.14	<code>\xintiSqr</code>	61
.5	<code>\XINT_OQ</code>	35	.15	<code>\xintiPow</code>	62
.6	<code>\XINT_cuz</code>	36	.16	<code>\xintiDivision, \xintiQuo, \xintiRem</code>	65
.7	<code>\xintNum</code>	37	.17	<code>\xintiDivRound, \xintiiDivRound</code>	78
.8	<code>\xintSgn, \xintiiSgn, \XINT_Sgn,</code> <code>\XINT_cntSgn</code>	38	.18	<code>\xintiDivTrunc, \xintiiDivTrunc</code>	79
.9	<code>\xintiOpp</code>	39	.19	<code>\xintiMod, cshxintiiMod</code>	79
.10	<code>\xintiAbs</code>	39	.20	<code>\xintDec</code>	80
			.21	<code>\xintInc</code>	81

Got split off from *xint* with release 1.1. Adds `\xintiiDivRound`. Does not load *xinttools*.

Since release *xint* 1.09a these macros doing arithmetic operations apply systematically `\xint2num` 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. 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.

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2 \catcode13=5 % ^^M
3 \endlinechar=13 %
4 \catcode123=1 % {
5 \catcode125=2 % }
6 \catcode64=11 % @
7 \catcode35=6 % #
8 \catcode44=12 % ,
9 \catcode45=12 % -
10 \catcode46=12 % .
11 \catcode58=12 % :
12 \let\z\endgroup
13 \expandafter\let\expandafter\x\csname ver@xintcore.sty\endcsname
14 \expandafter\let\expandafter\w\csname ver@xintkernel.sty\endcsname
15 \expandafter
16 \ifx\csname PackageInfo\endcsname\relax
17 \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
18 \else
19 \def\y#1#2{\PackageInfo{#1}{#2}}%
20 \fi
21 \expandafter
22 \ifx\csname numexpr\endcsname\relax
23 \y{xintcore}{\numexpr not available, aborting input}%
24 \aftergroup\endinput

```

```

25 \else
26   \ifx\x\relax % plain-TeX, first loading of xintcore.sty
27   \ifx\w\relax % but xintkernel.sty not yet loaded.
28     \def\z{\endgroup\input xintkernel.sty\relax}%
29   \fi
30 \else
31   \def\empty {}%
32   \ifx\x\empty % LaTeX, first loading,
33   % variable is initialized, but \ProvidesPackage not yet seen
34   \ifx\w\relax % xintkernel.sty not yet loaded.
35     \def\z{\endgroup\RequirePackage{xintkernel}}%
36   \fi
37 \else
38   \aftergroup\endinput % xinttools already loaded.
39 \fi
40 \fi
41 \fi
42 \z%
43 \XINTsetupcatcodes% defined in xintkernel.sty

```

3.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintcore}%
46 [2014/10/28 v1.1 Expandable arithmetic on big integers (jfb)]%

```

3.3 More token management, constants

```

47 \def\xint_minus_thenstop { -}%
48 \def\xint_gob_til_zeros_iii #1000{}%
49 \def\xint_gob_til_zeros_iv #10000{}%
50 \def\xint_gob_til_one #11{}%
51 \def\xint_gob_til_G #1G{}%
52 \def\xint_gob_til_minus #1-{}%
53 \def\xint_gob_til_relax #1\relax {}%
54 \def\xint_exchangetwo_keepbraces #1#2{{#2}{#1}}%
55 \def\xint_exchangetwo_keepbraces_thenstop #1#2{ {#2}{#1}}%
56 \def\xint_UDzerofork #10#2#3\krof {#2}%
57 \def\xint_UDsignfork #1-#2#3\krof {#2}%
58 \def\xint_UDwfork #1\W#2#3\krof {#2}%
59 \def\xint_UDzerosfork #100#2#3\krof {#2}%
60 \def\xint_UDonezerofork #110#2#3\krof {#2}%
61 \def\xint_UDsignsfork #1--#2#3\krof {#2}%
62 \chardef\xint_c_ix 9
63 \chardef\xint_c_x 10
64 \chardef\xint_c_ii^v 32 % not used in xint, common to xintfrac and xintbinhex
65 \chardef\xint_c_ii^vi 64
66 \mathchardef\xint_c_ixixixix 9999
67 \mathchardef\xint_c_x^iv 10000
68 \newcount\xint_c_x^viii \xint_c_x^viii 100000000

```

3.4 \backslash XINT_RQ

cette macro renverse et ajoute le nombre minimal de zéros à la fin pour que la longueur soit alors multiple de 4

3 Package *xintcore* implementation

`\romannumeral0\XINT_RQ {}<le truc à renverser>\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

```
69 \def\XINT_RQ #1#2#3#4#5#6#7#8#9%
70 {%
71   \xint_gob_til_R #9\XINT_RQ_end_a\R\XINT_RQ {#9#8#7#6#5#4#3#2#1}%
72 }%
73 \def\XINT_RQ_end_a\R\XINT_RQ #1#2\Z
74 {%
75   \XINT_RQ_end_b #1\Z
76 }%
77 \def\XINT_RQ_end_b #1#2#3#4#5#6#7#8%
78 {%
79   \xint_gob_til_R
80     #8\XINT_RQ_end_viii
81     #7\XINT_RQ_end_vii
82     #6\XINT_RQ_end_vi
83     #5\XINT_RQ_end_v
84     #4\XINT_RQ_end_iv
85     #3\XINT_RQ_end_iii
86     #2\XINT_RQ_end_ii
87     \R\XINT_RQ_end_i
88     \Z #2#3#4#5#6#7#8%
89 }%
90 \def\XINT_RQ_end_viii #1\Z #2#3#4#5#6#7#8#9\Z { #9}%
91 \def\XINT_RQ_end_vii #1\Z #2#3#4#5#6#7#8#9\Z { #8#9000}%
92 \def\XINT_RQ_end_vi #1\Z #2#3#4#5#6#7#8#9\Z { #7#8#900}%
93 \def\XINT_RQ_end_v #1\Z #2#3#4#5#6#7#8#9\Z { #6#7#8#90}%
94 \def\XINT_RQ_end_iv #1\Z #2#3#4#5#6#7#8#9\Z { #5#6#7#8#9}%
95 \def\XINT_RQ_end_iii #1\Z #2#3#4#5#6#7#8#9\Z { #4#5#6#7#8#9000}%
96 \def\XINT_RQ_end_ii #1\Z #2#3#4#5#6#7#8#9\Z { #3#4#5#6#7#8#900}%
97 \def\XINT_RQ_end_i #1\Z #2#3#4#5#6#7#8\Z { #1#2#3#4#5#6#7#80}%

```

3.5 `\XINT_OQ`

```
98 \def\XINT_OQ #1#2#3#4#5#6#7#8#9%
99 {%
100   \xint_gob_til_R #9\XINT_OQ_end_a\R\XINT_OQ {#9#8#7#6#5#4#3#2#1}%
101 }%
102 \def\XINT_OQ_end_a\R\XINT_OQ #1#2\Z
103 {%
104   \XINT_OQ_end_b #1\Z
105 }%
106 \def\XINT_OQ_end_b #1#2#3#4#5#6#7#8%
107 {%
108   \xint_gob_til_R
109     #8\XINT_OQ_end_viii
110     #7\XINT_OQ_end_vii
111     #6\XINT_OQ_end_vi
112     #5\XINT_OQ_end_v
113     #4\XINT_OQ_end_iv
114     #3\XINT_OQ_end_iii

```

3 Package *xintcore* implementation

```
115      #2\XINT_OQ_end_ii
116      \R\XINT_OQ_end_i
117      \Z #2#3#4#5#6#7#8%
118 }%
119 \def\XINT_OQ_end_viii #1\Z #2#3#4#5#6#7#8#9\Z { #9}%
120 \def\XINT_OQ_end_vii #1\Z #2#3#4#5#6#7#8#9\Z { #8#90000000}%
121 \def\XINT_OQ_end_vi #1\Z #2#3#4#5#6#7#8#9\Z { #7#8#9000000}%
122 \def\XINT_OQ_end_v #1\Z #2#3#4#5#6#7#8#9\Z { #6#7#8#900000}%
123 \def\XINT_OQ_end_iv #1\Z #2#3#4#5#6#7#8#9\Z { #5#6#7#8#90000}%
124 \def\XINT_OQ_end_iii #1\Z #2#3#4#5#6#7#8#9\Z { #4#5#6#7#8#9000}%
125 \def\XINT_OQ_end_ii #1\Z #2#3#4#5#6#7#8#9\Z { #3#4#5#6#7#8#900}%
126 \def\XINT_OQ_end_i #1\Z #2#3#4#5#6#7#8\Z { #1#2#3#4#5#6#7#80}%

```

3.6 \XINT_cuz

```
127 \edef\xint_cleanupzeros_andstop #1#2#3#4%
128 {%
129     \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4\relax
130 }%
131 \def\xint_cleanupzeros_nostop #1#2#3#4%
132 {%
133     \the\numexpr #1#2#3#4\relax
134 }%
135 \def\XINT_rev_andcuz #1%
136 {%
137     \expandafter\xint_cleanupzeros_andstop
138     \romannumeral0\XINT_rord_main {}#1%
139     \xint_relax
140     \xint_bye\xint_bye\xint_bye\xint_bye
141     \xint_bye\xint_bye\xint_bye\xint_bye
142     \xint_relax
143 }%

```

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\Z avec SEPT \W

```
144 \def\XINT_cuz #1%
145 {%
146     \XINT_cuz_loop #1\W\W\W\W\W\W\W\Z%
147 }%
148 \def\XINT_cuz_loop #1#2#3#4#5#6#7#8%
149 {%
150     \xint_gob_til_W #8\xint_cuz_end_a\W
151     \xint_gob_til_Z #8\xint_cuz_end_A\Z
152     \XINT_cuz_check_a {#1#2#3#4#5#6#7#8}%
153 }%
154 \def\xint_cuz_end_a #1\XINT_cuz_check_a #2%
155 {%
156     \xint_cuz_end_b #2%
157 }%
158 \edef\xint_cuz_end_b #1#2#3#4#5\Z
159 {%
160     \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4\relax

```

```

161 }%
162 \def\xint_cuz_end_A \Z\XINT_cuz_check_a #1{ 0}%
163 \def\XINT_cuz_check_a #1%
164 {%
165   \expandafter\XINT_cuz_check_b\the\numexpr #1\relax
166 }%
167 \def\XINT_cuz_check_b #1%
168 {%
169   \xint_gob_til_zero #1\xint_cuz_backtoloop 0\XINT_cuz_stop #1%
170 }%
171 \def\XINT_cuz_stop #1\W #2\Z{ #1}%
172 \def\xint_cuz_backtoloop 0\XINT_cuz_stop 0{\XINT_cuz_loop }%

```

3.7 \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.

```

173 \def\xintiNum {\romannumeral0\xintinum }%
174 \def\xintinum #1%
175 {%
176   \expandafter\XINT_num_loop
177   \romannumeral-‘0#1\xint_relax\xint_relax\xint_relax\xint_relax
178   \xint_relax\xint_relax\xint_relax\xint_relax\Z
179 }%
180 \let\xintNum\xintiNum \let\xintnum\xintinum
181 \def\XINT_num #1%
182 {%
183   \XINT_num_loop #1\xint_relax\xint_relax\xint_relax\xint_relax
184   \xint_relax\xint_relax\xint_relax\xint_relax\Z
185 }%
186 \def\XINT_num_loop #1#2#3#4#5#6#7#8%
187 {%
188   \xint_gob_til_xint_relax #8\XINT_num_end\xint_relax
189   \XINT_num_NumEight #1#2#3#4#5#6#7#8%
190 }%
191 \edef\XINT_num_end\xint_relax\XINT_num_NumEight #1\xint_relax #2\Z
192 {%
193   \noexpand\expandafter\space\noexpand\the\numexpr #1+\xint_c_\relax
194 }%
195 \def\XINT_num_NumEight #1#2#3#4#5#6#7#8%
196 {%
197   \ifnum \numexpr #1#2#3#4#5#6#7#8+\xint_c_= \xint_c_
198     \xint_afterfi {\expandafter\XINT_num_keepsign_a
199     \the\numexpr #1#2#3#4#5#6#7#81\relax}%
200   \else
201     \xint_afterfi {\expandafter\XINT_num_finish

```

3 Package *xintcore* implementation

```
202             \the\numexpr #1#2#3#4#5#6#7#8\relax}%
203     \fi
204 }%
205 \def\XINT_num_keepsign_a #1%
206 {%
207     \xint_gob_til_one#1\XINT_num_gobackto loop 1\XINT_num_keepsign_b
208 }%
209 \def\XINT_num_gobackto loop 1\XINT_num_keepsign_b {\XINT_num_loop }%
210 \def\XINT_num_keepsign_b #1{\XINT_num_loop -}%
211 \def\XINT_num_finish #1\xint_relax #2\Z { #1}%
```

3.8 `\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.

```
212 \def\xintiisgn {\romannumeral0\xintiisgn }%
213 \def\xintiisgn #1%
214 {%
215     \expandafter\XINT_sgn \romannumeral-‘0#1\Z%
216 }%
217 \def\xintSgn {\romannumeral0\xintsgn }%
218 \def\xintsgn #1%
219 {%
220     \expandafter\XINT_sgn \romannumeral0\xintnum{#1}\Z%
221 }%
222 \def\XINT_sgn #1#2\Z
223 {%
224     \xint_UDzerominusfork
225     #1-{ 0}%
226     0#1{ -1}%
227     0-{ 1}%
228     \krof
229 }%
230 \def\XINT_Sgn #1#2\Z
231 {%
232     \xint_UDzerominusfork
233     #1-{0}%
234     0#1{-1}%
235     0-{1}%
236     \krof
237 }%
238 \def\XINT_cntSgn #1#2\Z
239 {%
240     \xint_UDzerominusfork
241     #1-\xint_c_
242     0#1\m@ne % I will not allocate a count only for -1?
243     0-\xint_c_i
244     \krof
245 }%
```

3.9 `\xintiOpp`

`\xintnum` added in 1.09a

```

246 \def\xintiiOpp {\romannumeral0\xintiiopp }%
247 \def\xintiiopp #1%
248 {%
249   \expandafter\XINT_opp \romannumeral-‘0#1%
250 }%
251 \def\xintiOpp {\romannumeral0\xintiopp }%
252 \def\xintiopp #1%
253 {%
254   \expandafter\XINT_opp \romannumeral0\xintnum{#1}%
255 }%
256 \let\xintOpp\xintiOpp \let\xintopp\xintiopp
257 \def\XINT_Opp #1{\romannumeral0\XINT_opp #1}%
258 \def\XINT_opp #1%
259 {%
260   \xint_UDzerominusfork
261   #1-{ 0}%      zero
262   0#1{ }%      negative
263   0-{ -#1}%    positive
264   \krof
265 }%

```

3.10 `\xintiAbs`

Release 1.09a has now `\xintiabs` which does `\xintnum` (contrarily to some other i-macros, but similarly as `\xintiAdd` etc...) and this is inherited by `DecSplit`, by `Sqr`, and macros of `xintgcd.sty`.

```

266 \def\xintiiAbs {\romannumeral0\xintiiabs }%
267 \def\xintiiabs #1%
268 {%
269   \expandafter\XINT_abs \romannumeral-‘0#1%
270 }%
271 \def\xintiAbs {\romannumeral0\xintiabs }%
272 \def\xintiabs #1%
273 {%
274   \expandafter\XINT_abs \romannumeral0\xintnum{#1}%
275 }%
276 \let\xintAbs\xintiAbs \let\xintabs\xintiabs
277 \def\XINT_Abs #1{\romannumeral0\XINT_abs #1}%
278 \def\XINT_abs #1%
279 {%
280   \xint_UDsignfork
281   #1{ }%
282   -{ #1}%
283   \krof
284 }%

```

ARITHMETIC OPERATIONS: ADDITION, SUBTRACTION, SUMS, MULTIPLICATION, PRODUCTS, FACTORIAL, POWERS, EUCLIDEAN DIVISION.

3 Package *xintcore* implementation

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 I: `\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.

```
285 \def\XINT_add_A #1#2#3#4#5#6%
286 {%
287     \xint_gob_til_W #3\xint_add_az\W
288     \XINT_add_AB #1{#3#4#5#6}{#2}%
289 }%
290 \def\xint_add_az\W\XINT_add_AB #1#2%
291 {%
292     \XINT_add_AC_checkcarry #1%
293 }%
```

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

```
294 \def\XINT_add_AB #1#2#3#4\W\X\Y\Z #5#6#7#8%
295 {%
296     \xint_gob_til_W #5\xint_add_bz\W
297     \XINT_add_ABE #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
298 }%
299 \def\XINT_add_ABE #1#2#3#4#5#6%
300 {%
301     \expandafter\XINT_add_ABEA\the\numexpr #1+10#5#4#3#2+#6.%
302 }%
303 \def\XINT_add_ABEA #1#2#3.#4%
304 {%
305     \XINT_add_A #2{#3#4}%
306 }%
```

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

```
307 \def\xint_add_bz\W\XINT_add_ABE #1#2#3#4#5#6%
308 {%
309     \expandafter\XINT_add_CC\the\numexpr #1+10#5#4#3#2.%
310 }%
```


3 Package *xintcore* implementation

```

311 \def\XINT_add_CC #1#2#3.#4%
312 {%
313   \XINT_add_AC_checkcarry #2{#3#4}% on va examiner et \'eliminer #2
314 }%

```

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

```

315 \def\XINT_add_AC_checkcarry #1%
316 {%
317   \xint_gob_til_zero #1\xint_add_AC_nocarry 0\XINT_add_C
318 }%
319 \def\xint_add_AC_nocarry 0\XINT_add_C #1#2\W\X\Y\Z
320 {%
321   \expandafter
322   \xint_cleanupzeros_andstop
323   \romannumeral0%
324   \XINT_rord_main {}#2%
325   \xint_relax
326   \xint_bye\xint_bye\xint_bye\xint_bye
327   \xint_bye\xint_bye\xint_bye\xint_bye
328   \xint_relax
329   #1%
330 }%
331 \def\XINT_add_C #1#2#3#4#5%
332 {%
333   \xint_gob_til_W #2\xint_add_cz\W
334   \XINT_add_CD {#5#4#3#2}{#1}%
335 }%
336 \def\XINT_add_CD #1%
337 {%
338   \expandafter\XINT_add_CC\the\numexpr 1+10#1.%
339 }%
340 \def\xint_add_cz\W\XINT_add_CD #1#2{ 1#2}%

```

Addition II: \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*

```

341 \def\XINT_addr_A #1#2#3#4#5#6%
342 {%
343   \xint_gob_til_W #3\xint_addr_az\W
344   \XINT_addr_B #1{#3#4#5#6}{#2}%
345 }%
346 \def\xint_addr_az\W\XINT_addr_B #1#2%
347 {%
348   \XINT_addr_AC_checkcarry #1%

```

3 Package *xintcore* implementation

```

349 }%
350 \def\XINT_addr_B #1#2#3#4\W\X\Y\Z #5#6#7#8%
351 {%
352     \xint_gob_til_W #5\xint_addr_bz\W
353     \XINT_addr_E #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
354 }%
355 \def\XINT_addr_E #1#2#3#4#5#6%
356 {%
357     \expandafter\XINT_addr_ABEA\the\numexpr #1+10#5#4#3#2+#6\relax
358 }%
359 \def\XINT_addr_ABEA #1#2#3#4#5#6#7%
360 {%
361     \XINT_addr_A #2{#7#6#5#4#3}%
362 }%
363 \def\xint_addr_bz\W\XINT_addr_E #1#2#3#4#5#6%
364 {%
365     \expandafter\XINT_addr_CC\the\numexpr #1+10#5#4#3#2\relax
366 }%
367 \def\XINT_addr_CC #1#2#3#4#5#6#7%
368 {%
369     \XINT_addr_AC_checkcarry #2{#7#6#5#4#3}%
370 }%
371 \def\XINT_addr_AC_checkcarry #1%
372 {%
373     \xint_gob_til_zero #1\xint_addr_AC_nocarry 0\XINT_addr_C
374 }%
375 \def\xint_addr_AC_nocarry 0\XINT_addr_C #1#2\W\X\Y\Z { #1#2}%
376 \def\XINT_addr_C #1#2#3#4#5%
377 {%
378     \xint_gob_til_W #2\xint_addr_cz\W
379     \XINT_addr_D {#5#4#3#2}{#1}%
380 }%
381 \def\XINT_addr_D #1%
382 {%
383     \expandafter\XINT_addr_CC\the\numexpr 1+10#1\relax
384 }%
385 \def\xint_addr_cz\W\XINT_addr_D #1#2{ #21000}%

```

ADDITION III, \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 multiplication.

```

386 \def\XINT_addm_A #1#2#3#4#5#6%
387 {%
388     \xint_gob_til_W #3\xint_addm_az\W
389     \XINT_addm_AB #1{#3#4#5#6}{#2}%
390 }%
391 \def\xint_addm_az\W\XINT_addm_AB #1#2%
392 {%

```

3 Package *xintcore* implementation

```
393 \XINT_addm_AC_checkcarry #1%
394 }%
395 \def\XINT_addm_AB #1#2#3#4\W\X\Y\Z #5#6#7#8%
396 {%
397 \XINT_addm_ABE #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
398 }%
399 \def\XINT_addm_ABE #1#2#3#4#5#6%
400 {%
401 \expandafter\XINT_addm_ABEA\the\numexpr #1+10#5#4#3#2+#6.%
402 }%
403 \def\XINT_addm_ABEA #1#2#3.#4%
404 {%
405 \XINT_addm_A #2{#3#4}%
406 }%
407 \def\XINT_addm_AC_checkcarry #1%
408 {%
409 \xint_gob_til_zero #1\xint_addm_AC_nocarry 0\XINT_addm_C
410 }%
411 \def\xint_addm_AC_nocarry 0\XINT_addm_C #1#2\W\X\Y\Z
412 {%
413 \expandafter
414 \xint_cleanupzeros_andstop
415 \romannumeral0%
416 \XINT_rord_main {}#2%
417 \xint_relax
418 \xint_bye\xint_bye\xint_bye\xint_bye
419 \xint_bye\xint_bye\xint_bye\xint_bye
420 \xint_relax
421 #1%
422 }%
423 \def\XINT_addm_C #1#2#3#4#5%
424 {%
425 \xint_gob_til_W
426 #5\xint_addm_cw
427 #4\xint_addm_cx
428 #3\xint_addm_cy
429 #2\xint_addm_cz
430 \W\XINT_addm_CD {#5#4#3#2}{#1}%
431 }%
432 \def\XINT_addm_CD #1%
433 {%
434 \expandafter\XINT_addm_CC\the\numexpr 1+10#1.%
435 }%
436 \def\XINT_addm_CC #1#2#3.#4%
437 {%
438 \XINT_addm_AC_checkcarry #2{#3#4}%
439 }%
440 \def\xint_addm_cw
441 #1\xint_addm_cx
442 #2\xint_addm_cy
443 #3\xint_addm_cz
444 \W\XINT_addm_CD
```

3 Package *xintcore* implementation

```

445 {%
446   \expandafter\XINT_addm_CDw\the\numexpr 1+#1#2#3.%
447 }%
448 \def\XINT_addm_CDw #1.#2#3\X\Y\Z
449 {%
450   \XINT_addm_end #1#3%
451 }%
452 \def\xint_addm_cx
453   #1\xint_addm_cy
454   #2\xint_addm_cz
455   \W\XINT_addm_CD
456 {%
457   \expandafter\XINT_addm_CDx\the\numexpr 1+#1#2.%
458 }%
459 \def\XINT_addm_CDx #1.#2#3\Y\Z
460 {%
461   \XINT_addm_end #1#3%
462 }%
463 \def\xint_addm_cy
464   #1\xint_addm_cz
465   \W\XINT_addm_CD
466 {%
467   \expandafter\XINT_addm_CDy\the\numexpr 1+#1.%
468 }%
469 \def\XINT_addm_CDy #1.#2#3\Z
470 {%
471   \XINT_addm_end #1#3%
472 }%
473 \def\xint_addm_cz\W\XINT_addm_CD #1#2#3{\XINT_addm_end #1#3}%
474 \edef\XINT_addm_end #1#2#3#4#5%
475   {\noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5\relax}%

```

ADDITION IV, variante \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 terminer en 0000. Utilisé par la multiplication servant pour le calcul des puissances.

```

476 \def\XINT_addp_A #1#2#3#4#5#6%
477 {%
478   \xint_gob_til_W #3\xint_addp_az\W
479   \XINT_addp_AB #1{#3#4#5#6}{#2}%
480 }%
481 \def\xint_addp_az\W\XINT_addp_AB #1#2%
482 {%
483   \XINT_addp_AC_checkcarry #1%
484 }%
485 \def\XINT_addp_AC_checkcarry #1%
486 {%
487   \xint_gob_til_zero #1\xint_addp_AC_nocarry 0\XINT_addp_C
488 }%

```

3 Package *xintcore* implementation

```
489 \def\xint_addp_AC_nocarry 0\XINT_addp_C
490 {%
491   \XINT_addp_F
492 }%
493 \def\XINT_addp_AB #1#2#3#4\W\X\Y\Z #5#6#7#8%
494 {%
495   \XINT_addp_ABE #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
496 }%
497 \def\XINT_addp_ABE #1#2#3#4#5#6%
498 {%
499   \expandafter\XINT_addp_ABEA\the\numexpr #1+10#5#4#3#2+#6\relax
500 }%
501 \def\XINT_addp_ABEA #1#2#3#4#5#6#7%
502 {%
503   \XINT_addp_A #2{#7#6#5#4#3}%<-- attention on met donc \'a droite
504 }%
505 \def\XINT_addp_C #1#2#3#4#5%
506 {%
507   \xint_gob_til_W
508   #5\xint_addp_cw
509   #4\xint_addp_cx
510   #3\xint_addp_cy
511   #2\xint_addp_cz
512   \W\XINT_addp_CD {#5#4#3#2}{#1}%
513 }%
514 \def\XINT_addp_CD #1%
515 {%
516   \expandafter\XINT_addp_CC\the\numexpr 1+10#1\relax
517 }%
518 \def\XINT_addp_CC #1#2#3#4#5#6#7%
519 {%
520   \XINT_addp_AC_checkcarry #2{#7#6#5#4#3}%
521 }%
522 \def\xint_addp_cw
523   #1\xint_addp_cx
524   #2\xint_addp_cy
525   #3\xint_addp_cz
526   \W\XINT_addp_CD
527 {%
528   \expandafter\XINT_addp_CDw\the\numexpr \xint_c_i+10#1#2#3\relax
529 }%
530 \def\XINT_addp_CDw #1#2#3#4#5#6%
531 {%
532   \xint_gob_til_zeros_iv #2#3#4#5\XINT_addp_endDw_zeros
533   0000\XINT_addp_endDw #2#3#4#5%
534 }%
535 \def\XINT_addp_endDw_zeros 0000\XINT_addp_endDw 0000#1\X\Y\Z{ #1}%
536 \def\XINT_addp_endDw #1#2#3#4#5\X\Y\Z{ #5#4#3#2#1}%
537 \def\xint_addp_cx
538   #1\xint_addp_cy
539   #2\xint_addp_cz
540   \W\XINT_addp_CD
```

3 Package *xintcore* implementation

```
541 {%
542   \expandafter\XINT_addp_CDx\the\numexpr \xint_c_i+100#1#2\relax
543 }%
544 \def\XINT_addp_CDx #1#2#3#4#5#6%
545 {%
546   \xint_gob_til_zeros_iv #2#3#4#5\XINT_addp_endDx_zeros
547   0000\XINT_addp_endDx #2#3#4#5%
548 }%
549 \def\XINT_addp_endDx_zeros 0000\XINT_addp_endDx 0000#1\Y\Z{ #1}%
550 \def\XINT_addp_endDx #1#2#3#4#5\Y\Z{ #5#4#3#2#1}%
551 \def\xint_addp_cy #1\xint_addp_cz\W\XINT_addp_CD
552 {%
553   \expandafter\XINT_addp_CDy\the\numexpr \xint_c_i+1000#1\relax
554 }%
555 \def\XINT_addp_CDy #1#2#3#4#5#6%
556 {%
557   \xint_gob_til_zeros_iv #2#3#4#5\XINT_addp_endDy_zeros
558   0000\XINT_addp_endDy #2#3#4#5%
559 }%
560 \def\XINT_addp_endDy_zeros 0000\XINT_addp_endDy 0000#1\Z{ #1}%
561 \def\XINT_addp_endDy #1#2#3#4#5\Z{ #5#4#3#2#1}%
562 \def\xint_addp_cz\W\XINT_addp_CD #1#2{ #21000}%
563 \def\XINT_addp_F #1#2#3#4#5%
564 {%
565   \xint_gob_til_W
566   #5\xint_addp_Gw
567   #4\xint_addp_Gx
568   #3\xint_addp_Gy
569   #2\xint_addp_Gz
570   \W\XINT_addp_G {#2#3#4#5}{#1}%
571 }%
572 \def\XINT_addp_G #1#2%
573 {%
574   \XINT_addp_F {#2#1}%
575 }%
576 \def\xint_addp_Gw
577   #1\xint_addp_Gx
578   #2\xint_addp_Gy
579   #3\xint_addp_Gz
580   \W\XINT_addp_G #4%
581 {%
582   \xint_gob_til_zeros_iv #3#2#10\XINT_addp_endGw_zeros
583   0000\XINT_addp_endGw #3#2#10%
584 }%
585 \def\XINT_addp_endGw_zeros 0000\XINT_addp_endGw 0000#1\X\Y\Z{ #1}%
586 \def\XINT_addp_endGw #1#2#3#4#5\X\Y\Z{ #5#1#2#3#4}%
587 \def\xint_addp_Gx
588   #1\xint_addp_Gy
589   #2\xint_addp_Gz
590   \W\XINT_addp_G #3%
591 {%
592   \xint_gob_til_zeros_iv #2#100\XINT_addp_endGx_zeros
```

3 Package *xintcore* implementation

```
593          0000\XINT_addp_endGx #2#100%
594 }%
595 \def\XINT_addp_endGx_zeros 0000\XINT_addp_endGx 0000#1\Y\Z{ #1}%
596 \def\XINT_addp_endGx #1#2#3#4#5\Y\Z{ #5#1#2#3#4}%
597 \def\xint_addp_Gy
598   #1\xint_addp_Gz
599   \W\XINT_addp_G #2%
600 {%
601   \xint_gob_til_zeros_iv #1000\XINT_addp_endGy_zeros
602   0000\XINT_addp_endGy #1000%
603 }%
604 \def\XINT_addp_endGy_zeros 0000\XINT_addp_endGy 0000#1\Z{ #1}%
605 \def\XINT_addp_endGy #1#2#3#4#5\Z{ #5#1#2#3#4}%
606 \def\xint_addp_Gz\W\XINT_addp_G #1#2{ #2}%
```

3.11 \xintiAdd

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.

```
607 \def\xintiiAdd {\romannumeral0\xintiiadd }%
608 \def\xintiiadd #1{\expandafter\xint_iiadd\romannumeral-‘0#1\Z }%
609 \def\xint_iiadd #1#2\Z #3%
610 {%
611   \expandafter\XINT_add_fork\expandafter #1\romannumeral-‘0#3\Z #2\Z
612 }%
613 \def\xintiAdd {\romannumeral0\xintiadd }%
614 \def\xintiadd #1%
615 {%
616   \expandafter\xint_add\romannumeral0\xintnum{#1}\Z
617 }%
618 \def\xint_add #1#2\Z #3%
619 {%
620   \expandafter\XINT_add_fork\expandafter #1\romannumeral0\xintnum{#3}\Z #2\Z
621 }%
622 \let\xintAdd\xintiAdd \let\xintadd\xintiadd
623 \def\XINT_add_fork #1#2%
624 {%
625   \xint_UDzerofork
626   #1\XINT_add_firstiszero
627   #2\XINT_add_secondiszero
628   0{}%
629   \krof
630   \xint_UDsignsfork
631   #1#2\XINT_add_minusminus
632   #1-\XINT_add_minusplus
633   #2-\XINT_add_plusminus
634   --\XINT_add_plusplus
635   \krof #1#2%
636 }%
```

3 Package *xintcore* implementation

```
637 \def\XINT_add_firstiszero #1\krof #2#3\Z #4\Z { #3}%
638 \def\XINT_add_secondiszero #1\krof #2#3\Z #4\Z { #2#4}%
639 \def\XINT_add_plusplus #1#2#3\Z #4\Z {\XINT_add_pre {#1#4}{#2#3}}%
640 \def\XINT_add_minusminus #1#2#3\Z #4\Z
641   {\expandafter\XINT_minus_thenstop\romannumeral0\XINT_add_pre {#4}{#3}}%
642 \def\XINT_add_minusplus #1#2#3\Z #4\Z {\XINT_sub_pre {#2#3}{#4}}%
643 \def\XINT_add_plusminus #1#2#3\Z #4\Z {\XINT_sub_pre {#1#4}{#3}}%
```

positive summands

```
644 \def\XINT_add_pre #1%
645 {%
646   \expandafter\XINT_add_pre_b\expandafter
647   {\romannumeral0\XINT_RQ {}#1\R\R\R\R\R\R\R\R\Z}%
648 }%
649 \def\XINT_add_pre_b #1#2%
650 {%
651   \expandafter\XINT_add_A
652   \expandafter0\expandafter{\expandafter}%
653   \romannumeral0\XINT_RQ {}#2\R\R\R\R\R\R\R\R\Z
654   \W\X\Y\Z #1\W\X\Y\Z
655 }%
```

3.12 \xintiSub

Release 1.09a has \xintnum added into \xintiSub.

```
656 \def\xintiiSub {\romannumeral0\xintiisub}%
657 \def\xintiisub #1{\expandafter\xint_iisub\romannumeral-‘0#1\Z}%
658 \def\xint_iisub #1#2\Z #3%
659 {%
660   \expandafter\XINT_sub_fork\expandafter #1\romannumeral-‘0#3\Z #2\Z
661 }%
662 \def\xintiSub {\romannumeral0\xintisub}%
663 \def\xintisub #1%
664 {%
665   \expandafter\xint_sub\romannumeral0\xintnum{#1}\Z
666 }%
667 \def\xint_sub #1#2\Z #3%
668 {%
669   \expandafter\XINT_sub_fork\expandafter #1\romannumeral0\xintnum{#3}\Z #2\Z
670 }%
671 \let\xintSub\xintiSub \let\xintsub\xintisub
672 \def\XINT_sub_fork #1#2%
673 {%
674   \xint_UDzerofork
675   #1\XINT_sub_firstiszero
676   #2\XINT_sub_secondiszero
677   0}%
678 \krof
679 \xint_UDsignsfork
680   #1#2\XINT_sub_minusminus
681   #1-\XINT_sub_minusplus
```


3 Package *xintcore* implementation

```

682      #2-\XINT_sub_plusminus
683      --\XINT_sub_plusplus
684      \krof #1#2%
685 }%
686 \def\XINT_sub_firstiszero #1\krof #2#3\Z #4\Z {\XINT_opp #3}%
687 \def\XINT_sub_secondiszero #1\krof #2#3\Z #4\Z { #2#4}%
688 \def\XINT_sub_plusplus #1#2#3\Z #4\Z {\XINT_sub_pre {#1#4}{#2#3}}%
689 \def\XINT_sub_minusminus #1#2#3\Z #4\Z {\XINT_sub_pre {#3}{#4}}%
690 \def\XINT_sub_minusplus #1#2#3\Z #4\Z
691   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_add_pre {#4}{#2#3}}%
692 \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

```

693 \def\XINT_sub_pre #1%
694 {%
695   \expandafter\XINT_sub_pre_b\expandafter
696   {\romannumeral0\XINT_RQ {}#1\R\R\R\R\R\R\R\R\Z }%
697 }%
698 \def\XINT_sub_pre_b #1#2%
699 {%
700   \expandafter\XINT_sub_A
701   \expandafter1\expandafter{\expandafter}%
702   \romannumeral0\XINT_RQ {}#2\R\R\R\R\R\R\R\R\Z
703   \W\X\Y\Z #1 \W\X\Y\Z
704 }%

```

\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.

```

705 \def\XINT_sub_A #1#2#3\W\X\Y\Z #4#5#6#7%
706 {%
707   \xint_gob_til_W
708   #4\xint_sub_az
709   \W\XINT_sub_B #1{#4#5#6#7}{#2}#3\W\X\Y\Z
710 }%
711 \def\XINT_sub_B #1#2#3#4#5#6#7%
712 {%
713   \xint_gob_til_W
714   #4\xint_sub_bz
715   \W\XINT_sub_onestep #1#2{#7#6#5#4}{#3}%
716 }%

```

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.

```

717 \def\XINT_sub_onestep #1#2#3#4#5#6%
718 {%
719   \expandafter\XINT_sub_backtoA\the\numexpr 11#5#4#3#2-#6+#1-\xint_c_i.%
720 }%

```

ON PRODUIT LE RÉSULTAT DANS LE BON ORDRE

```

721 \def\XINT_sub_backtoA #1#2#3.#4%
722 {%
723   \XINT_sub_A #2{#3#4}%
724 }%
725 \def\xint_sub_bz
726   \W\XINT_sub_onestep #1#2#3#4#5#6#7%
727 {%
728   \xint_UDzerofork
729     #1\XINT_sub_C    % une retenue
730     0\XINT_sub_D    % pas de retenue
731   \krof
732   {#7}#2#3#4#5%
733 }%
734 \def\XINT_sub_D #1#2\W\X\Y\Z
735 {%
736   \expandafter
737   \xint_cleanupzeros_andstop
738   \romannumeral0%
739   \XINT_rord_main {}#2%
740   \xint_relax
741     \xint_bye\xint_bye\xint_bye\xint_bye
742     \xint_bye\xint_bye\xint_bye\xint_bye
743   \xint_relax
744   #1%
745 }%
746 \def\XINT_sub_C #1#2#3#4#5%
747 {%
748   \xint_gob_til_W
749   #2\xint_sub_cz
750   \W\XINT_sub_AC_onestep {#5#4#3#2}{#1}%
751 }%
752 \def\XINT_sub_AC_onestep #1%
753 {%
754   \expandafter\XINT_sub_backtoC\the\numexpr 11#1-\xint_c_i.%
755 }%
756 \def\XINT_sub_backtoC #1#2#3.#4%
757 {%
758   \XINT_sub_AC_checkcarry #2{#3#4}% la retenue va \^etre examin\'ee
759 }%
760 \def\XINT_sub_AC_checkcarry #1%
761 {%
762   \xint_gob_til_one #1\xint_sub_AC_nocarry 1\XINT_sub_C
763 }%
764 \def\xint_sub_AC_nocarry 1\XINT_sub_C #1#2\W\X\Y\Z
765 {%
766   \expandafter
767   \XINT_cuz_loop
768   \romannumeral0%
769   \XINT_rord_main {}#2%
770   \xint_relax
771     \xint_bye\xint_bye\xint_bye\xint_bye

```

3 Package *xintcore* implementation

```
772      \xint_bye\xint_bye\xint_bye\xint_bye
773      \xint_relax
774      #1\W\W\W\W\W\W\W\Z
775 }%
776 \def\xint_sub_cz\W\XINT_sub_AC_onestep #1%
777 {%
778     \XINT_cuz
779 }%
780 \def\xint_sub_az\W\XINT_sub_B #1#2#3#4#5#6#7%
781 {%
782     \xint_gob_til_W
783     #4\xint_sub_ez
784     \W\XINT_sub_Eenter #1{#3}#4#5#6#7%
785 }%
```

le premier nombre continue, le résultat sera < 0 .

```
786 \def\XINT_sub_Eenter #1#2%
787 {%
788     \expandafter
789     \XINT_sub_E\expandafter1\expandafter{\expandafter}%
790     \romannumeral0%
791     \XINT_rord_main {}#2%
792     \xint_relax
793     \xint_bye\xint_bye\xint_bye\xint_bye
794     \xint_bye\xint_bye\xint_bye\xint_bye
795     \xint_relax
796     \W\X\Y\Z #1%
797 }%
798 \def\XINT_sub_E #1#2#3#4#5#6%
799 {%
800     \xint_gob_til_W #3\xint_sub_F\W
801     \XINT_sub_Eonestep #1{#6#5#4#3}{#2}%
802 }%
803 \def\XINT_sub_Eonestep #1#2%
804 {%
805     \expandafter\XINT_sub_backtoE\the\numexpr 109999-#2+#1.%
806 }%
807 \def\XINT_sub_backtoE #1#2#3.#4%
808 {%
809     \XINT_sub_E #2{#3#4}%
810 }%
811 \def\xint_sub_F\W\XINT_sub_Eonestep #1#2#3#4%
812 {%
813     \xint_UDonezerofork
814     #4#1{\XINT_sub_Fdec 0}% soustraire 1. Et faire signe -
815     #1#4{\XINT_sub_Finc 1}% additionner 1. Et faire signe -
816     10\XINT_sub_DD % terminer. Mais avec signe -
817     \krof
818     {#3}%
819 }%
820 \def\XINT_sub_DD {\expandafter\xint_minus_thenstop\romannumeral0\XINT_sub_D }%
821 \def\XINT_sub_Fdec #1#2#3#4#5#6%
```

3 Package *xintcore* implementation

```

822 {%
823   \xint_gob_til_W #3\xint_sub_Fdec_finish\W
824   \XINT_sub_Fdec_onestep #1{#6#5#4#3}{#2}%
825 }%
826 \def\XINT_sub_Fdec_onestep #1#2%
827 {%
828   \expandafter\XINT_sub_backtoFdec\the\numexpr 11#2+#1-\xint_c_i.%
829 }%
830 \def\XINT_sub_backtoFdec #1#2#3.#4%
831 {%
832   \XINT_sub_Fdec #2{#3#4}%
833 }%
834 \def\xint_sub_Fdec_finish\W\XINT_sub_Fdec_onestep #1#2%
835 {%
836   \expandafter\xint_minus_thenstop\romannumeral0\XINT_cuz
837 }%
838 \def\XINT_sub_Finc #1#2#3#4#5#6%
839 {%
840   \xint_gob_til_W #3\xint_sub_Finc_finish\W
841   \XINT_sub_Finc_onestep #1{#6#5#4#3}{#2}%
842 }%
843 \def\XINT_sub_Finc_onestep #1#2%
844 {%
845   \expandafter\XINT_sub_backtoFinc\the\numexpr 10#2+#1.%
846 }%
847 \def\XINT_sub_backtoFinc #1#2#3.#4%
848 {%
849   \XINT_sub_Finc #2{#3#4}%
850 }%
851 \def\xint_sub_Finc_finish\W\XINT_sub_Finc_onestep #1#2#3%
852 {%
853   \xint_UDzerofork
854   #1{\expandafter\expandafter\expandafter
855     \xint_minus_thenstop\xint_cleanupzeros_nostop}%
856   0{ -1}%
857   \krof
858   #3%
859 }%
860 \def\xint_sub_ez\W\XINT_sub_Eenter #1%
861 {%
862   \xint_UDzerofork
863   #1\XINT_sub_K % il y a une retenue
864   0\XINT_sub_L % pas de retenue
865   \krof
866 }%
867 \def\XINT_sub_L #1\W\X\Y\Z {\XINT_cuz_loop #1\W\W\W\W\W\W\Z }%
868 \def\XINT_sub_K #1%
869 {%
870   \expandafter
871   \XINT_sub_KK\expandafter1\expandafter{\expandafter}%
872   \romannumeral0%
873   \XINT_rord_main {}#1%

```

3 Package *xintcore* implementation

```
874 \xint_relax
875 \xint_bye\xint_bye\xint_bye\xint_bye
876 \xint_bye\xint_bye\xint_bye\xint_bye
877 \xint_relax
878 }%
879 \def\xINT_sub_KK #1#2#3#4#5#6%
880 {%
881 \xint_gob_til_W #3\xint_sub_KK_finish\W
882 \XINT_sub_KK_onestep #1{#6#5#4#3}{#2}%
883 }%
884 \def\xINT_sub_KK_onestep #1#2%
885 {%
886 \expandafter\xINT_sub_backtoKK\the\numexpr 109999-#2+#1.%
887 }%
888 \def\xINT_sub_backtoKK #1#2#3.#4%
889 {%
890 \XINT_sub_KK #2{#3#4}%
891 }%
892 \def\xint_sub_KK_finish\W\xINT_sub_KK_onestep #1#2#3%
893 {%
894 \expandafter\xint_minus_thenstop
895 \romannumeral0\xINT_cuz_loop #3\W\W\W\W\W\W\W\Z
896 }%
```

3.13 *\xintiMul*

1.09a adds *\xintnum*

```
897 \def\xintiiMul {\romannumeral0\xintiimul }%
898 \def\xintiimul #1%
899 {%
900 \expandafter\xint_iimul\expandafter {\romannumeral-‘0#1}%
901 }%
902 \def\xint_iimul #1#2%
903 {%
904 \expandafter\xINT_mul_fork \romannumeral-‘0#2\Z #1\Z
905 }%
906 \def\xintiMul {\romannumeral0\xintimul }%
907 \def\xintimul #1%
908 {%
909 \expandafter\xint_mul\expandafter {\romannumeral0\xintnum{#1}}%
910 }%
911 \def\xint_mul #1#2%
912 {%
913 \expandafter\xINT_mul_fork \romannumeral0\xintnum{#2}\Z #1\Z
914 }%
915 \let\xintMul\xintiMul \let\xintmul\xintimul
916 \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

3 Package *xintcore* implementation

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.

```

917 \def\XINT_mul_fork #1#2\Z #3#4\Z
918 {%
919   \xint_UDzerofork
920     #1\XINT_mul_zero
921     #3\XINT_mul_zero
922     0{}}%
923   \krof
924   \xint_UDsignsfork
925     #1#3\XINT_mul_minusminus      % #1 = #3 = -
926     #1-{\XINT_mul_minusplus #3}%  % #1 = -
927     #3-{\XINT_mul_plusminus #1}%   % #3 = -
928     --{\XINT_mul_plusplus #1#3}%
929   \krof
930   {#2}{#4}%
931 }%
932 \def\XINT_mul_zero #1\krof #2#3{ 0}%
933 \def\XINT_mul_minusminus #1#2%
934 {%
935   \expandafter\XINT_mul_choice_a
936   \expandafter{\romannumeral0\xintlength {#2}}%
937   {\romannumeral0\xintlength {#1}}{#1}{#2}%
938 }%
939 \def\XINT_mul_minusplus #1#2#3%
940 {%
941   \expandafter\xint_minus_thenstop\romannumeral0\expandafter
942   \XINT_mul_choice_a
943   \expandafter{\romannumeral0\xintlength {#1#3}}%
944   {\romannumeral0\xintlength {#2}}{#2}{#1#3}%
945 }%
946 \def\XINT_mul_plusminus #1#2#3%
947 {%
948   \expandafter\xint_minus_thenstop\romannumeral0\expandafter
949   \XINT_mul_choice_a
950   \expandafter{\romannumeral0\xintlength {#3}}%
951   {\romannumeral0\xintlength {#1#2}}{#1#2}{#3}%
952 }%
953 \def\XINT_mul_plusplus #1#2#3#4%
954 {%
955   \expandafter\XINT_mul_choice_a
956   \expandafter{\romannumeral0\xintlength {#2#4}}%
957   {\romannumeral0\xintlength {#1#3}}{#1#3}{#2#4}%
958 }%
959 \def\XINT_mul_choice_a #1#2%
960 {%
961   \expandafter\XINT_mul_choice_b\expandafter{#2}{#1}%
962 }%
963 \def\XINT_mul_choice_b #1#2%
964 {%

```

3 Package *xintcore* implementation

```

965 \ifnum #1<\xint_c_v
966 \expandafter\XINT_mul_choice_littlebyfirst
967 \else
968 \ifnum #2<\xint_c_v
969 \expandafter\expandafter\expandafter\XINT_mul_choice_littlebysecond
970 \else
971 \expandafter\expandafter\expandafter\XINT_mul_choice_compare
972 \fi
973 \fi
974 {#1}{#2}%
975 }%
976 \def\XINT_mul_choice_littlebyfirst #1#2#3#4%
977 {%
978 \expandafter\XINT_mul_M
979 \expandafter{\the\numexpr #3\expandafter}%
980 \romannumeral0\XINT_RQ {}#4\R\R\R\R\R\R\R\R\Z \Z\Z\Z\Z
981 }%
982 \def\XINT_mul_choice_littlebysecond #1#2#3#4%
983 {%
984 \expandafter\XINT_mul_M
985 \expandafter{\the\numexpr #4\expandafter}%
986 \romannumeral0\XINT_RQ {}#3\R\R\R\R\R\R\R\R\Z \Z\Z\Z\Z
987 }%
988 \def\XINT_mul_choice_compare #1#2%
989 {%
990 \ifnum #1>#2
991 \expandafter \XINT_mul_choice_i
992 \else
993 \expandafter \XINT_mul_choice_ii
994 \fi
995 {#1}{#2}%
996 }%
997 \def\XINT_mul_choice_i #1#2%
998 {%
999 \ifnum #1<\numexpr\ifcase \numexpr (#2-\xint_c_iii)/\xint_c_iv\relax
1000 \or 330\or 168\or 109\or 80\or 66\or 52\else 0\fi\relax
1001 \expandafter\XINT_mul_choice_same
1002 \else
1003 \expandafter\XINT_mul_choice_permute
1004 \fi
1005 }%
1006 \def\XINT_mul_choice_ii #1#2%
1007 {%
1008 \ifnum #2<\numexpr\ifcase \numexpr (#1-\xint_c_iii)/\xint_c_iv\relax
1009 \or 330\or 168\or 109\or 80\or 66\or 52\else 0\fi\relax
1010 \expandafter\XINT_mul_choice_permute
1011 \else
1012 \expandafter\XINT_mul_choice_same
1013 \fi
1014 }%
1015 \def\XINT_mul_choice_same #1#2%
1016 {%

```

3 Package *xintcore* implementation

```

1017 \expandafter\XINT_mul_enter
1018 \romannumeral0\XINT_RQ {}#1\R\R\R\R\R\R\R\R\Z
1019 \Z\Z\Z\Z #2\W\W\W\W
1020 }%
1021 \def\XINT_mul_choice_permute #1#2%
1022 {%
1023 \expandafter\XINT_mul_enter
1024 \romannumeral0\XINT_RQ {}#2\R\R\R\R\R\R\R\R\Z
1025 \Z\Z\Z\Z #1\W\W\W\W
1026 }%

```

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`.

```

1027 \def\XINT_mul_Ar #1#2#3#4#5#6%
1028 {%
1029 \xint_gob_til_Z #6\xint_mul_br\Z\XINT_mul_Br #1{#6#5#4#3}{#2}%
1030 }%
1031 \def\xint_mul_br\Z\XINT_mul_Br #1#2%
1032 {%
1033 \XINT_addr_AC_checkcarry #1%
1034 }%
1035 \def\XINT_mul_Br #1#2#3#4\W\X\Y\Z #5#6#7#8%
1036 {%
1037 \expandafter\XINT_mul_ABEAr
1038 \the\numexpr #1+10#2+#8#7#6#5.{#3}#4\W\X\Y\Z
1039 }%
1040 \def\XINT_mul_ABEAr #1#2#3#4#5#6.#7%
1041 {%
1042 \XINT_mul_Ar #2{#7#6#5#4#3}%
1043 }%

```

<< 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.

```

1044 \def\XINT_mul_Mr #1%
1045 {%
1046 \expandafter\XINT_mul_Mr_checkifzeroorone\expandafter{\the\numexpr #1}%
1047 }%
1048 \def\XINT_mul_Mr_checkifzeroorone #1%
1049 {%
1050 \ifcase #1
1051 \expandafter\XINT_mul_Mr_zero
1052 \or
1053 \expandafter\XINT_mul_Mr_one
1054 \else
1055 \expandafter\XINT_mul_Nr
1056 \fi
1057 {0000}{#1}%
1058 }%
1059 \def\XINT_mul_Mr_zero #1\Z\Z\Z\Z { 0000}%

```


3 Package *xintcore* implementation

```

1060 \def\XINT_mul_Mr_one #1#2#3#4\Z\Z\Z { #4}%
1061 \def\XINT_mul_Nr #1#2#3#4#5#6#7%
1062 {%
1063   \xint_gob_til_Z #4\xint_mul_pr\Z\XINT_mul_Pr {#1}{#3}{#7#6#5#4}{#2}{#3}%
1064 }%
1065 \def\XINT_mul_Pr #1#2#3%
1066 {%
1067   \expandafter\XINT_mul_Lr\the\numexpr \xint_c_x^viii+#1+#2*#3\relax
1068 }%
1069 \def\XINT_mul_Lr #1#2#3#4#5#6#7#8#9%
1070 {%
1071   \XINT_mul_Nr {#1#2#3#4}{#9#8#7#6#5}%
1072 }%
1073 \def\xint_mul_pr\Z\XINT_mul_Pr #1#2#3#4#5%
1074 {%
1075   \xint_gob_til_zeros_iv #1\XINT_mul_Mr_end_nocarry 0000%
1076   \XINT_mul_Mr_end_carry #1{#4}%
1077 }%
1078 \def\XINT_mul_Mr_end_nocarry 0000\XINT_mul_Mr_end_carry 0000#1{ #1}%
1079 \def\XINT_mul_Mr_end_carry #1#2#3#4#5{ #5#4#3#2#1}%

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

1080 \def\XINT_mul_M #1%
1081 {%
1082   \expandafter\XINT_mul_M_checkifzeroorone\expandafter{\the\numexpr #1}%
1083 }%
1084 \def\XINT_mul_M_checkifzeroorone #1%
1085 {%
1086   \ifcase #1
1087     \expandafter\XINT_mul_M_zero
1088   \or
1089     \expandafter\XINT_mul_M_one
1090   \else
1091     \expandafter\XINT_mul_N
1092   \fi
1093   {0000}{#1}%
1094 }%
1095 \def\XINT_mul_M_zero #1\Z\Z\Z { 0}%
1096 \def\XINT_mul_M_one #1#2#3#4\Z\Z\Z
1097 {%
1098   \expandafter\xint_cleanupzeros_andstop\romannumeral0\xintreverseorder{#4}%
1099 }%
1100 \def\XINT_mul_N #1#2#3#4#5#6#7%
1101 {%
1102   \xint_gob_til_Z #4\xint_mul_p\Z\XINT_mul_P {#1}{#3}{#7#6#5#4}{#2}{#3}%
1103 }%
1104 \def\XINT_mul_P #1#2#3%
1105 {%
1106   \expandafter\XINT_mul_L\the\numexpr \xint_c_x^viii+#1+#2*#3\relax

```

3 Package *xintcore* implementation

```

1107 }%
1108 \def\XINT_mul_L 1#1#2#3#4#5#6#7#8#9%
1109 {%
1110   \XINT_mul_N {#1#2#3#4}{#5#6#7#8#9}%
1111 }%
1112 \def\xint_mul_p\Z\XINT_mul_P #1#2#3#4#5%
1113 {%
1114   \XINT_mul_M_end #1#4%
1115 }%
1116 \edef\XINT_mul_M_end #1#2#3#4#5#6#7#8%
1117 {%
1118   \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7#8\relax
1119 }%

```

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 significatif) 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 carrée: 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.

```

1120 \def\XINT_mul_enter #1\Z\Z\Z\Z #2#3#4#5%
1121 {%
1122   \xint_gob_til_W #5\XINT_mul_exit_a\W
1123   \XINT_mul_start {#2#3#4#5}#1\Z\Z\Z\Z
1124 }%
1125 \def\XINT_mul_exit_a\W\XINT_mul_start #1%
1126 {%
1127   \XINT_mul_exit_b #1%
1128 }%
1129 \def\XINT_mul_exit_b #1#2#3#4%
1130 {%
1131   \xint_gob_til_W
1132   #2\XINT_mul_exit_ci
1133   #3\XINT_mul_exit_cii
1134   \W\XINT_mul_exit_ciii #1#2#3#4%
1135 }%
1136 \def\XINT_mul_exit_ciii #1\W #2\Z\Z\Z\Z \W\W\W
1137 {%
1138   \XINT_mul_M {#1}#2\Z\Z\Z\Z
1139 }%
1140 \def\XINT_mul_exit_cii\W\XINT_mul_exit_ciii #1\W\W #2\Z\Z\Z\Z \W\W
1141 {%
1142   \XINT_mul_M {#1}#2\Z\Z\Z\Z

```

3 Package *xintcore* implementation

```

1143 }%
1144 \def\XINT_mul_exit_ci\W\XINT_mul_exit_cii
1145         \W\XINT_mul_exit_ciii #1\W\W\W #2\Z\Z\Z\Z \W
1146 {%
1147     \XINT_mul_M {#1}#2\Z\Z\Z\Z
1148 }%
1149 \def\XINT_mul_start #1#2\Z\Z\Z\Z
1150 {%
1151     \expandafter\XINT_mul_main\expandafter
1152     {\romannumeral0\XINT_mul_Mr {#1}#2\Z\Z\Z\Z}#2\Z\Z\Z\Z
1153 }%
1154 \def\XINT_mul_main #1#2\Z\Z\Z\Z #3#4#5#6%
1155 {%
1156     \xint_gob_til_W #6\XINT_mul_finish_a\W
1157     \XINT_mul_compute {#3#4#5#6}{#1}#2\Z\Z\Z\Z
1158 }%
1159 \def\XINT_mul_compute #1#2#3\Z\Z\Z\Z
1160 {%
1161     \expandafter\XINT_mul_main\expandafter
1162     {\romannumeral0\expandafter
1163     \XINT_mul_Ar\expandafter0\expandafter{\expandafter}%
1164     \romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z
1165     \W\X\Y\Z 0000#2\W\X\Y\Z }#3\Z\Z\Z\Z
1166 }%

```

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.

```

1167 \def\XINT_mul_finish_a\W\XINT_mul_compute #1%
1168 {%
1169     \XINT_mul_finish_b #1%
1170 }%
1171 \def\XINT_mul_finish_b #1#2#3#4%
1172 {%
1173     \xint_gob_til_W
1174     #1\XINT_mul_finish_c
1175     #2\XINT_mul_finish_ci
1176     #3\XINT_mul_finish_cii
1177     \W\XINT_mul_finish_ciii #1#2#3#4%
1178 }%
1179 \def\XINT_mul_finish_ciii #1\W #2#3\Z\Z\Z\Z \W\W\W
1180 {%
1181     \expandafter\XINT_addm_A\expandafter0\expandafter{\expandafter}%
1182     \romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z \W\X\Y\Z 000#2\W\X\Y\Z
1183 }%
1184 \def\XINT_mul_finish_cii
1185     \W\XINT_mul_finish_ciii #1\W\W #2#3\Z\Z\Z\Z \W\W
1186 {%
1187     \expandafter\XINT_addm_A\expandafter0\expandafter{\expandafter}%
1188     \romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z \W\X\Y\Z 00#2\W\X\Y\Z
1189 }%

```

3 Package *xintcore* implementation

```

1190 \def\XINT_mul_finish_ci #1\XINT_mul_finish_ciii #2\W\W\W #3#4\Z\Z\Z\Z \W
1191 {%
1192     \expandafter\XINT_addm_A\expandafter0\expandafter{\expandafter}%
1193     \romannumeral0\XINT_mul_Mr {#2}#4\Z\Z\Z\Z \W\X\Y\Z 0#3\W\X\Y\Z
1194 }%
1195 \def\XINT_mul_finish_c #1\XINT_mul_finish_ciii \W\W\W\W #2#3\Z\Z\Z\Z
1196 {%
1197     \expandafter\xint_cleanupzeros_andstop\romannumeral0\xintreverseorder{#2}%
1198 }%

```

Variante de la Multiplication

\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 version de \XINT_mul_enter. Je pourrais économiser des macros et fusionner \XINT_mul_enter et \XINT_mulr_enter. Une autre fois.

```

1199 \def\XINT_mulr_enter #1\Z\Z\Z\Z #2#3#4#5%
1200 {%
1201     \xint_gob_til_W #5\XINT_mulr_exit_a\W
1202     \XINT_mulr_start {#2#3#4#5}#1\Z\Z\Z\Z
1203 }%
1204 \def\XINT_mulr_exit_a\W\XINT_mulr_start #1%
1205 {%
1206     \XINT_mulr_exit_b #1%
1207 }%
1208 \def\XINT_mulr_exit_b #1#2#3#4%
1209 {%
1210     \xint_gob_til_W
1211     #2\XINT_mulr_exit_ci
1212     #3\XINT_mulr_exit_cii
1213     \W\XINT_mulr_exit_ciii #1#2#3#4%
1214 }%
1215 \def\XINT_mulr_exit_ciii #1\W #2\Z\Z\Z\Z \W\W\W
1216 {%
1217     \XINT_mul_Mr {#1}#2\Z\Z\Z\Z
1218 }%
1219 \def\XINT_mulr_exit_cii\W\XINT_mulr_exit_ciii #1\W\W #2\Z\Z\Z\Z \W\W
1220 {%
1221     \XINT_mul_Mr {#1}#2\Z\Z\Z\Z
1222 }%
1223 \def\XINT_mulr_exit_ci\W\XINT_mulr_exit_cii
1224     \W\XINT_mulr_exit_ciii #1\W\W\W #2\Z\Z\Z\Z \W
1225 {%
1226     \XINT_mul_Mr {#1}#2\Z\Z\Z\Z
1227 }%
1228 \def\XINT_mulr_start #1#2\Z\Z\Z\Z
1229 {%
1230     \expandafter\XINT_mulr_main\expandafter
1231     {\romannumeral0\XINT_mul_Mr {#1}#2\Z\Z\Z\Z}#2\Z\Z\Z\Z
1232 }%

```

3 Package *xintcore* implementation

```
1233 \def\XINT_mulr_main #1#2\Z\Z\Z\Z #3#4#5#6%
1234 {%
1235     \xint_gob_til_W #6\XINT_mulr_finish_a\W
1236     \XINT_mulr_compute {#3#4#5#6}{#1}#2\Z\Z\Z\Z
1237 }%
1238 \def\XINT_mulr_compute #1#2#3\Z\Z\Z\Z
1239 {%
1240     \expandafter\XINT_mulr_main\expandafter
1241     {\romannumeral0\expandafter
1242     \XINT_mul_Ar\expandafter0\expandafter{\expandafter}%
1243     \romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z
1244     \W\X\Y\Z 0000#2\W\X\Y\Z }#3\Z\Z\Z\Z
1245 }%
1246 \def\XINT_mulr_finish_a\W\XINT_mulr_compute #1%
1247 {%
1248     \XINT_mulr_finish_b #1%
1249 }%
1250 \def\XINT_mulr_finish_b #1#2#3#4%
1251 {%
1252     \xint_gob_til_W
1253     #1\XINT_mulr_finish_c
1254     #2\XINT_mulr_finish_ci
1255     #3\XINT_mulr_finish_cii
1256     \W\XINT_mulr_finish_ciii #1#2#3#4%
1257 }%
1258 \def\XINT_mulr_finish_ciii #1\W #2#3\Z\Z\Z\Z \W\W\W
1259 {%
1260     \expandafter\XINT_addp_A\expandafter0\expandafter{\expandafter}%
1261     \romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z \W\X\Y\Z 000#2\W\X\Y\Z
1262 }%
1263 \def\XINT_mulr_finish_cii
1264     \W\XINT_mulr_finish_ciii #1\W\W #2#3\Z\Z\Z\Z \W\W
1265 {%
1266     \expandafter\XINT_addp_A\expandafter0\expandafter{\expandafter}%
1267     \romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z \W\X\Y\Z 00#2\W\X\Y\Z
1268 }%
1269 \def\XINT_mulr_finish_ci #1\XINT_mulr_finish_ciii #2\W\W\W #3#4\Z\Z\Z\Z \W
1270 {%
1271     \expandafter\XINT_addp_A\expandafter0\expandafter{\expandafter}%
1272     \romannumeral0\XINT_mul_Mr {#2}#4\Z\Z\Z\Z \W\X\Y\Z 0#3\W\X\Y\Z
1273 }%
1274 \def\XINT_mulr_finish_c #1\XINT_mulr_finish_ciii \W\W\W\W #2#3\Z\Z\Z\Z { #2}%

```

3.14 *\xintiSqr*

```
1275 \def\xintiiSqr {\romannumeral0\xintiisqr }%
1276 \def\xintiisqr #1%
1277 {%
1278     \expandafter\XINT_sqr\expandafter {\romannumeral0\xintiiaabs{#1}}%
1279 }%
1280 \def\xintiSqr {\romannumeral0\xintisqr }%
1281 \def\xintisqr #1%
1282 {%

```

```

1283 \expandafter\XINT_sqr\expandafter {\romannumeral0\xintiabs{#1}}%
1284 }%
1285 \let\xintSqr\xintiSqr \let\xintsqr\xintisqr
1286 \def\XINT_sqr #1%
1287 {%
1288 \expandafter\XINT_mul_enter
1289 \romannumeral0%
1290 \XINT_RQ {}#1\R\R\R\R\R\R\R\R\Z
1291 \Z\Z\Z\Z #1\W\W\W\W
1292 }%

```

3.15 \xintiPow

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).

```

1293 \def\xintiiPow {\romannumeral0\xintiipow }%
1294 \def\xintiipow #1%
1295 {%
1296 \expandafter\xint_pow\romannumeral-'0#1\Z%
1297 }%
1298 \def\xintiPow {\romannumeral0\xintipow }%
1299 \def\xintipow #1%
1300 {%
1301 \expandafter\xint_pow\romannumeral0\xintnum{#1}\Z%
1302 }%
1303 \let\xintPow\xintiPow \let\xintpow\xintipow
1304 \def\xint_pow #1#2\Z
1305 {%
1306 \xint_UDsignfork
1307 #1\XINT_pow_Aneg
1308 -\XINT_pow_Anonneg
1309 \krof
1310 #1{#2}%
1311 }%
1312 \def\XINT_pow_Aneg #1#2#3%
1313 {%
1314 \expandafter\XINT_pow_Aneg_\expandafter{\the\numexpr #3}\#2\Z
1315 }%

```

3 Package *xintcore* implementation

```

1316 \def\XINT_pow_Aneg_ #1%
1317 {%
1318   \ifodd #1
1319     \expandafter\XINT_pow_Aneg_Bodd
1320   \fi
1321   \XINT_pow_Anonneg_ {#1}%
1322 }%
1323 \def\XINT_pow_Aneg_Bodd #1%
1324 {%
1325   \expandafter\XINT_opp\romannumeral0\XINT_pow_Anonneg_
1326 }%

```

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

```

1327 \def\XINT_pow_Anonneg_ #1#2#3%
1328 {%
1329   \expandafter\XINT_pow_Anonneg_\expandafter {\the\numexpr #3}#1#2\Z
1330 }%

```

#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.

```

1331 \def\XINT_pow_Anonneg_ #1#2#3\Z
1332 {%
1333   \if\relax #3\relax\xint_dothis
1334     {\ifcase #2 \expandafter\XINT_pow_AisZero
1335       \or\expandafter\XINT_pow_AisOne
1336       \else\expandafter\XINT_pow_AatleastTwo
1337     \fi }\fi
1338   \xint_orthat \XINT_pow_AatleastTwo {#1}{#2#3}%
1339 }%
1340 \def\XINT_pow_AisOne #1#2{ 1}%

```

#1 = B

```

1341 \def\XINT_pow_AisZero #1#2%
1342 {%
1343   \ifcase\XINT_cntSgn #1\Z
1344     \xint_afterfi { 1}%
1345   \or
1346     \xint_afterfi { 0}%
1347   \else
1348     \xint_afterfi {\xintError:DivisionByZero\space 0}%
1349   \fi
1350 }%
1351 \def\XINT_pow_AatleastTwo #1%
1352 {%
1353   \ifcase\XINT_cntSgn #1\Z
1354     \expandafter\XINT_pow_BisZero
1355   \or
1356     \expandafter\XINT_pow_checkBsize
1357   \else
1358     \expandafter\XINT_pow_BisNegative

```

3 Package *xintcore* implementation

```

1359 \fi
1360 {#1}%
1361 }%
1362 \edef\XINT_pow_BisNegative #1#2%
1363 {\noexpand\xintError:FractionRoundedToZero\space 0}%
1364 \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.

1365 \def\XINT_pow_checkBsize #1%
1366 {%
1367 \ifnum #1>1000000
1368 \expandafter\XINT_pow_BtooBig
1369 \else
1370 \expandafter\XINT_pow_loopI
1371 \fi
1372 {#1}%
1373 }%
1374 \edef\XINT_pow_BtooBig #1#2{\noexpand\xintError:ExponentTooBig\space 0}%
1375 \def\XINT_pow_loopI #1%
1376 {%
1377 \ifnum #1=\xint_c_i\XINT_pow_Iend\fi
1378 \ifodd #1
1379 \expandafter\XINT_pow_loopI_odd
1380 \else
1381 \expandafter\XINT_pow_loopI_even
1382 \fi
1383 {#1}%
1384 }%
1385 \edef\XINT_pow_Iend\fi #1\fi #2#3{\noexpand\fi\space #3}%
1386 \def\XINT_pow_loopI_even #1#2%
1387 {%
1388 \expandafter\XINT_pow_loopI\expandafter
1389 {\the\numexpr #1/\xint_c_ii\expandafter}\expandafter
1390 {\romannumeral0\xintiisqr {#2}}%
1391 }%
1392 \def\XINT_pow_loopI_odd #1#2%
1393 {%
1394 \expandafter\XINT_pow_loopI_odda\expandafter
1395 {\romannumeral0\XINT_RQ {}#2\R\R\R\R\R\R\R\R\Z }{#1}{#2}%
1396 }%
1397 \def\XINT_pow_loopI_odda #1#2#3%
1398 {%
1399 \expandafter\XINT_pow_loopII\expandafter
1400 {\the\numexpr #2/\xint_c_ii-\xint_c_i\expandafter}\expandafter
1401 {\romannumeral0\xintiisqr {#3}}{#1}%
1402 }%
1403 \def\XINT_pow_loopII #1%
1404 {%
1405 \ifnum #1 = \xint_c_i\XINT_pow_IIend\fi
1406 \ifodd #1
1407 \expandafter\XINT_pow_loopII_odd

```



```

1408 \else
1409 \expandafter\XINT_pow_loopII_even
1410 \fi
1411 {#1}%
1412 }%
1413 \def\XINT_pow_loopII_even #1#2%
1414 {%
1415 \expandafter\XINT_pow_loopII\expandafter
1416 {\the\numexpr #1/\xint_c_ii\expandafter}\expandafter
1417 {\romannumeral0\xintiisqr {#2}}%
1418 }%
1419 \def\XINT_pow_loopII_odd #1#2#3%
1420 {%
1421 \expandafter\XINT_pow_loopII_odda\expandafter
1422 {\romannumeral0\XINT_mulr_enter #3\Z\Z\Z #2\W\W\W\W}{#1}{#2}%
1423 }%
1424 \def\XINT_pow_loopII_odda #1#2#3%
1425 {%
1426 \expandafter\XINT_pow_loopII\expandafter
1427 {\the\numexpr #2/\xint_c_ii-\xint_c_i\expandafter}\expandafter
1428 {\romannumeral0\xintiisqr {#3}}{#1}%
1429 }%
1430 \def\XINT_pow_IIend\fi #1\fi #2#3#4%
1431 {%
1432 \fi\XINT_mul_enter #4\Z\Z\Z\Z #3\W\W\W\W
1433 }%

```

3.16 `\xintiDivision`, `\xintiQuo`, `\xintiRem`

The 1.09a release inserted the use of `\xintnum`. The `\xintiDivision` 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) still does not impact the input save stack. One can have a produced quotient longer than $4 \times 5000 = 20000$ digits, and no need to worry about consequences propagating to `\xintTrunc`, `\xin-`

3 Package *xintcore* implementation

`tRound`, `\xintFloat`, `\xintFloatSqrt`, etc... and all other places using the division. See also `\xintXTrunc` in this context.

```

1434 \def\xintiQuo {\romannumeral0\xinti quo }%
1435 \def\xintiiRem {\romannumeral0\xintii rem }%
1436 \def\xinti quo {\expandafter\xint_firstoftwo_thenstop\romannumeral0\xinti idivision }%
1437 \def\xintii rem {\expandafter\xint_secondoftwo_thenstop\romannumeral0\xintii idivision }%
1438 \def\xintiQuo {\romannumeral0\xinti quo }%
1439 \def\xintiRem {\romannumeral0\xinti rem }%
1440 \def\xinti quo {\expandafter\xint_firstoftwo_thenstop\romannumeral0\xinti idivision }%
1441 \def\xinti rem {\expandafter\xint_secondoftwo_thenstop\romannumeral0\xinti idivision }%
1442 \let\xintQuo\xintiQuo\let\xintquo\xinti quo % deprecated (1.1)
1443 \let\xintRem\xintiRem\let\xintrem\xinti rem % deprecated (1.1)

```

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

```

1444 \def\xintiDivision {\romannumeral0\xinti division }%
1445 \def\xintidivision #1{\expandafter\XINT_division\romannumeral0\xintnum{\#1}\Z }%
1446 \let\xintDivision\xintiDivision \let\xintdivision\xintidivision % deprecated
1447 \def\XINT_division #1#2\Z #3{\expandafter\XINT_iidivision_a\expandafter #1%
1448 \romannumeral0\xintnum{\#3}\Z #2\Z }%
1449 \def\xintiiDivision {\romannumeral0\xintii division }%
1450 \def\xintiiidivision #1{\expandafter\XINT_iidivision \romannumeral-'\#1\Z }%
1451 \def\XINT_iidivision #1#2\Z #3{\expandafter\XINT_iidivision_a\expandafter #1%
1452 \romannumeral-'\#3\Z #2\Z }%
1453 \def\XINT_iidivision_a #1#2\Z #1 de A, #2 de B.
1454 {%
1455 \if0#2\xint_dothis\XINT_iidivision_divbyzero\fi
1456 \if0#1\xint_dothis\XINT_iidivision_aiszero\fi
1457 \if-#2\xint_dothis{\expandafter\XINT_iidivision_bneg
1458 \romannumeral0\XINT_iidivision_bpos #1}\fi
1459 \xint_orthat{\XINT_iidivision_bpos #1#2}%
1460 }%
1461 \def\XINT_iidivision_divbyzero #1\Z #2\Z {\xintError:DivisionByZero\space {0}{0}}%
1462 \def\XINT_iidivision_aiszero #1\Z #2\Z { {0}{0}}%
1463 \def\XINT_iidivision_bneg #1\Z q->-q, r unchanged
1464 {\expandafter\space\expandafter{\romannumeral0\XINT_opp #1}}%
1465 \def\XINT_iidivision_bpos #1\Z
1466 {%
1467 \xint_UDsignfork
1468 #1\XINT_iidivision_aneg
1469 -{\XINT_iidivision_apos #1}%
1470 \krof
1471 }%
1472 \def\XINT_iidivision_apos #1#2\Z #3\Z{\XINT_div_prepare {\#2}{\#1\#3}}%
1473 \def\XINT_iidivision_aneg #1\Z #2\Z
1474 {\expandafter
1475 \XINT_iidivision_aneg_b\romannumeral0\XINT_div_prepare {\#1}{\#2}{\#1}}%
1476 \def\XINT_iidivision_aneg_b #1#2\Z{\if0\XINT_Sgn #2\Z
1477 \expandafter\XINT_iidivision_aneg_rzero
1478 \else
1479 \expandafter\XINT_iidivision_aneg_rpos

```

3 Package *xintcore* implementation

```

1480 \fi {#1}{#2}}%
1481 \def\XINT_iidivision_aneg_rzero #1#2#3{ {-#1}{0}}% necessarily q was >0
1482 \def\XINT_iidivision_aneg_rpos #1%
1483 {%
1484 \expandafter\XINT_iidivision_aneg_end\expandafter
1485 {\expandafter-\romannumeral0\xintinc {#1}}% q-> -(1+q)
1486 }%
1487 \def\XINT_iidivision_aneg_end #1#2#3%
1488 {%
1489 \expandafter\xint_exchangetwo_keepbraces_thenstop
1490 \expandafter{\romannumeral0\XINT_sub_pre {#3}{#2}}{#1}% r-> b-r
1491 }%

```

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

```

1492 \def\XINT_div_prepare #1%
1493 {%
1494 \expandafter \XINT_div_prepareB_aa \expandafter
1495 {\romannumeral0\xintlength {#1}}{#1}% B > 0 ici
1496 }%
1497 \def\XINT_div_prepareB_aa #1%
1498 {%
1499 \ifnum #1=\xint_c_i
1500 \expandafter\XINT_div_prepareB_onedigit
1501 \else
1502 \expandafter\XINT_div_prepareB_a
1503 \fi
1504 {#1}%
1505 }%
1506 \def\XINT_div_prepareB_a #1%
1507 {%
1508 \expandafter\XINT_div_prepareB_c\expandafter
1509 {\the\numexpr \xint_c_iv*((#1+\xint_c_i)/\xint_c_iv)}{#1}%
1510 }%

```

B=1 and B=2 treated specially.

```

1511 \def\XINT_div_prepareB_onedigit #1#2%
1512 {%
1513 \ifcase#2
1514 \or\expandafter\XINT_div_BisOne
1515 \or\expandafter\XINT_div_BisTwo
1516 \else\expandafter\XINT_div_prepareB_e
1517 \fi {000}{0}{4}{#2}%
1518 }%
1519 \def\XINT_div_BisOne #1#2#3#4#5{ {#5}{0}}%
1520 \def\XINT_div_BisTwo #1#2#3#4#5%
1521 {%
1522 \expandafter\expandafter\expandafter\XINT_div_BisTwo_a
1523 \ifodd\xintiilDg{#5} \expandafter1\else \expandafter0\fi {#5}%
1524 }%
1525 \edef\XINT_div_BisTwo_a #1#2%
1526 {%

```

3 Package *xintcore* implementation

```

1527 \noexpand\expandafter\space\noexpand\expandafter
1528 {\noexpand\romannumeral0\noexpand\xinthalft {#2}}{#1}%
1529 }%

```

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

```

1530 \def\XINT_div_prepareB_c #1#2%
1531 {%
1532 \csname XINT_div_prepareB_d\romannumeral\numexpr#1-#2\endcsname
1533 {#1}%
1534 }%
1535 \def\XINT_div_prepareB_d {\XINT_div_prepareB_e {}{0000}}%
1536 \def\XINT_div_prepareB_di {\XINT_div_prepareB_e {0}{000}}%
1537 \def\XINT_div_prepareB_dii {\XINT_div_prepareB_e {00}{00}}%
1538 \def\XINT_div_prepareB_diii {\XINT_div_prepareB_e {000}{0}}%
1539 \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_checksngx 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

```

1540 \def\XINT_div_prepareB_e #1#2#3#4%
1541 {%
1542 \ifnum#3=\xint_c_iv\expandafter\XINT_div_prepareLittleB_f
1543 \else\expandafter\XINT_div_prepareB_f
1544 \fi
1545 #4#1{#3}{#2}{#1}%
1546 }%

```

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.

```

1547 \def\XINT_div_prepareB_f #1#2#3#4#5#{%
1548 \expandafter\XINT_div_prepareB_g
1549 \the\numexpr #1#2#3#4+\xint_c_i\expandafter
1550 .\the\numexpr (#1#2#3#4+\xint_c_i)/\xint_c_ii\expandafter
1551 .\romannumeral0\xintreverseorder {#1#2#3#4#5}.{#1#2#3#4}%
1552 }%
1553 \def\XINT_div_prepareLittleB_f #1#{%
1554 \expandafter\XINT_div_prepareB_g \the\numexpr #1/\xint_c_ii.{}.{#1}%
1555 }%

```

#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.

```

1556 \def\XINT_div_prepareB_g #1.#2.#3.#4#5#6#7#8%
1557 {%
1558 \XINT_div_prepareA_a {#8#7}{#5}{#1}{#2}{#4}{#3}{#6}%
1559 }%

```

3 Package *xintcore* implementation

```

A, K, {x'yx}, B«C»

1560 \def\XINT_div_prepareA_a #1%
1561 {%
1562   \expandafter\XINT_div_prepareA_b\expandafter
1563     {\romannumeral0\xintlength {#1}}{#1}%
1564 }%

L0, A, K, {x'yx}, B«C»

1565 \def\XINT_div_prepareA_b #1%
1566 {%
1567   \expandafter\XINT_div_prepareA_c\expandafter
1568     {\the\numexpr \xint_c_iv*((#1+\xint_c_iv)/\xint_c_iv)}{#1}%
1569 }%

L, L0, A, K, {x'yx}, B, «C»

1570 \def\XINT_div_prepareA_c #1#2%
1571 {%
1572   \csname XINT_div_prepareA_d\romannumeral\numexpr #1-#2\endcsname
1573     {#1}%
1574 }%
1575 \def\XINT_div_prepareA_d {\XINT_div_prepareA_e {}}%
1576 \def\XINT_div_prepareA_di {\XINT_div_prepareA_e {0}}%
1577 \def\XINT_div_prepareA_dii {\XINT_div_prepareA_e {00}}%
1578 \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»

1579 \def\XINT_div_prepareA_e #1#2#3#4#5%
1580 {%
1581   \XINT_div_start_a {#2}{#4}{#1#3}%5%
1582 }%

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

1583 \def\XINT_div_start_a #1#2%
1584 {%
1585   \ifnum #2=\xint_c_iv \expandafter\XINT_div_little_b
1586   \else
1587     \ifnum #1 < #2
1588       \expandafter\expandafter\expandafter\XINT_div_III_aa
1589     \else
1590       \expandafter\expandafter\expandafter\XINT_div_start_b
1591     \fi
1592   \fi
1593   {#1}{#2}%
1594 }%

L, K, A, x',y,x, B, «C».

1595 \def\XINT_div_III_aa #1#2#3#4#5#6#7%
1596 {%
1597   \expandafter\expandafter\expandafter
1598     \XINT_div_III_b\xint_cleanupzeros_nostop #3.{0000}%
1599 }%

```

3 Package *xintcore* implementation

R.Q«c».

```

1600 \def\XINT_div_III_b #1%
1601 {%
1602   \if0#1%
1603     \expandafter\XINT_div_III_bRzero
1604   \else
1605     \expandafter\XINT_div_III_bRpos
1606   \fi
1607   #1%
1608 }%
1609 \def\XINT_div_III_bRzero 0.#1#2%
1610 {%
1611   \expandafter\space\expandafter
1612   {\romannumeral0\XINT_cuz_loop #1\W\W\W\W\W\W\W\Z}{0}%
1613 }%
1614 \def\XINT_div_III_bRpos #1.#2#3%
1615 {%
1616   \expandafter\XINT_div_III_c \XINT_div_cleanR #1#3.{#2}%
1617 }%
1618 \def\XINT_div_III_c #1#2%
1619 {%
1620   \expandafter\space\expandafter
1621   {\romannumeral0\XINT_cuz_loop #2\W\W\W\W\W\W\W\Z}{#1}%
1622 }%

```

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

```

1623 \def\XINT_div_start_b #1#2#3#4#5#6%
1624 {%
1625   \XINT_div_start_c {#2}.#3.{#6}{{#1}{#2}}{{#4}{#5}}{{#6}}%
1626 }%

```

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

```

1627 \def\XINT_div_start_c #1#2.#3#4#5#6%
1628 {%
1629   \ifnum #1=\xint_c_iv\XINT_div_start_ca\fi
1630   \expandafter\XINT_div_start_c\expandafter
1631   {\the\numexpr #1-\xint_c_iv}#2#3#4#5#6.%
1632 }%
1633 \def\XINT_div_start_ca\fi\expandafter\XINT_div_start_c\expandafter
1634   #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.

```

1635 \def\XINT_div_start_d #1#2.#3.#4#5#6%
1636 {%
1637   \XINT_div_I_a {#1}{#4}{#2}{#6}{0000}#5{#3}{#6}{}%
1638 }%

```

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

3 Package *xintcore* implementation

```

1639 \def\XINT_div_I_a #1#2%
1640 {%
1641   \expandafter\XINT_div_I_b\the\numexpr #1/#2.{#1}{#2}%
1642 }%
1643 \def\XINT_div_I_b #1%
1644 {%
1645   \xint_gob_til_zero #1\XINT_div_I_czero 0\XINT_div_I_c #1%
1646 }%

  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»

1647 \def\XINT_div_I_czero 0%
1648   \XINT_div_I_c 0.#1#2#3#4#5{\XINT_div_I_g {#5}{#3}}%
1649 \def\XINT_div_I_c #1.#2#3%
1650 {%
1651   \expandafter\XINT_div_I_da\the\numexpr #2-#1*#3.#1.%
1652 }%

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

1653 \def\XINT_div_I_da #1.%
1654 {%
1655   \ifnum #1>\xint_c_ix
1656     \expandafter\XINT_div_I_dP
1657   \else
1658     \ifnum #1<\xint_c_
1659       \expandafter\expandafter\expandafter\XINT_div_I_dN
1660     \else
1661       \expandafter\expandafter\expandafter\XINT_div_I_db
1662     \fi
1663   \fi
1664 }%
1665 \def\XINT_div_I_dN #1.%
1666 {%
1667   \expandafter\XINT_div_I_dP\the\numexpr #1-\xint_c_i.%
1668 }%
1669 \def\XINT_div_I_db #1.#2#3% #1=q=un chiffre, #2=alpha, #3=B
1670 {%
1671   \expandafter\XINT_div_I_dc\expandafter
1672   {\romannumeral0\expandafter\XINT_div_sub_xp\expandafter
1673    {\romannumeral0\xintreverseorder{#2}}}%
1674   {\romannumeral0\XINT_mul_Mr {#1}{#3}\Z\Z\Z }%
1675   #1{#2}{#3}%
1676 }%
1677 \def\XINT_div_I_dc #1#2%
1678 {%
1679   \if-#1% s'arranger pour que si n'egatif on ait renvoy'e alpha=-.
1680     \expandafter\xint_firstoftwo
1681   \else\expandafter\xint_secondoftwo\fi
1682   {\expandafter\XINT_div_I_dP\the\numexpr #2-\xint_c_i.}%
1683   {\XINT_div_I_e {#1}{#2}}%
1684 }%

```

3 Package *xintcore* implementation

```

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

1685 \def\XINT_div_I_e #1#2#3#4#5%
1686 {%
1687     \expandafter\XINT_div_I_f \the\numexpr \xint_c_x^iv+#2+#5{#1}%
1688 }%

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»

1689 \def\XINT_div_I_dP #1.#2#3#4%
1690 {%
1691     \expandafter \XINT_div_I_f \the\numexpr \xint_c_x^iv+#1+#4\expandafter
1692     {\romannumeral0\expandafter\XINT_div_sub_xpxp\expandafter
1693     {\romannumeral0\xintreverseorder{#2}}}%
1694     {\romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z }}%
1695 }%

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

1696 \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}

1697 \def\XINT_div_I_g #1#2#3#4#5#6#7#8#9%
1698 {%
1699     \ifnum#3=#4
1700         \expandafter\XINT_div_III_ab
1701     \else
1702         \expandafter\XINT_div_I_h
1703     \fi
1704     {#5}#2.#7.{#5}{#6}{#4}{#3}}{#8}{#9#1}%
1705 }%

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

1706 \def\XINT_div_III_ab #1#2.#3.#4#5%
1707 {%
1708     \expandafter\XINT_div_III_b
1709     \romannumeral0\XINT_cuz_loop #2#3\W\W\W\W\W\W\Z.%
1710 }%

#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»

1711 \def\XINT_div_I_h #1.#2#3#4#5#6.#7#8%
1712 {%
1713     \XINT_div_II_b #1#2#3#4#5.{#8}{#7}{#6}{#8}%
1714 }%

{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

```


3 Package *xintcore* implementation

```

1715 \def\XINT_div_II_b #1#2#3#4#5#6#7#8#9%
1716 {%
1717     \xint_gob_til_zeros_iv #2#3#4#5\XINT_div_II_skipc 0000%
1718     \XINT_div_II_c #1{#2#3#4#5}{#6#7#8#9}%
1719 }%

    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»

1720 \def\XINT_div_II_skipc 0000\XINT_div_II_c #1#2#3#4#5.#6#7%
1721 {%
1722     \XINT_div_II_k #7{#4#5}{#6}{0000}%
1723 }%

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

1724 \def\XINT_div_II_c #1#2#3#4%
1725 {%
1726     \expandafter\XINT_div_II_d\the\numexpr (#3#4+#2)/#1+\xint_c_ixixixix\relax
1727     {#1}{#2}#3#4%
1728 }%

    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»

1729 \def\XINT_div_II_d 1#1#2#3#4#5#6#7.#8%
1730 {%
1731     \expandafter\XINT_div_II_e
1732     \romannumeral0\expandafter\XINT_div_sub_xp xp\expandafter
1733     {\romannumeral0\xintreverseorder{#7}}%
1734     {\romannumeral0\XINT_mul_Mr {#1#2#3#4}#8\Z\Z\Z }.%
1735     {#5}{#6}{#8}{#1#2#3#4}%
1736 }%

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

1737 \def\XINT_div_II_e #1#2#3#4%
1738 {%
1739     \xint_gob_til_zeros_iv #1#2#3#4\XINT_div_II_skipf 0000%
1740     \XINT_div_II_f #1#2#3#4%
1741 }%

    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»

1742 \def\XINT_div_II_skipf 0000\XINT_div_II_f 0000#1.#2#3#4#5#6%
1743 {%
1744     \XINT_div_II_k #6{#1}{#4}{#5}%
1745 }%

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

1746 \def\XINT_div_II_f #1#2#3#4#5#6#7#8#9.%
1747 {%
1748     \XINT_div_II_fa {#1#2#3#4#5#6#7#8}{#1#2#3#4#5#6#7#8#9}%
1749 }%

```

3 Package *xintcore* implementation

```

1750 \def\XINT_div_II_fa #1#2#3#4%
1751 {%
1752   \expandafter\XINT_div_II_g\expandafter
1753     {\the\numexpr (#1+#4)/#3-\xint_c_i}{#2}%
1754 }%

#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»

1755 \def\XINT_div_II_g #1#2#3#4%
1756 {%
1757   \expandafter \XINT_div_II_h
1758   \the\numexpr #4+#1+\xint_c_x^iv\expandafter\expandafter\expandafter
1759   {\expandafter\xint_gobble_iv
1760   \romannumeral0\expandafter\XINT_div_sub_xp xp\expandafter
1761     {\romannumeral0\xintreverseorder{#2}}}%
1762     {\romannumeral0\XINT_mul_Mr {#1}#3\Z\Z\Z\Z }{#3}%
1763 }%

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»

1764 \def\XINT_div_II_h #1#2#3#4#5#6#7%
1765 {%
1766   \XINT_div_II_k #7{#5}{#6}{#1#2#3#4}%
1767 }%

{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»

1768 \def\XINT_div_II_k #1#2#3#4#5%
1769 {%
1770   \expandafter\XINT_div_II_l \the\numexpr #4-\xint_c_iv.{#3}#1{#2}#5.%
1771 }%
1772 \def\XINT_div_II_l #1.#2#3#4#5#6#7#8#9%
1773 {%
1774   \XINT_div_II_m {{#1}{#2}}{{#3}{#4}}{{#5}{#6}}{#7#8#9}#6#7#8#9%
1775 }%

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

1776 \def\XINT_div_II_m #1#2#3#4.#5#6%
1777 {%
1778   \XINT_div_I_a {#3}{#2}{#4}{#5}{#6}#1%
1779 }%

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

1780 \def\XINT_div_little_b #1#2#3#4#5#6#7%
1781 {%
1782   \XINT_div_little_c #3.{{#4}{#6}}{{#1}}%
1783 }%

```

3 Package *xintcore* implementation

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

```
1784 \def\XINT_div_little_c #1#2#3#4#5.#6#7%
1785 {%
1786   \XINT_div_littleI_a {#1#2#3#4}#6{#7}{#5}%
1787 }%
```

a, y, x, L, alpha',«c» On calcule ici (contrairement à la phase I générale) le vrai quotient euclidien 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.

```
1788 \def\XINT_div_littleI_a #1#2#3%
1789 {%
1790   \expandafter\XINT_div_littleI_b
1791   \the\numexpr (#1+#2)/#3-\xint_c_i{#1}{#2}{#3}%
1792 }%
```

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».

```
1793 \def\XINT_div_littleI_b #1%
1794 {%
1795   \xint_gob_til_zero #1\XINT_div_littleI_skip 0\XINT_div_littleI_c #1%
1796 }%
1797 \def\XINT_div_littleI_skip 0\XINT_div_littleI_c 0#1#2#3#4#5%
1798   {\XINT_div_littleII_a {#4}{#1}#5.{#2}{#3}{0000}}%
1799 \def\XINT_div_littleI_c #1#2#3#4%
1800 {%
1801   \expandafter\expandafter\expandafter\XINT_div_littleI_e
1802   \expandafter\expandafter\expandafter
1803   {\expandafter\xint_gobble_i\the\numexpr \xint_c_x^iv+#2-#1*#4}#1{#3}{#4}%
1804 }%
```

#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

```
1805 \def\XINT_div_littleI_e #1#2#3#4#5%
1806   {\XINT_div_littleII_a {#4}{#1}#5.{#3}{000#2}}%
```

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

```
1807 \def\XINT_div_littleII_a #1%
1808 {%
1809   \ifnum#1=\xint_c_iv
1810     \expandafter\XINT_div_littleIII_ab
1811   \else
1812     \expandafter\XINT_div_littleII_b
1813   \fi {#1}%
1814 }%
```

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

```
1815 \def\XINT_div_littleIII_ab #1#2#3.#4%
```

3 Package *xintcore* implementation

```

1816 {%
1817   \expandafter\XINT_div_III_b\the\numexpr #2#3.%
1818 }%

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

1819 \def\XINT_div_littleII_b #1%
1820 {%
1821   \expandafter\XINT_div_littleII_c\expandafter {\the\numexpr #1-\xint_c_iv}%
1822 }%

{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.

1823 \def\XINT_div_littleII_c #1#2#3#4#5#6#7.#8%
1824 {%
1825   \XINT_div_littleII_d {#2#3#4#5#6}#8{#1}{#7}%
1826 }%
1827 \def\XINT_div_littleII_d #1#2#3%
1828 {%
1829   \expandafter\XINT_div_littleII_e\the\numexpr (#1+#2)/#3+\xint_c_ixixixix.%
1830   {#1}{#2}{#3}%
1831 }%

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»

1832 \def\XINT_div_littleII_e 1#1.#2#3#4%
1833 {%
1834   \expandafter\expandafter\expandafter\XINT_div_littleII_f
1835   \expandafter\xint_gobble_i\the\numexpr \xint_c_x^iv+#2-#1*#4.%
1836   {#1}{#3}{#4}}%
1837 }%

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

1838 \def\XINT_div_littleII_f #1.#2#3#4#5#6%
1839 {%
1840   \XINT_div_littleII_a {#4}{#1}#5.{#3}{#6#2}%
1841 }%

```

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

3 Package *xintcore* implementation

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.

```
1842 \def\XINT_div_sub_xpxp #1#2% #1=alpha d\'ej\'a renvers\'e, #2 se d\'eveloppe en qB
1843 {%
1844   \expandafter\XINT_div_sub_xpxp_b #2\W\X\Y\Z #1\W\X\Y\Z
1845 }%
1846 \def\XINT_div_sub_xpxp_b
1847 {%
1848   \XINT_div_sub_A 1{%
1849 }%
1850 \def\XINT_div_sub_A #1#2#3#4#5#6%
1851 {%
1852   \xint_gob_til_W #3\xint_div_sub_az\W
1853   \XINT_div_sub_B #1{#3#4#5#6}{#2}%
1854 }%
1855 \def\XINT_div_sub_B #1#2#3#4\W\X\Y\Z #5#6#7#8%
1856 {%
1857   \xint_gob_til_W #5\xint_div_sub_bz\W
1858   \XINT_div_sub_onestep #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
1859 }%
1860 \def\XINT_div_sub_onestep #1#2#3#4#5#6%
1861 {%
1862   \expandafter\XINT_div_sub_backtoA
1863   \the\numexpr 11#6-#5#4#3#2+#1-\xint_c_i.%
1864 }%
1865 \def\XINT_div_sub_backtoA #1#2#3.#4%
1866 {%
1867   \XINT_div_sub_A #2{#3#4}%
1868 }%
```

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 -

```
1869 \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.

```
1870 \def\xint_div_sub_az\W\XINT_div_sub_B #1#2{\XINT_div_sub_C #1}%
1871 \def\XINT_div_sub_C #1#2#3#4#5#6%
1872 {%
1873   \xint_gob_til_W #3\xint_div_sub_cz\W
1874   \XINT_div_sub_C_onestep #1{#6#5#4#3}{#2}%
1875 }%
1876 \def\XINT_div_sub_C_onestep #1#2%
1877 {%
1878   \expandafter\XINT_div_sub_backtoC \the\numexpr 11#2+#1-\xint_c_i.%
1879 }%
1880 \def\XINT_div_sub_backtoC #1#2#3.#4%
1881 {%
```

```
1882 \XINT_div_sub_C #2{#3#4}%
1883 }%
```

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 -

```
1884 \def\xint_div_sub_cz\W\XINT_div_sub_C_onestep #1#2%
1885 {%
1886   \if#10% retenue
1887     \expandafter\xint_div_sub_neg
1888   \else\expandafter\xint_div_sub_ok
1889   \fi
1890 }%
1891 \def\xint_div_sub_neg #1{ -}%
1892 \def\xint_div_sub_ok #1{ #1}%
```

3.17 \xintiDivRound, \xintiiDivRound

v1.1, transferred from first release of bnumexpr.

```
1893 \def\xintiDivRound {\romannumeral0\xintidivround }%
1894 \def\xintidivround #1{\expandafter\XINT_iidivround\romannumeral0\xintnum{#1}\Z }%
1895 \def\xintiiDivRound {\romannumeral0\xintiidivround }%
1896 \def\xintiidivround #1{\expandafter\XINT_iidivround \romannumeral-‘0#1\Z }%
1897 \def\XINT_iidivround #1#2\Z #3{\expandafter\XINT_iidivround_a\expandafter #1%
1898   \romannumeral-‘0#3\Z #2\Z }%
1899 \def\XINT_iidivround_a #1#2# #1 de A, #2 de B.
1900 {%
1901   \if0#2\xint_dothis\XINT_iidivround_divbyzero\fi
1902   \if0#1\xint_dothis\XINT_iidivround_aiszero\fi
1903   \if-#2\xint_dothis{\XINT_iidivround_bneg #1}\fi
1904   \xint_orthat{\XINT_iidivround_bpos #1#2}%
1905 }%
1906 \def\XINT_iidivround_divbyzero #1\Z #2\Z {\xintError:DivisionByZero\space 0}%
1907 \def\XINT_iidivround_aiszero #1\Z #2\Z { 0}%
1908 \def\XINT_iidivround_bpos #1%
1909 {%
1910   \xint_UDsignfork
1911     #1{\xintiiopp\XINT_iidivround_pos {}}%
1912     -{\XINT_iidivround_pos #1}%
1913   \krof
1914 }%
1915 \def\XINT_iidivround_bneg #1%
1916 {%
1917   \xint_UDsignfork
1918     #1{\XINT_iidivround_pos {}}%
1919     -{\xintiiopp\XINT_iidivround_pos #1}%
1920   \krof
1921 }%
1922 \def\XINT_iidivround_pos #1#2\Z #3\Z{\expandafter\XINT_iidivround_pos_a
1923   \romannumeral0\XINT_div_prepare {#2}{#1#30}}%
1924 \def\XINT_iidivround_pos_a #1#2{\xintReverseOrder {#1\XINT_iidivround_pos_b}\Z }%
1925 \def\XINT_iidivround_pos_b #1#2{\xint_gob_til_Z #2\XINT_iidivround_pos_small\Z
```

```

1926 \XINT_iidivround_pos_c #1#2}%
1927 \def\XINT_iidivround_pos_c #1#2\Z {\ifnum #1>\xint_c_iv
1928 \expandafter\XINT_iidivround_pos_up
1929 \else \expandafter\xintreverseorder
1930 \fi {#2}}%
1931 \def\XINT_iidivround_pos_up #1{\xintinc {\xintReverseOrder{#1}}}%
1932 \def\XINT_iidivround_pos_small\Z\XINT_iidivround_pos_c #1#2%
1933 {\ifnum #1>\xint_c_iv\expandafter\xint_secondoftwo\else\expandafter
1934 \xint_firstoftwo\fi { 0}{ 1}}%

```

3.18 \xintiDivTrunc, \xintiiDivTrunc

```

1935 \def\xintiDivTrunc {\romannumeral0\xintidivtrunc }%
1936 \def\xintidivtrunc #1{\expandafter\XINT_iidivtrunc\romannumeral0\xintnum{#1}\Z }%
1937 \def\xintiiDivTrunc {\romannumeral0\xintiidivtrunc }%
1938 \def\xintiidivtrunc #1{\expandafter\XINT_iidivtrunc \romannumeral-‘0#1\Z }%
1939 \def\XINT_iidivtrunc #1#2\Z #3{\expandafter\XINT_iidivtrunc_a\expandafter #1%
1940 \romannumeral-‘0#3\Z #2\Z }%
1941 \def\XINT_iidivtrunc_a #1#2% #1 de A, #2 de B.
1942 {%
1943 \if0#2\xint_dothis\XINT_iidivround_divbyzero\fi
1944 \if0#1\xint_dothis\XINT_iidivround_aiszero\fi
1945 \if-#2\xint_dothis{\XINT_iidivtrunc_bneg #1}\fi
1946 \xint_orthat{\XINT_iidivtrunc_bpos #1#2}%
1947 }%
1948 \def\XINT_iidivtrunc_bpos #1%
1949 {%
1950 \xint_UDsignfork
1951 #1{\xintiiopp\XINT_iidivtrunc_pos {}}%
1952 -{\XINT_iidivtrunc_pos #1}%
1953 \krof
1954 }%
1955 \def\XINT_iidivtrunc_bneg #1%
1956 {%
1957 \xint_UDsignfork
1958 #1{\XINT_iidivtrunc_pos {}}%
1959 -{\xintiiopp\XINT_iidivtrunc_pos #1}%
1960 \krof
1961 }%
1962 \def\XINT_iidivtrunc_pos #1#2\Z #3\Z%
1963 {\expandafter\xint_firstoftwo_thenstop\romannumeral0\XINT_div_prepare {#2}{#1#3}}%

```

3.19 \xintiMod, cshxintiiMod

```

1964 \def\xintiMod {\romannumeral0\xintimod }%
1965 \def\xintimod #1{\expandafter\XINT_iimod\romannumeral0\xintnum{#1}\Z }%
1966 \def\xintiiMod {\romannumeral0\xintiimod }%
1967 \def\xintiimod #1{\expandafter\XINT_iimod \romannumeral-‘0#1\Z }%
1968 \def\XINT_iimod #1#2\Z #3{\expandafter\XINT_iimod_a\expandafter #1%
1969 \romannumeral-‘0#3\Z #2\Z }%
1970 \def\XINT_iimod_a #1#2% #1 de A, #2 de B.
1971 {%
1972 \if0#2\xint_dothis\XINT_iidivround_divbyzero\fi
1973 \if0#1\xint_dothis\XINT_iidivround_aiszero\fi

```

3 Package *xintcore* implementation

```
1974 \if-#2\xint_dothis{\XINT_iimod_bneg #1}\fi
1975 \xint_orthat{\XINT_iimod_bpos #1#2}%
1976 }%
1977 \def\xint_iimod_bpos #1%
1978 {%
1979 \xint_UDsignfork
1980 #1{\xintiiopp\xint_iimod_pos {}}%
1981 -{\XINT_iimod_pos #1}%
1982 \krof
1983 }%
1984 \def\xint_iimod_bneg #1%
1985 {%
1986 \xint_UDsignfork
1987 #1{\xintiiopp\xint_iimod_pos {}}%
1988 -{\XINT_iimod_pos #1}%
1989 \krof
1990 }%
1991 \def\xint_iimod_pos #1#2\Z #3\Z%
1992 {\expandafter\xint_secondoftwo_thenstop\romannumeral0\xint_div_prepare {#2}{#1#3}}%
```

3.20 *\xintDec*

v1.08

```
1993 \def\xintDec {\romannumeral0\xintdec }%
1994 \def\xintdec #1%
1995 {%
1996 \expandafter\XINT_dec\romannumeral-'0#1%
1997 \R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W\W
1998 }%
1999 \def\XINT_dec #1%
2000 {%
2001 \xint_UDzerominusfork
2002 #1-\XINT_dec_zero
2003 0#1\XINT_dec_neg
2004 0-{\XINT_dec_pos #1}%
2005 \krof
2006 }%
2007 \def\XINT_dec_zero #1\W\W\W\W\W\W\W { -1}%
2008 \def\XINT_dec_neg
2009 {\expandafter\xint_minus_thenstop\romannumeral0\XINT_inc_pos }%
2010 \def\XINT_dec_pos
2011 {%
2012 \expandafter\XINT_dec_a \expandafter{\expandafter}%
2013 \romannumeral0\XINT_OQ {}%
2014 }%
2015 \def\XINT_dec_a #1#2#3#4#5#6#7#8#9%
2016 {%
2017 \expandafter\XINT_dec_b
2018 \the\numexpr 11#9#8#7#6#5#4#3#2-\xint_c_i\relax {#1}%
2019 }%
2020 \def\XINT_dec_b 1#1%
2021 {%
2022 \xint_gob_til_one #1\XINT_dec_A 1\XINT_dec_c
```


3 Package *xintcore* implementation

```
2023 }%
2024 \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}}%
2025 \def\XINT_dec_A 1\XINT_dec_c #1#2#3#4#5#6#7#8#9%
2026   {\XINT_dec_B {#1#2#3#4#5#6#7#8#9}}%
2027 \def\XINT_dec_B #1#2\W\W\W\W\W\W\W\W
2028 {%
2029   \expandafter\XINT_dec_cleanup
2030   \romannumeral0\XINT_rord_main {}#2%
2031   \xint_relax
2032   \xint_bye\xint_bye\xint_bye\xint_bye
2033   \xint_bye\xint_bye\xint_bye\xint_bye
2034   \xint_relax
2035   #1%
2036 }%
2037 \edef\XINT_dec_cleanup #1#2#3#4#5#6#7#8%
2038   {\noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7#8\relax }%
```

3.21 *\xintInc*

v1.08

```
2039 \def\xintInc {\romannumeral0\xintinc }%
2040 \def\xintinc #1%
2041 {%
2042   \expandafter\XINT_inc\romannumeral-‘0#1%
2043   \R\R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W\W
2044 }%
2045 \def\XINT_inc #1%
2046 {%
2047   \xint_UDzerominusfork
2048   #1-\XINT_inc_zero
2049   0#1\XINT_inc_neg
2050   0-{\XINT_inc_pos #1}%
2051   \krof
2052 }%
2053 \def\XINT_inc_zero #1\W\W\W\W\W\W\W\W { 1}%
2054 \def\XINT_inc_neg {\expandafter\XINT_opp\romannumeral0\XINT_dec_pos }%
2055 \def\XINT_inc_pos
2056 {%
2057   \expandafter\XINT_inc_a \expandafter{\expandafter}%
2058   \romannumeral0\XINT_OQ {}%
2059 }%
2060 \def\XINT_inc_a #1#2#3#4#5#6#7#8#9%
2061 {%
2062   \xint_gob_til_W #9\XINT_inc_end\W
2063   \expandafter\XINT_inc_b
2064   \the\numexpr 10#9#8#7#6#5#4#3#2+\xint_c_i\relax {#1}%
2065 }%
2066 \def\XINT_inc_b 1#1%
2067 {%
2068   \xint_gob_til_zero #1\XINT_inc_A 0\XINT_inc_c
2069 }%
2070 \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}}%
```

```

2071 \def\XINT_inc_A 0\XINT_inc_c #1#2#3#4#5#6#7#8#9%
2072           {\XINT_dec_B {#1#2#3#4#5#6#7#8#9}}%
2073 \def\XINT_inc_end\W #1\relax #2{ 1#2}%
2074 \XINT_restorecatcodes_endinput%

```

4 Package `xint` implementation

.1	Package identification	83	.25	<code>\xintORof</code>	92
.2	More token management	83	.26	<code>\xintXORof</code>	93
.3	<code>\xintSgnFork</code>	83	.27	<code>\xintGeq</code>	93
.4	<code>\xintIsOne</code>	84	.28	<code>\xintiMax, \xintiiMax</code>	95
.5	<code>\XINT_SQ</code>	84	.29	<code>\xintMaxof</code>	96
.6	<code>\xintRev</code>	84	.30	<code>\xintiMin, \xintiiMin</code>	96
.7	<code>\xintLen</code>	85	.31	<code>\xintMinof</code>	98
.8	<code>\xintBool, \xintToggle</code>	85	.32	<code>\xintSum</code>	98
.9	<code>\xintifSgn</code>	86	.33	<code>\xintPrd</code>	99
.10	<code>\xintifZero, \xintifNotZero</code>	86	.34	<code>\xintFac</code>	100
.11	<code>\xintifOne</code>	87	.35	<code>\xintFDg</code>	102
.12	<code>\xintifTrueAelseB, \xintifFalseAelseB</code>	87	.36	<code>\xintLDg</code>	102
.13	<code>\xintifCmp</code>	87	.37	<code>\xintMON, \xintMMON</code>	102
.14	<code>\xintifEq</code>	88	.38	<code>\xintOdd</code>	103
.15	<code>\xintifGt</code>	88	.39	<code>\xintDSL</code>	104
.16	<code>\xintifLt</code>	88	.40	<code>\xintDSR</code>	104
.17	<code>\xintifOdd</code>	88	.41	<code>\xintDSH, \xintDSHr</code>	105
.18	<code>\xintCmp</code>	89	.42	<code>\xintDSx</code>	106
.19	<code>\xintEq, \xintGt, \xintLt</code>	91	.43	<code>\xintDecSplit, \xintDecSplitL, \xint-</code> <code>DecSplitR</code>	108
.20	<code>\xintNEq, \xintGtorEq, \xintLtorEq</code>	91	.44	<code>\xintDouble</code>	112
.21	<code>\xintIsZero, \xintIsNotZero</code>	91	.45	<code>\xintHalf</code>	113
.22	<code>\xintIsTrue, \xintNot, \xintIsFalse</code>	92	.46	<code>\xintiiSqrt, \xintiiSquareRoot</code>	114
.23	<code>\xintAND, \xintOR, \xintXOR</code>	92	.47	<code>\xintiiE</code>	117
.24	<code>\xintANDof</code>	92			

The basic arithmetic routines `\xintiiAdd`, `\xintiiSub`, `\xintiiMul`, `\xintiiQuo` and `\xintiiPow` have been moved to new package `xintcore`.

```

1 \beginingroup\catcode61\catcode48\catcode32=10\relax%
2   \catcode13=5    % ^^M
3   \endlinechar=13 %
4   \catcode123=1   % {
5   \catcode125=2   % }
6   \catcode64=11   % @
7   \catcode35=6    % #
8   \catcode44=12   % ,
9   \catcode45=12   % -
10  \catcode46=12   % .
11  \catcode58=12   % :
12  \let\z\endgroup
13  \expandafter\let\expandafter\x\csname ver@xint.sty\endcsname
14  \expandafter\let\expandafter\w\csname ver@xintcore.sty\endcsname
15  \expandafter
16    \ifx\csname PackageInfo\endcsname\relax
17    \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
18    \else

```

```

19   \def\y#1#2{\PackageInfo{#1}{#2}}%
20   \fi
21   \expandafter
22   \ifx\csname numexpr\endcsname\relax
23     \y{xint}{\numexpr not available, aborting input}%
24     \aftergroup\endinput
25   \else
26     \ifx\x\relax % plain-TeX, first loading of xintcore.sty
27       \ifx\w\relax % but xintkernel.sty not yet loaded.
28         \def\z{\endgroup\input xintcore.sty\relax}%
29       \fi
30     \else
31       \def\empty {}%
32       \ifx\x\empty % LaTeX, first loading,
33       % variable is initialized, but \ProvidesPackage not yet seen
34       \ifx\w\relax % xintcore.sty not yet loaded.
35         \def\z{\endgroup\RequirePackage{xintcore}}%
36       \fi
37     \else
38       \aftergroup\endinput % xint already loaded.
39     \fi
40   \fi
41 \fi
42 \z%
43 \XINTsetupcatcodes% defined in xintkernel.sty (loaded by xintcore.sty)

```

4.1 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xint}%
46 [2014/10/28 v1.1 Expandable operations on big integers (jfB)]%

```

4.2 More token management

```

47 \long\def\xint_firstofthree #1#2#3{#1}%
48 \long\def\xint_secondofthree #1#2#3{#2}%
49 \long\def\xint_thirdofthree #1#2#3{#3}%
50 \long\def\xint_firstofthree_thenstop #1#2#3{ #1}% 1.09i
51 \long\def\xint_secondofthree_thenstop #1#2#3{ #2}%
52 \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 `-1`, `0` or `1`. 1.09i has `_afterstop`, renamed `_thenstop` later, for efficiency.

```

53 \def\xintSgnFork {\romannumeral0\xintsngfork }%
54 \def\xintsngfork #1%
55 {%
56   \ifcase #1 \expandafter\xint_secondofthree_thenstop
57             \or\expandafter\xint_thirdofthree_thenstop
58             \else\expandafter\xint_firstofthree_thenstop
59   \fi
60 }%

```

4.4 `\xintIsOne`

Added in 1.03. Attention: `\XINT_isOne` does not do any expansion. Release 1.09a defines `\xintIsOne` which is more user-friendly. Will be modified if `xintfrac` is loaded.

```

61 \def\xintIsOne    {\romannumeral0\xintisone }%
62 \def\xintisone    #1{\expandafter\XINT_isone\romannumeral0\xintnum{#1}\W\Z }%
63 \def\XINT_isOne  #1{\romannumeral0\XINT_isone #1\W\Z }%
64 \def\XINT_isone  #1#2%
65 {%
66     \xint_gob_til_one #1\XINT_isone_b 1%
67     \expandafter\space\expandafter 0\xint_gob_til_Z #2%
68 }%
69 \def\XINT_isone_b #1\xint_gob_til_Z #2%
70 {%
71     \xint_gob_til_W #2\XINT_isone_yes \W
72     \expandafter\space\expandafter 0\xint_gob_til_Z
73 }%
74 \def\XINT_isone_yes #1\Z { 1}%

```

4.5 `\XINT_SQ`

```

75 \def\XINT_SQ  #1#2#3#4#5#6#7#8%
76 {%
77     \xint_gob_til_R #8\XINT_SQ_end_a\R\XINT_SQ {#8#7#6#5#4#3#2#1}%
78 }%
79 \def\XINT_SQ_end_a\R\XINT_SQ  #1#2\Z
80 {%
81     \XINT_SQ_end_b #1\Z
82 }%
83 \def\XINT_SQ_end_b  #1#2#3#4#5#6#7%
84 {%
85     \xint_gob_til_R
86         #7\XINT_SQ_end_vii
87         #6\XINT_SQ_end_vi
88         #5\XINT_SQ_end_v
89         #4\XINT_SQ_end_iv
90         #3\XINT_SQ_end_iii
91         #2\XINT_SQ_end_ii
92         \R\XINT_SQ_end_i
93         \Z #2#3#4#5#6#7%
94 }%
95 \def\XINT_SQ_end_vii  #1\Z #2#3#4#5#6#7#8\Z { #8}%
96 \def\XINT_SQ_end_vi   #1\Z #2#3#4#5#6#7#8\Z { #7#8000000}%
97 \def\XINT_SQ_end_v    #1\Z #2#3#4#5#6#7#8\Z { #6#7#800000}%
98 \def\XINT_SQ_end_iv   #1\Z #2#3#4#5#6#7#8\Z { #5#6#7#80000}%
99 \def\XINT_SQ_end_iii  #1\Z #2#3#4#5#6#7#8\Z { #4#5#6#7#8000}%
100 \def\XINT_SQ_end_ii   #1\Z #2#3#4#5#6#7#8\Z { #3#4#5#6#7#800}%
101 \def\XINT_SQ_end_i     \Z #1#2#3#4#5#6#7\Z { #1#2#3#4#5#6#70}%

```

4.6 `\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.

```

102 \def\xintRev {\romannumeral0\xintrev }%
103 \def\xintrev #1%
104 {%
105   \expandafter\XINT_rev_fork
106   \romannumeral-‘0#1\xint_relax % empty #1 ok, \xint_relax stops expansion
107     \xint_bye\xint_bye\xint_bye\xint_bye
108     \xint_bye\xint_bye\xint_bye\xint_bye
109   \xint_relax
110 }%
111 \def\XINT_rev_fork #1%
112 {%
113   \xint_UDsignfork
114   #1{\expandafter\xint_minus_thenstop\romannumeral0\XINT_rord_main {}}%
115   -{\XINT_rord_main {}}#1}%
116   \krof
117 }%

```

4.7 `\xintLen`

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

```

118 \def\xintLen {\romannumeral0\xintlen }%
119 \def\xintlen #1%
120 {%
121   \expandafter\XINT_len_fork
122   \romannumeral0\xintnum{#1}\xint_relax\xint_relax\xint_relax\xint_relax
123     \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
124 }%
125 \def\XINT_Len #1% variant which does not expand via \xintnum.
126 {%
127   \romannumeral0\XINT_len_fork
128   #1\xint_relax\xint_relax\xint_relax\xint_relax
129     \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
130 }%
131 \def\XINT_len_fork #1%
132 {%
133   \expandafter\XINT_length_loop
134   \xint_UDsignfork
135   #1{0.}%
136   -{0.#1}%
137   \krof
138 }%

```

4.8 `\xintBool`, `\xintToggle`

1.09c

```

139 \def\xintBool #1{\romannumeral-‘0%
140   \csname if#1\endcsname\expandafter1\else\expandafter0\fi }%
141 \def\xintToggle #1{\romannumeral-‘0\iftoggle{#1}{1}{0}}%

```

4.9 `\xintifSgn`

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`)

```

142 \def\xintifSgn {\romannumeral0\xintifsgn }%
143 \def\xintifsgn #1%
144 {%
145     \ifcase \xintSgn{#1}
146         \expandafter\xint_secondofthree_thenstop
147     \or\expandafter\xint_thirdofthree_thenstop
148     \else\expandafter\xint_firstofthree_thenstop
149     \fi
150 }%
151 \def\xintiiifSgn {\romannumeral0\xintiiifsgn }%
152 \def\xintiiifsgn #1%
153 {%
154     \ifcase \xintiiSgn{#1}
155         \expandafter\xint_secondofthree_thenstop
156     \or\expandafter\xint_thirdofthree_thenstop
157     \else\expandafter\xint_firstofthree_thenstop
158     \fi
159 }%
```

4.10 `\xintifZero`, `\xintifNotZero`

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.

```

160 \def\xintifZero {\romannumeral0\xintifzero }%
161 \def\xintifzero #1%
162 {%
163     \if0\xintSgn{#1}%
164     \expandafter\xint_firstoftwo_thenstop
165     \else
166     \expandafter\xint_secondoftwo_thenstop
167     \fi
168 }%
169 \def\xintifNotZero {\romannumeral0\xintifnotzero }%
170 \def\xintifnotzero #1%
171 {%
172     \if0\xintSgn{#1}%
173     \expandafter\xint_secondoftwo_thenstop
174     \else
175     \expandafter\xint_firstoftwo_thenstop
176     \fi
177 }%
178 \def\xintiiifZero {\romannumeral0\xintiiifzero }%
179 \def\xintiiifzero #1%
```

```

180 {%
181   \if0\xintiiSgn{#1}%
182     \expandafter\xint_firstoftwo_thenstop
183   \else
184     \expandafter\xint_secondoftwo_thenstop
185   \fi
186 }%
187 \def\xintiiifNotZero {\romannumeral0\xintiiifnotzero }%
188 \def\xintiiifnotzero #1%
189 {%
190   \if0\xintiiSgn{#1}%
191     \expandafter\xint_secondoftwo_thenstop
192   \else
193     \expandafter\xint_firstoftwo_thenstop
194   \fi
195 }%

```

4.11 \xintifOne

added in 1.09i.

```

196 \def\xintifOne {\romannumeral0\xintifone }%
197 \def\xintifone #1%
198 {%
199   \if1\xintIsOne{#1}%
200     \expandafter\xint_firstoftwo_thenstop
201   \else
202     \expandafter\xint_secondoftwo_thenstop
203   \fi
204 }%

```

4.12 \xintifTrueAelseB, \xintifFalseAelseB

1.09i. Warning, \xintifTrueFalse, \xintifTrue deprecated, to be removed

```

205 \let\xintifTrueAelseB\xintifNotZero
206 \let\xintifFalseAelseB\xintifZero
207 \let\xintifTrue\xintifNotZero
208 \let\xintifTrueFalse\xintifNotZero

```

4.13 \xintifCmp

1.09e \xintifCmp {n}{m}{if n<m}{if n=m}{if n>m}.

```

209 \def\xintifCmp {\romannumeral0\xintifcmp }%
210 \def\xintifcmp #1#2%
211 {%
212   \ifcase\xintCmp {#1}{#2}
213     \expandafter\xint_secondofthree_thenstop
214   \or\expandafter\xint_thirdofthree_thenstop
215   \else\expandafter\xint_firstofthree_thenstop
216   \fi
217 }%

```

4.14 `\xintifEq`

1.09a `\xintifEq {n}{m}{YES if n=m}{NO if n<>m}`.

```

218 \def\xintifEq {\romannumeral0\xintifeq }%
219 \def\xintifeq #1#2%
220 {%
221   \if0\xintCmp{#1}{#2}%
222     \expandafter\xint_firstoftwo_thenstop
223   \else\expandafter\xint_secondoftwo_thenstop
224   \fi
225 }%
```

4.15 `\xintifGt`

1.09a `\xintifGt {n}{m}{YES if n>m}{NO if n<=m}`.

```

226 \def\xintifGt {\romannumeral0\xintifgt }%
227 \def\xintifgt #1#2%
228 {%
229   \if1\xintCmp{#1}{#2}%
230     \expandafter\xint_firstoftwo_thenstop
231   \else\expandafter\xint_secondoftwo_thenstop
232   \fi
233 }%
```

4.16 `\xintifLt`

1.09a `\xintifLt {n}{m}{YES if n<m}{NO if n>=m}`. Restyled in 1.09i

```

234 \def\xintifLt {\romannumeral0\xintiflt }%
235 \def\xintiflt #1#2%
236 {%
237   \ifnum\xintCmp{#1}{#2}<\xint_c_
238     \expandafter\xint_firstoftwo_thenstop
239   \else \expandafter\xint_secondoftwo_thenstop
240   \fi
241 }%
```

4.17 `\xintifOdd`

1.09e. Restyled in 1.09i.

```

242 \def\xintifOdd {\romannumeral0\xintifodd }%
243 \def\xintifodd #1%
244 {%
245   \if\xintOdd{#1}1%
246     \expandafter\xint_firstoftwo_thenstop
247   \else
248     \expandafter\xint_secondoftwo_thenstop
249   \fi
250 }%
```


4.18 `\xintCmp`

Release 1.09a has `\xintnum` inserted into `\xintCmp`. Unnecessary `\xintiCmp` suppressed in 1.09f.

```

251 \def\xintCmp {\romannumeral0\xintcmp }%
252 \def\xintcmp #1%
253 {%
254   \expandafter\xint_cmp\expandafter{\romannumeral0\xintnum{#1}}%
255 }%
256 \def\xint_cmp #1#2%
257 {%
258   \expandafter\XINT_cmp_fork \romannumeral0\xintnum{#2}\Z #1\Z
259 }%
260 \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*

```

261 \def\XINT_cmp_fork #1#2\Z #3#4\Z
262 {%
263   \xint_UDsignsfork
264     #1#3\XINT_cmp_minusminus
265     #1-\XINT_cmp_minusplus
266     #3-\XINT_cmp_plusminus
267     --{\xint_UDzerosfork
268       #1#3\XINT_cmp_zerozero
269       #10\XINT_cmp_zeroplus
270       #30\XINT_cmp_pluszero
271       00\XINT_cmp_plusplus
272     \krof }%
273   \krof
274   {#2}{#4}#1#3%
275 }%
276 \def\XINT_cmp_minusplus #1#2#3#4{ 1}%
277 \def\XINT_cmp_plusminus #1#2#3#4{ -1}%
278 \def\XINT_cmp_zerozero #1#2#3#4{ 0}%
279 \def\XINT_cmp_zeroplus #1#2#3#4{ 1}%
280 \def\XINT_cmp_pluszero #1#2#3#4{ -1}%
281 \def\XINT_cmp_plusplus #1#2#3#4%
282 {%
283   \XINT_cmp_pre {#4#2}{#3#1}%
284 }%
285 \def\XINT_cmp_minusminus #1#2#3#4%
286 {%
287   \XINT_cmp_pre {#1}{#2}%
288 }%
289 \def\XINT_cmp_pre #1%
290 {%
291   \expandafter\XINT_cmp_pre_b\expandafter
292   {\romannumeral0\XINT_RQ }{#1\R\R\R\R\R\R\R\R\Z }%
293 }%
294 \def\XINT_cmp_pre_b #1#2%
295 {%

```

4 Package *xint* implementation

```

296 \expandafter\XINT_cmp_A
297 \expandafter1\expandafter{\expandafter}%
298 \romannumeral0\XINT_RQ {}#2\R\R\R\R\R\R\R\R\Z
299 \W\X\Y\Z #1\W\X\Y\Z
300 }%

```

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

```

301 \def\XINT_cmp_A #1#2#3\W\X\Y\Z #4#5#6#7%
302 {%
303 \xint_gob_til_W #4\xint_cmp_az\W
304 \XINT_cmp_B #1{#4#5#6#7}{#2}{#3\W\X\Y\Z
305 }%
306 \def\XINT_cmp_B #1#2#3#4#5#6#7%
307 {%
308 \xint_gob_til_W#4\xint_cmp_bz\W
309 \XINT_cmp_onestep #1#2{#7#6#5#4}{#3}%
310 }%
311 \def\XINT_cmp_onestep #1#2#3#4#5#6%
312 {%
313 \expandafter\XINT_cmp_backtoA\the\numexpr 11#5#4#3#2-#6+#1-\xint_c_i.%
314 }%
315 \def\XINT_cmp_backtoA #1#2#3.#4%
316 {%
317 \XINT_cmp_A #2{#3#4}%
318 }%
319 \def\xint_cmp_bz\W\XINT_cmp_onestep #1\Z { 1}%
320 \def\xint_cmp_az\W\XINT_cmp_B #1#2#3#4#5#6#7%
321 {%
322 \xint_gob_til_W #4\xint_cmp_ez\W
323 \XINT_cmp_Eenter #1{#3}{#4#5#6#7}%
324 }%
325 \def\XINT_cmp_Eenter #1\Z { -1}%
326 \def\xint_cmp_ez\W\XINT_cmp_Eenter #1%
327 {%
328 \xint_UDzerofork
329 #1\XINT_cmp_K % il y a une retenue
330 0\XINT_cmp_L % pas de retenue
331 \krof
332 }%
333 \def\XINT_cmp_K #1\Z { -1}%
334 \def\XINT_cmp_L #1{\XINT_OneIfPositive_main #1}%
335 \def\XINT_OneIfPositive #1%
336 {%
337 \XINT_OneIfPositive_main #1\W\X\Y\Z%
338 }%
339 \def\XINT_OneIfPositive_main #1#2#3#4%
340 {%
341 \xint_gob_til_Z #4\xint_OneIfPositive_terminated\Z

```

4 Package `xint` implementation

```
342 \XINT_OneIfPositive_onestep #1#2#3#4%
343 }%
344 \def\xint_OneIfPositive_terminated\Z\XINT_OneIfPositive_onestep\W\X\Y\Z { 0}%
345 \def\XINT_OneIfPositive_onestep #1#2#3#4%
346 {%
347 \expandafter\XINT_OneIfPositive_check\the\numexpr #1#2#3#4\relax
348 }%
349 \def\XINT_OneIfPositive_check #1%
350 {%
351 \xint_gob_til_zero #1\xint_OneIfPositive_backtomain 0%
352 \XINT_OneIfPositive_finish #1%
353 }%
354 \def\XINT_OneIfPositive_finish #1\W\X\Y\Z{ 1}%
355 \def\xint_OneIfPositive_backtomain 0\XINT_OneIfPositive_finish 0%
356 {\XINT_OneIfPositive_main }%
```

4.19 `\xintEq`, `\xintGt`, `\xintLt`

1.09a.

```
357 \def\xintEq {\romannumeral0\xinteq }%
358 \def\xinteq #1#2{\xintifeq{#1}{#2}{1}{0}}%
359 \def\xintGt {\romannumeral0\xintgt }%
360 \def\xintgt #1#2{\xintifgt{#1}{#2}{1}{0}}%
361 \def\xintLt {\romannumeral0\xintlts }%
362 \def\xintlts #1#2{\xintiflt{#1}{#2}{1}{0}}%
```

4.20 `\xintNEq`, `\xintGtorEq`, `\xintLtorEq`

1.1. Pour `xintexpr`

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

4.21 `\xintIsZero`, `\xintIsNotZero`

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

```
366 \def\xintIsZero {\romannumeral0\xintiszero }%
367 \def\xintiszero #1{\if0\xintSgn{#1}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%
368 \def\xintIsNotZero {\romannumeral0\xintisnotzero }%
369 \def\xintisnotzero
370 #1{\if0\xintSgn{#1}\xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi}%
371 \def\xintiiIsZero {\romannumeral0\xintiiiszero }%
372 \def\xintiiiszero #1{\if0\xintiiSgn{#1}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%
373 \def\xintiiIsNotZero {\romannumeral0\xintiiisnotzero }%
374 \def\xintiiisnotzero
375 #1{\if0\xintiiSgn{#1}\xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi}%

```

4.22 `\xintIsTrue`, `\xintNot`, `\xintIsFalse`

1.09c

```
376 \let\xintIsTrue\xintIsNotZero
377 \let\xintNot\xintIsZero
378 \let\xintIsFalse\xintIsZero
```

4.23 `\xintAND`, `\xintOR`, `\xintXOR`

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

```
379 \def\xintAND {\romannumeral0\xintand }%
380 \def\xintand #1#2{\if0\xintSgn{#1}\expandafter\xint_firstoftwo
381               \else\expandafter\xint_secondoftwo\fi
382               { 0}{\xintisnotzero{#2}}}%
383 \def\xintOR {\romannumeral0\xintor }%
384 \def\xintor #1#2{\if0\xintSgn{#1}\expandafter\xint_firstoftwo
385               \else\expandafter\xint_secondoftwo\fi
386               {\xintisnotzero{#2}}{ 1}}%
387 \def\xintXOR {\romannumeral0\xintxor }%
388 \def\xintxor #1#2{\if\xintIsZero{#1}\xintIsZero{#2}%
389               \xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi }%
```

4.24 `\xintANDof`

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

```
390 \def\xintANDof {\romannumeral0\xintandof }%
391 \def\xintandof #1{\expandafter\XINT_andof_a\romannumeral-‘0#1\relax }%
392 \def\XINT_andof_a #1{\expandafter\XINT_andof_b\romannumeral-‘0#1\Z }%
393 \def\XINT_andof_b #1%
394       {\xint_gob_til_relax #1\XINT_andof_e\relax\XINT_andof_c #1}%
395 \def\XINT_andof_c #1\Z
396       {\xintifTrueAelseB {#1}{\XINT_andof_a}{\XINT_andof_no}}%
397 \def\XINT_andof_no #1\relax { 0}%
398 \def\XINT_andof_e #1\Z { 1}%
```

4.25 `\xintORof`

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

```
399 \def\xintORof {\romannumeral0\xintorof }%
400 \def\xintorof #1{\expandafter\XINT_orof_a\romannumeral-‘0#1\relax }%
401 \def\XINT_orof_a #1{\expandafter\XINT_orof_b\romannumeral-‘0#1\Z }%
402 \def\XINT_orof_b #1%
403       {\xint_gob_til_relax #1\XINT_orof_e\relax\XINT_orof_c #1}%
404 \def\XINT_orof_c #1\Z
405       {\xintifTrueAelseB {#1}{\XINT_orof_yes}{\XINT_orof_a}}%
406 \def\XINT_orof_yes #1\relax { 1}%
407 \def\XINT_orof_e #1\Z { 0}%
```

4.26 `\xintXORof`

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

```

408 \def\xintXORof      {\romannumeral0\xintxorof }%
409 \def\xintxorof      #1{\expandafter\XINT_xorof_a\expandafter
410                        0\romannumeral-‘0#1\relax }%
411 \def\XINT_xorof_a #1#2{\expandafter\XINT_xorof_b\romannumeral-‘0#2\Z #1}%
412 \def\XINT_xorof_b #1%
413                        {\xint_gob_til_relax #1\XINT_xorof_e\relax\XINT_xorof_c #1}%
414 \def\XINT_xorof_c #1\Z #2%
415                        {\xintifTrueAelseB {#1}{\if #20\xint_afterfi{\XINT_xorof_a 1}%
416                        \else\xint_afterfi{\XINT_xorof_a 0}\fi}%
417                        {\XINT_xorof_a #2}%
418                        }%
419 \def\XINT_xorof_e #1\Z #2{ #2}%

```

4.27 `\xintGeq`

Release 1.09a has `\xintnum` added into `\xintGeq`. Unused and useless `\xintiGeq` removed in 1.09e.
PLUS GRAND OU ÉGAL attention compare les **valeurs absolues**

```

420 \def\xintGeq {\romannumeral0\xintgeq }%
421 \def\xintgeq #1%
422 {%
423   \expandafter\xint_geq\expandafter {\romannumeral0\xintnum{#1}}%
424 }%
425 \def\xint_geq #1#2%
426 {%
427   \expandafter\XINT_geq_fork \romannumeral0\xintnum{#2}\Z #1\Z
428 }%
429 \def\XINT_Geq #1#2{\romannumeral0\XINT_geq_fork #2\Z #1\Z }%

```

PLUS GRAND OU ÉGAL ATTENTION, TESTE les VALEURS ABSOLUES

```

430 \def\XINT_geq_fork #1#2\Z #3#4\Z
431 {%
432   \xint_UDzerofork
433   #1\XINT_geq_secondiszero % |#1#2|=0
434   #3\XINT_geq_firstiszero % |#1#2|>0
435   0{\xint_UDsignsfork
436       #1#3\XINT_geq_minusminus
437       #1-\XINT_geq_minusplus
438       #3-\XINT_geq_plusminus
439       --\XINT_geq_plusplus
440       \krof }%
441   \krof
442   {#2}{#4}#1#3%
443 }%
444 \def\XINT_geq_secondiszero #1#2#3#4{ 1}%
445 \def\XINT_geq_firstiszero #1#2#3#4{ 0}%
446 \def\XINT_geq_plusplus #1#2#3#4{\XINT_geq_pre {#4#2}{#3#1}}%
447 \def\XINT_geq_minusminus #1#2#3#4{\XINT_geq_pre {#2}{#1}}%

```

4 Package *xint* implementation

```

448 \def\XINT_geq_minusplus #1#2#3#4{\XINT_geq_pre {#4#2}{#1}}%
449 \def\XINT_geq_plusminus #1#2#3#4{\XINT_geq_pre {#2}{#3#1}}%
450 \def\XINT_geq_pre #1%
451 {%
452   \expandafter\XINT_geq_pre_b\expandafter
453   {\romannumeral0\XINT_RQ {}}#1\R\R\R\R\R\R\R\R\Z}%
454}%
455 \def\XINT_geq_pre_b #1#2%
456 {%
457   \expandafter\XINT_geq_A
458   \expandafter1\expandafter{\expandafter}%
459   \romannumeral0\XINT_RQ {}}#2\R\R\R\R\R\R\R\R\Z
460   \W\X\Y\Z #1 \W\X\Y\Z
461}%

```

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 $N1 > N2$

```

462 \def\XINT_geq_A #1#2#3\W\X\Y\Z #4#5#6#7%
463 {%
464   \xint_gob_til_W #4\xint_geq_az\W
465   \XINT_geq_B #1{#4#5#6#7}{#2}{#3\W\X\Y\Z
466}%
467 \def\XINT_geq_B #1#2#3#4#5#6#7%
468 {%
469   \xint_gob_til_W #4\xint_geq_bz\W
470   \XINT_geq_onestep #1#2{#7#6#5#4}{#3}%
471}%
472 \def\XINT_geq_onestep #1#2#3#4#5#6%
473 {%
474   \expandafter\XINT_geq_backtoA\the\numexpr 11#5#4#3#2-#6+#1-\xint_c_i.%
475}%
476 \def\XINT_geq_backtoA #1#2#3.#4%
477 {%
478   \XINT_geq_A #2{#3#4}%
479}%
480 \def\xint_geq_bz\W\XINT_geq_onestep #1\W\X\Y\Z { 1}%
481 \def\xint_geq_az\W\XINT_geq_B #1#2#3#4#5#6#7%
482 {%
483   \xint_gob_til_W #4\xint_geq_ez\W
484   \XINT_geq_Eenter #1%
485}%
486 \def\XINT_geq_Eenter #1\W\X\Y\Z { 0}%
487 \def\xint_geq_ez\W\XINT_geq_Eenter #1%
488 {%
489   \xint_UDzerofork
490   #1{ 0} % il y a une retenue
491   0{ 1} % pas de retenue
492   \krof

```

493 }%

4.28 `\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.

```

494 \def\xintiMax {\romannumeral0\xintimax }%
495 \def\xintimax #1%
496 {%
497   \expandafter\xint_max\expandafter {\romannumeral0\xintnum{#1}}%
498 }%
499 \def\xint_max #1#2%
500 {%
501   \expandafter\XINT_max_pre\expandafter {\romannumeral0\xintnum{#2}}{#1}%
502 }%
503 \def\xintiiMax {\romannumeral0\xintiimax }%
504 \def\xintiimax #1%
505 {%
506   \expandafter\xint_iimax\expandafter {\romannumeral-‘0#1}%
507 }%
508 \def\xint_iimax #1#2%
509 {%
510   \expandafter\XINT_max_pre\expandafter {\romannumeral-‘0#2}{#1}%
511 }%
512 \let\xintMax\xintiMax \let\xintmax\xintimax % deprecated, should be only with xintfrac
513 \def\XINT_max_pre #1#2{\XINT_max_fork #1\Z #2\Z {#2}{#1}}%
514 \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**

```

515 \def\XINT_max_fork #1#2\Z #3#4\Z
516 {%
517   \xint_UDsignsfork
518     #1#3\XINT_max_minusminus % A < 0, B < 0
519     #1-\XINT_max_minusplus % B < 0, A >= 0
520     #3-\XINT_max_plusminus % A < 0, B >= 0
521     --{\xint_UDzerosfork
522       #1#3\XINT_max_zerozero % A = B = 0
523       #10\XINT_max_zeropplus % B = 0, A > 0
524       #30\XINT_max_pluszero % A = 0, B > 0
525       00\XINT_max_plusplus % A, B > 0
526     \krof }%
527   \krof
528   {#2}{#4}#1#3%
529 }%

```

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

```

530 \def\XINT_max_zerozero #1#2#3#4{\xint_firstoftwo_thenstop }%
531 \def\XINT_max_zeropplus #1#2#3#4{\xint_firstoftwo_thenstop }%

```

4 Package `xint` implementation

```
532 \def\XINT_max_pluszero #1#2#3#4{\xint_secondoftwo_thenstop }%
533 \def\XINT_max_minusplus #1#2#3#4{\xint_firstoftwo_thenstop }%
534 \def\XINT_max_plusminus #1#2#3#4{\xint_secondoftwo_thenstop }%
535 \def\XINT_max_plusplus #1#2#3#4%
536 {%
537   \ifodd\XINT_Geq {#4#2}{#3#1}
538   \expandafter\xint_firstoftwo_thenstop
539   \else
540   \expandafter\xint_secondoftwo_thenstop
541   \fi
542 }%
```

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

```
543 \def\XINT_max_minusminus #1#2#3#4%
544 {%
545   \ifodd\XINT_Geq {#1}{#2}
546   \expandafter\xint_firstoftwo_thenstop
547   \else
548   \expandafter\xint_secondoftwo_thenstop
549   \fi
550 }%
```

4.29 `\xintMaxof`

New with 1.09a.

```
551 \def\xintiMaxof {\romannumeral0\xintimaxof }%
552 \def\xintimaxof #1{\expandafter\XINT_imaxof_a\romannumeral-‘0#1\relax }%
553 \def\XINT_imaxof_a #1{\expandafter\XINT_imaxof_b\romannumeral0\xintnum{#1}\Z }%
554 \def\XINT_imaxof_b #1\Z #2%
555   {\expandafter\XINT_imaxof_c\romannumeral-‘0#2\Z {#1}\Z}%
556 \def\XINT_imaxof_c #1%
557   {\xint_gob_til_relax #1\XINT_imaxof_e\relax\XINT_imaxof_d #1}%
558 \def\XINT_imaxof_d #1\Z
559   {\expandafter\XINT_imaxof_b\romannumeral0\xintimax {#1}}%
560 \def\XINT_imaxof_e #1\Z #2\Z { #2}%
561 \let\xintMaxof\xintiMaxof \let\xintmaxof\xintimaxof
```

4.30 `\xintiMin`, `\xintiiMin`

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

```
562 \def\xintiMin {\romannumeral0\xintimin }%
563 \def\xintimin #1%
564 {%
565   \expandafter\xint_min\expandafter {\romannumeral0\xintnum{#1}}%
566 }%
567 \def\xint_min #1#2%
568 {%
569   \expandafter\XINT_min_pre\expandafter {\romannumeral0\xintnum{#2}}{#1}%
570 }%
```


4 Package `xint` implementation

```

571 \def\xintiMin {\romannumeral0\xintiimin }%
572 \def\xintiimin #1%
573 {%
574   \expandafter\xint_iimin\expandafter {\romannumeral-‘0#1}%
575 }%
576 \def\xint_iimin #1#2%
577 {%
578   \expandafter\XINT_min_pre\expandafter {\romannumeral-‘0#2}{#1}%
579 }%
580 \let\xintMin\xintiMin \let\xintmin\xintimin % deprecated
581 \def\XINT_min_pre #1#2{\XINT_min_fork #1\Z #2\Z {#2}{#1}}%
582 \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*

583 \def\XINT_min_fork #1#2\Z #3#4\Z
584 {%
585   \xint_UDsignsfork
586     #1#3\XINT_min_minusminus % A < 0, B < 0
587     #1-\XINT_min_minusplus % B < 0, A >= 0
588     #3-\XINT_min_plusminus % A < 0, B >= 0
589     --{\xint_UDzerosfork
590       #1#3\XINT_min_zerozero % A = B = 0
591       #10\XINT_min_zeroplus % B = 0, A > 0
592       #30\XINT_min_pluszero % A = 0, B > 0
593       00\XINT_min_plusplus % A, B > 0
594     \krof }%
595   \krof
596   {#2}{#4}#1#3%
597 }%

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

598 \def\XINT_min_zerozero #1#2#3#4{\xint_firstoftwo_thenstop }%
599 \def\XINT_min_zeroplus #1#2#3#4{\xint_secondoftwo_thenstop }%
600 \def\XINT_min_pluszero #1#2#3#4{\xint_firstoftwo_thenstop }%
601 \def\XINT_min_minusplus #1#2#3#4{\xint_secondoftwo_thenstop }%
602 \def\XINT_min_plusminus #1#2#3#4{\xint_firstoftwo_thenstop }%
603 \def\XINT_min_plusplus #1#2#3#4%
604 {%
605   \ifodd\XINT_Geq {#4#2}{#3#1}
606   \expandafter\xint_secondoftwo_thenstop
607   \else
608   \expandafter\xint_firstoftwo_thenstop
609   \fi
610 }%

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

611 \def\XINT_min_minusminus #1#2#3#4%
612 {%
613   \ifodd\XINT_Geq {#1}{#2}
614   \expandafter\xint_secondoftwo_thenstop
615   \else

```

```

616     \expandafter\xint_firstoftwo_thenstop
617   \fi
618 }%

```

4.31 *\xintMinof*

1.09a

```

619 \def\xintiMinof      {\romannumeral0\xintiminof }%
620 \def\xintiminof      #1{\expandafter\XINT_iminof_a\romannumeral-‘0#1\relax }%
621 \def\XINT_iminof_a  #1{\expandafter\XINT_iminof_b\romannumeral0\xintnum{#1}\Z }%
622 \def\XINT_iminof_b  #1\Z #2%
623     {\expandafter\XINT_iminof_c\romannumeral-‘0#2\Z {#1}\Z}%
624 \def\XINT_iminof_c  #1%
625     {\xint_gob_til_relax #1\XINT_iminof_e\relax\XINT_iminof_d #1}%
626 \def\XINT_iminof_d  #1\Z
627     {\expandafter\XINT_iminof_b\romannumeral0\xintimin {#1}}%
628 \def\XINT_iminof_e  #1\Z #2\Z { #2}%
629 \let\xintMinof\xintiMinof \let\xintminof\xintiminof

```

4.32 *\xintSum*

```

\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.

```

630 \def\xintiiSum {\romannumeral0\xintiisum }%
631 \def\xintiisum #1{\xintiisumexpr #1\relax }%
632 \def\xintiiSumExpr {\romannumeral0\xintiisumexpr }%
633 \def\xintiisumexpr {\expandafter\XINT_sumexpr\romannumeral-‘0}%
634 \let\xintSum\xintiiSum \let\xintsum\xintiisum
635 \let\xintSumExpr\xintiiSumExpr \let\xintsumexpr\xintiisumexpr
636 \def\XINT_sumexpr {\XINT_sum_loop {0000}{0000}}%
637 \def\XINT_sum_loop #1#2#3%
638 {%
639     \expandafter\XINT_sum_checksign\romannumeral-‘0#3\Z {#1}{#2}%
640 }%
641 \def\XINT_sum_checksign #1%
642 {%
643     \xint_gob_til_relax #1\XINT_sum_finished\relax
644     \xint_gob_til_zero #1\XINT_sum_skipzeroinput0%
645     \xint_UDsignfork
646     #1\XINT_sum_N
647     -{\XINT_sum_P #1}%
648     \krof
649 }%
650 \def\XINT_sum_finished #1\Z #2#3%
651 {%

```

```

652 \XINT_sub_A 1{ }#3\W\X\Y\Z #2\W\X\Y\Z
653 }%
654 \def\XINT_sum_skipzeroinput #1\krof #2\Z {\XINT_sum_loop }%
655 \def\XINT_sum_P #1\Z #2%
656 {%
657 \expandafter\XINT_sum_loop\expandafter
658 {\romannumeral0\expandafter
659 \XINT_addr_A\expandafter0\expandafter{\expandafter}%
660 \romannumeral0\XINT_RQ { }#1\R\R\R\R\R\R\R\R\Z
661 \W\X\Y\Z #2\W\X\Y\Z }%
662 }%
663 \def\XINT_sum_N #1\Z #2#3%
664 {%
665 \expandafter\XINT_sum_NN\expandafter
666 {\romannumeral0\expandafter
667 \XINT_addr_A\expandafter0\expandafter{\expandafter}%
668 \romannumeral0\XINT_RQ { }#1\R\R\R\R\R\R\R\R\Z
669 \W\X\Y\Z #3\W\X\Y\Z }{#2}%
670 }%
671 \def\XINT_sum_NN #1#2{\XINT_sum_loop {#2}{#1}}%

```

4.33 *\xintPrd*

\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.

```

672 \def\xintiiPrd {\romannumeral0\xintiiprd }%
673 \def\xintiiprd #1{\xintiiprdexpr #1\relax }%
674 \let\xintPrd\xintiiPrd
675 \let\xintprd\xintiiprd
676 \def\xintiiPrdExpr {\romannumeral0\xintiiprdexpr }%
677 \def\xintiiprdexpr {\expandafter\XINT_prdexpr\romannumeral-‘0}%
678 \let\xintPrdExpr\xintiiPrdExpr
679 \let\xintprdexpr\xintiiprdexpr
680 \def\XINT_prdexpr {\XINT_prod_loop_a 1\Z }%
681 \def\XINT_prod_loop_a #1\Z #2%
682 {\expandafter\XINT_prod_loop_b \romannumeral-‘0#2\Z #1\Z \Z}%
683 \def\XINT_prod_loop_b #1%

```

```

684   {\xint_gob_til_relax #1\XINT_prod_finished\relax\XINT_prod_loop_c #1}%
685 \def\XINT_prod_loop_c
686   {\expandafter\XINT_prod_loop_a\romannumeral0\XINT_mul_fork }%
687 \def\XINT_prod_finished #1\Z #2\Z \Z { #2}%

```

4.34 `\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...

```

688 \def\xintiFac {\romannumeral0\xintifac }%
689 \def\xintifac #1%
690 {%
691   \expandafter\XINT_fac_fork\expandafter{\the\numexpr #1}%
692 }%
693 \let\xintFac\xintiFac \let\xintfac\xintifac
694 \def\XINT_fac_fork #1%
695 {%
696   \ifcase\XINT_cntSgn #1\Z
697     \xint_afterfi{\expandafter\space\expandafter 1\xint_gobble_i }%
698   \or
699     \expandafter\XINT_fac_checklength
700   \else
701     \xint_afterfi{\expandafter\xintError:FactorialOfNegativeNumber
702                   \expandafter\space\expandafter 1\xint_gobble_i }%
703   \fi
704   {#1}%
705 }%
706 \def\XINT_fac_checklength #1%
707 {%
708   \ifnum #1>100000
709     \xint_afterfi{\expandafter\xintError:FactorialOfTooBigNumber
710                   \expandafter\space\expandafter 1\xint_gobble_i }%
711   \else
712     \xint_afterfi{\ifnum #1>\xint_c_ixixixix
713                   \expandafter\XINT_fac_big_loop
714                   \else
715                     \expandafter\XINT_fac_loop
716                   \fi }%
717   \fi
718   {#1}%
719 }%
720 \def\XINT_fac_big_loop #1{\XINT_fac_big_loop_main {10000}{#1}}%
721 \def\XINT_fac_big_loop_main #1#2#3%
722 {%

```

4 Package *xint* implementation

```

723 \ifnum #1<#2
724 \expandafter
725 \XINT_fac_big_loop_main
726 \expandafter
727 {\the\numexpr #1+1\expandafter }%
728 \else
729 \expandafter\XINT_fac_big_docomputation
730 \fi
731 {#2}{#3{#1}}%
732 }%
733 \def\XINT_fac_big_docomputation #1#2%
734 {%
735 \expandafter \XINT_fac_bigcompute_loop \expandafter
736 {\romannumeral0\XINT_fac_loop {9999}}#2\relax
737 }%
738 \def\XINT_fac_bigcompute_loop #1#2%
739 {%
740 \xint_gob_til_relax #2\XINT_fac_bigcompute_end\relax
741 \expandafter\XINT_fac_bigcompute_loop\expandafter
742 {\expandafter\XINT_mul_enter
743 \romannumeral0\XINT_RQ { }#2\R\R\R\R\R\R\R\Z
744 \Z\Z\Z\Z #1\W\W\W\W }%
745 }%
746 \def\XINT_fac_bigcompute_end #1#2#3#4#5%
747 {%
748 \XINT_fac_bigcompute_end_ #5%
749 }%
750 \def\XINT_fac_bigcompute_end_ #1\R #2\Z \W\X\Y\Z #3\W\X\Y\Z { #3}%
751 \def\XINT_fac_loop #1{\XINT_fac_loop_main 1{1000}{#1}}%
752 \def\XINT_fac_loop_main #1#2#3%
753 {%
754 \ifnum #3>#1
755 \else
756 \expandafter\XINT_fac_loop_exit
757 \fi
758 \expandafter\XINT_fac_loop_main\expandafter
759 {\the\numexpr #1+1\expandafter }\expandafter
760 {\romannumeral0\XINT_mul_Mr {#1}#2\Z\Z\Z\Z }%
761 {#3}%
762 }%
763 \def\XINT_fac_loop_exit #1#2#3#4#5#6#7%
764 {%
765 \XINT_fac_loop_exit_ #6%
766 }%
767 \def\XINT_fac_loop_exit_ #1#2#3%
768 {%
769 \XINT_mul_M
770 }%

```


 DECIMAL OPERATIONS: FIRST DIGIT, LASTDIGIT, ODDNESS, MULTIPLICATION BY TEN, QUOTIENT BY TEN, QUOTIENT OR MULTIPLICATION BY POWER OF TEN, SPLIT OPERATION.

4.35 `\xintFDg`

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.

```

771 \def\xintiFDg {\romannumeral0\xintiifdg }%
772 \def\xintiifdg #1%
773 {%
774   \expandafter\XINT_fdg \romannumeral-'0#1\W\Z
775 }%
776 \def\xintFDg {\romannumeral0\xintfdg }%
777 \def\xintfdg #1%
778 {%
779   \expandafter\XINT_fdg \romannumeral0\xintnum{#1}\W\Z
780 }%
781 \def\XINT_FDg #1{\romannumeral0\XINT_fdg #1\W\Z }%
782 \def\XINT_fdg #1#2#3\Z
783 {%
784   \xint_UDzerominusfork
785   #1-{ 0}%    zero
786   0#1{ #2}%   negative
787   0-{ #1}%    positive
788   \krof
789 }%
```

4.36 `\xintLDg`

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 `\xintiLDg` is for defining `\xintiiOdd` which is used once (currently) elsewhere .

```

790 \def\xintiLDg {\romannumeral0\xintiildg }%
791 \def\xintiildg #1%
792 {%
793   \expandafter\XINT_ldg\expandafter {\romannumeral-'0#1}%
794 }%
795 \def\xintLDg {\romannumeral0\xintlbg }%
796 \def\xintlbg #1%
797 {%
798   \expandafter\XINT_ldg\expandafter {\romannumeral0\xintnum{#1}}%
799 }%
800 \def\XINT_LDg #1{\romannumeral0\XINT_ldg {#1}}%
801 \def\XINT_ldg #1%
802 {%
803   \expandafter\XINT_ldg_\romannumeral0\xintreverseorder {#1}\Z
804 }%
805 \def\XINT_ldg_ #1#2\Z{ #1}%
```

4.37 `\xintMON`, `\xintMMON`

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

```

806 \def\xintiiMON {\romannumeral0\xintiimon }%
```

```

807 \def\xintiimon #1%
808 {%
809   \ifodd\xintiiLDg {#1}
810     \xint_afterfi{ -1}%
811   \else
812     \xint_afterfi{ 1}%
813   \fi
814 }%
815 \def\xintiiMMON {\romannumeral0\xintiimmon }%
816 \def\xintiimmon #1%
817 {%
818   \ifodd\xintiiLDg {#1}
819     \xint_afterfi{ 1}%
820   \else
821     \xint_afterfi{ -1}%
822   \fi
823 }%
824 \def\xintMON {\romannumeral0\xintmon }%
825 \def\xintmon #1%
826 {%
827   \ifodd\xintLDg {#1}
828     \xint_afterfi{ -1}%
829   \else
830     \xint_afterfi{ 1}%
831   \fi
832 }%
833 \def\xintMMON {\romannumeral0\xintmmon }%
834 \def\xintmmon #1%
835 {%
836   \ifodd\xintLDg {#1}
837     \xint_afterfi{ 1}%
838   \else
839     \xint_afterfi{ -1}%
840   \fi
841 }%

```

4.38 `\xintOdd`

1.05 has `\xintiOdd`, whereas `\xintOdd` parses through `\xintNum`. Inadvertently, 1.09a redefined `\xintiLDg` so `\xintiOdd` also parsed through `\xintNum`. Anyway, having a `\xintOdd` and a `\xintiOdd` was silly. Removed in 1.09f, now `\xintOdd` and `\xintiiOdd`.

```

842 \def\xintiiOdd {\romannumeral0\xintiiodd }%
843 \def\xintiiodd #1%
844 {%
845   \ifodd\xintiiLDg{#1}
846     \xint_afterfi{ 1}%
847   \else
848     \xint_afterfi{ 0}%
849   \fi
850 }%
851 \def\xintiiEven {\romannumeral0\xintiieven }%
852 \def\xintiieven #1%

```

```

853 {%
854   \ifodd\xintiilDg{#1}
855     \xint_afterfi{ 0}%
856   \else
857     \xint_afterfi{ 1}%
858   \fi
859 }%
860 \def\xintOdd {\romannumeral0\xintodd }%
861 \def\xintodd #1%
862 {%
863   \ifodd\xintLDg{#1}
864     \xint_afterfi{ 1}%
865   \else
866     \xint_afterfi{ 0}%
867   \fi
868 }%
869 \def\xintEven {\romannumeral0\xinteven }%
870 \def\xinteven #1%
871 {%
872   \ifodd\xintLDg{#1}
873     \xint_afterfi{ 0}%
874   \else
875     \xint_afterfi{ 1}%
876   \fi
877 }%

```

4.39 `\xintDSL`

DECIMAL SHIFT LEFT (=MULTIPLICATION PAR 10)

```

878 \def\xintDSL {\romannumeral0\xintdsl }%
879 \def\xintdsl #1%
880 {%
881   \expandafter\XINT_dsl \romannumeral-'0#1\Z
882 }%
883 \def\XINT_DSL #1{\romannumeral0\XINT_dsl #1\Z }%
884 \def\XINT_dsl #1%
885 {%
886   \xint_gob_til_zero #1\xint_dsl_zero 0\XINT_dsl_ #1%
887 }%
888 \def\xint_dsl_zero 0\XINT_dsl_ 0#1\Z { 0}%
889 \def\XINT_dsl_ #1\Z { #10}%

```

4.40 `\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

```

890 \def\xintDSR {\romannumeral0\xintdsr }%
891 \def\xintdsr #1%
892 {%
893   \expandafter\XINT_dsr_a\expandafter {\romannumeral-'0#1}\W\Z
894 }%

```



```

895 \def\XINT_DSR #1{\romannumeral0\XINT_dsr_a {#1}\W\Z }%
896 \def\XINT_dsr_a
897 {%
898   \expandafter\XINT_dsr_b\romannumeral0\xintreverseorder
899 }%
900 \def\XINT_dsr_b #1#2#3\Z
901 {%
902   \xint_gob_til_W #2\xint_dsr_onedigit\W
903   \xint_gob_til_minus #2\xint_dsr_onedigit-%
904   \expandafter\XINT_dsr_removev
905   \romannumeral0\xintreverseorder {#2#3}%
906 }%
907 \def\xint_dsr_onedigit #1\xintreverseorder #2{ 0}%
908 \def\XINT_dsr_removev #1\W { }%

```

4.41 \xintDSH, \xintDSHr

DECIMAL SHIFTS \xintDSH {x}{A}

si $x \leq 0$, fait $A \rightarrow A \cdot 10^{|x|}$. v1.03 corrige l'oversight pour $A=0$.

si $x > 0$, et $A \geq 0$, fait $A \rightarrow \text{quo}(A, 10^x)$

si $x > 0$, et $A < 0$, fait $A \rightarrow -\text{quo}(-A, 10^x)$

(donc pour $x > 0$ c'est comme DSR itéré x fois)

\xintDSHr donne le 'reste' (si $x \leq 0$ donne zéro).

Release 1.06 now feeds x to a \numexpr first. I will have to revise this code at some point.

```

909 \def\xintDSHr {\romannumeral0\xintdshr }%
910 \def\xintdshr #1%
911 {%
912   \expandafter\XINT_dshr_checkxpositive \the\numexpr #1\relax\Z
913 }%
914 \def\XINT_dshr_checkxpositive #1%
915 {%
916   \xint_UDzerominusfork
917   0#1\XINT_dshr_xzeroorneg
918   #1-\XINT_dshr_xzeroorneg
919   0-\XINT_dshr_xpositive
920   \krof #1%
921 }%
922 \def\XINT_dshr_xzeroorneg #1\Z #2{ 0}%
923 \def\XINT_dshr_xpositive #1\Z
924 {%
925   \expandafter\xint_secondoftwo_thenstop\romannumeral0\xintdsx {#1}%
926 }%
927 \def\xintDSH {\romannumeral0\xintdsh }%
928 \def\xintdsh #1#2%
929 {%
930   \expandafter\xint_dsh\expandafter {\romannumeral-'0#2}{#1}%
931 }%
932 \def\xint_dsh #1#2%
933 {%
934   \expandafter\XINT_dsh_checksignx \the\numexpr #2\relax\Z {#1}%
935 }%
936 \def\XINT_dsh_checksignx #1%

```

```

937 {%
938   \xint_UDzerominusfork
939   #1-\XINT_dsh_xiszero
940   0#1\XINT_dsx_xisNeg_checkA      % on passe direct dans DSx
941   0-\XINT_dsh_xisPos #1}%
942   \krof
943 }%
944 \def\XINT_dsh_xiszero #1\Z #2{ #2}%
945 \def\XINT_dsh_xisPos #1\Z #2%
946 {%
947   \expandafter\xint_firstoftwo_thenstop
948   \romannumeral0\XINT_dsx_checksiga #2\Z {#1}% via DSx
949 }%

```

4.42 \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 $A \rightarrow A \cdot 10^{|x|}$
si $x \geq 0$, et $A \geq 0$, fait $A \rightarrow \{ \text{quo}(A, 10^x) \} \{ \text{rem}(A, 10^x) \}$
si $x \geq 0$, et $A < 0$, d'abord on calcule $\{ \text{quo}(-A, 10^x) \} \{ \text{rem}(-A, 10^x) \}$
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 $40000 = 8 \times 5000$ 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 fail, 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.

```

950 \def\xintDSx {\romannumeral0\xintdsx }%
951 \def\xintdsx #1#2%
952 {%
953   \expandafter\xint_dsx\expandafter {\romannumeral-'0#2}{#1}%
954 }%
955 \def\xint_dsx #1#2%
956 {%
957   \expandafter\XINT_dsx_checksiga \the\numexpr #2\relax\Z {#1}%
958 }%
959 \def\XINT_DSx #1#2{\romannumeral0\XINT_dsx_checksiga #1\Z {#2}}%
960 \def\XINT_dsx #1#2{\XINT_dsx_checksiga #1\Z {#2}}%
961 \def\XINT_dsx_checksiga #1%
962 {%

```

4 Package `xint` implementation

```

963 \xint_UDzerominusfork
964 #1-\XINT_dsx_xisZero
965 0#1\XINT_dsx_xisNeg_checkA
966 0-\XINT_dsx_xisPos #1}%
967 \krof
968 }%
969 \def\XINT_dsx_xisZero #1\Z #2{ {#2}{0}}% attention comme x > 0
970 \def\XINT_dsx_xisNeg_checkA #1\Z #2%
971 {%
972 \XINT_dsx_xisNeg_checkA_ #2\Z {#1}%
973 }%
974 \def\XINT_dsx_xisNeg_checkA_ #1#2\Z #3%
975 {%
976 \xint_gob_til_zero #1\XINT_dsx_xisNeg_Azero 0%
977 \XINT_dsx_xisNeg_checkx {#3}{#3}{ }\Z {#1#2}%
978 }%
979 \def\XINT_dsx_xisNeg_Azero #1\Z #2{ 0}%
980 \def\XINT_dsx_xisNeg_checkx #1%
981 {%
982 \ifnum #1>10000000
983 \xint_afterfi
984 {\xintError:TooBigDecimalShift
985 \expandafter\space\expandafter 0\xint_gobble_iv }%
986 \else
987 \expandafter \XINT_dsx_zeroloop
988 \fi
989 }%
990 \def\XINT_dsx_addzerosnofuss #1{\XINT_dsx_zeroloop {#1}{ }\Z }%
991 \def\XINT_dsx_zeroloop #1#2%
992 {%
993 \ifnum #1<\xint_c_ix \XINT_dsx_exita\fi
994 \expandafter\XINT_dsx_zeroloop\expandafter
995 {\the\numexpr #1-\xint_c_viii}{#200000000}%
996 }%
997 \def\XINT_dsx_exita\fi\expandafter\XINT_dsx_zeroloop
998 {%
999 \fi\expandafter\XINT_dsx_exitb
1000 }%
1001 \def\XINT_dsx_exitb #1#2%
1002 {%
1003 \expandafter\expandafter\expandafter
1004 \XINT_dsx_addzeros\csname xint_gobble_\romannumeral -#1\endcsname #2%
1005 }%
1006 \def\XINT_dsx_addzeros #1\Z #2{ #2#1}%
1007 \def\XINT_dsx_xisPos #1\Z #2%
1008 {%
1009 \XINT_dsx_checksignA #2\Z {#1}%
1010 }%
1011 \def\XINT_dsx_checksignA #1%
1012 {%
1013 \xint_UDzerominusfork
1014 #1-\XINT_dsx_AisZero

```

4 Package `xint` implementation

```

1015      0#1\XINT_dsx_AisNeg
1016      0-{\XINT_dsx_AisPos #1}%
1017      \krof
1018 }%
1019 \def\XINT_dsx_AisZero #1\Z #2{ {0}{0}}%
1020 \def\XINT_dsx_AisNeg #1\Z #2%
1021 {%
1022     \expandafter\XINT_dsx_AisNeg_dosplit_andcheckfirst
1023     \romannumeral0\XINT_split_checksizex {#2}{#1}%
1024 }%
1025 \def\XINT_dsx_AisNeg_dosplit_andcheckfirst #1%
1026 {%
1027     \XINT_dsx_AisNeg_checkiffirstempty #1\Z
1028 }%
1029 \def\XINT_dsx_AisNeg_checkiffirstempty #1%
1030 {%
1031     \xint_gob_til_Z #1\XINT_dsx_AisNeg_finish_zero\Z
1032     \XINT_dsx_AisNeg_finish_notzero #1%
1033 }%
1034 \def\XINT_dsx_AisNeg_finish_zero\Z
1035     \XINT_dsx_AisNeg_finish_notzero\Z #1%
1036 {%
1037     \expandafter\XINT_dsx_end
1038     \expandafter {\romannumeral0\XINT_num {-#1}}{0}%
1039 }%
1040 \def\XINT_dsx_AisNeg_finish_notzero #1\Z #2%
1041 {%
1042     \expandafter\XINT_dsx_end
1043     \expandafter {\romannumeral0\XINT_num {#2}}{-#1}%
1044 }%
1045 \def\XINT_dsx_AisPos #1\Z #2%
1046 {%
1047     \expandafter\XINT_dsx_AisPos_finish
1048     \romannumeral0\XINT_split_checksizex {#2}{#1}%
1049 }%
1050 \def\XINT_dsx_AisPos_finish #1#2%
1051 {%
1052     \expandafter\XINT_dsx_end
1053     \expandafter {\romannumeral0\XINT_num {#2}}%
1054     {\romannumeral0\XINT_num {#1}}%
1055 }%
1056 \edef\XINT_dsx_end #1#2%
1057 {%
1058     \noexpand\expandafter\space\noexpand\expandafter{#2}{#1}%
1059 }%

```

4.43 `\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

4 Package *xint* implementation

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 `xintiabs` rather without that overhead.

```

1060 \def\xintDecSplitL {\romannumeral0\xintdecsplitl}%
1061 \def\xintDecSplitR {\romannumeral0\xintdecsplitr}%
1062 \def\xintdecsplitl
1063 {%
1064   \expandafter\xint_firstoftwo_thenstop
1065   \romannumeral0\xintdecsplit
1066 }%
1067 \def\xintdecsplitr
1068 {%
1069   \expandafter\xint_secondoftwo_thenstop
1070   \romannumeral0\xintdecsplit
1071 }%
1072 \def\xintDecSplit {\romannumeral0\xintdecsplit}%
1073 \def\xintdecsplit #1#2%
1074 {%
1075   \expandafter \xint_split \expandafter
1076   {\romannumeral0\xintiabs {#2}}{#1}%   fait expansion de A
1077 }%
1078 \def\xint_split #1#2%
1079 {%
1080   \expandafter\XINT_split_checksizex\expandafter{\the\numexpr #2}{#1}%
1081 }%
1082 \def\XINT_split_checksizex #1% 999999999 is anyhow very big, could be reduced
1083 {%
1084   \ifnum\numexpr\XINT_Abs{#1}>999999999
1085     \xint_afterfi {\xintError:TooBigDecimalSplit\XINT_split_bigx}%
1086   \else
1087     \expandafter\XINT_split_xfork
1088   \fi
1089   #1\Z
1090 }%
1091 \def\XINT_split_bigx  #1\Z #2%
1092 {%
1093   \ifcase\XINT_cntSgn #1\Z
1094   \or \xint_afterfi { }{#2}% positive big x
1095   \else
1096     \xint_afterfi { }{#2}% negative big x
1097   \fi
1098 }%
1099 \def\XINT_split_xfork #1%

```

4 Package `xint` implementation

```

1100 {%
1101   \xint_UDzerominusfork
1102   #1-\XINT_split_zerosplit
1103   0#1\XINT_split_fromleft
1104   0-\XINT_split_fromright #1}%
1105   \krof
1106 }%
1107 \def\XINT_split_zerosplit #1\Z #2{ {#2}{}}%
1108 \def\XINT_split_fromleft #1\Z #2%
1109 {%
1110   \XINT_split_fromleft_loop {#1}{#2\W\W\W\W\W\W\W\W\Z
1111 }%
1112 \def\XINT_split_fromleft_loop #1%
1113 {%
1114   \ifnum #1<\xint_c_viii\XINT_split_fromleft_exita\fi
1115   \expandafter\XINT_split_fromleft_loop_perhaps\expandafter
1116   {\the\numexpr #1-\xint_c_viii\expandafter}\XINT_split_fromleft_eight
1117 }%
1118 \def\XINT_split_fromleft_eight #1#2#3#4#5#6#7#8#9{#9{#1#2#3#4#5#6#7#8#9}}%
1119 \def\XINT_split_fromleft_loop_perhaps #1#2%
1120 {%
1121   \xint_gob_til_W #2\XINT_split_fromleft_toofar\W
1122   \XINT_split_fromleft_loop {#1}%
1123 }%
1124 \def\XINT_split_fromleft_toofar\W\XINT_split_fromleft_loop #1#2#3\Z
1125 {%
1126   \XINT_split_fromleft_toofar_b #2\Z
1127 }%
1128 \def\XINT_split_fromleft_toofar_b #1\W #2\Z { {#1}{}}%
1129 \def\XINT_split_fromleft_exita\fi
1130   \expandafter\XINT_split_fromleft_loop_perhaps\expandafter #1#2%
1131   {\fi \XINT_split_fromleft_exitb #1}%
1132 \def\XINT_split_fromleft_exitb\the\numexpr #1-\xint_c_viii\expandafter
1133 {%
1134   \csname XINT_split_fromleft_endsplit_\romannumeral #1\endcsname
1135 }%
1136 \def\XINT_split_fromleft_endsplit_ #1#2\W #3\Z { {#1}{#2}}%
1137 \def\XINT_split_fromleft_endsplit_i #1#2%
1138   {\XINT_split_fromleft_checkiftoofar #2{#1#2}}%
1139 \def\XINT_split_fromleft_endsplit_ii #1#2#3%
1140   {\XINT_split_fromleft_checkiftoofar #3{#1#2#3}}%
1141 \def\XINT_split_fromleft_endsplit_iii #1#2#3#4%
1142   {\XINT_split_fromleft_checkiftoofar #4{#1#2#3#4}}%
1143 \def\XINT_split_fromleft_endsplit_iv #1#2#3#4#5%
1144   {\XINT_split_fromleft_checkiftoofar #5{#1#2#3#4#5}}%
1145 \def\XINT_split_fromleft_endsplit_v #1#2#3#4#5#6%
1146   {\XINT_split_fromleft_checkiftoofar #6{#1#2#3#4#5#6}}%
1147 \def\XINT_split_fromleft_endsplit_vi #1#2#3#4#5#6#7%
1148   {\XINT_split_fromleft_checkiftoofar #7{#1#2#3#4#5#6#7}}%
1149 \def\XINT_split_fromleft_endsplit_vii #1#2#3#4#5#6#7#8%
1150   {\XINT_split_fromleft_checkiftoofar #8{#1#2#3#4#5#6#7#8}}%
1151 \def\XINT_split_fromleft_checkiftoofar #1#2#3\W #4\Z

```

4 Package `xint` implementation

```

1152 {%
1153   \xint_gob_til_W #1\XINT_split_fromleft_wenttoofar\W
1154   \space {#2}{#3}%
1155 }%
1156 \def\XINT_split_fromleft_wenttoofar\W\space #1%
1157 {%
1158   \XINT_split_fromleft_wenttoofar_b #1\Z
1159 }%
1160 \def\XINT_split_fromleft_wenttoofar_b #1\W #2\Z { {#1}}%
1161 \def\XINT_split_fromright #1\Z #2%
1162 {%
1163   \expandafter \XINT_split_fromright_a \expandafter
1164   {\romannumeral0\xintreverseorder {#2}}{#1}{#2}%
1165 }%
1166 \def\XINT_split_fromright_a #1#2%
1167 {%
1168   \XINT_split_fromright_loop {#2}{#1}\W\W\W\W\W\W\W\W\Z
1169 }%
1170 \def\XINT_split_fromright_loop #1%
1171 {%
1172   \ifnum #1<\xint_c_viii\XINT_split_fromright_exita\fi
1173   \expandafter\XINT_split_fromright_loop_perhaps\expandafter
1174   {\the\numexpr #1-\xint_c_viii\expandafter }\XINT_split_fromright_eight
1175 }%
1176 \def\XINT_split_fromright_eight #1#2#3#4#5#6#7#8#9{#9{#9#8#7#6#5#4#3#2#1}}%
1177 \def\XINT_split_fromright_loop_perhaps #1#2%
1178 {%
1179   \xint_gob_til_W #2\XINT_split_fromright_toofar\W
1180   \XINT_split_fromright_loop {#1}%
1181 }%
1182 \def\XINT_split_fromright_toofar\W\XINT_split_fromright_loop #1#2#3\Z { {}}%
1183 \def\XINT_split_fromright_exita\fi
1184   \expandafter\XINT_split_fromright_loop_perhaps\expandafter #1#2%
1185   {\fi \XINT_split_fromright_exitb #1}%
1186 \def\XINT_split_fromright_exitb\the\numexpr #1-\xint_c_viii\expandafter
1187 {%
1188   \csname XINT_split_fromright_endsplit_\romannumeral #1\endcsname
1189 }%
1190 \edef\XINT_split_fromright_endsplit_ #1#2\W #3\Z #4%
1191 {%
1192   \noexpand\expandafter\space\noexpand\expandafter
1193   {\noexpand\romannumeral0\noexpand\xintreverseorder {#2}}{#1}%
1194 }%
1195 \def\XINT_split_fromright_endsplit_i #1#2%
1196   {\XINT_split_fromright_checkiftoofar #2{#2#1}}%
1197 \def\XINT_split_fromright_endsplit_ii #1#2#3%
1198   {\XINT_split_fromright_checkiftoofar #3{#3#2#1}}%
1199 \def\XINT_split_fromright_endsplit_iii #1#2#3#4%
1200   {\XINT_split_fromright_checkiftoofar #4{#4#3#2#1}}%
1201 \def\XINT_split_fromright_endsplit_iv #1#2#3#4#5%
1202   {\XINT_split_fromright_checkiftoofar #5{#5#4#3#2#1}}%
1203 \def\XINT_split_fromright_endsplit_v #1#2#3#4#5#6%

```

```

1204      {\XINT_split_fromright_checkiftoofar #6{#6#5#4#3#2#1}}%
1205 \def\XINT_split_fromright_endsplit_vi #1#2#3#4#5#6#7%
1206      {\XINT_split_fromright_checkiftoofar #7{#7#6#5#4#3#2#1}}%
1207 \def\XINT_split_fromright_endsplit_vii #1#2#3#4#5#6#7#8%
1208      {\XINT_split_fromright_checkiftoofar #8{#8#7#6#5#4#3#2#1}}%
1209 \def\XINT_split_fromright_checkiftoofar #1%
1210 {%
1211   \xint_gob_til_W #1\XINT_split_fromright_wenttoofar\W
1212   \XINT_split_fromright_endsplit_
1213 }%
1214 \def\XINT_split_fromright_wenttoofar\W\XINT_split_fromright_endsplit_ #1\Z #2%
1215   { {}{#2}}%

```

4.44 `\xintDouble`

`v1.08`

```

1216 \def\xintDouble {\romannumeral0\xintdouble }%
1217 \def\xintdouble #1%
1218 {%
1219   \expandafter\XINT_dbl\romannumeral-'0#1%
1220   \R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W
1221 }%
1222 \def\XINT_dbl #1%
1223 {%
1224   \xint_UDzerominusfork
1225   #1-\XINT_dbl_zero
1226   0#1\XINT_dbl_neg
1227   0-{\XINT_dbl_pos #1}%
1228   \krof
1229 }%
1230 \def\XINT_dbl_zero #1\Z \W\W\W\W\W\W\W { 0}%
1231 \def\XINT_dbl_neg
1232   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_dbl_pos }%
1233 \def\XINT_dbl_pos
1234 {%
1235   \expandafter\XINT_dbl_a \expandafter{\expandafter}\expandafter 0%
1236   \romannumeral0\XINT_SQ {}%
1237 }%
1238 \def\XINT_dbl_a #1#2#3#4#5#6#7#8#9%
1239 {%
1240   \xint_gob_til_W #9\XINT_dbl_end_a\W
1241   \expandafter\XINT_dbl_b
1242   \the\numexpr \xint_c_x^viii+#2+\xint_c_ii*#9#8#7#6#5#4#3\relax {#1}%
1243 }%
1244 \def\XINT_dbl_b 1#1#2#3#4#5#6#7#8#9%
1245 {%
1246   \XINT_dbl_a {#2#3#4#5#6#7#8#9}{#1}%
1247 }%
1248 \def\XINT_dbl_end_a #1+#2+#3\relax #4%
1249 {%
1250   \expandafter\XINT_dbl_end_b #2#4%
1251 }%

```



```

1252 \edef\XINT_dbl_end_b #1#2#3#4#5#6#7#8%
1253 {%
1254     \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7#8\relax
1255 }%

```

4.45 `\xintHalf`

`v1.08`

```

1256 \def\xintHalf {\romannumeral0\xinthalf }%
1257 \def\xinthalf #1%
1258 {%
1259     \expandafter\XINT_half\romannumeral-‘0#1%
1260     \R\R\R\R\R\R\R\Z \W\W\W\W\W\W
1261 }%
1262 \def\XINT_half #1%
1263 {%
1264     \xint_UDzerominusfork
1265     #1-\XINT_half_zero
1266     0#1\XINT_half_neg
1267     0-{ \XINT_half_pos #1}%
1268     \krof
1269 }%
1270 \def\XINT_half_zero #1\Z \W\W\W\W\W\W { 0}%
1271 \def\XINT_half_neg {\expandafter\XINT_opp\romannumeral0\XINT_half_pos }%
1272 \def\XINT_half_pos {\expandafter\XINT_half_a\romannumeral0\XINT_SQ {}}%
1273 \def\XINT_half_a #1#2#3#4#5#6#7#8%
1274 {%
1275     \xint_gob_til_W #8\XINT_half_dont\W
1276     \expandafter\XINT_half_b
1277     \the\numexpr \xint_c_x^viii+\xint_c_v*#7#6#5#4#3#2#1\relax #8%
1278 }%
1279 \edef\XINT_half_dont\W\expandafter\XINT_half_b
1280     \the\numexpr \xint_c_x^viii+\xint_c_v*#1#2#3#4#5#6#7\relax \W\W\W\W\W\W
1281 {%
1282     \noexpand\expandafter\space
1283     \noexpand\the\numexpr (#1#2#3#4#5#6#7+\xint_c_i)/\xint_c_ii-\xint_c_i \relax
1284 }%
1285 \def\XINT_half_b 1#1#2#3#4#5#6#7#8%
1286 {%
1287     \XINT_half_c {#2#3#4#5#6#7}{#1}%
1288 }%
1289 \def\XINT_half_c #1#2#3#4#5#6#7#8#9%
1290 {%
1291     \xint_gob_til_W #3\XINT_half_end_a #2\W
1292     \expandafter\XINT_half_d
1293     \the\numexpr \xint_c_x^viii+\xint_c_v*#9#8#7#6#5#4#3+#2\relax {#1}%
1294 }%
1295 \def\XINT_half_d 1#1#2#3#4#5#6#7#8#9%
1296 {%
1297     \XINT_half_c {#2#3#4#5#6#7#8#9}{#1}%
1298 }%
1299 \def\XINT_half_end_a #1\W #2\relax #3%

```

```

1300 {%
1301   \xint_gob_til_zero #1\XINT_half_end_b 0\space #1#3%
1302 }%
1303 \edef\XINT_half_end_b 0\space 0#1#2#3#4#5#6#7%
1304 {%
1305   \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7\relax
1306 }%

```

4.46 `\xintiiSqrt`, `\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`, etc...

```

1307 \def\xintiiSqrt {\romannumeral0\xintiisqrt }%
1308 \def\xintiisqrt
1309   {\expandafter\XINT_sqrt_post\romannumeral0\xintiisquareroot }%
1310 \def\XINT_sqrt_post #1#2{\XINT_dec_pos #1\R\R\R\R\R\R\R\R\Z
1311   \W\W\W\W\W\W\W\W }%
1312 \def\xintiiSquareRoot {\romannumeral0\xintiisquareroot }%
1313 \def\xintiisquareroot #1%
1314   {\expandafter\XINT_sqrt_checkin\romannumeral-‘0#1\Z}%
1315 \def\xintiSqrt {\romannumeral0\xintisqrt }%
1316 \def\xintisqrt
1317   {\expandafter\XINT_sqrt_post\romannumeral0\xintisquareroot }%
1318 \def\xintiSquareRoot {\romannumeral0\xintisquareroot }%
1319 \def\xintisquareroot #1%
1320   {\expandafter\XINT_sqrt_checkin\romannumeral0\xintnum{#1}\Z}%
1321 \def\XINT_sqrt_checkin #1%
1322 {%
1323   \xint_UDzerominusfork
1324   #1-\XINT_sqrt_iszero
1325   0#1\XINT_sqrt_isneg
1326   0-{\XINT_sqrt #1}%
1327   \krof
1328 }%
1329 \def\XINT_sqrt_iszero #1\Z { 1.}%
1330 \edef\XINT_sqrt_isneg #1\Z {\noexpand\xintError:RootOfNegative\space 1.}%
1331 \def\XINT_sqrt #1\Z
1332 {%
1333   \expandafter\XINT_sqrt_start\expandafter
1334     {\romannumeral0\xintlenght {#1}}{#1}%
1335 }%
1336 \def\XINT_sqrt_start #1%
1337 {%
1338   \ifnum #1<\xint_c_x
1339     \expandafter\XINT_sqrt_small_a
1340   \else
1341     \expandafter\XINT_sqrt_big_a
1342   \fi

```

4 Package *xint* implementation

```

1343     {#1}%
1344 }%
1345 \def\XINT_sqrt_small_a #1{\XINT_sqrt_a {#1}\XINT_sqrt_small_d }%
1346 \def\XINT_sqrt_big_a #1{\XINT_sqrt_a {#1}\XINT_sqrt_big_d }%
1347 \def\XINT_sqrt_a #1%
1348 {%
1349     \ifodd #1
1350         \expandafter\XINT_sqrt_bB
1351     \else
1352         \expandafter\XINT_sqrt_bA
1353     \fi
1354     {#1}%
1355 }%
1356 \def\XINT_sqrt_bA #1#2#3%
1357 {%
1358     \XINT_sqrt_bA_b #3\Z #2{#1}{#3}%
1359 }%
1360 \def\XINT_sqrt_bA_b #1#2#3\Z
1361 {%
1362     \XINT_sqrt_c {#1#2}%
1363 }%
1364 \def\XINT_sqrt_bB #1#2#3%
1365 {%
1366     \XINT_sqrt_bB_b #3\Z #2{#1}{#3}%
1367 }%
1368 \def\XINT_sqrt_bB_b #1#2\Z
1369 {%
1370     \XINT_sqrt_c #1%
1371 }%
1372 \def\XINT_sqrt_c #1#2%
1373 {%
1374     \expandafter #2\expandafter
1375     {\the\numexpr\ifnum #1>\xint_c_iii
1376         \ifnum #1>\xint_c_viii
1377             \ifnum #1>15 \ifnum #1>24 \ifnum #1>35
1378                 \ifnum #1>48 \ifnum #1>63 \ifnum #1>80
1379                 10\else 9\fi \else 8\fi \else 7\fi \else 6\fi
1380                 \else 5\fi \else 4\fi \else 3\fi \else 2\fi \relax }%
1381 }%
1382 \def\XINT_sqrt_small_d #1#2%
1383 {%
1384     \expandafter\XINT_sqrt_small_e\expandafter
1385     {\the\numexpr #1\ifcase \numexpr #2/\xint_c_ii-\xint_c_i\relax
1386         \or 0\or 00\or 000\or 0000\fi }%
1387 }%
1388 \def\XINT_sqrt_small_e #1#2%
1389 {%
1390     \expandafter\XINT_sqrt_small_f\expandafter {\the\numexpr #1*#1-#2}{#1}%
1391 }%
1392 \def\XINT_sqrt_small_f #1#2%
1393 {%
1394     \expandafter\XINT_sqrt_small_g\expandafter

```

4 Package `xint` implementation

```

1395   {\the\numexpr ((#1+#2)/(\xint_c_ii*#2))-\xint_c_i}{#1}{#2}%
1396 }%
1397 \def\xint_sqrt_small_g #1%
1398 {%
1399   \ifnum #1>\xint_c_
1400     \expandafter\xint_sqrt_small_h
1401   \else
1402     \expandafter\xint_sqrt_small_end
1403   \fi
1404   {#1}%
1405 }%
1406 \def\xint_sqrt_small_h #1#2#3%
1407 {%
1408   \expandafter\xint_sqrt_small_f\expandafter
1409   {\the\numexpr #2-\xint_c_ii*#1*#3+#1*#1\expandafter}\expandafter
1410   {\the\numexpr #3-#1}%
1411 }%
1412 \def\xint_sqrt_small_end #1#2#3{ {#3}{#2}}%
1413 \def\xint_sqrt_big_d #1#2%
1414 {%
1415   \ifodd #2
1416     \expandafter\expandafter\expandafter\xint_sqrt_big_eB
1417   \else
1418     \expandafter\expandafter\expandafter\xint_sqrt_big_eA
1419   \fi
1420   \expandafter {\the\numexpr #2/\xint_c_ii }{#1}%
1421 }%
1422 \def\xint_sqrt_big_eA #1#2#3%
1423 {%
1424   \xint_sqrt_big_eA_a #3\Z {#2}{#1}{#3}%
1425 }%
1426 \def\xint_sqrt_big_eA_a #1#2#3#4#5#6#7#8#9\Z
1427 {%
1428   \xint_sqrt_big_eA_b {#1#2#3#4#5#6#7#8}%
1429 }%
1430 \def\xint_sqrt_big_eA_b #1#2%
1431 {%
1432   \expandafter\xint_sqrt_big_f
1433   \romannumeral0\xint_sqrt_small_e {#2000}{#1}{#1}%
1434 }%
1435 \def\xint_sqrt_big_eB #1#2#3%
1436 {%
1437   \xint_sqrt_big_eB_a #3\Z {#2}{#1}{#3}%
1438 }%
1439 \def\xint_sqrt_big_eB_a #1#2#3#4#5#6#7#8#9%
1440 {%
1441   \xint_sqrt_big_eB_b {#1#2#3#4#5#6#7#8#9}%
1442 }%
1443 \def\xint_sqrt_big_eB_b #1#2\Z #3%
1444 {%
1445   \expandafter\xint_sqrt_big_f
1446   \romannumeral0\xint_sqrt_small_e {#30000}{#1}{#1}%

```

```

1447 }%
1448 \def\XINT_sqrt_big_f #1#2#3#4%
1449 {%
1450   \expandafter\XINT_sqrt_big_f_a\expandafter
1451   {\the\numexpr #2+#3\expandafter}\expandafter
1452   {\romannumeral0\XINT_dsx_addzerosnofuss
1453     {\numexpr #4-\xint_c_iv\relax}{#1}}{#4}%
1454 }%
1455 \def\XINT_sqrt_big_f_a #1#2#3#4%
1456 {%
1457   \expandafter\XINT_sqrt_big_g\expandafter
1458   {\romannumeral0\xintiisub
1459     {\XINT_dsx_addzerosnofuss
1460       {\numexpr \xint_c_ii*#3-\xint_c_viii\relax}{#1}}{#4}}%
1461   {#2}{#3}%
1462 }%
1463 \def\XINT_sqrt_big_g #1#2%
1464 {%
1465   \expandafter\XINT_sqrt_big_j
1466   \romannumeral0\xintiidivision{#1}%
1467   {\romannumeral0\XINT_dbl_pos #2\R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W }{#2}%
1468 }%
1469 \def\XINT_sqrt_big_j #1%
1470 {%
1471   \if0\XINT_Sgn #1\Z
1472     \expandafter \XINT_sqrt_big_end
1473   \else \expandafter \XINT_sqrt_big_k
1474   \fi {#1}%
1475 }%
1476 \def\XINT_sqrt_big_k #1#2#3%
1477 {%
1478   \expandafter\XINT_sqrt_big_l\expandafter
1479   {\romannumeral0\xintiisub {#3}{#1}}%
1480   {\romannumeral0\xintiiadd {#2}{\xintiiSqr {#1}}}%
1481 }%
1482 \def\XINT_sqrt_big_l #1#2%
1483 {%
1484   \expandafter\XINT_sqrt_big_g\expandafter
1485   {#2}{#1}%
1486 }%
1487 \def\XINT_sqrt_big_end #1#2#3#4{ {#3}{#2}}%

```

4.47 \xintiiE

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

```

1488 \def\xintiiE {\romannumeral0\xintiiE }% used in \xintMod.
1489 \def\xintiiE #1#2%
1490   {\expandafter\XINT_iiE\the\numexpr #2\expandafter.\expandafter{\romannumeral-‘0#1}}%
1491 \def\XINT_iiE #1.#2{\ifnum#1>\xint_c_ \xint_dothis{\xint_dsh {#2}{-#1}}\fi
1492   \xint_orthat{ #2}}%
1493 \XINT_restorecatcodes_endinput%

```

5 Package *xintbinhex* implementation

.1	Catcodes, ε -TeX and reload detection	118	.6	<code>\xintBinToDec</code>	125
.2	Package identification	119	.7	<code>\xintBinToHex</code>	128
.3	Constants, etc...	119	.8	<code>\xintHexToBin</code>	128
.4	<code>\xintDecToHex</code> , <code>\xintDecToBin</code>	121	.9	<code>\xintCHexToBin</code>	129
.5	<code>\xintHexToDec</code>	124			

The commenting is currently (2014/10/28) 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.

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2 \catcode13=5 % ^^M
3 \endlinechar=13 %
4 \catcode123=1 % {
5 \catcode125=2 % }
6 \catcode64=11 % @
7 \catcode35=6 % #
8 \catcode44=12 % ,
9 \catcode45=12 % -
10 \catcode46=12 % .
11 \catcode58=12 % :
12 \let\z\endgroup
13 \expandafter\let\expandafter\x\csname ver@xintbinhex.sty\endcsname
14 \expandafter\let\expandafter\w\csname ver@xintcore.sty\endcsname
15 \expandafter
16 \ifx\csname PackageInfo\endcsname\relax
17 \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
18 \else
19 \def\y#1#2{\PackageInfo{#1}{#2}}%
20 \fi
21 \expandafter
22 \ifx\csname numexpr\endcsname\relax
23 \y{xintbinhex}{\numexpr not available, aborting input}%
24 \aftergroup\endinput
25 \else
26 \ifx\x\relax % plain-TeX, first loading of xintbinhex.sty
27 \ifx\w\relax % but xintcore.sty not yet loaded.
28 \def\z{\endgroup\input xintcore.sty\relax}%
29 \fi
30 \else
31 \def\empty {}%
32 \ifx\x\empty % LaTeX, first loading,
33 % variable is initialized, but \ProvidesPackage not yet seen
34 \ifx\w\relax % xintcore.sty not yet loaded.
35 \def\z{\endgroup\RequirePackage{xintcore}}%
36 \fi
37 \else
38 \aftergroup\endinput % xintbinhex already loaded.
39 \fi

```

```

40 \fi
41 \fi
42 \z%
43 \XINTsetupcatcodes% defined in xintkernel.sty

```

5.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintbinhex}%
46 [2014/10/28 v1.1 Expandable binary and hexadecimal conversions (jfb)]%

```

5.3 Constants, etc...

v1.08

```

47 \chardef\xint_c_xvi          16
48 % \chardef\xint_c_ii^v       32 % already in xint.sty
49 % \chardef\xint_c_ii^vi      64 % already in xint.sty
50 \chardef\xint_c_ii^vii       128
51 \mathchardef\xint_c_ii^viii  256
52 \mathchardef\xint_c_ii^xii   4096
53 \newcount\xint_c_ii^xv \xint_c_ii^xv 32768
54 \newcount\xint_c_ii^xvi \xint_c_ii^xvi 65536
55 \newcount\xint_c_x^v \xint_c_x^v 100000
56 \newcount\xint_c_x^ix \xint_c_x^ix 1000000000
57 \def\xINT_tmpa #1{\ifx\relax#1\else
58 \expandafter\edef\csname XINT_sdth_#1\endcsname
59 {\ifcase #1 0\or 1\or 2\or 3\or 4\or 5\or 6\or 7\or
60 8\or 9\or A\or B\or C\or D\or E\or F\fi}%
61 \expandafter\xINT_tmpa\fi }%
62 \XINT_tmpa {0}{1}{2}{3}{4}{5}{6}{7}{8}{9}{10}{11}{12}{13}{14}{15}\relax
63 \def\xINT_tmpa #1{\ifx\relax#1\else
64 \expandafter\edef\csname XINT_sdtb_#1\endcsname
65 {\ifcase #1
66 0000\or 0001\or 0010\or 0011\or 0100\or 0101\or 0110\or 0111\or
67 1000\or 1001\or 1010\or 1011\or 1100\or 1101\or 1110\or 1111\fi}%
68 \expandafter\xINT_tmpa\fi }%
69 \XINT_tmpa {0}{1}{2}{3}{4}{5}{6}{7}{8}{9}{10}{11}{12}{13}{14}{15}\relax
70 \let\xINT_tmpa\relax
71 \expandafter\def\csname XINT_sbtd_0000\endcsname {0}%
72 \expandafter\def\csname XINT_sbtd_0001\endcsname {1}%
73 \expandafter\def\csname XINT_sbtd_0010\endcsname {2}%
74 \expandafter\def\csname XINT_sbtd_0011\endcsname {3}%
75 \expandafter\def\csname XINT_sbtd_0100\endcsname {4}%
76 \expandafter\def\csname XINT_sbtd_0101\endcsname {5}%
77 \expandafter\def\csname XINT_sbtd_0110\endcsname {6}%
78 \expandafter\def\csname XINT_sbtd_0111\endcsname {7}%
79 \expandafter\def\csname XINT_sbtd_1000\endcsname {8}%
80 \expandafter\def\csname XINT_sbtd_1001\endcsname {9}%
81 \expandafter\def\csname XINT_sbtd_1010\endcsname {10}%
82 \expandafter\def\csname XINT_sbtd_1011\endcsname {11}%
83 \expandafter\def\csname XINT_sbtd_1100\endcsname {12}%
84 \expandafter\def\csname XINT_sbtd_1101\endcsname {13}%
85 \expandafter\def\csname XINT_sbtd_1110\endcsname {14}%

```

5 Package *xintbinhex* implementation

```

86 \expandafter\def\csname XINT_sbtd_1111\endcsname {15}%
87 \expandafter\let\csname XINT_sbth_0000\expandafter\endcsname
88     \csname XINT_sbtd_0000\endcsname
89 \expandafter\let\csname XINT_sbth_0001\expandafter\endcsname
90     \csname XINT_sbtd_0001\endcsname
91 \expandafter\let\csname XINT_sbth_0010\expandafter\endcsname
92     \csname XINT_sbtd_0010\endcsname
93 \expandafter\let\csname XINT_sbth_0011\expandafter\endcsname
94     \csname XINT_sbtd_0011\endcsname
95 \expandafter\let\csname XINT_sbth_0100\expandafter\endcsname
96     \csname XINT_sbtd_0100\endcsname
97 \expandafter\let\csname XINT_sbth_0101\expandafter\endcsname
98     \csname XINT_sbtd_0101\endcsname
99 \expandafter\let\csname XINT_sbth_0110\expandafter\endcsname
100     \csname XINT_sbtd_0110\endcsname
101 \expandafter\let\csname XINT_sbth_0111\expandafter\endcsname
102     \csname XINT_sbtd_0111\endcsname
103 \expandafter\let\csname XINT_sbth_1000\expandafter\endcsname
104     \csname XINT_sbtd_1000\endcsname
105 \expandafter\let\csname XINT_sbth_1001\expandafter\endcsname
106     \csname XINT_sbtd_1001\endcsname
107 \expandafter\def\csname XINT_sbth_1010\endcsname {A}%
108 \expandafter\def\csname XINT_sbth_1011\endcsname {B}%
109 \expandafter\def\csname XINT_sbth_1100\endcsname {C}%
110 \expandafter\def\csname XINT_sbth_1101\endcsname {D}%
111 \expandafter\def\csname XINT_sbth_1110\endcsname {E}%
112 \expandafter\def\csname XINT_sbth_1111\endcsname {F}%
113 \expandafter\def\csname XINT_shtb_0\endcsname {0000}%
114 \expandafter\def\csname XINT_shtb_1\endcsname {0001}%
115 \expandafter\def\csname XINT_shtb_2\endcsname {0010}%
116 \expandafter\def\csname XINT_shtb_3\endcsname {0011}%
117 \expandafter\def\csname XINT_shtb_4\endcsname {0100}%
118 \expandafter\def\csname XINT_shtb_5\endcsname {0101}%
119 \expandafter\def\csname XINT_shtb_6\endcsname {0110}%
120 \expandafter\def\csname XINT_shtb_7\endcsname {0111}%
121 \expandafter\def\csname XINT_shtb_8\endcsname {1000}%
122 \expandafter\def\csname XINT_shtb_9\endcsname {1001}%
123 \def\XINT_shtb_A {1010}%
124 \def\XINT_shtb_B {1011}%
125 \def\XINT_shtb_C {1100}%
126 \def\XINT_shtb_D {1101}%
127 \def\XINT_shtb_E {1110}%
128 \def\XINT_shtb_F {1111}%
129 \def\XINT_shtb_G {}%
130 \def\XINT_smallhex #1%
131 {%
132     \expandafter\XINT_smallhex_a\expandafter
133     {\the\numexpr (#1+\xint_c_viii)/\xint_c_xvi-\xint_c_i}{#1}%
134 }%
135 \def\XINT_smallhex_a #1#2%
136 {%
137     \csname XINT_sdth_#1\expandafter\expandafter\expandafter\endcsname

```



```

138 \csname XINT_sdth\_the\numexpr #2-\xint_c_xvi*#1\endcsname
139 }%
140 \def\XINT_smallbin #1%
141 {%
142 \expandafter\XINT_smallbin_a\expandafter
143 {\the\numexpr (#1+\xint_c_viii)/\xint_c_xvi-\xint_c_i}{#1}%
144 }%
145 \def\XINT_smallbin_a #1#2%
146 {%
147 \csname XINT_sdtb_#1\expandafter\expandafter\expandafter\endcsname
148 \csname XINT_sdtb\_the\numexpr #2-\xint_c_xvi*#1\endcsname
149 }%

```

5.4 \xintDecToHex, \xintDecToBin

v1.08

```

150 \def\xintDecToHex {\romannumeral0\xintdectohex }%
151 \def\xintdectohex #1%
152 {\expandafter\XINT_dth_checkin\romannumeral-'0#1\W\W\W\W \T}%
153 \def\XINT_dth_checkin #1%
154 {%
155 \xint_UDsignfork
156 #1\XINT_dth_N
157 -{\XINT_dth_P #1}%
158 \krof
159 }%
160 \def\XINT_dth_N {\expandafter\xint_minus_thenstop\romannumeral0\XINT_dth_P }%
161 \def\XINT_dth_P {\expandafter\XINT_dth_III\romannumeral-'0\XINT_dtbh_I {0.}}%
162 \def\xintDecToBin {\romannumeral0\xintdectobin }%
163 \def\xintdectobin #1%
164 {\expandafter\XINT_dtb_checkin\romannumeral-'0#1\W\W\W\W \T }%
165 \def\XINT_dtb_checkin #1%
166 {%
167 \xint_UDsignfork
168 #1\XINT_dtb_N
169 -{\XINT_dtb_P #1}%
170 \krof
171 }%
172 \def\XINT_dtb_N {\expandafter\xint_minus_thenstop\romannumeral0\XINT_dtb_P }%
173 \def\XINT_dtb_P {\expandafter\XINT_dtb_III\romannumeral-'0\XINT_dtbh_I {0.}}%
174 \def\XINT_dtbh_I #1#2#3#4#5%
175 {%
176 \xint_gob_til_W #5\XINT_dtbh_II_a\W\XINT_dtbh_I_a {}{#2#3#4#5}#1\Z.%
177 }%
178 \def\XINT_dtbh_II_a\W\XINT_dtbh_I_a #1#2{\XINT_dtbh_II_b #2}%
179 \def\XINT_dtbh_II_b #1#2#3#4%
180 {%
181 \xint_gob_til_W
182 #1\XINT_dtbh_II_c
183 #2\XINT_dtbh_II_ci
184 #3\XINT_dtbh_II_cii
185 \W\XINT_dtbh_II_ciii #1#2#3#4%

```

5 Package *xintbinhex* implementation

```

186 }%
187 \def\XINT_dtbh_II_c \W\XINT_dtbh_II_ci
188         \W\XINT_dtbh_II_cii
189         \W\XINT_dtbh_II_ciii \W\W\W\W {{{}%
190 \def\XINT_dtbh_II_ci #1\XINT_dtbh_II_ciii #2\W\W\W
191     {\XINT_dtbh_II_d {{}{#2}{0}}}%
192 \def\XINT_dtbh_II_cii\W\XINT_dtbh_II_ciii #1#2\W\W
193     {\XINT_dtbh_II_d {{}{#1#2}{00}}}%
194 \def\XINT_dtbh_II_ciii #1#2#3\W
195     {\XINT_dtbh_II_d {{}{#1#2#3}{000}}}%
196 \def\XINT_dtbh_I_a #1#2#3.%
197 {%
198     \xint_gob_til_Z #3\XINT_dtbh_I_z\Z
199     \expandafter\XINT_dtbh_I_b\the\numexpr #2+#30000.{#1}%
200 }%
201 \def\XINT_dtbh_I_b #1.%
202 {%
203     \expandafter\XINT_dtbh_I_c\the\numexpr
204     (#1+\xint_c_ii^xv)/\xint_c_ii^xvi-\xint_c_i.#1.%
205 }%
206 \def\XINT_dtbh_I_c #1.#2.%
207 {%
208     \expandafter\XINT_dtbh_I_d\expandafter
209     {\the\numexpr #2-\xint_c_ii^xvi*#1}{#1}%
210 }%
211 \def\XINT_dtbh_I_d #1#2#3{\XINT_dtbh_I_a {#3#1.}{#2}}%
212 \def\XINT_dtbh_I_z\Z\expandafter\XINT_dtbh_I_b\the\numexpr #1+#2.%
213 {%
214     \ifnum #1=\xint_c_ \expandafter\XINT_dtbh_I_end_zb\fi
215     \XINT_dtbh_I_end_za {#1}%
216 }%
217 \def\XINT_dtbh_I_end_za #1#2{\XINT_dtbh_I {#2#1.}}%
218 \def\XINT_dtbh_I_end_zb\XINT_dtbh_I_end_za #1#2{\XINT_dtbh_I {#2}}%
219 \def\XINT_dtbh_II_d #1#2#3#4.%
220 {%
221     \xint_gob_til_Z #4\XINT_dtbh_II_z\Z
222     \expandafter\XINT_dtbh_II_e\the\numexpr #2+#4#3.{#1}{#3}%
223 }%
224 \def\XINT_dtbh_II_e #1.%
225 {%
226     \expandafter\XINT_dtbh_II_f\the\numexpr
227     (#1+\xint_c_ii^xv)/\xint_c_ii^xvi-\xint_c_i.#1.%
228 }%
229 \def\XINT_dtbh_II_f #1.#2.%
230 {%
231     \expandafter\XINT_dtbh_II_g\expandafter
232     {\the\numexpr #2-\xint_c_ii^xvi*#1}{#1}%
233 }%
234 \def\XINT_dtbh_II_g #1#2#3{\XINT_dtbh_II_d {#3#1.}{#2}}%
235 \def\XINT_dtbh_II_z\Z\expandafter\XINT_dtbh_II_e\the\numexpr #1+#2.%
236 {%
237     \ifnum #1=\xint_c_ \expandafter\XINT_dtbh_II_end_zb\fi

```

5 Package *xintbinhex* implementation

```

238 \XINT_dtbh_II_end_za {#1}%
239 }%
240 \def\XINT_dtbh_II_end_za #1#2#3{{#2#1.\Z.}%
241 \def\XINT_dtbh_II_end_zb\XINT_dtbh_II_end_za #1#2#3{{#2\Z.}%
242 \def\XINT_dth_III #1#2.%
243 {%
244 \xint_gob_til_Z #2\XINT_dth_end\Z
245 \expandafter\XINT_dth_III\expandafter
246 {\romannumeral-'0\XINT_dth_small #2.#1}%
247 }%
248 \def\XINT_dth_small #1.%
249 {%
250 \expandafter\XINT_smallhex\expandafter
251 {\the\numexpr (#1+\xint_c_ii^vii)/\xint_c_ii^viii-\xint_c_i\expandafter}%
252 \romannumeral-'0\expandafter\XINT_smallhex\expandafter
253 {\the\numexpr
254 #1-((#1+\xint_c_ii^vii)/\xint_c_ii^viii-\xint_c_i)*\xint_c_ii^viii}%
255 }%
256 \def\XINT_dth_end\Z\expandafter\XINT_dth_III\expandafter #1#2\T
257 {%
258 \XINT_dth_end_b #1%
259 }%
260 \def\XINT_dth_end_b #1.{\XINT_dth_end_c }%
261 \def\XINT_dth_end_c #1{\xint_gob_til_zero #1\XINT_dth_end_d 0\space #1}%
262 \def\XINT_dth_end_d 0\space 0#1%
263 {%
264 \xint_gob_til_zero #1\XINT_dth_end_e 0\space #1%
265 }%
266 \def\XINT_dth_end_e 0\space 0#1%
267 {%
268 \xint_gob_til_zero #1\XINT_dth_end_f 0\space #1%
269 }%
270 \def\XINT_dth_end_f 0\space 0{ }%
271 \def\XINT_dtb_III #1#2.%
272 {%
273 \xint_gob_til_Z #2\XINT_dtb_end\Z
274 \expandafter\XINT_dtb_III\expandafter
275 {\romannumeral-'0\XINT_dtb_small #2.#1}%
276 }%
277 \def\XINT_dtb_small #1.%
278 {%
279 \expandafter\XINT_smallbin\expandafter
280 {\the\numexpr (#1+\xint_c_ii^vii)/\xint_c_ii^viii-\xint_c_i\expandafter}%
281 \romannumeral-'0\expandafter\XINT_smallbin\expandafter
282 {\the\numexpr
283 #1-((#1+\xint_c_ii^vii)/\xint_c_ii^viii-\xint_c_i)*\xint_c_ii^viii}%
284 }%
285 \def\XINT_dtb_end\Z\expandafter\XINT_dtb_III\expandafter #1#2\T
286 {%
287 \XINT_dtb_end_b #1%
288 }%
289 \def\XINT_dtb_end_b #1.{\XINT_dtb_end_c }%

```

```

290 \def\XINT_dtb_end_c #1#2#3#4#5#6#7#8%
291 {%
292   \expandafter\XINT_dtb_end_d\the\numexpr #1#2#3#4#5#6#7#8\relax
293 }%
294 \edef\XINT_dtb_end_d #1#2#3#4#5#6#7#8#9%
295 {%
296   \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7#8#9\relax
297 }%

```

5.5 \xintHexToDec

v1.08

```

298 \def\xintHexToDec {\romannumeral0\xinthextodec }%
299 \def\xinthextodec #1%
300   {\expandafter\XINT_htd_checkin\romannumeral-‘0#1\W\W\W\W \T }%
301 \def\XINT_htd_checkin #1%
302 {%
303   \xint_UDsignfork
304   #1\XINT_htd_neg
305   -{\XINT_htd_I {0000}#1}%
306   \krof
307 }%
308 \def\XINT_htd_neg {\expandafter\xint_minus_thenstop
309   \romannumeral0\XINT_htd_I {0000}}%
310 \def\XINT_htd_I #1#2#3#4#5%
311 {%
312   \xint_gob_til_W #5\XINT_htd_II_a\W
313   \XINT_htd_I_a  {}{"#2#3#4#5}#1\Z\Z\Z\Z
314 }%
315 \def\XINT_htd_II_a \W\XINT_htd_I_a #1#2{\XINT_htd_II_b #2}%
316 \def\XINT_htd_II_b "#1#2#3#4%
317 {%
318   \xint_gob_til_W
319   #1\XINT_htd_II_c
320   #2\XINT_htd_II_ci
321   #3\XINT_htd_II_cii
322   \W\XINT_htd_II_ciii #1#2#3#4%
323 }%
324 \def\XINT_htd_II_c \W\XINT_htd_II_ci
325   \W\XINT_htd_II_cii
326   \W\XINT_htd_II_ciii \W\W\W\W #1\Z\Z\Z\Z\T
327 {%
328   \expandafter\xint_cleanupzeros_andstop
329   \romannumeral0\XINT_rord_main {}#1%
330   \xint_relax
331   \xint_bye\xint_bye\xint_bye\xint_bye
332   \xint_bye\xint_bye\xint_bye\xint_bye
333   \xint_relax
334 }%
335 \def\XINT_htd_II_ci #1\XINT_htd_II_ciii
336   #2\W\W\W\W {\XINT_htd_II_d {}{"#2}{\xint_c_xvi}}%
337 \def\XINT_htd_II_cii\W\XINT_htd_II_ciii

```

5 Package *xintbinhex* implementation

```
338          #1#2\W\W {\XINT_htd_II_d {}{"#1#2}{\xint_c_ii^viii}}%
339 \def\XINT_htd_II_ciii #1#2#3\W {\XINT_htd_II_d {}{"#1#2#3}{\xint_c_ii^xii}}%
340 \def\XINT_htd_I_a #1#2#3#4#5#6%
341 {%
342   \xint_gob_til_Z #3\XINT_htd_I_end_a\Z
343   \expandafter\XINT_htd_I_b\the\numexpr
344   #2+\xint_c_ii^xvi*#6#5#4#3+\xint_c_x^ix\relax {#1}%
345 }%
346 \def\XINT_htd_I_b #1#2#3#4#5#6#7#8#9{\XINT_htd_I_c {#1#2#3#4#5}{#9#8#7#6}}%
347 \def\XINT_htd_I_c #1#2#3{\XINT_htd_I_a {#3#2}{#1}}%
348 \def\XINT_htd_I_end_a\Z\expandafter\XINT_htd_I_b\the\numexpr #1+#2\relax
349 {%
350   \expandafter\XINT_htd_I_end_b\the\numexpr \xint_c_x^v+#1\relax
351 }%
352 \def\XINT_htd_I_end_b #1#2#3#4#5%
353 {%
354   \xint_gob_til_zero #1\XINT_htd_I_end_bz0%
355   \XINT_htd_I_end_c #1#2#3#4#5%
356 }%
357 \def\XINT_htd_I_end_c #1#2#3#4#5#6{\XINT_htd_I {#6#5#4#3#2#1000}}%
358 \def\XINT_htd_I_end_bz0\XINT_htd_I_end_c 0#1#2#3#4%
359 {%
360   \xint_gob_til_zeros_iv #1#2#3#4\XINT_htd_I_end_bzz 0000%
361   \XINT_htd_I_end_D {#4#3#2#1}%
362 }%
363 \def\XINT_htd_I_end_D #1#2{\XINT_htd_I {#2#1}}%
364 \def\XINT_htd_I_end_bzz 0000\XINT_htd_I_end_D #1{\XINT_htd_I }%
365 \def\XINT_htd_II_d #1#2#3#4#5#6#7%
366 {%
367   \xint_gob_til_Z #4\XINT_htd_II_end_a\Z
368   \expandafter\XINT_htd_II_e\the\numexpr
369   #2+#3*#7#6#5#4+\xint_c_x^viii\relax {#1}{#3}%
370 }%
371 \def\XINT_htd_II_e #1#2#3#4#5#6#7#8{\XINT_htd_II_f {#1#2#3#4}{#5#6#7#8}}%
372 \def\XINT_htd_II_f #1#2#3{\XINT_htd_II_d {#2#3}{#1}}%
373 \def\XINT_htd_II_end_a\Z\expandafter\XINT_htd_II_e
374   \the\numexpr #1+#2\relax #3#4\T
375 {%
376   \XINT_htd_II_end_b #1#3%
377 }%
378 \edef\XINT_htd_II_end_b #1#2#3#4#5#6#7#8%
379 {%
380   \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7#8\relax
381 }%
```

5.6 *\xintBinToDec*

v1.08

```
382 \def\xintBinToDec {\romannumeral0\xintbintodec }%
383 \def\xintbintodec #1{\expandafter\XINT_btd_checkin
384   \romannumeral-'0#1\W\W\W\W\W\W\W\W \T }%
385 \def\XINT_btd_checkin #1%
```

5 Package *xintbinhex* implementation

```

386 {%
387   \xint_UDsignfork
388     #1\XINT_btd_neg
389     -{\XINT_btd_I {0000000}\#1}%
390   \krof
391 }%
392 \def\XINT_btd_neg {\expandafter\xint_minus_thenstop
393                   \romannumeral0\XINT_btd_I {0000000}}%
394 \def\XINT_btd_I #1#2#3#4#5#6#7#8#9%
395 {%
396   \xint_gob_til_W #9\XINT_btd_II_a {#2#3#4#5#6#7#8#9}\W
397   \XINT_btd_I_a {}{\csname XINT_sbtd_#2#3#4#5\endcsname*\xint_c_xvi+%
398               \csname XINT_sbtd_#6#7#8#9\endcsname}%
399   #1\Z\Z\Z\Z\Z\Z
400 }%
401 \def\XINT_btd_II_a #1\W\XINT_btd_I_a #2#3{\XINT_btd_II_b #1}%
402 \def\XINT_btd_II_b #1#2#3#4#5#6#7#8%
403 {%
404   \xint_gob_til_W
405     #1\XINT_btd_II_c
406     #2\XINT_btd_II_ci
407     #3\XINT_btd_II_cii
408     #4\XINT_btd_II_ciii
409     #5\XINT_btd_II_civ
410     #6\XINT_btd_II_cv
411     #7\XINT_btd_II_cvi
412     \W\XINT_btd_II_cvii #1#2#3#4#5#6#7#8%
413 }%
414 \def\XINT_btd_II_c #1\XINT_btd_II_cvii \W\W\W\W\W\W\W #2\Z\Z\Z\Z\Z\Z\T
415 {%
416   \expandafter\XINT_btd_II_c_end
417   \romannumeral0\XINT_rord_main {}#2%
418   \xint_relax
419     \xint_bye\xint_bye\xint_bye\xint_bye
420     \xint_bye\xint_bye\xint_bye\xint_bye
421   \xint_relax
422 }%
423 \edef\XINT_btd_II_c_end #1#2#3#4#5#6%
424 {%
425   \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6\relax
426 }%
427 \def\XINT_btd_II_ci #1\XINT_btd_II_cvii #2\W\W\W\W\W\W\W
428   {\XINT_btd_II_d {}{#2}{\xint_c_ii }}%
429 \def\XINT_btd_II_cii #1\XINT_btd_II_cvii #2\W\W\W\W\W\W\W
430   {\XINT_btd_II_d {}{\csname XINT_sbtd_00#2\endcsname }{\xint_c_iv }}%
431 \def\XINT_btd_II_ciii #1\XINT_btd_II_cvii #2\W\W\W\W\W\W\W
432   {\XINT_btd_II_d {}{\csname XINT_sbtd_0#2\endcsname }{\xint_c_viii }}%
433 \def\XINT_btd_II_civ #1\XINT_btd_II_cvii #2\W\W\W\W\W\W\W
434   {\XINT_btd_II_d {}{\csname XINT_sbtd_#2\endcsname }{\xint_c_xvi }}%
435 \def\XINT_btd_II_cv #1\XINT_btd_II_cvii #2#3#4#5#6\W\W\W\W
436 {%
437   \XINT_btd_II_d {}{\csname XINT_sbtd_#2#3#4#5\endcsname*\xint_c_ii+%

```

5 Package *xintbinhex* implementation

```

438             #6}{\xint_c_ii^v }%
439 }%
440 \def\xINT_btd_II_cvi #1\xINT_btd_II_cvii #2#3#4#5#6#7\W\W
441 {%
442     \XINT_btd_II_d {}{\csname XINT_sbtd_#2#3#4#5\endcsname*\xint_c_iv+%
443         \csname XINT_sbtd_00#6#7\endcsname}{\xint_c_ii^vi }%
444 }%
445 \def\xINT_btd_II_cvii #1#2#3#4#5#6#7\W
446 {%
447     \XINT_btd_II_d {}{\csname XINT_sbtd_#1#2#3#4\endcsname*\xint_c_viii+%
448         \csname XINT_sbtd_0#5#6#7\endcsname}{\xint_c_ii^vii }%
449 }%
450 \def\xINT_btd_II_d #1#2#3#4#5#6#7#8#9%
451 {%
452     \xint_gob_til_Z #4\xINT_btd_II_end_a\Z
453     \expandafter\xINT_btd_II_e\the\numexpr
454     #2+(\xint_c_x^ix+#3*#9#8#7#6#5#4)\relax {#1}{#3}%
455 }%
456 \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}}%
457 \def\xINT_btd_II_f #1#2#3{\xINT_btd_II_d {#2#3}{#1}}%
458 \def\xINT_btd_II_end_a\Z\expandafter\xINT_btd_II_e
459     \the\numexpr #1+(\xint_c_ii^viii*#8#7#6#5#4#3+\xint_c_x^ix)\relax {#1}%
460 {%
461     \XINT_btd_II_end_b #1#3%
462 }%
463 \edef\xINT_btd_II_end_b #1#2#3#4#5#6#7#8#9%
464 {%
465     \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7#8#9\relax
466 }%
467 \def\xINT_btd_I_a #1#2#3#4#5#6#7#8%
468 {%
469     \xint_gob_til_Z #3\xINT_btd_I_end_a\Z
470     \expandafter\xINT_btd_I_b\the\numexpr
471     #2+\xint_c_ii^viii*#8#7#6#5#4#3+\xint_c_x^ix\relax {#1}%
472 }%
473 \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}}%
474 \def\xINT_btd_I_c #1#2#3{\xINT_btd_I_a {#3#2}{#1}}%
475 \def\xINT_btd_I_end_a\Z\expandafter\xINT_btd_I_b
476     \the\numexpr #1+\xint_c_ii^viii #2\relax
477 {%
478     \expandafter\xINT_btd_I_end_b\the\numexpr 1000+#1\relax
479 }%
480 \def\xINT_btd_I_end_b 1#1#2#3%
481 {%
482     \xint_gob_til_zeros_iii #1#2#3\xINT_btd_I_end_bz 000%
483     \XINT_btd_I_end_c #1#2#3%
484 }%
485 \def\xINT_btd_I_end_c #1#2#3#4{\xINT_btd_I {#4#3#2#1000}}%
486 \def\xINT_btd_I_end_bz 000\xINT_btd_I_end_c 000{\xINT_btd_I }%

```

5.7 `\xintBinToHex`

v1.08

```

487 \def\xintBinToHex {\romannumeral0\xintbinto hex }%
488 \def\xintbinto hex #1%
489 {%
490   \expandafter\XINT_bth_checkin
491           \romannumeral0\expandafter\XINT_num_loop
492           \romannumeral-'0#1\xint_relax\xint_relax
493           \xint_relax\xint_relax
494           \xint_relax\xint_relax\xint_relax\xint_relax\Z
495   \R\R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W\W
496 }%
497 \def\XINT_bth_checkin #1%
498 {%
499   \xint_UDsignfork
500   #1\XINT_bth_N
501   -{\XINT_bth_P #1}%
502   \krof
503 }%
504 \def\XINT_bth_N {\expandafter\xint_minus_thenstop\romannumeral0\XINT_bth_P }%
505 \def\XINT_bth_P {\expandafter\XINT_bth_I\expandafter{\expandafter}%
506               \romannumeral0\XINT_OQ {}}%
507 \def\XINT_bth_I #1#2#3#4#5#6#7#8#9%
508 {%
509   \xint_gob_til_W #9\XINT_bth_end_a\W
510   \expandafter\expandafter\expandafter
511   \XINT_bth_I
512   \expandafter\expandafter\expandafter
513   {\csname XINT_sbth_#9#8#7#6\expandafter\expandafter\expandafter\endcsname
514   \csname XINT_sbth_#5#4#3#2\endcsname #1}%
515 }%
516 \def\XINT_bth_end_a\W \expandafter\expandafter\expandafter
517   \XINT_bth_I \expandafter\expandafter\expandafter #1%
518 {%
519   \XINT_bth_end_b #1%
520 }%
521 \def\XINT_bth_end_b #1\endcsname #2\endcsname #3%
522 {%
523   \xint_gob_til_zero #3\XINT_bth_end_z 0\space #3%
524 }%
525 \def\XINT_bth_end_z0\space 0{ }%

```

5.8 `\xintHexToBin`

v1.08

```

526 \def\xintHexToBin {\romannumeral0\xinthextobin }%
527 \def\xinthextobin #1%
528 {%
529   \expandafter\XINT_htb_checkin\romannumeral-'0#1GGGGGGGG\T
530 }%

```



```

531 \def\XINT_htb_checkin #1%
532 {%
533     \xint_UDsignfork
534     #1\XINT_htb_N
535     -{\XINT_htb_P #1}%
536     \krof
537 }%
538 \def\XINT_htb_N {\expandafter\xint_minus_thenstop\romannumeral0\XINT_htb_P }%
539 \def\XINT_htb_P {\XINT_htb_I_a {}}%
540 \def\XINT_htb_I_a #1#2#3#4#5#6#7#8#9%
541 {%
542     \xint_gob_til_G #9\XINT_htb_II_a G%
543     \expandafter\expandafter\expandafter
544     \XINT_htb_I_b
545     \expandafter\expandafter\expandafter
546     {\csname XINT_shtb_#2\expandafter\expandafter\expandafter\endcsname
547     \csname XINT_shtb_#3\expandafter\expandafter\expandafter\endcsname
548     \csname XINT_shtb_#4\expandafter\expandafter\expandafter\endcsname
549     \csname XINT_shtb_#5\expandafter\expandafter\expandafter\endcsname
550     \csname XINT_shtb_#6\expandafter\expandafter\expandafter\endcsname
551     \csname XINT_shtb_#7\expandafter\expandafter\expandafter\endcsname
552     \csname XINT_shtb_#8\expandafter\expandafter\expandafter\endcsname
553     \csname XINT_shtb_#9\endcsname }{#1}%
554 }%
555 \def\XINT_htb_I_b #1#2{\XINT_htb_I_a {#2#1}}%
556 \def\XINT_htb_II_a G\expandafter\expandafter\expandafter\XINT_htb_I_b
557 {%
558     \expandafter\expandafter\expandafter \XINT_htb_II_b
559 }%
560 \def\XINT_htb_II_b #1#2#3\T
561 {%
562     \XINT_num_loop #2#1%
563     \xint_relax\xint_relax\xint_relax\xint_relax
564     \xint_relax\xint_relax\xint_relax\xint_relax\Z
565 }%

```

5.9 \xintCHexToBin

v1.08

```

566 \def\xintCHexToBin {\romannumeral0\xintchextobin }%
567 \def\xintchextobin #1%
568 {%
569     \expandafter\XINT_chtb_checkin\romannumeral-'0#1%
570     \R\R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W\W
571 }%
572 \def\XINT_chtb_checkin #1%
573 {%
574     \xint_UDsignfork
575     #1\XINT_chtb_N
576     -{\XINT_chtb_P #1}%
577     \krof
578 }%

```

```

579 \def\XINT_chtb_N {\expandafter\xint_minus_thenstop\romannumeral0\XINT_chtb_P }%
580 \def\XINT_chtb_P {\expandafter\XINT_chtb_I\expandafter{\expandafter}%
581 \romannumeral0\XINT_OQ {}}%
582 \def\XINT_chtb_I #1#2#3#4#5#6#7#8#9%
583 {%
584 \xint_gob_til_W #9\XINT_chtb_end_a\W
585 \expandafter\expandafter\expandafter
586 \XINT_chtb_I
587 \expandafter\expandafter\expandafter
588 {\csname XINT_shtb_#9\expandafter\expandafter\expandafter\endcsname
589 \csname XINT_shtb_#8\expandafter\expandafter\expandafter\endcsname
590 \csname XINT_shtb_#7\expandafter\expandafter\expandafter\endcsname
591 \csname XINT_shtb_#6\expandafter\expandafter\expandafter\endcsname
592 \csname XINT_shtb_#5\expandafter\expandafter\expandafter\endcsname
593 \csname XINT_shtb_#4\expandafter\expandafter\expandafter\endcsname
594 \csname XINT_shtb_#3\expandafter\expandafter\expandafter\endcsname
595 \csname XINT_shtb_#2\endcsname
596 #1}%
597 }%
598 \def\XINT_chtb_end_a\W\expandafter\expandafter\expandafter
599 \XINT_chtb_I\expandafter\expandafter\expandafter #1%
600 {%
601 \XINT_chtb_end_b #1%
602 \xint_relax\xint_relax\xint_relax\xint_relax
603 \xint_relax\xint_relax\xint_relax\xint_relax\Z
604 }%
605 \def\XINT_chtb_end_b #1\W#2\W#3\W#4\W#5\W#6\W#7\W#8\W\endcsname
606 {%
607 \XINT_num_loop
608 }%
609 \XINT_restorecatcodes_endinput%

```

6 Package *xintgcd* implementation

.1	Catcodes, ε - \TeX and reload detection	130	.7	<code>\xintBezoutAlgorithm</code>	138
.2	Package identification	131	.8	<code>\xintGCDof</code>	140
.3	<code>\xintGCD</code>	131	.9	<code>\xintLCMof</code>	140
.4	<code>\xintLCM</code>	132	.10	<code>\xintTypesetEuclideanAlgorithm</code>	140
.5	<code>\xintBezout</code>	133	.11	<code>\xintTypesetBezoutAlgorithm</code>	141
.6	<code>\xintEuclideanAlgorithm</code>	136			

The commenting is currently (2014/10/28) very sparse. Release 1.09h has modified a bit the `\xintTypesetEuclideanAlgorithm` 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`.

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.

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2 \catcode13=5 % ^^M
3 \endlinechar=13 %

```

```

4 \catcode123=1 % {
5 \catcode125=2 % }
6 \catcode64=11 % @
7 \catcode35=6 % #
8 \catcode44=12 % ,
9 \catcode45=12 % -
10 \catcode46=12 % .
11 \catcode58=12 % :
12 \let\z\endgroup
13 \expandafter\let\expandafter\x\csname ver@xintgcd.sty\endcsname
14 \expandafter\let\expandafter\w\csname ver@xintcore.sty\endcsname
15 \expandafter
16 \ifx\csname PackageInfo\endcsname\relax
17 \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
18 \else
19 \def\y#1#2{\PackageInfo{#1}{#2}}%
20 \fi
21 \expandafter
22 \ifx\csname numexpr\endcsname\relax
23 \y{xintgcd}{numexpr not available, aborting input}%
24 \aftergroup\endinput
25 \else
26 \ifx\x\relax % plain-TeX, first loading of xintgcd.sty
27 \ifx\w\relax % but xintcore.sty not yet loaded.
28 \def\z{\endgroup\input xintcore.sty\relax}%
29 \fi
30 \else
31 \def\empty {}%
32 \ifx\x\empty % LaTeX, first loading,
33 % variable is initialized, but \ProvidesPackage not yet seen
34 \ifx\w\relax % xintcore.sty not yet loaded.
35 \def\z{\endgroup\RequirePackage{xintcore}}%
36 \fi
37 \else
38 \aftergroup\endinput % xintgcd already loaded.
39 \fi
40 \fi
41 \fi
42 \z%
43 \XINTsetupcatcodes% defined in xintkernel.sty

```

6.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintgcd}%
46 [2014/10/28 v1.1 Euclidean algorithm with xint package (jfb)]%

```

6.3 *\xintGCD*

The macros of 1.09a benefits from the *\xintnum* which has been inserted inside *\xintiabs* in *xint*; this is a little overhead but is more convenient for the user and also makes it easier to use into *\xintexpressions*.

```

47 \def\xintGCD {\romannumeral0\xintgcd }%

```

```

48 \def\xintgcd #1%
49 {%
50   \expandafter\XINT_gcd\expandafter{\romannumeral0\xintiabs {#1}}%
51 }%
52 \def\XINT_gcd #1#2%
53 {%
54   \expandafter\XINT_gcd_fork\romannumeral0\xintiabs {#2}\Z #1\Z
55 }%

```

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

```

56 \def\XINT_gcd_fork #1#2\Z #3#4\Z
57 {%
58   \xint_UDzerofork
59     #1\XINT_gcd_BisZero
60     #3\XINT_gcd_AisZero
61     0\XINT_gcd_loop
62   \krof
63   {#1#2}{#3#4}%
64 }%
65 \def\XINT_gcd_AisZero #1#2{ #1}%
66 \def\XINT_gcd_BisZero #1#2{ #2}%
67 \def\XINT_gcd_CheckRem #1#2\Z
68 {%
69   \xint_gob_til_zero #1\xint_gcd_end0\XINT_gcd_loop {#1#2}%
70 }%
71 \def\xint_gcd_end0\XINT_gcd_loop #1#2{ #2}%

```

#1=B, #2=A

```

72 \def\XINT_gcd_loop #1#2%
73 {%
74   \expandafter\expandafter\expandafter
75     \XINT_gcd_CheckRem
76   \expandafter\xint_seconddoftwo
77   \romannumeral0\XINT_div_prepare {#1}{#2}\Z
78   {#1}%
79 }%

```

6.4 \xintLCM

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.

```

80 \def\xintLCM {\romannumeral0\xintlcm}%
81 \def\xintlcm #1%
82 {%
83   \expandafter\XINT_lcm\expandafter{\romannumeral0\xintiabs {#1}}%
84 }%
85 \def\XINT_lcm #1#2%
86 {%
87   \expandafter\XINT_lcm_fork\romannumeral0\xintiabs {#2}\Z #1\Z
88 }%
89 \def\XINT_lcm_fork #1#2\Z #3#4\Z

```

```

90 {%
91   \xint_UDzerofork
92   #1\XINT_lcm_BisZero
93   #3\XINT_lcm_AisZero
94   0\expandafter
95   \krof
96   \XINT_lcm_notzero\expandafter{\romannumeral0\XINT_gcd_loop {#1#2}{#3#4}}%
97   {#1#2}{#3#4}%
98 }%
99 \def\XINT_lcm_AisZero #1#2#3#4#5{ 0}%
100 \def\XINT_lcm_BisZero #1#2#3#4#5{ 0}%
101 \def\XINT_lcm_notzero #1#2#3{\xintiimul {#2}{\xintiiQuo{#3}{#1}}}%

```

6.5 \xintBezout

1.09a inserts use of \xintnum

```

102 \def\xintBezout {\romannumeral0\xintbezout }%
103 \def\xintbezout #1%
104 {%
105   \expandafter\xint_bezout\expandafter {\romannumeral0\xintnum{#1}}%
106 }%
107 \def\xint_bezout #1#2%
108 {%
109   \expandafter\XINT_bezout_fork \romannumeral0\xintnum{#2}\Z #1\Z
110 }%

```

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

```

111 \def\XINT_bezout_fork #1#2\Z #3#4\Z
112 {%
113   \xint_UDzerosfork
114   #1#3\XINT_bezout_botharezero
115   #10\XINT_bezout_secondiszero
116   #30\XINT_bezout_firstiszero
117   00{\xint_UDsignsfork
118     #1#3\XINT_bezout_minusminus % A < 0, B < 0
119     #1-\XINT_bezout_minusplus % A > 0, B < 0
120     #3-\XINT_bezout_plusminus % A < 0, B > 0
121     --\XINT_bezout_plusplus % A > 0, B > 0
122   \krof }%
123   \krof
124   {#2}{#4}#1#3{#3#4}{#1#2}% #1#2=B, #3#4=A
125 }%
126 \edef\XINT_bezout_botharezero #1#2#3#4#5#6%
127 {%
128   \noexpand\xintError:NoBezoutForZeros\space {0}{0}{0}{0}{0}%
129 }%

```

attention première entrée doit être ici (-1)^n donc 1

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

```

130 \def\XINT_bezout_firstiszero #1#2#3#4#5#6%
131 {%

```

```

132 \xint_UDsignfork
133   #3{ {0}{#3#1}{0}{1}{#1}}%
134   -{ {0}{#3#1}{0}{-1}{#1}}%
135 \krof
136 }%

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

137 \def\XINT_bezout_secondiszero #1#2#3#4#5#6%
138 {%
139   \xint_UDsignfork
140   #4{ {#4#2}{0}{-1}{0}{#2}}%
141   -{ {#4#2}{0}{1}{0}{#2}}%
142 \krof
143 }%

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

144 \def\XINT_bezout_minusminus #1#2#3#4%
145 {%
146   \expandafter\XINT_bezout_mm_post
147   \romannumeral0\XINT_bezout_loop_a 1{#1}{#2}1001%
148 }%
149 \def\XINT_bezout_mm_post #1#2%
150 {%
151   \expandafter\XINT_bezout_mm_postb\expandafter
152   {\romannumeral0\xintiopp{#2}}{\romannumeral0\xintiopp{#1}}%
153 }%
154 \def\XINT_bezout_mm_postb #1#2%
155 {%
156   \expandafter\XINT_bezout_mm_postc\expandafter {#2}{#1}%
157 }%
158 \edef\XINT_bezout_mm_postc #1#2#3#4#5%
159 {%
160   \space {#4}{#5}{#1}{#2}{#3}%
161 }%

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

162 \def\XINT_bezout_minusplus #1#2#3#4%
163 {%
164   \expandafter\XINT_bezout_mp_post
165   \romannumeral0\XINT_bezout_loop_a 1{#1}{#4#2}1001%
166 }%
167 \def\XINT_bezout_mp_post #1#2%
168 {%
169   \expandafter\XINT_bezout_mp_postb\expandafter
170   {\romannumeral0\xintiopp {#2}}{#1}%
171 }%
172 \edef\XINT_bezout_mp_postb #1#2#3#4#5%
173 {%
174   \space {#4}{#5}{#2}{#1}{#3}%
175 }%

```

plusminus $A < 0, B > 0$

```

176 \def\XINT_bezout_plusminus #1#2#3#4%
177 {%
178   \expandafter\XINT_bezout_pm_post
179   \romannumeral0\XINT_bezout_loop_a 1{#3#1}{#2}1001%
180 }%
181 \def\XINT_bezout_pm_post #1%
182 {%
183   \expandafter \XINT_bezout_pm_postb \expandafter
184   {\romannumeral0\xintiio{#1}}%
185 }%
186 \edef\XINT_bezout_pm_postb #1#2#3#4#5%
187 {%
188   \space {#4}{#5}{#1}{#2}{#3}%
189 }%

```

plusplus

```

190 \def\XINT_bezout_plusplus #1#2#3#4%
191 {%
192   \expandafter\XINT_bezout_pp_post
193   \romannumeral0\XINT_bezout_loop_a 1{#3#1}{#4#2}1001%
194 }%

```

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

```

195 \edef\XINT_bezout_pp_post #1#2#3#4#5%
196 {%
197   \space {#4}{#5}{#1}{#2}{#3}%
198 }%

```

$n = 0$: $1BA\alpha(0)\beta(0)\alpha(-1)\beta(-1)$

n général: $\{(-1)^n\}\{r(n)\}\{r(n-1)\}\{\alpha(n-1)\}\{\beta(n-1)\}\{\alpha(n-2)\}\{\beta(n-2)\}$

$\#2 = B, \#3 = A$

```

199 \def\XINT_bezout_loop_a #1#2#3%
200 {%
201   \expandafter\XINT_bezout_loop_b
202   \expandafter{\the\numexpr -#1\expandafter}%
203   \romannumeral0\XINT_div_prepare {#2}{#3}{#2}%
204 }%

```

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)

$\{-(-1)^n\}\{q(n)\}\{r(n)\}\{r(n-1)\}\{\alpha(n-1)\}\{\beta(n-1)\}\{\alpha(n-2)\}\{\beta(n-2)\}$

```

205 \def\XINT_bezout_loop_b #1#2#3#4#5#6#7#8%
206 {%
207   \expandafter \XINT_bezout_loop_c \expandafter
208   {\romannumeral0\xintiioadd{\XINT_Mul{#5}{#2}}{#7}}%
209   {\romannumeral0\xintiioadd{\XINT_Mul{#6}{#2}}{#8}}%
210   {#1}{#3}{#4}{#5}{#6}%
211 }%

```

```

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

212 \def\XINT_bezout_loop_c #1#2%
213 {%
214   \expandafter \XINT_bezout_loop_d \expandafter
215     {#2}{#1}%
216 }%

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

217 \def\XINT_bezout_loop_d #1#2#3#4#5%
218 {%
219   \XINT_bezout_loop_e #4\Z {#3}{#5}{#2}{#1}%
220 }%

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

221 \def\XINT_bezout_loop_e #1#2\Z
222 {%
223   \xint_gob_til_zero #1\xint_bezout_loop_exit0\XINT_bezout_loop_f
224   {#1#2}%
225 }%

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

226 \def\XINT_bezout_loop_f #1#2%
227 {%
228   \XINT_bezout_loop_a {#2}{#1}%
229 }%

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

230 \def\xint_bezout_loop_exit0\XINT_bezout_loop_f #1#2%
231 {%
232   \ifcase #2
233   \or \expandafter\XINT_bezout_exiteven
234   \else\expandafter\XINT_bezout_exitodd
235   \fi
236 }%
237 \edef\XINT_bezout_exiteven #1#2#3#4#5%
238 {%
239   \space {#5}{#4}{#1}%
240 }%
241 \edef\XINT_bezout_exitodd #1#2#3#4#5%
242 {%
243   \space {-#5}{-#4}{#1}%
244 }%

```

6.6 \xintEuclideanAlgorithm

Pour Euclide: $\{N\}\{A\}\{D=r(n)\}\{B\}\{q_1\}\{r_1\}\{q_2\}\{r_2\}\{q_3\}\{r_3\}\dots\{q_N\}\{r_N=0\}$
 $u_{<2n>} = u_{<2n+3>}u_{<2n+2>} + u_{<2n+4>}$ à la n ième étape

```

245 \def\xintEuclideanAlgorithm {\romannumeral0\xinteucclideanalgorithm }%
246 \def\xinteucclideanalgorithm #1%

```



```

247 {%
248   \expandafter \XINT_euc \expandafter{\romannumeral0\xintiabs {#1}}%
249 }%
250 \def\XINT_euc #1#2%
251 {%
252   \expandafter\XINT_euc_fork \romannumeral0\xintiabs {#2}\Z #1\Z
253 }%

```

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

```

254 \def\XINT_euc_fork #1#2\Z #3#4\Z
255 {%
256   \xint_UDzerofork
257   #1\XINT_euc_BisZero
258   #3\XINT_euc_AisZero
259   0\XINT_euc_a
260   \krof
261   {0}{#1#2}{#3#4}{#3#4}{#1#2}}\Z
262 }%

```

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}

```

263 \def\XINT_euc_AisZero #1#2#3#4#5#6{ {1}{0}{#2}{#2}{0}{0}}%
264 \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

```

265 \def\XINT_euc_a #1#2#3%
266 {%
267   \expandafter\XINT_euc_b
268   \expandafter {\the\numexpr #1+1\expandafter }%
269   \romannumeral0\XINT_div_prepare {#2}{#3}{#2}%
270 }%

```

{n+1}{q(n+1)}{r(n+1)}{rn}{{qn}{rn}}...

```

271 \def\XINT_euc_b #1#2#3#4%
272 {%
273   \XINT_euc_c #3\Z {#1}{#3}{#4}{{#2}{#3}}%
274 }%

```

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.

```

275 \def\XINT_euc_c #1#2\Z
276 {%
277   \xint_gob_til_zero #1\xint_euc_end0\XINT_euc_a
278 }%

```

{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}

```

279 \def\xint_euc_end0\XINT_euc_a #1#2#3#4\Z%
280 {%
281   \expandafter\xint_euc_end_
282   \romannumeral0%
283   \XINT_rord_main {}#4{{#1}{{#3}}}%
284   \xint_relax
285   \xint_bye\xint_bye\xint_bye\xint_bye
286   \xint_bye\xint_bye\xint_bye\xint_bye
287   \xint_relax
288 }%
289 \edef\xint_euc_end_ #1#2#3%
290 {%
291   \space {{#1}{{#3}}#2}%
292 }%

```

6.7 \xintBezoutAlgorithm

Pour Bezout: objectif, renvoyer

$\{N\}\{A\}\{0\}\{1\}\{D=r(n)\}\{B\}\{1\}\{0\}\{q_1\}\{r_1\}\{\alpha_1=q_1\}\{\beta_1=1\}$
 $\{q_2\}\{r_2\}\{\alpha_2\}\{\beta_2\}\dots\{q_N\}\{r_N=0\}\{\alpha_N=A/D\}\{\beta_N=B/D\}$
 $\alpha_0=1, \beta_0=0, \alpha(-1)=0, \beta(-1)=1$

```

293 \def\xintBezoutAlgorithm {\romannumeral0\xintbezoutalgorithm }%
294 \def\xintbezoutalgorithm #1%
295 {%
296   \expandafter \XINT_bezalg \expandafter{\romannumeral0\xintiabs {#1}}%
297 }%
298 \def\XINT_bezalg #1#2%
299 {%
300   \expandafter\XINT_bezalg_fork \romannumeral0\xintiabs {#2}\Z #1\Z
301 }%

```

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

```

302 \def\XINT_bezalg_fork #1#2\Z #3#4\Z
303 {%
304   \xint_UDzerofork
305   #1\XINT_bezalg_BisZero
306   #3\XINT_bezalg_AisZero
307   0\XINT_bezalg_a
308   \krof
309   0{{#1#2}{{#3#4}}1001{{#3#4}{{#1#2}}}}\Z
310 }%
311 \def\XINT_bezalg_AisZero #1#2#3\Z{ {1}{0}{0}{1}{{#2}{{#2}}{1}{0}{0}{0}{0}{1}}}%
312 \def\XINT_bezalg_BisZero #1#2#3#4\Z{ {1}{0}{0}{1}{{#3}{{#3}}{1}{0}{0}{0}{0}{1}}}%

```

pour préparer l'étape n+1 il faut $\{n\}\{r(n)\}\{r(n-1)\}\{\alpha(n)\}\{\beta(n)\}\{\alpha(n-1)\}\{\beta(n-1)\}\{q(n)\}\{r(n)\}\{\alpha(n-1)\}\{\beta(n-1)\}$
division de #3 par #2

```

313 \def\XINT_bezalg_a #1#2#3%
314 {%
315   \expandafter\XINT_bezalg_b
316   \expandafter {\the\numexpr #1+1\expandafter }%
317   \romannumeral0\XINT_div_prepare {#2}{{#3}}#2}%
318 }%

```

```

{#n+1}{#q(n+1)}{#r(n+1)}{#r(n)}{#alpha(n)}{#beta(n)}{#alpha(n-1)}{#beta(n-1)}...

319 \def\XINT_bezalg_b #1#2#3#4#5#6#7#8%
320 {%
321   \expandafter\XINT_bezalg_c\expandafter
322   {\romannumeral0\xintiiadd {\xintiiMul {#6}{#2}}{#8}}%
323   {\romannumeral0\xintiiadd {\xintiiMul {#5}{#2}}{#7}}%
324   {#1}{#2}{#3}{#4}{#5}{#6}%
325 }%

{#beta(n+1)}{#alpha(n+1)}{#n+1}{#q(n+1)}{#r(n+1)}{#r(n)}{#alpha(n)}{#beta(n)}

326 \def\XINT_bezalg_c #1#2#3#4#5#6%
327 {%
328   \expandafter\XINT_bezalg_d\expandafter {#2}{#3}{#4}{#5}{#6}{#1}%
329 }%

{#alpha(n+1)}{#n+1}{#q(n+1)}{#r(n+1)}{#r(n)}{#beta(n+1)}

330 \def\XINT_bezalg_d #1#2#3#4#5#6#7#8%
331 {%
332   \XINT_bezalg_e #4\Z {#2}{#4}{#5}{#1}{#6}{#7}{#8}{{#3}{#4}{#1}{#6}}%
333 }%

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.

334 \def\XINT_bezalg_e #1#2\Z
335 {%
336   \xint_gob_til_zero #1\xint_bezalg_end0\XINT_bezalg_a
337 }%

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}

338 \def\xint_bezalg_end0\XINT_bezalg_a #1#2#3#4#5#6#7#8\Z
339 {%
340   \expandafter\xint_bezalg_end_
341   \romannumeral0%
342   \XINT_rord_main {}#8{{#1}{#3}}%
343   \xint_relax
344   \xint_bye\xint_bye\xint_bye\xint_bye
345   \xint_bye\xint_bye\xint_bye\xint_bye
346   \xint_relax
347 }%

{#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}
```

```
348 \edef\xint_bezalg_end_ #1#2#3#4%
349 {%
350   \space {#1}{#3}{0}{1}{#2}{#4}{1}{0}%
351 }%
```

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 ...

```
352 \def\xintGCDof      {\romannumeral0\xintgcdof }%
353 \def\xintgcdof      #1{\expandafter\xintgcdof_a\romannumeral-‘0#1\relax }%
354 \def\xintgcdof_a    #1{\expandafter\xintgcdof_b\romannumeral-‘0#1\Z }%
355 \def\xintgcdof_b    #1\Z #2{\expandafter\xintgcdof_c\romannumeral-‘0#2\Z {#1}\Z}%
356 \def\xintgcdof_c    #1{\xint_gob_til_relax #1\xintgcdof_e\relax\xintgcdof_d #1}%
357 \def\xintgcdof_d    #1\Z {\expandafter\xintgcdof_b\romannumeral0\xintgcd {#1}}%
358 \def\xintgcdof_e    #1\Z #2\Z { #2}%
```

6.9 \xintLCMof

New with 1.09a

```
359 \def\xintLCMof      {\romannumeral0\xintlcmof }%
360 \def\xintlcmof      #1{\expandafter\xintlcmof_a\romannumeral-‘0#1\relax }%
361 \def\xintlcmof_a    #1{\expandafter\xintlcmof_b\romannumeral-‘0#1\Z }%
362 \def\xintlcmof_b    #1\Z #2{\expandafter\xintlcmof_c\romannumeral-‘0#2\Z {#1}\Z}%
363 \def\xintlcmof_c    #1{\xint_gob_til_relax #1\xintlcmof_e\relax\xintlcmof_d #1}%
364 \def\xintlcmof_d    #1\Z {\expandafter\xintlcmof_b\romannumeral0\xintlcm {#1}}%
365 \def\xintlcmof_e    #1\Z #2\Z { #2}%
```

6.10 \xintTypesetEuclideanAlgorithm

TYPESETTING

Organisation:

$\{N\}\{A\}\{D\}\{B\}\{q_1\}\{r_1\}\{q_2\}\{r_2\}\{q_3\}\{r_3\}...\{q_N\}\{r_N=0\}$

$\backslash U1 = N = \text{nombre d'étapes}$, $\backslash U3 = \text{PGCD}$, $\backslash U2 = A$, $\backslash U4=B$ $q_1 = \backslash U5$, $q_2 = \backslash U7 \rightarrow q_n = \backslash U<2n+3>$, $r_n = \backslash U<2n+4>$ $bn = rn$. $B = r_0$. $A=r(-1)$

$r(n-2) = q(n)r(n-1)+r(n)$ (n e étape)

$\backslash U\{2n\} = \backslash U\{2n+3\} \times \backslash U\{2n+2\} + \backslash U\{2n+4\}$, n e étape. (avec n entre 1 et N)

1.09h uses `\xintloop`, and `\par` rather than `\endgraf`; and `\par` rather than `\hfill\break`

```
366 \def\xintTypesetEuclideanAlgorithm {%
367   \unless\ifdefined\xintAssignArray
368     \errmessage
369     {xintgcd: package xinttools is required for \string\xintTypesetEuclideanAlgorithm}%
370   \expandafter\xint_gobble_iii
371   \fi
372   \XINT_TypesetEuclideanAlgorithm
373 }%
374 \def\XINT_TypesetEuclideanAlgorithm #1#2%
```

```

375 {% l'algo remplace #1 et #2 par |#1| et |#2|
376   \par
377   \begingroup
378     \xintAssignArray\xintEuclideanAlgorithm {#1}{#2}\to\U
379     \edef\A{\U2}\edef\B{\U4}\edef\N{\U1}%
380     \setbox 0 \vbox{\halign {$##$\cr \A\cr \B \cr}}%
381     \count 255 1
382     \xintloop
383       \indent\hbox to \wd 0 {\hfil$\U{\numexpr 2*\count255\relax}$}%
384       ${} = \U{\numexpr 2*\count255 + 3\relax}
385       \times \U{\numexpr 2*\count255 + 2\relax}
386       + \U{\numexpr 2*\count255 + 4\relax}$%
387     \ifnum \count255 < \N
388       \par
389       \advance \count255 1
390     \repeat
391   \endgroup
392 }%

```

6.11 \xintTypesetBezoutAlgorithm

Pour Bezout on a: $\{N\}\{A\}\{0\}\{1\}\{D=r(n)\}\{B\}\{1\}\{0\}\{q_1\}\{r_1\}\{\alpha_1=q_1\}\{\beta_1=1\}$
 $\{q_2\}\{r_2\}\{\alpha_2\}\{\beta_2\}\dots\{q_N\}\{r_N=0\}\{\alpha_N=A/D\}\{\beta_N=B/D\}$ Donc $4n+8$ termes: $U_1 = N$, $U_2 = A$,
 $U_5 = D$, $U_6 = B$, $q_1 = U_9$, $q_n = U_{4n+5}$, n au moins 1
 $r_n = U_{4n+6}$, 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

```

393 \def\xintTypesetBezoutAlgorithm {%
394   \unless\ifdefined\xintAssignArray
395     \errmessage
396     {xintgcd: package xinttools is required for \string\xintTypesetBezoutAlgorithm}%
397     \expandafter\xint_gobble_iii
398   \fi
399   \XINT_TypesetBezoutAlgorithm
400 }%
401 \def\XINT_TypesetBezoutAlgorithm #1#2%
402 {%
403   \par
404   \begingroup
405     \xintAssignArray\xintBezoutAlgorithm {#1}{#2}\to\BEZ
406     \edef\A{\BEZ2}\edef\B{\BEZ6}\edef\N{\BEZ1}% A = |#1|, B = |#2|
407     \setbox 0 \vbox{\halign {$##$\cr \A\cr \B \cr}}%
408     \count255 1
409     \xintloop
410       \indent\hbox to \wd 0 {\hfil$\BEZ{4*\count255 - 2}$}%
411       ${} = \BEZ{4*\count255 + 5}
412       \times \BEZ{4*\count255 + 2}
413       + \BEZ{4*\count255 + 6}$\hfill\break
414     \hbox to \wd 0 {\hfil$\BEZ{4*\count255 + 7}$}%
415     ${} = \BEZ{4*\count255 + 5}
416     \times \BEZ{4*\count255 + 3}

```

```

417      + \BEZ{4*\count255 - 1}$\hfill\break
418      \hbox to \wd 0 {\hfil$\BEZ{4*\count255 +8}$}%
419      ${} = \BEZ{4*\count255 + 5}
420      \times \BEZ{4*\count255 + 4}
421      + \BEZ{4*\count255 }$
422      \par
423      \ifnum \count255 < \N
424      \advance \count255 1
425      \repeat
426      \edef\U{\BEZ{4*\N + 4}}%
427      \edef\V{\BEZ{4*\N + 3}}%
428      \edef\D{\BEZ5}%
429      \ifodd\N
430      $\U\times\A - \V\times \B = -\D$%
431      \else
432      $\U\times\A - \V\times\B = \D$%
433      \fi
434      \par
435      \endgroup
436 }%
437 \XINT_restorecatcodes_endinput%

```

7 Package *xintfrac* implementation

.1	Catcodes, ε -TeX and reload detection	143	.29	<code>\xintTTrunc</code>	162
.2	Package identification	144	.30	<code>\xintNum</code>	162
.3	<code>\XINT_cntSgnFork</code>	144	.31	<code>\xintRound, \xintiRound</code>	162
.4	<code>\xintLen</code>	144	.32	<code>\xintXTrunc</code>	164
.5	<code>\XINT_lenrord_loop</code>	144	.33	<code>\xintDigits</code>	169
.6	<code>\XINT_outfrac</code>	145	.34	<code>\xintFloat</code>	169
.7	<code>\XINT_inFrac</code>	146	.35	<code>\xintPFloat</code>	173
.8	<code>\XINT_frac</code>	146	.36	<code>\XINTinFloat</code>	174
.9	<code>\XINT_factortens, \XINT_cuz_cnt</code>	148	.37	<code>\xintAdd</code>	176
.10	<code>\xintRaw</code>	150	.38	<code>\xintSub</code>	178
.11	<code>\xintPRaw</code>	150	.39	<code>\xintSum</code>	178
.12	<code>\xintRawWithZeros</code>	151	.40	<code>\xintMul</code>	178
.13	<code>\xintFloor</code>	151	.41	<code>\xintSqr</code>	179
.14	<code>\xintCeil</code>	152	.42	<code>\xintPow</code>	179
.15	<code>\xintNumerator</code>	152	.43	<code>\xintFac</code>	180
.16	<code>\xintDenominator</code>	152	.44	<code>\xintPrd</code>	180
.17	<code>\xintFrac</code>	152	.45	<code>\xintDiv</code>	181
.18	<code>\xintSignedFrac</code>	153	.46	<code>\xintDivFloor</code>	181
.19	<code>\xintFwOver</code>	153	.47	<code>\xintDivTrunc</code>	181
.20	<code>\xintSignedFwOver</code>	154	.48	<code>\xintDivRound</code>	181
.21	<code>\xintREZ</code>	154	.49	<code>\xintMod</code>	181
.22	<code>\xintE, \xintFloatE, \XINTinFloatE</code>	155	.50	<code>\XINTinFloatMod</code>	182
.23	<code>\xintIrr</code>	156	.51	<code>\xintIsOne</code>	182
.24	<code>\xintifInt</code>	158	.52	<code>\xintGeq</code>	183
.25	<code>\xintJrr</code>	158	.53	<code>\xintMax</code>	184
.26	<code>\xintTFrac</code>	159	.54	<code>\xintMaxof</code>	184
.27	<code>\XINTinFloatFrac</code>	160	.55	<code>\xintMin</code>	185
.28	<code>\xintTrunc, \xintiTrunc</code>	160	.56	<code>\xintMinof</code>	185

.57	<code>\xintCmp</code>	186	.65	<code>\XINTinFloatSum</code>	190
.58	<code>\xintAbs</code>	187	.66	<code>\XINTinFloatPrd</code>	191
.59	<code>\xintOpp</code>	187	.67	<code>\xintFloatPow, \XINTinFloatPow</code>	191
.60	<code>\xintSgn</code>	188	.68	<code>\xintFloatPower, \XINTinFloatPower</code>	194
.61	<code>\xintFloatAdd, \XINTinFloatAdd</code>	188	.69	<code>\xintFloatSqrt, \XINTinFloatSqrt</code>	196
.62	<code>\xintFloatSub, \XINTinFloatSub</code>	189	.70	<code>\XINTinFloatMaxof</code>	200
.63	<code>\xintFloatMul, \XINTinFloatMul</code>	189	.71	<code>\XINTinFloatMinof</code>	200
.64	<code>\xintFloatDiv, \XINTinFloatDiv</code>	190			

The commenting is currently (2014/10/28) 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.

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2 \catcode13=5 % ^^M
3 \endlinechar=13 %
4 \catcode123=1 % {
5 \catcode125=2 % }
6 \catcode64=11 % @
7 \catcode35=6 % #
8 \catcode44=12 % ,
9 \catcode45=12 % -
10 \catcode46=12 % .
11 \catcode58=12 % :
12 \let\z\endgroup
13 \expandafter\let\expandafter\x\csname ver@xintfrac.sty\endcsname
14 \expandafter\let\expandafter\w\csname ver@xint.sty\endcsname
15 \expandafter
16 \ifx\csname PackageInfo\endcsname\relax
17 \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
18 \else
19 \def\y#1#2{\PackageInfo{#1}{#2}}%
20 \fi
21 \expandafter
22 \ifx\csname numexpr\endcsname\relax
23 \y{xintfrac}{\numexpr not available, aborting input}%
24 \aftergroup\endinput
25 \else
26 \ifx\x\relax % plain-TeX, first loading of xintfrac.sty
27 \ifx\w\relax % but xint.sty not yet loaded.
28 \def\z{\endgroup\input xint.sty\relax}%
29 \fi
30 \else
31 \def\empty {}%
32 \ifx\x\empty % LaTeX, first loading,
33 % variable is initialized, but \ProvidesPackage not yet seen
34 \ifx\w\relax % xint.sty not yet loaded.
35 \def\z{\endgroup\RequirePackage{xint}}%
36 \fi
37 \else
38 \aftergroup\endinput % xintfrac already loaded.

```

```

39      \fi
40    \fi
41  \fi
42 \z%
43 \XINTsetupcatcodes% defined in xintkernel.sty

```

7.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintfrac}%
46 [2014/10/28 v1.1 Expandable operations on fractions (jfb)]%
47 \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.

```

48 \def\xint_cntSgnFork #1%
49 {%
50   \ifcase #1\expandafter\xint_secondofthree
51     \or\expandafter\xint_thirdofthree
52     \else\expandafter\xint_firstofthree
53   \fi
54 }%

```

7.4 *\xintLen*

```

55 \def\xintLen {\romannumeral0\xintlen }%
56 \def\xintlen #1%
57 {%
58   \expandafter\xint_flen\romannumeral0\xint_infrac {#1}%
59 }%
60 \def\xint_flen #1#2#3%
61 {%
62   \expandafter\space
63   \the\numexpr -1+\XINT_Abs {#1}+\XINT_Len {#2}+\XINT_Len {#3}\relax
64 }%

```

7.5 *\XINT_lenrord_loop*

```

65 \def\xint_lenrord_loop #1#2#3#4#5#6#7#8#9%
66 {%  faire \romannumeral-'\XINT_lenrord_loop 0{}#1\Z\W\W\W\W\W\W\W\Z
67   \xint_gob_til_W #9\xint_lenrord_W\W
68   \expandafter\xint_lenrord_loop\expandafter
69   {\the\numexpr #1+7}{#9#8#7#6#5#4#3#2}%
70 }%
71 \def\xint_lenrord_W\W\expandafter\xint_lenrord_loop\expandafter #1#2#3\Z
72 {%
73   \expandafter\xint_lenrord_X\expandafter {#1}#2\Z
74 }%
75 \def\xint_lenrord_X #1#2\Z
76 {%
77   \XINT_lenrord_Y #2\R\R\R\R\R\R\T {#1}%
78 }%

```



```

79 \def\XINT_lenrord_Y #1#2#3#4#5#6#7#8\T
80 {%
81   \xint_gob_til_W
82     #7\XINT_lenrord_Z \xint_c_viii
83     #6\XINT_lenrord_Z \xint_c_vii
84     #5\XINT_lenrord_Z \xint_c_vi
85     #4\XINT_lenrord_Z \xint_c_v
86     #3\XINT_lenrord_Z \xint_c_iv
87     #2\XINT_lenrord_Z \xint_c_iii
88     \W\XINT_lenrord_Z \xint_c_ii \Z
89 }%
90 \def\XINT_lenrord_Z #1#2\Z #3% retourne: {longueur}renverse\Z
91 {%
92   \expandafter{\the\numexpr #3-#1\relax}%
93 }%

```

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 releases.

Months later (2014/10/22): perhaps I should document what this macro does before I forget? from $\{e\}\{N\}\{D\}$ it output $N/D[e]$, 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.

```

94 \def\XINT_outfrac #1#2#3%
95 {%
96   \ifcase\XINT_cntSgn #3\Z
97     \expandafter \XINT_outfrac_divisionbyzero
98   \or
99     \expandafter \XINT_outfrac_P
100  \else
101    \expandafter \XINT_outfrac_N
102  \fi
103  {\#2}\{\#3}\{\#1}%
104 }%
105 \def\XINT_outfrac_divisionbyzero #1#2{\xintError:DivisionByZero\space #1/0}%
106 \edef\XINT_outfrac_P #1#2%
107 {%
108   \noexpand\if0\noexpand\XINT_Sgn #1\noexpand\Z
109     \noexpand\expandafter\noexpand\XINT_outfrac_Zero
110   \noexpand\fi
111   \space #1/#2%
112 }%
113 \def\XINT_outfrac_Zero #1[#2]{ 0/1[0]}%
114 \def\XINT_outfrac_N #1#2%
115 {%
116   \expandafter\XINT_outfrac_N_a\expandafter
117   {\romannumeral0\XINT_opp #2}\{\romannumeral0\XINT_opp #1}%
118 }%
119 \def\XINT_outfrac_N_a #1#2%

```

```

120 {%
121   \expandafter\XINT_outfrac_P\expandafter {#2}{#1}%
122 }%

```

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.

```

123 \def\XINT_inFrac {\romannumeral0\XINT_infrac }%
124 \def\XINT_infrac #1%
125 {%
126   \expandafter\XINT_infrac_ \romannumeral-'0#1[\W]\Z\T
127 }%
128 \def\XINT_infrac_ #1[#2#3]#4\Z
129 {%
130   \xint_UDwfork
131     #2\XINT_infrac_A
132     \W\XINT_infrac_B
133   \krof
134   #1[#2#3]#4%
135 }%
136 \def\XINT_infrac_A #1[\W]\T
137 {%
138   \XINT_frac #1/\W\Z
139 }%
140 \def\XINT_infrac_B #1%
141 {%
142   \xint_gob_til_zero #1\XINT_infrac_Zero0\XINT_infrac_BB #1%
143 }%
144 \def\XINT_infrac_BB #1[\W]\T {\XINT_infrac_BC #1/\W\Z }%
145 \def\XINT_infrac_BC #1/#2#3\Z
146 {%
147   \xint_UDwfork
148     #2\XINT_infrac_BCa
149     \W{\expandafter\XINT_infrac_BCb \romannumeral-'0#2}%
150   \krof
151   #3\Z #1\Z
152 }%
153 \def\XINT_infrac_BCa \Z #1[#2]#3\Z { {#2}{#1}{1}}%
154 \def\XINT_infrac_BCb #1[#2]/\W\Z #3\Z { {#2}{#3}{#1}}%
155 \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`

```

156 \def\XINT_frac #1/#2#3\Z
157 {%

```

```

158 \xint_UDwfork
159 #2\XINT_frac_A
160 \W{\expandafter\XINT_frac_U \romannumeral-'\0#2}%
161 \krof
162 #3e\W\Z #1e\W\Z
163 }%
164 \def\XINT_frac_U #1e#2#3\Z
165 {%
166 \xint_UDwfork
167 #2\XINT_frac_Ua
168 \W{\XINT_frac_Ub #2}%
169 \krof
170 #3\Z #1\Z
171 }%
172 \def\XINT_frac_Ua \Z #1/\W\Z {\XINT_frac_B #1.\W\Z {0}}%
173 \def\XINT_frac_Ub #1/\W e\W\Z #2\Z {\XINT_frac_B #2.\W\Z {#1}}%
174 \def\XINT_frac_B #1.#2#3\Z
175 {%
176 \xint_UDwfork
177 #2\XINT_frac_Ba
178 \W{\XINT_frac_Bb #2}%
179 \krof
180 #3\Z #1\Z
181 }%
182 \def\XINT_frac_Ba \Z #1\Z {\XINT_frac_T {0}{#1}}%
183 \def\XINT_frac_Bb #1.\W\Z #2\Z
184 {%
185 \expandafter \XINT_frac_T \expandafter
186 {\romannumeral0\xintlength {#1}}{#2#1}%
187 }%
188 \def\XINT_frac_A e\W\Z {\XINT_frac_T {0}{1}{0}}%
189 \def\XINT_frac_T #1#2#3#4e#5#6\Z
190 {%
191 \xint_UDwfork
192 #5\XINT_frac-Ta
193 \W{\XINT_frac-Tb #5}%
194 \krof
195 #6\Z #4\Z {#1}{#2}{#3}%
196 }%
197 \def\XINT_frac-Ta \Z #1\Z {\XINT_frac_C #1.\W\Z {0}}%
198 \def\XINT_frac-Tb #1e\W\Z #2\Z {\XINT_frac_C #2.\W\Z {#1}}%
199 \def\XINT_frac_C #1.#2#3\Z
200 {%
201 \xint_UDwfork
202 #2\XINT_frac_Ca
203 \W{\XINT_frac_Cb #2}%
204 \krof
205 #3\Z #1\Z
206 }%
207 \def\XINT_frac_Ca \Z #1\Z {\XINT_frac_D {0}{#1}}%
208 \def\XINT_frac_Cb #1.\W\Z #2\Z
209 {%

```

```

210 \expandafter\XINT_frac_D\expandafter
211 {\romannumeral0\xintlength {#1}}{#2#1}%
212 }%
213 \def\XINT_frac_D #1#2#3#4#5#6%
214 {%
215 \expandafter \XINT_frac_E \expandafter
216 {\the\numexpr -#1+#3+#4-#6\expandafter}\expandafter
217 {\romannumeral0\XINT_num_loop #2%
218 \xint_relax\xint_relax\xint_relax\xint_relax
219 \xint_relax\xint_relax\xint_relax\xint_relax\Z }%
220 {\romannumeral0\XINT_num_loop #5%
221 \xint_relax\xint_relax\xint_relax\xint_relax
222 \xint_relax\xint_relax\xint_relax\xint_relax\Z }%
223 }%
224 \def\XINT_frac_E #1#2#3%
225 {%
226 \expandafter \XINT_frac_F #3\Z {#2}{#1}%
227 }%
228 \def\XINT_frac_F #1%
229 {%
230 \xint_UDzerominusfork
231 #1-\XINT_frac_Gdivisionbyzero
232 0#1\XINT_frac_Gneg
233 0-{\XINT_frac_Gpos #1}%
234 \krof
235 }%
236 \edef\XINT_frac_Gdivisionbyzero #1\Z #2#3%
237 {%
238 \noexpand\xintError:DivisionByZero\space {0}{#2}{0}%
239 }%
240 \def\XINT_frac_Gneg #1\Z #2#3%
241 {%
242 \expandafter\XINT_frac_H \expandafter{\romannumeral0\XINT_opp #2}{#3}{#1}%
243 }%
244 \def\XINT_frac_H #1#2{ {#2}{#1}}%
245 \def\XINT_frac_Gpos #1\Z #2#3{ {#3}{#2}{#1}}%

```

7.9 \XINT_factortens, \XINT_cuz_cnt

```

246 \def\XINT_factortens #1%
247 {%
248 \expandafter\XINT_cuz_cnt_loop\expandafter
249 {\expandafter}\romannumeral0\XINT_rord_main { }#1%
250 \xint_relax
251 \xint_bye\xint_bye\xint_bye\xint_bye
252 \xint_bye\xint_bye\xint_bye\xint_bye
253 \xint_relax
254 \R\R\R\R\R\R\R\Z
255 }%
256 \def\XINT_cuz_cnt #1%
257 {%
258 \XINT_cuz_cnt_loop { }#1\R\R\R\R\R\R\R\Z
259 }%

```

7 Package *xintfrac* implementation

```

260 \def\XINT_cuz_cnt_loop #1#2#3#4#5#6#7#8#9%
261 {%
262   \xint_gob_til_R #9\XINT_cuz_cnt_toofara \R
263   \expandafter\XINT_cuz_cnt_checka\expandafter
264   {\the\numexpr #1+8\relax}{#2#3#4#5#6#7#8#9}%
265 }%
266 \def\XINT_cuz_cnt_toofara\R
267   \expandafter\XINT_cuz_cnt_checka\expandafter #1#2%
268 {%
269   \XINT_cuz_cnt_toofarb {#1}#2%
270 }%
271 \def\XINT_cuz_cnt_toofarb #1#2\Z {\XINT_cuz_cnt_toofarc #2\Z {#1}}%
272 \def\XINT_cuz_cnt_toofarc #1#2#3#4#5#6#7#8%
273 {%
274   \xint_gob_til_R #2\XINT_cuz_cnt_toofard 7%
275       #3\XINT_cuz_cnt_toofard 6%
276       #4\XINT_cuz_cnt_toofard 5%
277       #5\XINT_cuz_cnt_toofard 4%
278       #6\XINT_cuz_cnt_toofard 3%
279       #7\XINT_cuz_cnt_toofard 2%
280       #8\XINT_cuz_cnt_toofard 1%
281       \Z #1#2#3#4#5#6#7#8%
282 }%
283 \def\XINT_cuz_cnt_toofard #1#2\Z #3\R #4\Z #5%
284 {%
285   \expandafter\XINT_cuz_cnt_toofare
286   \the\numexpr #3\relax \R\R\R\R\R\R\R\R\Z
287   {\the\numexpr #5-#1\relax}\R\Z
288 }%
289 \def\XINT_cuz_cnt_toofare #1#2#3#4#5#6#7#8%
290 {%
291   \xint_gob_til_R #2\XINT_cuz_cnt_stopc 1%
292       #3\XINT_cuz_cnt_stopc 2%
293       #4\XINT_cuz_cnt_stopc 3%
294       #5\XINT_cuz_cnt_stopc 4%
295       #6\XINT_cuz_cnt_stopc 5%
296       #7\XINT_cuz_cnt_stopc 6%
297       #8\XINT_cuz_cnt_stopc 7%
298       \Z #1#2#3#4#5#6#7#8%
299 }%
300 \def\XINT_cuz_cnt_checka #1#2%
301 {%
302   \expandafter\XINT_cuz_cnt_checkb\the\numexpr #2\relax \Z {#1}%
303 }%
304 \def\XINT_cuz_cnt_checkb #1%
305 {%
306   \xint_gob_til_zero #1\expandafter\XINT_cuz_cnt_loop\xint_gob_til_Z
307   0\XINT_cuz_cnt_stopa #1%
308 }%
309 \def\XINT_cuz_cnt_stopa #1\Z
310 {%
311   \XINT_cuz_cnt_stopb #1\R\R\R\R\R\R\R\R\Z %

```

```

312 }%
313 \def\XINT_cuz_cnt_stopb #1#2#3#4#5#6#7#8#9%
314 {%
315     \xint_gob_til_R #2\XINT_cuz_cnt_stopc 1%
316         #3\XINT_cuz_cnt_stopc 2%
317         #4\XINT_cuz_cnt_stopc 3%
318         #5\XINT_cuz_cnt_stopc 4%
319         #6\XINT_cuz_cnt_stopc 5%
320         #7\XINT_cuz_cnt_stopc 6%
321         #8\XINT_cuz_cnt_stopc 7%
322         #9\XINT_cuz_cnt_stopc 8%
323     \Z #1#2#3#4#5#6#7#8#9%
324 }%
325 \def\XINT_cuz_cnt_stopc #1#2\Z #3\R #4\Z #5%
326 {%
327     \expandafter\XINT_cuz_cnt_stopd\expandafter
328     {\the\numexpr #5-#1}\#3%
329 }%
330 \def\XINT_cuz_cnt_stopd #1#2\R #3\Z
331 {%
332     \expandafter\space\expandafter
333     {\romannumeral0\XINT_rord_main {}}\#2%
334     \xint_relax
335     \xint_bye\xint_bye\xint_bye\xint_bye
336     \xint_bye\xint_bye\xint_bye\xint_bye
337     \xint_relax }\#1}%
338 }%

```

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.

```

339 \def\xintRaw {\romannumeral0\xintraw }%
340 \def\xintraw
341 {%
342     \expandafter\XINT_raw\romannumeral0\XINT_infrac
343 }%
344 \def\XINT_raw #1#2#3{ #2/#3[#1]}%

```

7.11 `\xintPRaw`

1.09b

```

345 \def\xintPRaw {\romannumeral0\xintpraw }%
346 \def\xintpraw
347 {%
348     \expandafter\XINT_praw\romannumeral0\XINT_infrac
349 }%
350 \def\XINT_praw #1%
351 {%
352     \ifnum #1=\xint_c_ \expandafter\XINT_praw_a\fi \XINT_praw_A {#1}%
353 }%

```

```

354 \def\XINT_praw_A #1#2#3%
355 {%
356   \if\XINT_isOne{#3}1\expandafter\xint_firstoftwo
357     \else\expandafter\xint_secondoftwo
358   \fi { #2[#1]}{ #2/#3[#1]}%
359 }%
360 \def\XINT_praw_a\XINT_praw_A #1#2#3%
361 {%
362   \if\XINT_isOne{#3}1\expandafter\xint_firstoftwo
363     \else\expandafter\xint_secondoftwo
364   \fi { #2}{ #2/#3}%
365 }%

```

7.12 \xintRawWithZeros

This was called `\xintRaw` in versions earlier than 1.07

```

366 \def\xintRawWithZeros {\romannumeral0\xinrawwithzeros }%
367 \def\xinrawwithzeros
368 {%
369   \expandafter\XINT_rawz\romannumeral0\XINT_infrac
370 }%
371 \def\XINT_rawz #1%
372 {%
373   \ifcase\XINT_cntSgn #1\Z
374     \expandafter\XINT_rawz_Ba
375   \or
376     \expandafter\XINT_rawz_A
377   \else
378     \expandafter\XINT_rawz_Ba
379   \fi
380   {#1}%
381 }%
382 \def\XINT_rawz_A #1#2#3{\xint_dsh {#2}{-#1}/#3}%
383 \def\XINT_rawz_Ba #1#2#3{\expandafter\XINT_rawz_Bb
384   \expandafter{\romannumeral0\xint_dsh {#3}{#1}}{#2}}%
385 \def\XINT_rawz_Bb #1#2{ #2/#1}%

```

7.13 \xintFloor

1.09a, 1.1 for `\xintiFloor`/`\xintFloor`. Not efficient if big negative decimal exponent. Also sub-efficient if big positive decimal exponent.

```

386 \def\xintFloor {\romannumeral0\xintfloor }%
387 \def\xintfloor #1% devrais-je faire \xintREZ?
388   {\expandafter\XINT_ifloor \romannumeral0\xinrawwithzeros {#1}./1[0]}%
389 \def\xintiFloor {\romannumeral0\xintifloor }%
390 \def\xintifloor #1%
391   {\expandafter\XINT_ifloor \romannumeral0\xinrawwithzeros {#1}.}%
392 \def\XINT_ifloor #1/#2.{\xintiiquo {#1}{#2}}%

```

7.14 `\xintCeil`

1.09a

```

393 \def\xintCeil {\romannumeral0\xintceil }%
394 \def\xintceil #1{\xintiiopt {\xintFloor {\xintOpp{#1}}}%
395 \def\xintiCeil {\romannumeral0\xinticeil }%
396 \def\xinticeil #1{\xintiiopt {\xintiFloor {\xintOpp{#1}}}%

```

7.15 `\xintNumerator`

```

397 \def\xintNumerator {\romannumeral0\xintnumerator }%
398 \def\xintnumerator
399 {%
400   \expandafter\XINT_numer\romannumeral0\XINT_infrac
401 }%
402 \def\XINT_numer #1%
403 {%
404   \ifcase\XINT_cntSgn #1\Z
405     \expandafter\XINT_numer_B
406   \or
407     \expandafter\XINT_numer_A
408   \else
409     \expandafter\XINT_numer_B
410   \fi
411   {#1}%
412 }%
413 \def\XINT_numer_A #1#2#3{\xint_dsh {#2}{-#1}}%
414 \def\XINT_numer_B #1#2#3{ #2}%

```

7.16 `\xintDenominator`

```

415 \def\xintDenominator {\romannumeral0\xintdenominator }%
416 \def\xintdenominator
417 {%
418   \expandafter\XINT_denom\romannumeral0\XINT_infrac
419 }%
420 \def\XINT_denom #1%
421 {%
422   \ifcase\XINT_cntSgn #1\Z
423     \expandafter\XINT_denom_B
424   \or
425     \expandafter\XINT_denom_A
426   \else
427     \expandafter\XINT_denom_B
428   \fi
429   {#1}%
430 }%
431 \def\XINT_denom_A #1#2#3{ #3}%
432 \def\XINT_denom_B #1#2#3{\xint_dsh {#3}{#1}}%

```

7.17 `\xintFrac`

```

433 \def\xintFrac {\romannumeral0\xintfrac }%
434 \def\xintfrac #1%

```



```

435 {%
436   \expandafter\XINT_fracfrac_A\romannumeral0\XINT_infrac {#1}%
437 }%
438 \def\XINT_fracfrac_A #1{\XINT_fracfrac_B #1\Z }%
439 \catcode'\^=7
440 \def\XINT_fracfrac_B #1#2\Z
441 {%
442   \xint_gob_til_zero #1\XINT_fracfrac_C 0\XINT_fracfrac_D {10^{#1#2}}%
443 }%
444 \def\XINT_fracfrac_C 0\XINT_fracfrac_D #1#2#3%
445 {%
446   \if1\XINT_isOne {#3}%
447     \xint_afterfi {\expandafter\xint_firstoftwo_thenstop\xint_gobble_ii }%
448   \fi
449   \space
450   \frac {#2}{#3}%
451 }%
452 \def\XINT_fracfrac_D #1#2#3%
453 {%
454   \if1\XINT_isOne {#3}\XINT_fracfrac_E\fi
455   \space
456   \frac {#2}{#3}#1%
457 }%
458 \def\XINT_fracfrac_E \fi\space\frac #1#2{\fi \space #1\cdot }%

```

7.18 \xintSignedFrac

```

459 \def\xintSignedFrac {\romannumeral0\xintsignedfrac }%
460 \def\xintsignedfrac #1%
461 {%
462   \expandafter\XINT_sgnfrac_a\romannumeral0\XINT_infrac {#1}%
463 }%
464 \def\XINT_sgnfrac_a #1#2%
465 {%
466   \XINT_sgnfrac_b #2\Z {#1}%
467 }%
468 \def\XINT_sgnfrac_b #1%
469 {%
470   \xint_UDsignfork
471   #1\XINT_sgnfrac_N
472   -{\XINT_sgnfrac_P #1}%
473   \krof
474 }%
475 \def\XINT_sgnfrac_P #1\Z #2%
476 {%
477   \XINT_fracfrac_A {#2}{#1}%
478 }%
479 \def\XINT_sgnfrac_N
480 {%
481   \expandafter\xint_minus_thenstop\romannumeral0\XINT_sgnfrac_P
482 }%

```

7.19 \xintFwOver

```

483 \def\xintFwOver {\romannumeral0\xintfwover }%

```

```

484 \def\xintfwover #1%
485 {%
486   \expandafter\XINT_fwover_A\romannumeral0\XINT_infrac {#1}%
487 }%
488 \def\XINT_fwover_A #1{\XINT_fwover_B #1\Z }%
489 \def\XINT_fwover_B #1#2\Z
490 {%
491   \xint_gob_til_zero #1\XINT_fwover_C 0\XINT_fwover_D {10^{#1#2}}%
492 }%
493 \catcode'\^=11
494 \def\XINT_fwover_C #1#2#3#4#5%
495 {%
496   \if0\XINT_isOne {#5}\xint_afterfi { {#4\over #5}}%
497   \else\xint_afterfi { #4}%
498   \fi
499 }%
500 \def\XINT_fwover_D #1#2#3%
501 {%
502   \if0\XINT_isOne {#3}\xint_afterfi { {#2\over #3}}%
503   \else\xint_afterfi { #2\cdot }%
504   \fi
505   #1%
506 }%

```

7.20 \xintSignedFwOver

```

507 \def\xintSignedFwOver {\romannumeral0\xintsignedfwover }%
508 \def\xintsignedfwover #1%
509 {%
510   \expandafter\XINT_sgnfwover_a\romannumeral0\XINT_infrac {#1}%
511 }%
512 \def\XINT_sgnfwover_a #1#2%
513 {%
514   \XINT_sgnfwover_b #2\Z {#1}%
515 }%
516 \def\XINT_sgnfwover_b #1%
517 {%
518   \xint_UDsignfork
519   #1\XINT_sgnfwover_N
520   -{\XINT_sgnfwover_P #1}%
521   \krof
522 }%
523 \def\XINT_sgnfwover_P #1\Z #2%
524 {%
525   \XINT_fwover_A {#2}{#1}%
526 }%
527 \def\XINT_sgnfwover_N
528 {%
529   \expandafter\xint_minus_thenstop\romannumeral0\XINT_sgnfwover_P
530 }%

```

7.21 \xintREZ

```

531 \def\xintREZ {\romannumeral0\xintrez }%
532 \def\xintrez

```

```

533 {%
534   \expandafter\XINT_rez_A\romannumeral0\XINT_infrac
535 }%
536 \def\XINT_rez_A #1#2%
537 {%
538   \XINT_rez_AB #2\Z {#1}%
539 }%
540 \def\XINT_rez_AB #1%
541 {%
542   \xint_UDzerominusfork
543   #1-\XINT_rez_zero
544   0#1\XINT_rez_neg
545   0-{\XINT_rez_B #1}%
546   \krof
547 }%
548 \def\XINT_rez_zero #1\Z #2#3{ 0/1[0]}%
549 \def\XINT_rez_neg {\expandafter\xint_minus_thenstop\romannumeral0\XINT_rez_B }%
550 \def\XINT_rez_B #1\Z
551 {%
552   \expandafter\XINT_rez_C\romannumeral0\XINT_factortens {#1}%
553 }%
554 \def\XINT_rez_C #1#2#3#4%
555 {%
556   \expandafter\XINT_rez_D\romannumeral0\XINT_factortens {#4}{#3}{#2}{#1}%
557 }%
558 \def\XINT_rez_D #1#2#3#4#5%
559 {%
560   \expandafter\XINT_rez_E\expandafter
561   {\the\numexpr #3+#4-#2}{#1}{#5}%
562 }%
563 \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 `\XINTinFloatE`, although currently the latter is only used from `\xintfloatexpr` hence always with `\XINTdigits`, it comes equipped with its first argument withing 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.

```

564 \def\xintE {\romannumeral0\xinte }%
565 \def\xinte #1%

```

```

566 {%
567   \expandafter\XINT_e \romannumeral0\XINT_infrac {#1}%
568 }%
569 \def\XINT_e #1#2#3#4%
570 {%
571   \expandafter\XINT_e_end\expandafter{\the\numexpr #1+#4}{#2}{#3}%
572 }%
573 \def\XINT_e_end #1#2#3{ #2/#3[#1]}%
574 \def\xintFloatE {\romannumeral0\xintfloate }%
575 \def\xintfloate #1{\XINT_floate_chkopt #1\xint_relax }%
576 \def\XINT_floate_chkopt #1%
577 {%
578   \ifx [#1\expandafter\XINT_floate_opt
579     \else\expandafter\XINT_floate_noopt
580   \fi #1%
581 }%
582 \def\XINT_floate_noopt #1\xint_relax
583 {%
584   \expandafter\XINT_floate_a\expandafter\XINTdigits
585     \romannumeral0\XINT_infrac {#1}%
586 }%
587 \def\XINT_floate_opt [\xint_relax #1]#2%
588 {%
589   \expandafter\XINT_floate_a\expandafter
590     {\the\numexpr #1\expandafter}\romannumeral0\XINT_infrac {#2}%
591 }%
592 \def\XINT_floate_a #1#2#3#4#5%
593 {%
594   \expandafter\expandafter\expandafter\XINT_float_a
595   \expandafter\xint_exchangetwo_keepbraces\expandafter
596     {\the\numexpr #2+#5}{#1}{#3}{#4}\XINT_float_Q
597 }%
598 \def\XINTinFloatE {\romannumeral0\XINTinfloate }%
599 \def\XINTinfloate {\expandafter\XINT_infloate\romannumeral0\XINTinfloat [\XINTdigits]}%
600 \def\XINT_infloate #1[#2]#3%
601   {\expandafter\XINT_infloate_end\expandafter {\the\numexpr #3+#2}{#1}}%
602 \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.

```

603 \def\xintIrr {\romannumeral0\xintirr }%
604 \def\xintirr #1%
605 {%
606   \expandafter\XINT_irr_start\romannumeral0\xintrawwithzeros {#1}\Z
607 }%
608 \def\XINT_irr_start #1#2/#3\Z
609 {%
610   \if0\XINT_isOne {#3}%
611     \xint_afterfi

```

```

612      {\xint_UDsignfork
613        #1\XINT_irr_negative
614        -{\XINT_irr_nonneg #1}%
615        \krof}%
616    \else
617      \xint_afterfi{\XINT_irr_denomisone #1}%
618    \fi
619    #2\Z {#3}%
620 }%
621 \def\XINT_irr_denomisone #1\Z #2{ #1/1}% changed in 1.08
622 \def\XINT_irr_negative #1\Z #2{\XINT_irr_D #1\Z #2\Z \xint_minus_thenstop}%
623 \def\XINT_irr_nonneg #1\Z #2{\XINT_irr_D #1\Z #2\Z \space}%
624 \def\XINT_irr_D #1#2\Z #3#4\Z
625 {%
626   \xint_UDzerosfork
627     #3#1\XINT_irr_indeterminate
628     #30\XINT_irr_divisionbyzero
629     #10\XINT_irr_zero
630     00\XINT_irr_loop_a
631   \krof
632   {#3#4}{#1#2}{#3#4}{#1#2}%
633 }%
634 \def\XINT_irr_indeterminate #1#2#3#4#5{\xintError:NaN\space 0/0}%
635 \def\XINT_irr_divisionbyzero #1#2#3#4#5{\xintError:DivisionByZero #5#2/0}%
636 \def\XINT_irr_zero #1#2#3#4#5{ 0/1}% changed in 1.08
637 \def\XINT_irr_loop_a #1#2%
638 {%
639   \expandafter\XINT_irr_loop_d
640   \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
641 }%
642 \def\XINT_irr_loop_d #1#2%
643 {%
644   \XINT_irr_loop_e #2\Z
645 }%
646 \def\XINT_irr_loop_e #1#2\Z
647 {%
648   \xint_gob_til_zero #1\xint_irr_loop_exit0\XINT_irr_loop_a {#1#2}%
649 }%
650 \def\xint_irr_loop_exit0\XINT_irr_loop_a #1#2#3#4%
651 {%
652   \expandafter\XINT_irr_loop_exitb\expandafter
653   {\romannumeral0\xintiigo {#3}{#2}}%
654   {\romannumeral0\xintiigo {#4}{#2}}%
655 }%
656 \def\XINT_irr_loop_exitb #1#2%
657 {%
658   \expandafter\XINT_irr_finish\expandafter {#2}{#1}%
659 }%
660 \def\XINT_irr_finish #1#2#3{#3#1/#2}% changed in 1.08

```

7.24 `\xintifInt`

1.09e. *xintfrac.sty* only. Nota bene (22/06/14) I just don't understand why this went through the costly *xintIrr* overhead !! shame on the author.

```

661 \def\xintifInt    {\romannumeral0\xintifint }%
662 \def\xintifint #1{\expandafter\XINT_ifint\romannumeral0\xintraawithzeros {#1}.}%
663 \def\XINT_ifint #1/#2.%
664 {%
665     \if 0\xintiiRem {#1}{#2}%
666     \expandafter\xint_firstoftwo_thenstop
667     \else
668     \expandafter\xint_secondoftwo_thenstop
669     \fi
670 }%
```

7.25 `\xintJrr`

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

```

671 \def\xintJrr {\romannumeral0\xintjrr }%
672 \def\xintjrr #1%
673 {%
674     \expandafter\XINT_jrr_start\romannumeral0\xintraawithzeros {#1}\Z
675 }%
676 \def\XINT_jrr_start #1#2/#3\Z
677 {%
678     \if0\XINT_isOne {#3}\xint_afterfi
679         {\xint_UDsignfork
680             #1\XINT_jrr_negative
681             -{\XINT_jrr_nonneg #1}%
682             \krof}%
683     \else
684         \xint_afterfi{\XINT_jrr_denomisone #1}%
685     \fi
686     #2\Z {#3}%
687 }%
688 \def\XINT_jrr_denomisone #1\Z #2{ #1/1}% changed in 1.08
689 \def\XINT_jrr_negative #1\Z #2{\XINT_jrr_D #1\Z #2\Z \xint_minus_thenstop }%
690 \def\XINT_jrr_nonneg #1\Z #2{\XINT_jrr_D #1\Z #2\Z \space}%
691 \def\XINT_jrr_D #1#2\Z #3#4\Z
692 {%
693     \xint_UDzerosfork
694         #3#1\XINT_jrr_indeterminate
695         #30\XINT_jrr_divisionbyzero
696         #10\XINT_jrr_zero
697         00\XINT_jrr_loop_a
698     \krof
699     {#3#4}{#1#2}1001%
700 }%
701 \def\XINT_jrr_indeterminate #1#2#3#4#5#6#7{\xintError:NaN\space 0/0}%
702 \def\XINT_jrr_divisionbyzero #1#2#3#4#5#6#7{\xintError:DivisionByZero #7#2/0}%
703 \def\XINT_jrr_zero #1#2#3#4#5#6#7{ 0/1}% changed in 1.08
```

```

704 \def\XINT_jrr_loop_a #1#2%
705 {%
706   \expandafter\XINT_jrr_loop_b
707   \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
708 }%
709 \def\XINT_jrr_loop_b #1#2#3#4#5#6#7%
710 {%
711   \expandafter \XINT_jrr_loop_c \expandafter
712     {\romannumeral0\xinttiadd{\XINT_Mul{#4}{#1}}{#6}}%
713     {\romannumeral0\xinttiadd{\XINT_Mul{#5}{#1}}{#7}}%
714     {#2}{#3}{#4}{#5}%
715 }%
716 \def\XINT_jrr_loop_c #1#2%
717 {%
718   \expandafter \XINT_jrr_loop_d \expandafter{#2}{#1}%
719 }%
720 \def\XINT_jrr_loop_d #1#2#3#4%
721 {%
722   \XINT_jrr_loop_e #3\Z {#4}{#2}{#1}%
723 }%
724 \def\XINT_jrr_loop_e #1#2\Z
725 {%
726   \xint_gob_til_zero #1\xint_jrr_loop_exit0\XINT_jrr_loop_a {#1#2}%
727 }%
728 \def\xint_jrr_loop_exit0\XINT_jrr_loop_a #1#2#3#4#5#6%
729 {%
730   \XINT_irr_finish {#3}{#4}%
731 }%

```

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 $x - \text{floor}(x)$. 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`.

```

732 \def\xintTFrac {\romannumeral0\xinttfrac }%
733 \def\xinttfrac #1{\expandafter\XINT_tfrac_fork\romannumeral0\xintrawithzeros {#1}\Z }%
734 \def\XINT_tfrac_fork #1%
735 {%
736   \xint_UDzerominusfork
737   #1-\XINT_tfrac_zero
738   0#1{\xinttiopp\XINT_tfrac_P }%
739   0-{\XINT_tfrac_P #1}%
740   \krof
741 }%
742 \def\XINT_tfrac_zero #1\Z { 0/1[0]}%
743 \def\XINT_tfrac_P #1/#2\Z {\expandafter\XINT_rez_AB
744   \romannumeral0\xintiirem{#1}{#2}\Z {0}{#2}}%

```

7.27 `\XINTinFloatFrac`

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 possibility of optional argument, for technical reasons having to do with `\xintNewExpr`.

```

745 \def\XINTinFloatFrac {\romannumeral0\XINTinfloatfrac [\XINTdigits]]%
746 \def\XINTinfloatfrac [#1]#2%
747 {%
748   \expandafter\XINT_infloatfrac_a\expandafter {\romannumeral0\xinttfrac{#2}}{#1}%
749 }%
750 \def\XINT_infloatfrac_a #1#2{\XINTinFloat [#2]{#1}}%
```

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 `\xintquo` was redefined to use `\xintnum`. Now uses `\xintiquo`, rather.

1.09j: minor improvements, `\XINT_trunc_E` was very strange and defined two never occurring 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.

```

751 \def\xintTrunc {\romannumeral0\xinttrunc }%
752 \def\xintiTrunc {\romannumeral0\xintitrunc }%
753 \def\xinttrunc #1%
754 {%
755   \expandafter\XINT_trunc\expandafter {\the\numexpr #1}%
756 }%
757 \def\XINT_trunc #1#2%
758 {%
759   \expandafter\XINT_trunc_G
760   \romannumeral0\expandafter\XINT_trunc_A
761   \romannumeral0\XINT_infrac {#2}{#1}{#1}%
762 }%
763 \def\xintitrunc #1%
764 {%
765   \expandafter\XINT_itrunc\expandafter {\the\numexpr #1}%
766 }%
767 \def\XINT_itrunc #1#2%
768 {%
769   \expandafter\XINT_itrunc_G
770   \romannumeral0\expandafter\XINT_trunc_A
771   \romannumeral0\XINT_infrac {#2}{#1}{#1}%
772 }%
773 \def\XINT_trunc_A #1#2#3#4%
774 {%
775   \expandafter\XINT_trunc_checkifzero
776   \expandafter{\the\numexpr #1+#4}#2\Z {#3}%
777 }%
778 \def\XINT_trunc_checkifzero #1#2#3\Z
779 {%
```


7 Package *xintfrac* implementation

```

780 \xint_gob_til_zero #2\XINT_trunc_iszero0\XINT_trunc_B {#1}{#2#3}%
781 }%
782 \def\XINT_trunc_iszero0\XINT_trunc_B #1#2#3{ 0\Z 0}%
783 \def\XINT_trunc_B #1%
784 {%
785 \ifcase\XINT_cntSgn #1\Z
786 \expandafter\XINT_trunc_D
787 \or
788 \expandafter\XINT_trunc_D
789 \else
790 \expandafter\XINT_trunc_C
791 \fi
792 {#1}%
793 }%
794 \def\XINT_trunc_C #1#2#3%
795 {%
796 \expandafter\XINT_trunc_CE\expandafter
797 {\romannumeral0\XINT_dsx_zero loop {-#1}{}\Z {#3}}{#2}%
798 }%
799 \def\XINT_trunc_CE #1#2{\XINT_trunc_E #2.{#1}}%
800 \def\XINT_trunc_D #1#2%
801 {%
802 \expandafter\XINT_trunc_E
803 \romannumeral0\XINT_dsx_zero loop {#1}{}\Z {#2}.%
804 }%
805 \def\XINT_trunc_E #1%
806 {%
807 \xint_UDsignfork
808 #1\XINT_trunc_Fneg
809 -{\XINT_trunc_Fpos #1}%
810 \krof
811 }%
812 \def\XINT_trunc_Fneg #1.#2{\expandafter\xint_firstoftwo_thenstop
813 \romannumeral0\XINT_div_prepare {#2}{#1}\Z \xint_minus_thenstop}%
814 \def\XINT_trunc_Fpos #1.#2{\expandafter\xint_firstoftwo_thenstop
815 \romannumeral0\XINT_div_prepare {#2}{#1}\Z \space }%
816 \def\XINT_itrunc_G #1#2\Z #3#4%
817 {%
818 \xint_gob_til_zero #1\XINT_trunc_zero 0#3#1#2%
819 }%
820 \def\XINT_trunc_zero 0#1#20{ 0}%
821 \def\XINT_trunc_G #1\Z #2#3%
822 {%
823 \xint_gob_til_zero #2\XINT_trunc_zero 0%
824 \expandafter\XINT_trunc_H\expandafter
825 {\the\numexpr\romannumeral0\xintlength {#1}-#3}{#3}{#1}#2%
826 }%
827 \def\XINT_trunc_H #1#2%
828 {%
829 \ifnum #1 > \xint_c_
830 \xint_afterfi {\XINT_trunc_Ha {#2}}%
831 \else

```

```

832      \xint_afterfi {\XINT_trunc_Hb {-#1}}% -0,--1,--2, ....
833      \fi
834 }%
835 \def\XINT_trunc_Ha
836 {%
837   \expandafter\XINT_trunc_Haa\romannumeral0\xintdecsplit
838 }%
839 \def\XINT_trunc_Haa #1#2#3%
840 {%
841   #3#1.#2%
842 }%
843 \def\XINT_trunc_Hb #1#2#3%
844 {%
845   \expandafter #3\expandafter0\expandafter.%
846   \romannumeral0\XINT_dsx_zeroloop {#1}{\Z {}}#2% #1=-0 autoris\'e !
847 }%

```

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`.

```

848 \def\xintTTrunc {\romannumeral0\xintttrunc }%
849 \def\xintttrunc #1%
850 {%
851   \expandafter\XINT_itrunc_G
852   \romannumeral0\expandafter\XINT_ttrunc_A
853   \romannumeral0\XINT_infrac {#1}0% this last 0 to let \XINT_itrunc_G be happy
854 }%
855 \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 (1.1) I just don't understand what was the point of going through `\xintIrr` if to raise an error 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, just before releasing 1.1 (I left it taking dust since June...), I did `\xintTTrunc`, and will map `\xintNum` to it

```

856 \let\xintNum \xintTTrunc
857 \let\xintnum \xintttrunc

```

7.31 `\xintRound`, `\xintiRound`

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

```

858 \def\xintRound {\romannumeral0\xintround }%
859 \def\xintiRound {\romannumeral0\xintiround }%

```

```

860 \def\xintround #1%
861 {%
862   \expandafter\XINT_round\expandafter {\the\numexpr #1}%
863 }%
864 \def\XINT_round
865 {%
866   \expandafter\XINT_trunc_G\romannumeral0\XINT_round_A
867 }%
868 \def\xintiround #1%
869 {%
870   \expandafter\XINT_iround\expandafter {\the\numexpr #1}%
871 }%
872 \def\XINT_iround
873 {%
874   \expandafter\XINT_itrunc_G\romannumeral0\XINT_round_A
875 }%
876 \def\XINT_round_A #1#2%
877 {%
878   \expandafter\XINT_round_B
879   \romannumeral0\expandafter\XINT_trunc_A
880   \romannumeral0\XINT_infrac {#2}{\the\numexpr #1+1\relax}{#1}%
881 }%
882 \def\XINT_round_B #1\Z
883 {%
884   \expandafter\XINT_round_C
885   \romannumeral0\XINT_rord_main {}#1%
886   \xint_relax
887   \xint_bye\xint_bye\xint_bye\xint_bye
888   \xint_bye\xint_bye\xint_bye\xint_bye
889   \xint_relax
890   \Z
891 }%
892 \def\XINT_round_C #1%
893 {%
894   \ifnum #1<5
895     \expandafter\XINT_round_Daa
896   \else
897     \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 $A \cdot 10^N$. 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 with 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`).

```

922 \def\xintXTrunc #1#2%
923 {%
924   \expandafter\XINT_xtrunc_a\expandafter
925   {\the\numexpr #1\expandafter}\romannumeral0\xintraW {#2}%
926 }%
927 \def\XINT_xtrunc_a #1%
928 {%
929   \expandafter\XINT_xtrunc_b\expandafter
930   {\the\numexpr\ifnum#1<\xint_c_i \xint_c_i-\fi #1}%
931 }%
932 \def\XINT_xtrunc_b #1%
933 {%
934   \expandafter\XINT_xtrunc_c\expandafter
935   {\the\numexpr (#1+\xint_c_ii^v)/\xint_c_ii^vi-\xint_c_i}{#1}%

```

[illegible]

```

988 }%
989 \def\XINT_xtrunc_negN_T #1%
990 {%
991   \ifnum \xint_c_<#1
992     \expandafter\XINT_xtrunc_negNA
993   \else
994     \expandafter\XINT_xtrunc_negNW
995   \fi {#1}%
996 }%
997 % #1=D-|N|>0, #2=|N|, #3=D, #4=R, #5=B, #6=Q
998 \def\XINT_xtrunc_unlock #10.{ }%
999 \def\XINT_xtrunc_negNA #1#2#3#4#5#6%
1000 {%
1001   \expandafter\XINT_xtrunc_negNB\expandafter
1002   {\romannumeral0\expandafter\expandafter\expandafter
1003     \XINT_xtrunc_unlock\expandafter\string
1004     \csname\XINT_xtrunc_b {#1}#4/#5[0]\expandafter\endcsname
1005     \expandafter}\expandafter
1006     {\the\numexpr\xintLength{#6}-#2}{#6}%
1007 }%
1008 \def\XINT_xtrunc_negNB #1#2#3{\XINT_xtrunc_negNC {#2}{#3}#1}%
1009 \def\XINT_xtrunc_negNC #1%
1010 {%
1011   \ifnum \xint_c_ < #1
1012     \expandafter\XINT_xtrunc_negNDa
1013   \else
1014     \expandafter\XINT_xtrunc_negNE
1015   \fi {#1}%
1016 }%
1017 \def\XINT_xtrunc_negNDa #1#2%
1018 {%
1019   \expandafter\XINT_xtrunc_negNDb%
1020   \romannumeral0\XINT_split_fromleft_loop {#1}{}#2\W\W\W\W\W\W\W\W\Z
1021 }%
1022 \def\XINT_xtrunc_negNDb #1#2{#1.#2}%
1023 \def\XINT_xtrunc_negNE #1#2%
1024 {%
1025   0.\romannumeral0\XINT_dsx_zero loop {-#1}{}Z {}#2%
1026 }%
1027 % #1=D-|N|<=0, #2=|N|, #3=D, #4=R, #5=B, #6=Q
1028 \def\XINT_xtrunc_negNW #1#2#3#4#5#6%
1029 {%
1030   \expandafter\XINT_xtrunc_negNX\expandafter
1031   {\romannumeral0\xintnum{\xintDecSplitL {-#1}{#6}}}{#3}%
1032 }%
1033 \def\XINT_xtrunc_negNX #1#2%
1034 {%
1035   \expandafter\XINT_xtrunc_negNC\expandafter
1036   {\the\numexpr\xintLength {#1}-#2}{#1}%
1037 }%
1038 \def\XINT_xtrunc_Q #1%
1039 {%

```

[illegible]

[illegible]


```

1144 \romannumeral0\expandafter\XINT_dsx_zeroloop\expandafter
1145     {\the\numexpr \xint_c_ii^vi-\xintLength{#1}}}\Z {}#1%
1146 \XINT_xtrunc_A {#3}{#2}%
1147 }%
1148 \def\XINT_xtrunc_transition\fi
1149 \expandafter\XINT_xtrunc_B\expandafter #1#2#3#4%
1150 {%
1151 \fi
1152 \ifnum #4=\xint_c_ \XINT_xtrunc_abort\fi
1153 \expandafter\XINT_xtrunc_x\expandafter
1154     {\romannumeral0\XINT_dsx_zeroloop {#4}}\Z {#2}}{#3}{#4}%
1155 }%
1156 \def\XINT_xtrunc_x #1#2%
1157 {%
1158 \expandafter\XINT_xtrunc_y\romannumeral0#2{#1}%
1159 }%
1160 \def\XINT_xtrunc_y #1#2#3%
1161 {%
1162 \romannumeral0\expandafter\XINT_dsx_zeroloop\expandafter
1163     {\the\numexpr #3-\xintLength{#1}}}\Z {}#1%
1164 }%
1165 \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`, release 1.09a uses `\XINTdigits` without underscore.

```

1166 \mathchardef\XINTdigits 16
1167 \def\xintDigits #1#2%
1168     {\afterassignment \xint_gobble_i \mathchardef\XINTdigits=}%
1169 \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 `\xintiquo` without this overhead.

```

1170 \def\xintFloat {\romannumeral0\xintfloat }%
1171 \def\xintfloat #1{\XINT_float_chkopt #1\xint_relax }%
1172 \def\XINT_float_chkopt #1%
1173 {%
1174 \ifx [#1\expandafter\XINT_float_opt
1175 \else\expandafter\XINT_float_noopt
1176 \fi #1%
1177 }%
1178 \def\XINT_float_noopt #1\xint_relax
1179 {%
1180 \expandafter\XINT_float_a\expandafter\XINTdigits
1181 \romannumeral0\XINT_infrac {#1}\XINT_float_Q

```

```

1182 }%
1183 \def\XINT_float_opt [\xint_relax #1]#2%
1184 {%
1185   \expandafter\XINT_float_a\expandafter
1186   {\the\numexpr #1\expandafter}%
1187   \romannumeral0\XINT_infrac {#2}\XINT_float_Q
1188 }%
1189 \def\XINT_float_a #1#2#3% #1=P, #2=n, #3=A, #4=B
1190 {%
1191   \XINT_float_fork #3\Z {#1}{#2}% #1 = precision, #2=n
1192 }%
1193 \def\XINT_float_fork #1%
1194 {%
1195   \xint_UDzerominusfork
1196   #1-\XINT_float_zero
1197   0#1\XINT_float_J
1198   0-\XINT_float_K #1}%
1199   \krof
1200 }%
1201 \def\XINT_float_zero #1\Z #2#3#4#5{ 0.e0}%
1202 \def\XINT_float_J {\expandafter\xint_minus_thenstop\romannumeral0\XINT_float_K }%
1203 \def\XINT_float_K #1\Z #2% #1=A, #2=P, #3=n, #4=B
1204 {%
1205   \expandafter\XINT_float_L\expandafter
1206   {\the\numexpr\xintLength{#1}\expandafter}\expandafter
1207   {\the\numexpr #2+\xint_c_ii}{#1}{#2}%
1208 }%
1209 \def\XINT_float_L #1#2%
1210 {%
1211   \ifnum #1>#2
1212     \expandafter\XINT_float_Ma
1213   \else
1214     \expandafter\XINT_float_Mc
1215   \fi {#1}{#2}%
1216 }%
1217 \def\XINT_float_Ma #1#2#3%
1218 {%
1219   \expandafter\XINT_float_Mb\expandafter
1220   {\the\numexpr #1-#2\expandafter\expandafter\expandafter}%
1221   \expandafter\expandafter\expandafter
1222   {\expandafter\xint_firstoftwo
1223     \romannumeral0\XINT_split_fromleft_loop {#2}{#3}\W\W\W\W\W\W\W\W\Z
1224     }{#2}%
1225 }%
1226 \def\XINT_float_Mb #1#2#3#4#5#6% #2=A', #3=P+2, #4=P, #5=n, #6=B
1227 {%
1228   \expandafter\XINT_float_N\expandafter
1229   {\the\numexpr\xintLength{#6}\expandafter}\expandafter
1230   {\the\numexpr #3\expandafter}\expandafter
1231   {\the\numexpr #1+#5}%
1232   {#6}{#3}{#2}{#4}%
1233 }% long de B, P+2, n', B, |A'|=P+2, A', P

```

```

1234 \def\XINT_float_Mc #1#2#3#4#5#6%
1235 {%
1236   \expandafter\XINT_float_N\expandafter
1237   {\romannumeral0\xintlength{#6}}{#2}{#5}{#6}{#1}{#3}{#4}%
1238 }% long de B, P+2, n, B, |A|, A, P
1239 \def\XINT_float_N #1#2%
1240 {%
1241   \ifnum #1>#2
1242     \expandafter\XINT_float_0
1243   \else
1244     \expandafter\XINT_float_P
1245   \fi {#1}{#2}%
1246 }%
1247 \def\XINT_float_0 #1#2#3#4%
1248 {%
1249   \expandafter\XINT_float_P\expandafter
1250   {\the\numexpr #2\expandafter}\expandafter
1251   {\the\numexpr #2\expandafter}\expandafter
1252   {\the\numexpr #3-#1+#2\expandafter\expandafter\expandafter}%
1253   \expandafter\expandafter\expandafter
1254   {\expandafter\xint_firstoftwo
1255     \romannumeral0\XINT_split_fromleft_loop {#2}{#4\W\W\W\W\W\W\W\W\Z
1256     }%
1257 }% |B|,P+2,n,B,|A|,A,P
1258 \def\XINT_float_P #1#2#3#4#5#6#7#8%
1259 {%
1260   \expandafter #8\expandafter {\the\numexpr #1-#5+#2-\xint_c_i}%
1261   {#6}{#4}{#7}{#3}%
1262 }% |B|-|A|+P+1,A,B,P,n
1263 \def\XINT_float_Q #1%
1264 {%
1265   \ifnum #1<\xint_c_
1266     \expandafter\XINT_float_Ri
1267   \else
1268     \expandafter\XINT_float_Rii
1269   \fi {#1}%
1270 }%
1271 \def\XINT_float_Ri #1#2#3%
1272 {%
1273   \expandafter\XINT_float_Sa
1274   \romannumeral0\xintiique {#2}%
1275   {\XINT_dsx_addzerosnofuss {-#1}{#3}}\Z {#1}%
1276 }%
1277 \def\XINT_float_Rii #1#2#3%
1278 {%
1279   \expandafter\XINT_float_Sa
1280   \romannumeral0\xintiique
1281   {\XINT_dsx_addzerosnofuss {#1}{#2}}{#3}\Z {#1}%
1282 }%
1283 \def\XINT_float_Sa #1%
1284 {%
1285   \if #1%

```

```

1286     \xint_afterfi {\XINT_float_Sb\XINT_float_Wb }%
1287   \else
1288     \xint_afterfi {\XINT_float_Sb\XINT_float_Wa }%
1289   \fi #1%
1290 }%
1291 \def\XINT_float_Sb #1#2\Z #3#4%
1292 {%
1293   \expandafter\XINT_float_T\expandafter
1294   {\the\numexpr #4+\xint_c_i\expandafter}%
1295   \romannumeral-‘0\XINT_lenrord_loop 0{ }#2\Z\W\W\W\W\W\W\W\Z #1{#3}{#4}%
1296 }%
1297 \def\XINT_float_T #1#2#3%
1298 {%
1299   \ifnum #2>#1
1300     \xint_afterfi{\XINT_float_U\XINT_float_Xb}%
1301   \else
1302     \xint_afterfi{\XINT_float_U\XINT_float_Xa #3}%
1303   \fi
1304 }%
1305 \def\XINT_float_U #1#2%
1306 {%
1307   \ifnum #2<\xint_c_v
1308     \expandafter\XINT_float_Va
1309   \else
1310     \expandafter\XINT_float_Vb
1311   \fi #1%
1312 }%
1313 \def\XINT_float_Va #1#2\Z #3%
1314 {%
1315   \expandafter#1%
1316   \romannumeral0\expandafter\XINT_float_Wa
1317   \romannumeral0\XINT_rord_main { }#2%
1318   \xint_relax
1319   \xint_bye\xint_bye\xint_bye\xint_bye
1320   \xint_bye\xint_bye\xint_bye\xint_bye
1321   \xint_relax \Z
1322 }%
1323 \def\XINT_float_Vb #1#2\Z #3%
1324 {%
1325   \expandafter #1%
1326   \romannumeral0\expandafter #3%
1327   \romannumeral0\XINT_addm_A 0{ }1000\W\X\Y\Z #2000\W\X\Y\Z \Z
1328 }%
1329 \def\XINT_float_Wa #1{ #1.}%
1330 \def\XINT_float_Wb #1#2%
1331   {\if #11\xint_afterfi{ 10.}\else\xint_afterfi{ #1.#2}\fi }%
1332 \def\XINT_float_Xa #1\Z #2#3#4%
1333 {%
1334   \expandafter\XINT_float_Y\expandafter
1335   {\the\numexpr #3+#4-#2}{#1}%
1336 }%
1337 \def\XINT_float_Xb #1\Z #2#3#4%

```

```

1338 {%
1339   \expandafter\XINT_float_Y\expandafter
1340   {\the\numexpr #3+#4+\xint_c_i-#2}{#1}%
1341 }%
1342 \def\XINT_float_Y #1#2{ #2e#1}%

```

7.35 \xintPFloat

1.1

```

1343 \def\xintPFloat {\romannumeral0\xintpfloat }%
1344 \def\xintpfloat #1{\XINT_pfloat_chkopt #1\xint_relax }%
1345 \def\XINT_pfloat_chkopt #1%
1346 {%
1347   \ifx [#1\expandafter\XINT_pfloat_opt
1348     \else\expandafter\XINT_pfloat_noopt
1349   \fi #1%
1350 }%
1351 \def\XINT_pfloat_noopt #1\xint_relax
1352 {%
1353   \expandafter\XINT_pfloat_a\expandafter\XINTdigits
1354   \romannumeral0\XINTinfloat [\XINTdigits]{#1}%
1355 }%
1356 \def\XINT_pfloat_opt [\xint_relax #1]#2%
1357 {%
1358   \expandafter\XINT_pfloat_a\expandafter {\the\numexpr #1\expandafter}%
1359   \romannumeral0\XINTinfloat [\numexpr #1\relax]#2}%
1360 }%
1361 \def\XINT_pfloat_a #1#2%
1362 {%
1363   \xint_UDzerominusfork
1364                                     #2-\XINT_pfloat_zero
1365                                     0#2\XINT_pfloat_neg
1366                                     0-{\XINT_pfloat_pos #2}%
1367   \krof {#1}%
1368 }%
1369 \def\XINT_pfloat_zero #1[#2]{ 0}%
1370 \def\XINT_pfloat_neg
1371   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_pfloat_pos {}}%
1372 \def\XINT_pfloat_pos #1#2#3[#4]%
1373 {%
1374   \ifnum#4>0 \xint_dothis\XINT_pfloat_no\fi
1375   \ifnum#4>\numexpr-#2\relax \xint_dothis\XINT_pfloat_b\fi
1376   \ifnum#4>\numexpr-#2-\xint_c_v\relax \xint_dothis\XINT_pfloat_B\fi
1377   \xint_orthat\XINT_pfloat_no {#2}{#4}{#1#3}%
1378 }%
1379 \def\XINT_pfloat_no #1#2%
1380 {%
1381   \expandafter\XINT_pfloat_no_b\expandafter{\the\numexpr #2+#1-\xint_c_i\relax}%
1382 }%
1383 \def\XINT_pfloat_no_b #1#2{\XINT_pfloat_no_c #2e#1}%
1384 \def\XINT_pfloat_no_c #1{ #1.}%
1385 \def\XINT_pfloat_b #1#2#3%

```

```

1386 {\expandafter\XINT_pfloat_c
1387 \romannumeral0\expandafter\XINT_split_fromleft_loop
1388 \expandafter {\the\numexpr #1+#2-\xint_c_i}#3\W\W\W\W\W\W\W\W\Z }%
1389 \def\XINT_pfloat_c #1#2{ #1.#2}% #2 peut être vide
1390 \def\XINT_pfloat_B #1#2#3%
1391 {\expandafter\XINT_pfloat_C
1392 \romannumeral0\XINT_dsx_zeroloop {\numexpr -#1-#2}{}\Z {}#3}%
1393 \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.

```

1394 \def\XINTinFloat {\romannumeral0\XINTinfloat }%
1395 \def\XINTinfloat [#1]#2%
1396 {%
1397 \expandafter\XINT_infloat_a\expandafter
1398 {\the\numexpr #1\expandafter}%
1399 \romannumeral0\XINT_infrac {#2}\XINT_infloat_Q
1400 }%
1401 \def\XINT_infloat_a #1#2#3% #1=P, #2=n, #3=A, #4=B
1402 {%
1403 \XINT_infloat_fork #3\Z {#1}{#2}% #1 = precision, #2=n
1404 }%
1405 \def\XINT_infloat_fork #1%
1406 {%
1407 \xint_UDzerominusfork
1408 #1-\XINT_infloat_zero
1409 0#1\XINT_infloat_J
1410 0-\XINT_float_K #1}%
1411 \krof
1412 }%
1413 \def\XINT_infloat_zero #1\Z #2#3#4#5{ 0[0]}%
1414 % the 0[0] was stupidly changed to 0/1[0] in 1.09i, with the result that the
1415 % Float addition would crash when an operand was zero
1416 \def\XINT_infloat_J {\expandafter-\romannumeral0\XINT_float_K }%
1417 \def\XINT_infloat_Q #1%
1418 {%
1419 \ifnum #1<\xint_c_
1420 \expandafter\XINT_infloat_Ri
1421 \else
1422 \expandafter\XINT_infloat_Rii

```

```

1423   \fi {#1}%
1424 }%
1425 \def\XINT_infloat_Ri #1#2#3%
1426 {%
1427   \expandafter\XINT_infloat_S\expandafter
1428   {\romannumeral0\xintiigo {#2}%
1429     {\XINT_dsx_addzerosnofuss {-#1}{#3}}}{#1}%
1430 }%
1431 \def\XINT_infloat_Rii #1#2#3%
1432 {%
1433   \expandafter\XINT_infloat_S\expandafter
1434   {\romannumeral0\xintiigo
1435     {\XINT_dsx_addzerosnofuss {#1}{#2}}{#3}}{#1}%
1436 }%
1437 \def\XINT_infloat_S #1#2#3%
1438 {%
1439   \expandafter\XINT_infloat_T\expandafter
1440   {\the\numexpr #3+\xint_c_i\expandafter}%
1441   \romannumeral-'0\XINT_lenrord_loop 0{#1}Z\W\W\W\W\W\W\W\Z
1442   {#2}%
1443 }%
1444 \def\XINT_infloat_T #1#2#3%
1445 {%
1446   \ifnum #2>#1
1447     \xint_afterfi{\XINT_infloat_U\XINT_infloat_Wb}%
1448   \else
1449     \xint_afterfi{\XINT_infloat_U\XINT_infloat_Wa #3}%
1450   \fi
1451 }%
1452 \def\XINT_infloat_U #1#2%
1453 {%
1454   \ifnum #2<\xint_c_v
1455     \expandafter\XINT_infloat_Va
1456   \else
1457     \expandafter\XINT_infloat_Vb
1458   \fi #1%
1459 }%
1460 \def\XINT_infloat_Va #1#2\Z
1461 {%
1462   \expandafter#1%
1463   \romannumeral0\XINT_rord_main {#2}%
1464   \xint_relax
1465   \xint_bye\xint_bye\xint_bye\xint_bye
1466   \xint_bye\xint_bye\xint_bye\xint_bye
1467   \xint_relax \Z
1468 }%
1469 \def\XINT_infloat_Vb #1#2\Z
1470 {%
1471   \expandafter #1%
1472   \romannumeral0\XINT_addm_A 0{1000\W\X\Y\Z #2000\W\X\Y\Z \Z
1473 }%
1474 \def\XINT_infloat_Wa #1\Z #2#3%

```

```

1475 {%
1476   \expandafter\XINT_infloat_X\expandafter
1477   {\the\numexpr #3+\xint_c_i-#2}{#1}%
1478 }%
1479 \def\XINT_infloat_Wb #1\Z #2#3%
1480 {%
1481   \expandafter\XINT_infloat_X\expandafter
1482   {\the\numexpr #3+\xint_c_ii-#2}{#1}%
1483 }%
1484 \def\XINT_infloat_X #1#2{ #2[#1]}%

```

7.37 \xintAdd

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

```

1485 \def\xintAdd {\romannumeral0\xintadd }%
1486 \def\xintadd #1{\expandafter\xint_fadd\romannumeral0\xintra {#1}}%
1487 \def\xint_fadd #1{\xint_gob_til_zero #1\XINT_fadd_Azero 0\XINT_fadd_a #1}%
1488 \def\XINT_fadd_Azero #1]{\xintra }%
1489 \def\XINT_fadd_a #1/#2[#3]#4%
1490   {\expandafter\XINT_fadd_b\romannumeral0\xintra {#4}{#3}{#1}{#2}}%
1491 \def\XINT_fadd_b #1{\xint_gob_til_zero #1\XINT_fadd_Bzero 0\XINT_fadd_c #1}%
1492 \def\XINT_fadd_Bzero #1]#2#3#4{ #3/#4[#2]}%
1493 \def\XINT_fadd_c #1/#2[#3]#4%
1494 {%
1495   \expandafter\XINT_fadd_Aa\expandafter{\the\numexpr #4-#3}{#3}{#4}{#1}{#2}%
1496 }%
1497 \def\XINT_fadd_Aa #1%
1498 {%
1499   \ifcase\XINT_cntSgn #1\Z
1500     \expandafter\XINT_fadd_B
1501   \or
1502     \expandafter \XINT_fadd_Ba
1503   \else
1504     \expandafter \XINT_fadd_Bb
1505   \fi {#1}%
1506 }%
1507 \def\XINT_fadd_B #1#2#3#4#5#6#7{\XINT_fadd_C {#4}{#5}{#7}{#6}[#3]}%
1508 \def\XINT_fadd_Ba #1#2#3#4#5#6#7%
1509 {%
1510   \expandafter\XINT_fadd_C\expandafter
1511   {\romannumeral0\XINT_dsx_zeroloop {#1}{}\Z {#6}}%
1512   {#7}{#5}{#4}[#2]}%
1513 }%
1514 \def\XINT_fadd_Bb #1#2#3#4#5#6#7%
1515 {%
1516   \expandafter\XINT_fadd_C\expandafter
1517   {\romannumeral0\XINT_dsx_zeroloop {-#1}{}\Z {#4}}%
1518   {#5}{#7}{#6}[#3]}%
1519 }%
1520 \def\XINT_fadd_C #1#2#3%
1521 {%
1522   \ifcase\romannumeral0\XINT_cmp_pre {#2}{#3} %<- intentional space here.

```


7 Package *xintfrac* implementation

```

1523     \expandafter\XINT_fadd_eq
1524     \or\expandafter\XINT_fadd_D
1525     \else\expandafter\XINT_fadd_Da
1526     \fi {#2}{#3}{#1}%
1527 }%
1528 \def\XINT_fadd_eq #1#2#3#4#5%
1529 {%
1530     \expandafter\XINT_fadd_G
1531     \romannumeral0\xintiiadd {#3}{#4}/#1[#5]%
1532 }%
1533 \def\XINT_fadd_D #1#2%
1534 {%
1535     \expandafter\XINT_fadd_E\romannumeral0\XINT_div_prepare {#2}{#1}{#1}{#2}%
1536 }%
1537 \def\XINT_fadd_E #1#2%
1538 {%
1539     \if0\XINT_Sgn #2\Z
1540         \expandafter\XINT_fadd_F
1541     \else\expandafter\XINT_fadd_K
1542     \fi {#1}%
1543 }%
1544 \def\XINT_fadd_F #1#2#3#4#5#6%
1545 {%
1546     \expandafter\XINT_fadd_G
1547     \romannumeral0\xintiiadd {\xintiiMul {#5}{#1}}{#4}/#2[#6]%
1548 }%
1549 \def\XINT_fadd_Da #1#2%
1550 {%
1551     \expandafter\XINT_fadd_Ea\romannumeral0\XINT_div_prepare {#1}{#2}{#1}{#2}%
1552 }%
1553 \def\XINT_fadd_Ea #1#2%
1554 {%
1555     \if0\XINT_Sgn #2\Z
1556         \expandafter\XINT_fadd_Fa
1557     \else\expandafter\XINT_fadd_K
1558     \fi {#1}%
1559 }%
1560 \def\XINT_fadd_Fa #1#2#3#4#5#6%
1561 {%
1562     \expandafter\XINT_fadd_G
1563     \romannumeral0\xintiiadd {\xintiiMul {#4}{#1}}{#5}/#3[#6]%
1564 }%
1565 \def\XINT_fadd_G #1{\if0#1\XINT_fadd_iszero\fi\space #1}%
1566 \def\XINT_fadd_K #1#2#3#4#5%
1567 {%
1568     \expandafter\XINT_fadd_L
1569     \romannumeral0\xintiiadd {\xintiiMul {#2}{#5}}{\xintiiMul {#3}{#4}}.%
1570     {{#2}{#3}}%
1571 }%
1572 \def\XINT_fadd_L #1{\if0#1\XINT_fadd_iszero\fi \XINT_fadd_M #1}%
1573 \def\XINT_fadd_M #1.#2{\expandafter\XINT_fadd_N \expandafter
1574     {\romannumeral0\xintiimul #2}{#1}}%

```

```

1575 \def\XINT_fadd_N #1#2{ #2/#1}%
1576 \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.

```

1577 \def\xintSub {\romannumeral0\xintsub }%
1578 \def\xintsub #1{\expandafter\xint_fsub\romannumeral0\xintra {#1}}%
1579 \def\xint_fsub #1{\xint_gob_til_zero #1\XINT_fsub_Azero 0\XINT_fsub_a #1}%
1580 \def\XINT_fsub_Azero #1{\xintopp }%
1581 \def\XINT_fsub_a #1/#2[#3]#4%
1582   {\expandafter\XINT_fsub_b\romannumeral0\xintra {#4}{#3}{#1}{#2}}%
1583 \def\XINT_fsub_b #1{\xint_UDzerominusfork
1584   #1-\XINT_fadd_Bzero
1585   0#1\XINT_fadd_c
1586   0-\XINT_fadd_c -#1}%
1587   \krof }%

```

7.39 \xintSum

```

1588 \def\xintSum {\romannumeral0\xintsum }%
1589 \def\xintsum #1{\xintsumexpr #1\relax }%
1590 \def\xintSumExpr {\romannumeral0\xintsumexpr }%
1591 \def\xintsumexpr {\expandafter\XINT_fsumexpr\romannumeral-'0}%
1592 \def\XINT_fsumexpr {\XINT_fsum_loop_a {0/1[0]}}%
1593 \def\XINT_fsum_loop_a #1#2%
1594 {%
1595   \expandafter\XINT_fsum_loop_b \romannumeral-'0#2\Z {#1}%
1596 }%
1597 \def\XINT_fsum_loop_b #1%
1598 {%
1599   \xint_gob_til_relax #1\XINT_fsum_finished\relax
1600   \XINT_fsum_loop_c #1%
1601 }%
1602 \def\XINT_fsum_loop_c #1\Z #2%
1603 {%
1604   \expandafter\XINT_fsum_loop_a\expandafter{\romannumeral0\xintadd {#2}{#1}}%
1605 }%
1606 \def\XINT_fsum_finished #1\Z #2{ #2}%

```

7.40 \xintMul

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

```

1607 \def\xintMul {\romannumeral0\xintmul }%
1608 \def\xintmul #1{\expandafter\xint_fmula\romannumeral0\xintra {#1}.}%
1609 \def\xint_fmula #1{\xint_gob_til_zero #1\XINT_fmula_zero 0\XINT_fmula_a #1}%
1610 \def\XINT_fmula_a #1[#2].#3%
1611   {\expandafter\XINT_fmula_b\romannumeral0\xintra {#3}#1[#2.]}%
1612 \def\XINT_fmula_b #1{\xint_gob_til_zero #1\XINT_fmula_zero 0\XINT_fmula_c #1}%
1613 \def\XINT_fmula_c #1/#2[#3]#4/#5[#6.]%
1614 {%
1615   \expandafter\XINT_fmula_d

```

```

1616 \expandafter{\the\numexpr #3+#6\expandafter}%
1617 \expandafter{\romannumeral0\xintiimul {#5}{#2}}%
1618 {\romannumeral0\xintiimul {#4}{#1}}%
1619 }%
1620 \def\XINT_fmulo_d #1#2#3%
1621 {%
1622 \expandafter \XINT_fmulo_e \expandafter{#3}{#1}{#2}%
1623 }%
1624 \def\XINT_fmulo_e #1#2{\XINT_outfrac {#2}{#1}}%
1625 \def\XINT_fmulo_zero #1.#2{ 0/1[0]}%

```

7.41 \xintSqr

1.1 modifs comme xintMul

```

1626 \def\xintSqr {\romannumeral0\xintsqr }%
1627 \def\xintsqr #1{\expandafter\xint_fsqr\romannumeral0\xintra {#1}}%
1628 \def\xint_fsqr #1{\xint_gob_til_zero #1\XINT_fsqr_zero 0\XINT_fsqr_a #1}%
1629 \def\xint_fsqr_a #1/#2[#3]%
1630 {%
1631 \expandafter\XINT_fsqr_b
1632 \expandafter{\the\numexpr #3+#3\expandafter}%
1633 \expandafter{\romannumeral0\xintiisqr {#2}}%
1634 {\romannumeral0\xintiisqr {#1}}%
1635 }%
1636 \def\XINT_fsqr_b #1#2#3{\expandafter \XINT_fmulo_e \expandafter{#3}{#1}{#2}}%
1637 \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.

```

1638 \def\xintPow {\romannumeral0\xintpow }%
1639 \def\xintpow #1%
1640 {%
1641 \expandafter\xint_fpow\expandafter {\romannumeral0\XINT_infrac {#1}}%
1642 }%
1643 \def\xint_fpow #1#2%
1644 {%
1645 \expandafter\XINT_fpow_fork\the\numexpr \xintNum{#2}\relax\Z #1%
1646 }%
1647 \def\XINT_fpow_fork #1#2\Z
1648 {%
1649 \xint_UDzerominusfork
1650 #1-\XINT_fpow_zero
1651 0#1\XINT_fpow_neg
1652 0-{\XINT_fpow_pos #1}%

```

```

1653 \krof
1654 {#2}%
1655 }%
1656 \def\XINT_fpow_zero #1#2#3#4{ 1/1[0]}%
1657 \def\XINT_fpow_pos #1#2#3#4#5%
1658 {%
1659 \expandafter\XINT_fpow_pos_A\expandafter
1660 {\the\numexpr #1#2*#3\expandafter}\expandafter
1661 {\romannumeral0\xintiipow {#5}{#1#2}}%
1662 {\romannumeral0\xintiipow {#4}{#1#2}}%
1663 }%
1664 \def\XINT_fpow_neg #1#2#3#4%
1665 {%
1666 \expandafter\XINT_fpow_pos_A\expandafter
1667 {\the\numexpr -#1*#2\expandafter}\expandafter
1668 {\romannumeral0\xintiipow {#3}{#1}}%
1669 {\romannumeral0\xintiipow {#4}{#1}}%
1670 }%
1671 \def\XINT_fpow_pos_A #1#2#3%
1672 {%
1673 \expandafter\XINT_fpow_pos_B\expandafter {#3}{#1}{#2}%
1674 }%
1675 \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`.

```

1676 \def\xintFac {\romannumeral0\xintfac }%
1677 \def\xintfac #1%
1678 {%
1679 \expandafter\XINT_fac_fork\expandafter{\the\numexpr \xintNum{#1}}%
1680 }%

```

7.44 \xintPrd

```

1681 \def\xintPrd {\romannumeral0\xintprd }%
1682 \def\xintprd #1{\xintprdexpr #1\relax }%
1683 \def\xintPrdExpr {\romannumeral0\xintprdexpr }%
1684 \def\xintprdexpr {\expandafter\XINT_fprdexpr \romannumeral-'0}%
1685 \def\XINT_fprdexpr {\XINT_fprod_loop_a {1/1[0]}}%
1686 \def\XINT_fprod_loop_a #1#2%
1687 {%
1688 \expandafter\XINT_fprod_loop_b \romannumeral-'0#2\Z {#1}%
1689 }%
1690 \def\XINT_fprod_loop_b #1%
1691 {%
1692 \xint_gob_til_relax #1\XINT_fprod_finished\relax
1693 \XINT_fprod_loop_c #1%
1694 }%

```

```

1695 \def\XINT_fprod_loop_c #1\Z #2%
1696 {%
1697   \expandafter\XINT_fprod_loop_a\expandafter{\romannumeral0\xintmul {#1}{#2}}%
1698 }%
1699 \def\XINT_fprod_finished #1\Z #2{ #2}%

```

7.45 \xintDiv

```

1700 \def\xintDiv {\romannumeral0\xintdiv }%
1701 \def\xintdiv #1%
1702 {%
1703   \expandafter\xint_fdiv\expandafter {\romannumeral0\XINT_infrac {#1}}%
1704 }%
1705 \def\xint_fdiv #1#2%
1706   {\expandafter\XINT_fdiv_A\romannumeral0\XINT_infrac {#2}{#1}}%
1707 \def\XINT_fdiv_A #1#2#3#4#5#6%
1708 {%
1709   \expandafter\XINT_fdiv_B
1710   \expandafter{\the\numexpr #4-#1\expandafter}%
1711   \expandafter{\romannumeral0\xintiimul {#2}{#6}}%
1712   {\romannumeral0\xintiimul {#3}{#5}}%
1713 }%
1714 \def\XINT_fdiv_B #1#2#3%
1715 {%
1716   \expandafter\XINT_fdiv_C
1717   \expandafter{#3}{#1}{#2}%
1718 }%
1719 \def\XINT_fdiv_C #1#2{\XINT_outfrac {#2}{#1}}%

```

7.46 \xintDivFloor

1.1

```

1720 \def\xintDivFloor {\romannumeral0\xintdivfloor }%
1721 \def\xintdivfloor #1#2{\xintfloor{\xintDiv {#1}{#2}}}%

```

7.47 \xintDivTrunc

1.1

```

1722 \def\xintDivTrunc {\romannumeral0\xintdivtrunc }%
1723 \def\xintdivtrunc #1#2{\xintitrunc 0{\xintDiv {#1}{#2}}}%

```

7.48 \xintDivRound

1.1

```

1724 \def\xintDivRound {\romannumeral0\xintdivround }%
1725 \def\xintdivround #1#2{\xintiround 0{\xintDiv {#1}{#2}}}%

```

7.49 \xintMod

1.1. `\xintMod {q1}{q2}` computes $q2 * t(q1/q2)$ with $t(q1/q2)$ truncated division of two arbitrary fractions $q1$ and $q2$. We put some efforts into minimizing the amount of computations.

```

1726 \def\xintMod {\romannumeral0\xintmod }%
1727 \def\xintmod #1{\expandafter\XINT_mod_a\romannumeral0\xintra{#1}.}%
1728 \def\XINT_mod_a #1#2.#3%
1729   {\expandafter\XINT_mod_b\expandafter #1\romannumeral0\xintra{#3}#2.}%
1730 \def\XINT_mod_b #1#2% #1 de A, #2 de B.
1731 {%
1732   \if0#2\xint_dothis\XINT_mod_divbyzero\fi
1733   \if0#1\xint_dothis\XINT_mod_aiszero\fi
1734   \if-#2\xint_dothis{\XINT_mod_bneg #1}\fi
1735   \xint_orthat{\XINT_mod_bpos #1#2}%
1736 }%
1737 \def\XINT_mod_bpos #1%
1738 {%
1739   \xint_UDsignfork
1740     #1{\xintiiopp\XINT_mod_pos {}}%
1741     -{\XINT_mod_pos #1}%
1742   \krof
1743 }%
1744 \def\XINT_mod_bneg #1%
1745 {%
1746   \xint_UDsignfork
1747     #1{\xintiiopp\XINT_mod_pos {}}%
1748     -{\XINT_mod_pos #1}%
1749   \krof
1750 }%
1751 \def\XINT_mod_divbyzero #1.{\xintError:DivisionByZero\space 0/1[0]}%
1752 \def\XINT_mod_aiszero #1.{ 0/1[0]}%
1753 \def\XINT_mod_pos #1#2/#3[#4]#5/#6[#7].%
1754 {%
1755   \expandafter\XINT_mod_pos_a
1756   \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.\expandafter
1757   {\romannumeral0\xintiimul {#6}{#3}}%      n fois u
1758   {\xintiiE{\xintiiMul {#1#5}{#3}}{#7-#4}}% m fois u
1759   {\xintiiE{\xintiiMul {#2}{#6}}{#4-#7}}%   t fois n
1760 }%
1761 \def\XINT_mod_pos_a #1.#2#3#4{\xintiirem {#3}{#4}/#2[#1]}%

```

7.50 \XINTinFloatMod

Pour emploi dans *xintexpr* 1.1

```

1762 \def\XINTinFloatMod {\romannumeral0\XINTinfloatmod [\XINTdigits]}%
1763 \def\XINTinfloatmod [#1]#2#3{\expandafter\XINT_infloatmod\expandafter
1764   {\romannumeral0\XINTinfloat[#1]{#2}}%
1765   {\romannumeral0\XINTinfloat[#1]{#3}}{#1}}%
1766 \def\XINT_infloatmod #1#2{\expandafter\XINT_infloatmod_a\expandafter {#2}{#1}}%
1767 \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.

```

1768 \def\xintIsOne {\romannumeral0\xintisone }%
1769 \def\xintisone #1{\expandafter\XINT_fracisone
1770      \romannumeral0\xintrawithzeros{#1}\Z }%
1771 \def\XINT_fracisone #1/#2\Z
1772      {\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.

```

1773 \def\xintGeq {\romannumeral0\xintgeq }%
1774 \def\xintgeq #1%
1775 {%
1776     \expandafter\xint_fgeq\expandafter {\romannumeral0\xintabs {#1}}%
1777 }%
1778 \def\xint_fgeq #1#2%
1779 {%
1780     \expandafter\XINT_fgeq_A \romannumeral0\xintabs {#2}#1%
1781 }%
1782 \def\XINT_fgeq_A #1%
1783 {%
1784     \xint_gob_til_zero #1\XINT_fgeq_Zii 0%
1785     \XINT_fgeq_B #1%
1786 }%
1787 \def\XINT_fgeq_Zii 0\XINT_fgeq_B #1[#2]#3[#4]{ 1}%
1788 \def\XINT_fgeq_B #1/#2[#3]#4#5/#6[#7]%
1789 {%
1790     \xint_gob_til_zero #4\XINT_fgeq_Zi 0%
1791     \expandafter\XINT_fgeq_C\expandafter
1792     {\the\numexpr #7-#3\expandafter}\expandafter
1793     {\romannumeral0\xintiimul {#4#5}{#2}}%
1794     {\romannumeral0\xintiimul {#6}{#1}}%
1795 }%
1796 \def\XINT_fgeq_Zi 0#1#2#3#4#5#6#7{ 0}%
1797 \def\XINT_fgeq_C #1#2#3%
1798 {%
1799     \expandafter\XINT_fgeq_D\expandafter
1800     {#3}{#1}{#2}%
1801 }%
1802 \def\XINT_fgeq_D #1#2#3%
1803 {%
1804     \expandafter\XINT_cntSgnFork\romannumeral-‘0\expandafter\XINT_cntSgn
1805     \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\Z
1806     { 0}{\XINT_fgeq_E #2\Z {#3}{#1}}{ 1}%
1807 }%
1808 \def\XINT_fgeq_E #1%
1809 {%
1810     \xint_UDsignfork
1811     #1\XINT_fgeq_Fd
1812     -{\XINT_fgeq_Fn #1}%
1813     \krof
1814 }%
1815 \def\XINT_fgeq_Fd #1\Z #2#3%

```

```

1816 {%
1817   \expandafter\XINT_fgeq_Fe\expandafter
1818   {\romannumeral0\XINT_dsx_addzerosnofuss {#1}{#3}}{#2}%
1819 }%
1820 \def\XINT_fgeq_Fe #1#2{\XINT_geq_pre {#2}{#1}}%
1821 \def\XINT_fgeq_Fn #1\Z #2#3%
1822 {%
1823   \expandafter\XINT_geq_pre\expandafter
1824   {\romannumeral0\XINT_dsx_addzerosnofuss {#1}{#2}}{#3}%
1825 }%

```

7.53 \xintMax

Rewritten completely in 1.08a.

```

1826 \def\xintMax {\romannumeral0\xintmax }%
1827 \def\xintmax #1%
1828 {%
1829   \expandafter\xint_fmax\expandafter {\romannumeral0\xintra #1}%
1830 }%
1831 \def\xint_fmax #1#2%
1832 {%
1833   \expandafter\XINT_fmax_A\romannumeral0\xintra #2}%1%
1834 }%
1835 \def\XINT_fmax_A #1#2/#3[#4]#5#6/#7[#8]%
1836 {%
1837   \xint_UDsignsfork
1838     #1#5\XINT_fmax_minusminus
1839     -#5\XINT_fmax_firstneg
1840     #1-\XINT_fmax_secondneg
1841     --\XINT_fmax_nonneg_a
1842   \krof
1843   #1#5{#2/#3[#4]}{#6/#7[#8]}%
1844 }%
1845 \def\XINT_fmax_minusminus --%
1846   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_fmin_nonneg_b }%
1847 \def\XINT_fmax_firstneg #1-#2#3{ #1#2}%
1848 \def\XINT_fmax_secondneg -#1#2#3{ #1#3}%
1849 \def\XINT_fmax_nonneg_a #1#2#3#4%
1850 {%
1851   \XINT_fmax_nonneg_b {#1#3}{#2#4}%
1852 }%
1853 \def\XINT_fmax_nonneg_b #1#2%
1854 {%
1855   \if0\romannumeral0\XINT_fgeq_A #1#2%
1856     \xint_afterfi{ #1}%
1857   \else \xint_afterfi{ #2}%
1858   \fi
1859 }%

```

7.54 \xintMaxof

```

1860 \def\xintMaxof {\romannumeral0\xintmaxof }%

```



```

1861 \def\xintmaxof    #1{\expandafter\XINT_maxof_a\romannumeral-‘0#1\relax }%
1862 \def\XINT_maxof_a #1{\expandafter\XINT_maxof_b\romannumeral0\xintra{#1}\Z }%
1863 \def\XINT_maxof_b #1\Z #2%
1864     {\expandafter\XINT_maxof_c\romannumeral-‘0#2\Z {#1}\Z}%
1865 \def\XINT_maxof_c #1%
1866     {\xint_gob_til_relax #1\XINT_maxof_e\relax\XINT_maxof_d #1}%
1867 \def\XINT_maxof_d #1\Z
1868     {\expandafter\XINT_maxof_b\romannumeral0\xintmax {#1}}%
1869 \def\XINT_maxof_e #1\Z #2\Z { #2}%

```

7.55 \xintMin

Rewritten completely in 1.08a.

```

1870 \def\xintMin {\romannumeral0\xintmin }%
1871 \def\xintmin #1%
1872 {%
1873     \expandafter\xint_fmin\expandafter {\romannumeral0\xintra{#1}}%
1874 }%
1875 \def\xint_fmin #1#2%
1876 {%
1877     \expandafter\XINT_fmin_A\romannumeral0\xintra{#2}#1%
1878 }%
1879 \def\XINT_fmin_A #1#2/#3[#4]#5#6/#7[#8]%
1880 {%
1881     \xint_UDsignsfork
1882     #1#5\XINT_fmin_minusminus
1883     -#5\XINT_fmin_firstneg
1884     #1-\XINT_fmin_secondneg
1885     --\XINT_fmin_nonneg_a
1886     \krof
1887     #1#5{#2/#3[#4]}{#6/#7[#8]}%
1888 }%
1889 \def\XINT_fmin_minusminus --%
1890     {\expandafter\xint_minus_thenstop\romannumeral0\XINT_fmax_nonneg_b }%
1891 \def\XINT_fmin_firstneg #1-#2#3{ -#3}%
1892 \def\XINT_fmin_secondneg -#1#2#3{ -#2}%
1893 \def\XINT_fmin_nonneg_a #1#2#3#4%
1894 {%
1895     \XINT_fmin_nonneg_b {#1#3}{#2#4}%
1896 }%
1897 \def\XINT_fmin_nonneg_b #1#2%
1898 {%
1899     \if0\romannumeral0\XINT_fgeq_A #1#2%
1900         \xint_afterfi{ #2}%
1901     \else \xint_afterfi{ #1}%
1902     \fi
1903 }%

```

7.56 \xintMinof

```

1904 \def\xintMinof    {\romannumeral0\xintminof }%
1905 \def\xintminof    #1{\expandafter\XINT_minof_a\romannumeral-‘0#1\relax }%
1906 \def\XINT_minof_a #1{\expandafter\XINT_minof_b\romannumeral0\xintra{#1}\Z }%

```

```

1907 \def\XINT_minof_b #1\Z #2%
1908     {\expandafter\XINT_minof_c\romannumeral-‘0#2\Z {#1}\Z}%
1909 \def\XINT_minof_c #1%
1910     {\xint_gob_til_relax #1\XINT_minof_e\relax\XINT_minof_d #1}%
1911 \def\XINT_minof_d #1\Z
1912     {\expandafter\XINT_minof_b\romannumeral0\xintmin {#1}}%
1913 \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. Incredibly, it seems that 1.08b introduced a bug in delimited arguments making the macro just non-functional when one of the input was zero! I did not detect this until working on release 1.09a, somehow I had not tested that \xintCmp just did NOT work! I must have done some last minute change...

```

1914 %\def\xintCmp {\romannumeral0\xintcmp }%
1915 \def\xintcmp #1%
1916 {%
1917     \expandafter\xint_fcmp\expandafter {\romannumeral0\xintra {#1}}%
1918 }%
1919 \def\xint_fcmp #1#2%
1920 {%
1921     \expandafter\XINT_fcmp_A\romannumeral0\xintra {#2}#1%
1922 }%
1923 \def\XINT_fcmp_A #1#2/#3[#4]#5#6/#7[#8]%
1924 {%
1925     \xint_UDsignsfork
1926     #1#5\XINT_fcmp_minusminus
1927     -#5\XINT_fcmp_firstneg
1928     #1-\XINT_fcmp_secondneg
1929     --\XINT_fcmp_nonneg_a
1930     \krof
1931     #1#5{#2/#3[#4]}{#6/#7[#8]}%
1932 }%
1933 \def\XINT_fcmp_minusminus --#1#2{\XINT_fcmp_B #2#1}%
1934 \def\XINT_fcmp_firstneg #1-#2#3{ -1}%
1935 \def\XINT_fcmp_secondneg -#1#2#3{ 1}%
1936 \def\XINT_fcmp_nonneg_a #1#2%
1937 {%
1938     \xint_UDzerosfork
1939     #1#2\XINT_fcmp_zerozero
1940     0#2\XINT_fcmp_firstzero
1941     #10\XINT_fcmp_secondzero
1942     00\XINT_fcmp_pos
1943     \krof
1944     #1#2%
1945 }%
1946 \def\XINT_fcmp_zerozero #1#2#3#4{ 0}% 1.08b had some [ and ] here!!!
1947 \def\XINT_fcmp_firstzero #1#2#3#4{ -1}% incredibly I never saw that until
1948 \def\XINT_fcmp_secondzero #1#2#3#4{ 1}% preparing 1.09a.
1949 \def\XINT_fcmp_pos #1#2#3#4%
1950 {%
1951     \XINT_fcmp_B #1#3#2#4%

```

```

1952 }%
1953 \def\XINT_fcmp_B #1/#2[#3]#4/#5[#6]%
1954 {%
1955   \expandafter\XINT_fcmp_C\expandafter
1956   {\the\numexpr #6-#3\expandafter}\expandafter
1957   {\romannumeral0\xintiimul {#4}{#2}}%
1958   {\romannumeral0\xintiimul {#5}{#1}}%
1959 }%
1960 \def\XINT_fcmp_C #1#2#3%
1961 {%
1962   \expandafter\XINT_fcmp_D\expandafter
1963   {#3}{#1}{#2}%
1964 }%
1965 \def\XINT_fcmp_D #1#2#3%
1966 {%
1967   \expandafter\XINT_cntSgnFork\romannumeral-'0\expandafter\XINT_cntSgn
1968   \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\Z
1969   { -1}{\XINT_fcmp_E #2\Z {#3}{#1}}{ 1}%
1970 }%
1971 \def\XINT_fcmp_E #1%
1972 {%
1973   \xint_UDsignfork
1974   #1\XINT_fcmp_Fd
1975   -{\XINT_fcmp_Fn #1}%
1976   \krof
1977 }%
1978 \def\XINT_fcmp_Fd #1\Z #2#3%
1979 {%
1980   \expandafter\XINT_fcmp_Fe\expandafter
1981   {\romannumeral0\XINT_dsx_addzerosnofuss {#1}{#3}}{#2}%
1982 }%
1983 \def\XINT_fcmp_Fe #1#2{\XINT_cmp_pre {#2}{#1}}%
1984 \def\XINT_fcmp_Fn #1\Z #2#3%
1985 {%
1986   \expandafter\XINT_cmp_pre\expandafter
1987   {\romannumeral0\XINT_dsx_addzerosnofuss {#1}{#2}}{#3}%
1988 }%

```

7.58 \xintAbs

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

```

1989 \def\xintAbs {\romannumeral0\xintabs }%
1990 \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 was written before \xintRaw)

```

1991 \def\xintOpp {\romannumeral0\xintopp }%
1992 \def\xintopp #1{\expandafter\XINT_opp\romannumeral0\xintraw {#1}}%

```

7.60 `\xintSgn`

Simplified in 1.09i. (original macro was written before `\xintRaw`)

```

1993 \def\xintSgn    {\romannumeral0\xintsgn }%
1994 \def\xintsgn #1{\expandafter\XINT_sgn\romannumeral0\xintraw {#1}\Z }%

```

7.61 `\xintFloatAdd`, `\XINTinFloatAdd`

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

```

1995 \def\xintFloatAdd    {\romannumeral0\xintfloatadd }%
1996 \def\xintfloatadd    #1{\XINT_fladd_chkopt \xintfloat #1\xint_relax }%
1997 \def\XINTinFloatAdd  {\romannumeral0\XINTinfloatadd }%
1998 \def\XINTinfloatadd  #1{\XINT_fladd_chkopt \XINTinfloat #1\xint_relax }%
1999 \def\XINT_fladd_chkopt #1#2%
2000 {%
2001   \ifx [#2\expandafter\XINT_fladd_opt
2002     \else\expandafter\XINT_fladd_noopt
2003   \fi  #1#2%
2004 }%
2005 \def\XINT_fladd_noopt #1#2\xint_relax #3%
2006 {%
2007   #1[\XINTdigits]{\XINT_FL_Add {\XINTdigits+\xint_c_ii}{#2}{#3}}%
2008 }%
2009 \def\XINT_fladd_opt #1[\xint_relax #2]#3#4%
2010 {%
2011   #1[#2]{\XINT_FL_Add {#2+\xint_c_ii}{#3}{#4}}%
2012 }%
2013 \def\XINT_FL_Add #1#2%
2014 {%
2015   \expandafter\XINT_FL_Add_a\expandafter{\the\numexpr #1\expandafter}%
2016   \expandafter{\romannumeral0\XINTinfloat [#1]{#2}}%
2017 }%
2018 \def\XINT_FL_Add_a #1#2#3%
2019 {%
2020   \expandafter\XINT_FL_Add_b\romannumeral0\XINTinfloat [#1]{#3}#2{#1}%
2021 }%
2022 \def\XINT_FL_Add_b #1%
2023 {%
2024   \xint_gob_til_zero #1\XINT_FL_Add_zero 0\XINT_FL_Add_c #1%
2025 }%
2026 \def\XINT_FL_Add_c #1[#2]#3%
2027 {%
2028   \xint_gob_til_zero #3\XINT_FL_Add_zerobis 0\XINT_FL_Add_d #1[#2]#3%
2029 }%
2030 \def\XINT_FL_Add_d #1[#2]#3[#4]#5%
2031 {%
2032   \ifnum \numexpr #2-#4-#5>\xint_c_i
2033     \expandafter \xint_secondofthree_thenstop
2034   \else
2035     \ifnum \numexpr #4-#2-#5>\xint_c_i
2036       \expandafter\expandafter\expandafter\xint_thirdofthree_thenstop

```

```

2037     \fi
2038   \fi
2039   \xintadd {#1[#2]}{#3[#4]}%
2040 }%
2041 \def\XINT_FL_Add_zero 0\XINT_FL_Add_c 0[0]#1[#2]#3{#1[#2]}%
2042 \def\XINT_FL_Add_zerobis 0\XINT_FL_Add_d #1[#2]0[0]#3{#1[#2]}%

```

7.62 \xintFloatSub, \XINTinFloatSub

1.07

```

2043 \def\xintFloatSub {\romannumeral0\xintfloatsub }%
2044 \def\xintfloatsub #1{\XINT_flsb_chkopt \xintfloat #1\xint_relax }%
2045 \def\XINTinFloatSub {\romannumeral0\XINTinfloatsub }%
2046 \def\XINTinfloatsub #1{\XINT_flsb_chkopt \XINTinfloat #1\xint_relax }%
2047 \def\XINT_flsb_chkopt #1#2%
2048 {%
2049   \ifx [#2\expandafter\XINT_flsb_opt
2050     \else\expandafter\XINT_flsb_noopt
2051   \fi #1#2%
2052 }%
2053 \def\XINT_flsb_noopt #1#2\xint_relax #3%
2054 {%
2055   #1[\XINTdigits]{\XINT_FL_Add {\XINTdigits+\xint_c_ii}{#2}{\xintOpp{#3}}}%
2056 }%
2057 \def\XINT_flsb_opt #1[\xint_relax #2]#3#4%
2058 {%
2059   #1[#2]{\XINT_FL_Add {#2+\xint_c_ii}{#3}{\xintOpp{#4}}}%
2060 }%

```

7.63 \xintFloatMul, \XINTinFloatMul

1.07

```

2061 \def\xintFloatMul {\romannumeral0\xintfloatmul}%
2062 \def\xintfloatmul #1{\XINT_flmul_chkopt \xintfloat #1\xint_relax }%
2063 \def\XINTinFloatMul {\romannumeral0\XINTinfloatmul }%
2064 \def\XINTinfloatmul #1{\XINT_flmul_chkopt \XINTinfloat #1\xint_relax }%
2065 \def\XINT_flmul_chkopt #1#2%
2066 {%
2067   \ifx [#2\expandafter\XINT_flmul_opt
2068     \else\expandafter\XINT_flmul_noopt
2069   \fi #1#2%
2070 }%
2071 \def\XINT_flmul_noopt #1#2\xint_relax #3%
2072 {%
2073   #1[\XINTdigits]{\XINT_FL_Mul {\XINTdigits+\xint_c_ii}{#2}{#3}}%
2074 }%
2075 \def\XINT_flmul_opt #1[\xint_relax #2]#3#4%
2076 {%
2077   #1[#2]{\XINT_FL_Mul {#2+\xint_c_ii}{#3}{#4}}%
2078 }%
2079 \def\XINT_FL_Mul #1#2%

```

```

2080 {%
2081   \expandafter\XINT_FL_Mul_a\expandafter{\the\numexpr #1\expandafter}%
2082   \expandafter{\romannumeral0\XINTinfloat [#1]{#2}}%
2083 }%
2084 \def\XINT_FL_Mul_a #1#2#3%
2085 {%
2086   \expandafter\XINT_FL_Mul_b\romannumeral0\XINTinfloat [#1]{#3}#2%
2087 }%
2088 \def\XINT_FL_Mul_b #1[#2]#3[#4]{\xintE{\xintiiMul {#1}{#3}}{#2+#4}}%

```

7.64 \xintFloatDiv, \XINTinFloatDiv

1.07

```

2089 \def\xintFloatDiv    {\romannumeral0\xintfloatdiv}%
2090 \def\xintfloatdiv    #1{\XINT_fldiv_chkopt \xintfloat #1\xint_relax }%
2091 \def\XINTinFloatDiv  {\romannumeral0\XINTinfloatdiv }%
2092 \def\XINTinfloatdiv  #1{\XINT_fldiv_chkopt \XINTinfloat #1\xint_relax }%
2093 \def\XINT_fldiv_chkopt #1#2%
2094 {%
2095   \ifx [#2\expandafter\XINT_fldiv_opt
2096   \else\expandafter\XINT_fldiv_noopt
2097   \fi #1#2%
2098 }%
2099 \def\XINT_fldiv_noopt #1#2\xint_relax #3%
2100 {%
2101   #1[\XINTdigits]{\XINT_FL_Div {\XINTdigits+\xint_c_ii}{#2}{#3}}%
2102 }%
2103 \def\XINT_fldiv_opt #1[\xint_relax #2]#3#4%
2104 {%
2105   #1[#2]{\XINT_FL_Div {#2+\xint_c_ii}{#3}{#4}}%
2106 }%
2107 \def\XINT_FL_Div #1#2%
2108 {%
2109   \expandafter\XINT_FL_Div_a\expandafter{\the\numexpr #1\expandafter}%
2110   \expandafter{\romannumeral0\XINTinfloat [#1]{#2}}%
2111 }%
2112 \def\XINT_FL_Div_a #1#2#3%
2113 {%
2114   \expandafter\XINT_FL_Div_b\romannumeral0\XINTinfloat [#1]{#3}#2%
2115 }%
2116 \def\XINT_FL_Div_b #1[#2]#3[#4]{\xintE{#3/#1}{#4-#2}}%

```

7.65 \XINTinFloatSum

1.09a: quick write-up, for use by `\xintfloatexpr`, will need to be thought through again. Renamed (and slightly modified) in 1.09h. Should be extended for optional precision. Should be rewritten for optimization.

```

2117 \def\XINTinFloatSum  {\romannumeral0\XINTinfloatsum }%
2118 \def\XINTinfloatsum #1{\expandafter\XINT_floatsum_a\romannumeral-`0#1\relax }%
2119 \def\XINT_floatsum_a #1{\expandafter\XINT_floatsum_b
2120   \romannumeral0\XINTinfloat[\XINTdigits]{#1}\Z }%

```

```

2121 \def\XINT_floatsum_b #1\Z #2%
2122     {\expandafter\XINT_floatsum_c\romannumeral-‘0#2\Z {#1}\Z}%
2123 \def\XINT_floatsum_c #1%
2124     {\xint_gob_til_relax #1\XINT_floatsum_e\relax\XINT_floatsum_d #1}%
2125 \def\XINT_floatsum_d #1\Z
2126     {\expandafter\XINT_floatsum_b\romannumeral0\XINTinfloatadd {#1}}%
2127 \def\XINT_floatsum_e #1\Z #2\Z { #2}%

```

7.66 \XINTinFloatPrd

1.09a: quick write-up, for use by `\xintfloatexpr`, will need to be thought through again. Renamed (and slightly modified) in 1.09h. Should be extended for optional precision. Should be rewritten for optimization.

```

2128 \def\XINTinFloatPrd {\romannumeral0\XINTinfloatprd}%
2129 \def\XINTinfloatprd #1{\expandafter\XINT_floatprd_a\romannumeral-‘0#1\relax}%
2130 \def\XINT_floatprd_a #1{\expandafter\XINT_floatprd_b
2131     \romannumeral0\XINTinfloat[\XINTdigits]{#1}\Z}%
2132 \def\XINT_floatprd_b #1\Z #2%
2133     {\expandafter\XINT_floatprd_c\romannumeral-‘0#2\Z {#1}\Z}%
2134 \def\XINT_floatprd_c #1%
2135     {\xint_gob_til_relax #1\XINT_floatprd_e\relax\XINT_floatprd_d #1}%
2136 \def\XINT_floatprd_d #1\Z
2137     {\expandafter\XINT_floatprd_b\romannumeral0\XINTinfloatmul {#1}}%
2138 \def\XINT_floatprd_e #1\Z #2\Z { #2}%

```

7.67 \xintFloatPow, \XINTinFloatPow

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

```

2139 \def\xintFloatPow {\romannumeral0\xintfloatpow}%
2140 \def\xintfloatpow #1{\XINT_flpow_chkopt \xintfloat #1\xint_relax}%
2141 \def\XINTinFloatPow {\romannumeral0\XINTinfloatpow}%
2142 \def\XINTinfloatpow #1{\XINT_flpow_chkopt \XINTinfloat #1\xint_relax}%
2143 \def\XINT_flpow_chkopt #1#2%
2144 {%
2145     \ifx [#2\expandafter\XINT_flpow_opt
2146         \else\expandafter\XINT_flpow_noopt
2147         \fi
2148     #1#2%
2149}%
2150 \def\XINT_flpow_noopt #1#2\xint_relax #3%
2151 {%
2152     \expandafter\XINT_flpow_checkB_start\expandafter
2153         {\the\numexpr #3\expandafter}\expandafter
2154         {\the\numexpr \XINTdigits}{#2}{#1[\XINTdigits]}%
2155}%
2156 \def\XINT_flpow_opt #1[\xint_relax #2]#3#4%
2157 {%
2158     \expandafter\XINT_flpow_checkB_start\expandafter
2159         {\the\numexpr #4\expandafter}\expandafter
2160         {\the\numexpr #2}{#3}{#1[#2]}%

```

```

2161 }%
2162 \def\XINT_flpow_checkB_start #1{\XINT_flpow_checkB_a #1\Z }%
2163 \def\XINT_flpow_checkB_a #1%
2164 {%
2165   \xint_UDzerominusfork
2166   #1-\XINT_flpow_BisZero
2167   0#1{\XINT_flpow_checkB_b 1}%
2168   0-{\XINT_flpow_checkB_b 0#1}%
2169   \krof
2170 }%
2171 \def\XINT_flpow_BisZero \Z #1#2#3{#3{1/1[0]}}%
2172 \def\XINT_flpow_checkB_b #1#2\Z #3%
2173 {%
2174   \expandafter\XINT_flpow_checkB_c \expandafter
2175   {\romannumeral0\xintlength{#2}}{#3}{#2}%1%
2176 }%
2177 \def\XINT_flpow_checkB_c #1#2%
2178 {%
2179   \expandafter\XINT_flpow_checkB_d \expandafter
2180   {\the\numexpr \expandafter\xintlength\expandafter
2181     {\the\numexpr #1*20/\xint_c_iii }+#1+#2+\xint_c_i }%
2182 }%
2183 \def\XINT_flpow_checkB_d #1#2#3#4%
2184 {%
2185   \expandafter \XINT_flpow_a
2186   \romannumeral0\XINTinfloat [#1]{#4}{#1}{#2}%3%
2187 }%
2188 \def\XINT_flpow_a #1%
2189 {%
2190   \xint_UDzerominusfork
2191   #1-\XINT_flpow_zero
2192   0#1{\XINT_flpow_b 1}%
2193   0-{\XINT_flpow_b 0#1}%
2194   \krof
2195 }%
2196 \def\XINT_flpow_b #1#2[#3]#4#5%
2197 {%
2198   \XINT_flpow_loopI {#5}{#2[#3]}\romannumeral0\XINTinfloatmul [#4]}%
2199   {#1*\ifodd #5 1\else 0\fi}%
2200 }%
2201 \def\XINT_flpow_zero [#1]#2#3#4#5%
2202 % xint is not equipped to signal infinity, the 2^31 will provoke
2203 % deliberately a number too big and arithmetic overflow in \XINT_float_Xb
2204 {%
2205   \if #41\xint_afterfi {\xintError:DivisionByZero #5{1[2147483648]}}%
2206   \else \xint_afterfi {#5{0[0]}}\fi
2207 }%
2208 \def\XINT_flpow_loopI #1%
2209 {%
2210   \ifnum #1=\xint_c_i\XINT_flpow_ItoIII\fi
2211   \ifodd #1
2212     \expandafter\XINT_flpow_loopI_odd

```



```

2213 \else
2214 \expandafter\XINT_flpow_loopI_even
2215 \fi
2216 {#1}%
2217 }%
2218 \def\XINT_flpow_ItoIII\fi #1\fi #2#3#4#5%
2219 {%
2220 \fi\expandafter\XINT_flpow_III\the\numexpr #5\relax #3%
2221 }%
2222 \def\XINT_flpow_loopI_even #1#2#3%
2223 {%
2224 \expandafter\XINT_flpow_loopI\expandafter
2225 {\the\numexpr #1/\xint_c_ii\expandafter}\expandafter
2226 {#3{#2}{#2}}{#3}%
2227 }%
2228 \def\XINT_flpow_loopI_odd #1#2#3%
2229 {%
2230 \expandafter\XINT_flpow_loopII\expandafter
2231 {\the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter}\expandafter
2232 {#3{#2}{#2}}{#3}{#2}%
2233 }%
2234 \def\XINT_flpow_loopII #1%
2235 {%
2236 \ifnum #1 = \xint_c_i\XINT_flpow_ItoIII\fi
2237 \ifodd #1
2238 \expandafter\XINT_flpow_loopII_odd
2239 \else
2240 \expandafter\XINT_flpow_loopII_even
2241 \fi
2242 {#1}%
2243 }%
2244 \def\XINT_flpow_loopII_even #1#2#3%
2245 {%
2246 \expandafter\XINT_flpow_loopII\expandafter
2247 {\the\numexpr #1/\xint_c_ii\expandafter}\expandafter
2248 {#3{#2}{#2}}{#3}%
2249 }%
2250 \def\XINT_flpow_loopII_odd #1#2#3#4%
2251 {%
2252 \expandafter\XINT_flpow_loopII_odda\expandafter
2253 {#3{#2}{#4}}{#1}{#2}{#3}%
2254 }%
2255 \def\XINT_flpow_loopII_odda #1#2#3#4%
2256 {%
2257 \expandafter\XINT_flpow_loopII\expandafter
2258 {\the\numexpr #2/\xint_c_ii-\xint_c_i\expandafter}\expandafter
2259 {#4{#3}{#3}}{#4}{#1}%
2260 }%
2261 \def\XINT_flpow_ItoIII\fi #1\fi #2#3#4#5#6%
2262 {%
2263 \fi\expandafter\XINT_flpow_III\the\numexpr #6\expandafter\relax
2264 #4{#3}{#5}%

```

```

2265 }%
2266 \def\XINT_flpow_III #1#2[#3]#4%
2267 {%
2268   \expandafter\XINT_flpow_IIIend\expandafter
2269   {\the\numexpr\if #41-\fi#3\expandafter}%
2270   \xint_UDzerofork
2271     #4{{#2}}%
2272     0{{1/#2}}%
2273   \krof #1%
2274 }%
2275 \def\XINT_flpow_IIIend #1#2#3#4%
2276 {%
2277   \xint_UDzerofork
2278   #3{#4{#2[#1]}}%
2279   0{#4{-#2[#1]}}%
2280   \krof
2281 }%

```

7.68 \xintFloatPower, \XINTinFloatPower

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

```

2282 \def\xintFloatPower {\romannumeral0\xintfloatpower}%
2283 \def\xintfloatpower #1{\XINT_flpower_chkopt \xintfloat #1\xint_relax }%
2284 \def\XINTinFloatPower {\romannumeral0\XINTinfloatpower}%
2285 \def\XINTinfloatpower #1{\XINT_flpower_chkopt \XINTinfloat #1\xint_relax }%
2286 \def\XINT_flpower_chkopt #1#2%
2287 {%
2288   \ifx [#2\expandafter\XINT_flpower_opt
2289   \else\expandafter\XINT_flpower_noopt
2290   \fi
2291   #1#2%
2292 }%
2293 \def\XINT_flpower_noopt #1#2\xint_relax #3%
2294 {%
2295   \expandafter\XINT_flpower_checkB_start\expandafter
2296   {\the\numexpr \XINTdigits\expandafter}\expandafter
2297   {\romannumeral0\xintnum{#3}}{#2}{#1[\XINTdigits]}%
2298 }%
2299 \def\XINT_flpower_opt #1[\xint_relax #2]#3#4%
2300 {%
2301   \expandafter\XINT_flpower_checkB_start\expandafter
2302   {\the\numexpr #2\expandafter}\expandafter
2303   {\romannumeral0\xintnum{#4}}{#3}{#1[#2]}%
2304 }%
2305 \def\XINT_flpower_checkB_start #1#2{\XINT_flpower_checkB_a #2\Z {#1}}%
2306 \def\XINT_flpower_checkB_a #1%
2307 {%
2308   \xint_UDzerominusfork
2309   #1-\XINT_flpower_BisZero
2310   0#1{\XINT_flpower_checkB_b 1}%
2311   0-{\XINT_flpower_checkB_b 0#1}%
2312   \krof

```

```

2313 }%
2314 \def\XINT_flpower_BisZero \Z #1#2#3{#3{1/1[0]}}%
2315 \def\XINT_flpower_checkB_b #1#2\Z #3%
2316 {%
2317   \expandafter\XINT_flpower_checkB_c \expandafter
2318   {\romannumeral0\xintlength{#2}}{#3}{#2}#1%
2319 }%
2320 \def\XINT_flpower_checkB_c #1#2%
2321 {%
2322   \expandafter\XINT_flpower_checkB_d \expandafter
2323   {\the\numexpr \expandafter\xintlength\expandafter
2324     {\the\numexpr #1*20/\xint_c_iii }+#1+#2+\xint_c_i }%
2325 }%
2326 \def\XINT_flpower_checkB_d #1#2#3#4%
2327 {%
2328   \expandafter \XINT_flpower_a
2329   \romannumeral0\XINTinfloat [#1]{#4}{#1}{#2}#3%
2330 }%
2331 \def\XINT_flpower_a #1%
2332 {%
2333   \xint_UDzerominusfork
2334   #1-\XINT_flpow_zero
2335   0#1{\XINT_flpower_b 1}%
2336   0-{\XINT_flpower_b 0#1}%
2337   \krof
2338 }%
2339 \def\XINT_flpower_b #1#2[#3]#4#5%
2340 {%
2341   \XINT_flpower_loopI {#5}{#2[#3]}\romannumeral0\XINTinfloatmul [#4]}%
2342   {#1*\xintiiOdd {#5}}%
2343 }%
2344 \def\XINT_flpower_loopI #1%
2345 {%
2346   \if1\XINT_isOne {#1}\XINT_flpower_ItoIII\fi
2347   \if1\xintiiOdd{#1}%
2348     \expandafter\expandafter\expandafter\XINT_flpower_loopI_odd
2349   \else
2350     \expandafter\expandafter\expandafter\XINT_flpower_loopI_even
2351   \fi
2352   \expandafter {\romannumeral0\xinthalff{#1}}%
2353 }%
2354 \def\XINT_flpower_ItoIII\fi #1\fi\expandafter #2#3#4#5%
2355 {%
2356   \fi\expandafter\XINT_flpow_III \the\numexpr #5\relax #3%
2357 }%
2358 \def\XINT_flpower_loopI_even #1#2#3%
2359 {%
2360   \expandafter\XINT_flpower_toI\expandafter {#3{#2}{#2}}{#1}{#3}%
2361 }%
2362 \def\XINT_flpower_loopI_odd #1#2#3%
2363 {%
2364   \expandafter\XINT_flpower_toII\expandafter {#3{#2}{#2}}{#1}{#3}{#2}%

```

```

2365 }%
2366 \def\XINT_flpower_toI #1#2{\XINT_flpower_loopI {#2}{#1}}%
2367 \def\XINT_flpower_toII #1#2{\XINT_flpower_loopII {#2}{#1}}%
2368 \def\XINT_flpower_loopII #1%
2369 {%
2370   \if1\XINT_isOne {#1}\XINT_flpower_IItoIII\fi
2371   \if1\xintiiOdd{#1}%
2372     \expandafter\expandafter\expandafter\XINT_flpower_loopII_odd
2373   \else
2374     \expandafter\expandafter\expandafter\XINT_flpower_loopII_even
2375   \fi
2376   \expandafter {\romannumeral0\xinthalft{#1}}%
2377 }%
2378 \def\XINT_flpower_loopII_even #1#2#3%
2379 {%
2380   \expandafter\XINT_flpower_toII\expandafter
2381   {#3{#2}{#2}}{#1}{#3}%
2382 }%
2383 \def\XINT_flpower_loopII_odd #1#2#3#4%
2384 {%
2385   \expandafter\XINT_flpower_loopII_odda\expandafter
2386   {#3{#2}{#4}}{#2}{#3}{#1}%
2387 }%
2388 \def\XINT_flpower_loopII_odda #1#2#3#4%
2389 {%
2390   \expandafter\XINT_flpower_toII\expandafter
2391   {#3{#2}{#2}}{#4}{#3}{#1}%
2392 }%
2393 \def\XINT_flpower_IItoIII\fi #1\fi\expandafter #2#3#4#5#6%
2394 {%
2395   \fi\expandafter\XINT_flpow_III\the\numexpr #6\expandafter\relax
2396   #4{#3}{#5}%
2397 }%

```

7.69 *\xintFloatSqrt*, *\XINTinFloatSqrt*

1.08

```

2398 \def\xintFloatSqrt {\romannumeral0\xintfloatsqrt }%
2399 \def\xintfloatsqrt #1{\XINT_flsqrt_chkopt \xintfloat #1\xint_relax }%
2400 \def\XINTinFloatSqrt {\romannumeral0\XINTinfloatsqrt }%
2401 \def\XINTinfloatsqrt #1{\XINT_flsqrt_chkopt \XINTinfloat #1\xint_relax }%
2402 \def\XINT_flsqrt_chkopt #1#2%
2403 {%
2404   \ifx [#2\expandafter\XINT_flsqrt_opt
2405     \else\expandafter\XINT_flsqrt_noopt
2406   \fi #1#2%
2407 }%
2408 \def\XINT_flsqrt_noopt #1#2\xint_relax
2409 {%
2410   #1[\XINTdigits]{\XINT_FL_sqrt \XINTdigits {#2}}%
2411 }%
2412 \def\XINT_flsqrt_opt #1[\xint_relax #2]#3%

```

```

2413 {%
2414   #1[#2]{\XINT_FL_sqrt {#2}{#3}}%
2415 }%
2416 \def\XINT_FL_sqrt #1%
2417 {%
2418   \ifnum\numexpr #1<\xint_c_xviii
2419     \xint_afterfi {\XINT_FL_sqrt_a\xint_c_xviii}%
2420   \else
2421     \xint_afterfi {\XINT_FL_sqrt_a {#1+\xint_c_i}}%
2422   \fi
2423 }%
2424 \def\XINT_FL_sqrt_a #1#2%
2425 {%
2426   \expandafter\XINT_FL_sqrt_checkifzeroorneg
2427   \romannumeral0\XINTinfloat [#1]{#2}%
2428 }%
2429 \def\XINT_FL_sqrt_checkifzeroorneg #1%
2430 {%
2431   \xint_UDzerominusfork
2432   #1-\XINT_FL_sqrt_iszero
2433   0#1\XINT_FL_sqrt_isneg
2434   0-\XINT_FL_sqrt_b #1}%
2435 \krof
2436 }%
2437 \def\XINT_FL_sqrt_iszero #1[#2]{0[0]}%
2438 \def\XINT_FL_sqrt_isneg #1[#2]{\xintError:RootOfNegative 0[0]}%
2439 \def\XINT_FL_sqrt_b #1[#2]%
2440 {%
2441   \ifodd #2
2442     \xint_afterfi{\XINT_FL_sqrt_c 01}%
2443   \else
2444     \xint_afterfi{\XINT_FL_sqrt_c {}0}%
2445   \fi
2446   {#1}{#2}%
2447 }%
2448 \def\XINT_FL_sqrt_c #1#2#3#4%
2449 {%
2450   \expandafter\XINT_flsqrt\expandafter {\the\numexpr #4-#2}{#3#1}%
2451 }%
2452 \def\XINT_flsqrt #1#2%
2453 {%
2454   \expandafter\XINT_sqrt_a
2455   \expandafter{\romannumeral0\xintlength {#2}}\XINT_flsqrt_big_d {#2}{#1}%
2456 }%
2457 \def\XINT_flsqrt_big_d #1#2%
2458 {%
2459   \ifodd #2
2460     \expandafter\expandafter\expandafter\XINT_flsqrt_big_eB
2461   \else
2462     \expandafter\expandafter\expandafter\XINT_flsqrt_big_eA
2463   \fi
2464   \expandafter {\the\numexpr (#2-\xint_c_i)/\xint_c_ii }{#1}%

```

```

2465 }%
2466 \def\XINT_flsqrt_big_eA #1#2#3%
2467 {%
2468   \XINT_flsqrt_big_eA_a #3\Z {#2}{#1}{#3}%
2469 }%
2470 \def\XINT_flsqrt_big_eA_a #1#2#3#4#5#6#7#8#9\Z
2471 {%
2472   \XINT_flsqrt_big_eA_b {#1#2#3#4#5#6#7#8}%
2473 }%
2474 \def\XINT_flsqrt_big_eA_b #1#2%
2475 {%
2476   \expandafter\XINT_flsqrt_big_f
2477   \romannumeral0\XINT_flsqrt_small_e {#2001}{#1}%
2478 }%
2479 \def\XINT_flsqrt_big_eB #1#2#3%
2480 {%
2481   \XINT_flsqrt_big_eB_a #3\Z {#2}{#1}{#3}%
2482 }%
2483 \def\XINT_flsqrt_big_eB_a #1#2#3#4#5#6#7#8#9%
2484 {%
2485   \XINT_flsqrt_big_eB_b {#1#2#3#4#5#6#7#8#9}%
2486 }%
2487 \def\XINT_flsqrt_big_eB_b #1#2\Z #3%
2488 {%
2489   \expandafter\XINT_flsqrt_big_f
2490   \romannumeral0\XINT_flsqrt_small_e {#30001}{#1}%
2491 }%
2492 \def\XINT_flsqrt_small_e #1#2%
2493 {%
2494   \expandafter\XINT_flsqrt_small_f\expandafter
2495   {\the\numexpr #1*#1-#2-\xint_c_i}{#1}%
2496 }%
2497 \def\XINT_flsqrt_small_f #1#2%
2498 {%
2499   \expandafter\XINT_flsqrt_small_g\expandafter
2500   {\the\numexpr (#1+#2)/(2*#2)-\xint_c_i}{#1}{#2}%
2501 }%
2502 \def\XINT_flsqrt_small_g #1%
2503 {%
2504   \ifnum #1>\xint_c_
2505     \expandafter\XINT_flsqrt_small_h
2506   \else
2507     \expandafter\XINT_flsqrt_small_end
2508   \fi
2509   {#1}%
2510 }%
2511 \def\XINT_flsqrt_small_h #1#2#3%
2512 {%
2513   \expandafter\XINT_flsqrt_small_f\expandafter
2514   {\the\numexpr #2-\xint_c_ii*#1*#3+#1*#1\expandafter}\expandafter
2515   {\the\numexpr #3-#1}%
2516 }%

```

7 Package *xintfrac* implementation

```

2517 \def\XINT_flsqrt_small_end #1#2#3%
2518 {%
2519   \expandafter\space\expandafter
2520   {\the\numexpr \xint_c_i+#3*\xint_c_x^iv-
2521     (#2*\xint_c_x^iv+#3)/(\xint_c_ii*#3)}%
2522 }%
2523 \def\XINT_flsqrt_big_f #1%
2524 {%
2525   \expandafter\XINT_flsqrt_big_fa\expandafter
2526   {\romannumeral0\xintiisqr {#1}}{#1}%
2527 }%
2528 \def\XINT_flsqrt_big_fa #1#2#3#4%
2529 {%
2530   \expandafter\XINT_flsqrt_big_fb\expandafter
2531   {\romannumeral0\XINT_dsx_addzerosnofuss
2532     {\numexpr #3-\xint_c_viii\relax}{#2}}%
2533   {\romannumeral0\xintiisub
2534     {\XINT_dsx_addzerosnofuss
2535       {\numexpr \xint_c_ii*(#3-\xint_c_viii)\relax}{#1}}{#4}}%
2536   {#3}%
2537 }%
2538 \def\XINT_flsqrt_big_fb #1#2%
2539 {%
2540   \expandafter\XINT_flsqrt_big_g\expandafter {#2}{#1}%
2541 }%
2542 \def\XINT_flsqrt_big_g #1#2%
2543 {%
2544   \expandafter\XINT_flsqrt_big_j
2545   \romannumeral0\xintiidivision
2546   {#1}{\romannumeral0\XINT_dbl_pos #2\R\R\R\R\R\R\Z \W\W\W\W\W\W }{#2}%
2547 }%
2548 \def\XINT_flsqrt_big_j #1%
2549 {%
2550   \if0\XINT_Sgn #1\Z
2551     \expandafter \XINT_flsqrt_big_end_a
2552   \else \expandafter \XINT_flsqrt_big_k
2553   \fi {#1}%
2554 }%
2555 \def\XINT_flsqrt_big_k #1#2#3%
2556 {%
2557   \expandafter\XINT_flsqrt_big_l\expandafter
2558   {\romannumeral0\XINT_sub_pre {#3}}{#1}}%
2559   {\romannumeral0\xintiiadd {#2}{\romannumeral0\XINT_sqr {#1}}}%
2560 }%
2561 \def\XINT_flsqrt_big_l #1#2%
2562 {%
2563   \expandafter\XINT_flsqrt_big_g\expandafter
2564   {#2}{#1}%
2565 }%
2566 \def\XINT_flsqrt_big_end_a #1#2#3#4#5%
2567 {%
2568   \expandafter\XINT_flsqrt_big_end_b\expandafter

```

```

2569 {\the\numexpr -#4+#5/\xint_c_ii\expandafter}\expandafter
2570 {\romannumeral0\xintiisub
2571 {\XINT_dsx_addzerosnofuss {#4}{#3}}}%
2572 {\xintHalf{\xintiiQuo{\XINT_dsx_addzerosnofuss {#4}{#2}}{#3}}}%
2573 }%
2574 \def\xINT_flsqrt_big_end_b #1#2{#2[#1]}%

```

7.70 \XINTinFloatMaxof

1.09a, for use by \xintNewFloatExpr. Name changed in 1.09h

```

2575 \def\xINTinFloatMaxof {\romannumeral0\xINTinfloatmaxof }%
2576 \def\xINTinfloatmaxof #1{\expandafter\xINT_flmaxof_a\romannumeral-‘0#1\relax }%
2577 \def\xINT_flmaxof_a #1{\expandafter\xINT_flmaxof_b
2578 \romannumeral0\xINTinfloat [\XINTdigits]{#1}\Z }%
2579 \def\xINT_flmaxof_b #1\Z #2%
2580 {\expandafter\xINT_flmaxof_c\romannumeral-‘0#2\Z {#1}\Z}%
2581 \def\xINT_flmaxof_c #1%
2582 {\xint_gob_til_relax #1\xINT_flmaxof_e\relax\xINT_flmaxof_d #1}%
2583 \def\xINT_flmaxof_d #1\Z
2584 {\expandafter\xINT_flmaxof_b\romannumeral0\xintmax
2585 {\XINTinFloat [\XINTdigits]{#1}}}%
2586 \def\xINT_flmaxof_e #1\Z #2\Z { #2}%

```

7.71 \XINTinFloatMinof

1.09a, for use by \xintNewFloatExpr. Name changed in 1.09h

```

2587 \def\xINTinFloatMinof {\romannumeral0\xINTinfloatminof }%
2588 \def\xINTinfloatminof #1{\expandafter\xINT_flminof_a\romannumeral-‘0#1\relax }%
2589 \def\xINT_flminof_a #1{\expandafter\xINT_flminof_b
2590 \romannumeral0\xINTinfloat [\XINTdigits]{#1}\Z }%
2591 \def\xINT_flminof_b #1\Z #2%
2592 {\expandafter\xINT_flminof_c\romannumeral-‘0#2\Z {#1}\Z}%
2593 \def\xINT_flminof_c #1%
2594 {\xint_gob_til_relax #1\xINT_flminof_e\relax\xINT_flminof_d #1}%
2595 \def\xINT_flminof_d #1\Z
2596 {\expandafter\xINT_flminof_b\romannumeral0\xintmin
2597 {\XINTinFloat [\XINTdigits]{#1}}}%
2598 \def\xINT_flminof_e #1\Z #2\Z { #2}%
2599 \XINT_restorecatcodes_endinput%

```

8 Package *xintseries* implementation

.1	Catcodes, ε -TeX and reload detection	201	.7	\xintRationalSeries	204
.2	Package identification	201	.8	\xintRationalSeriesX	205
.3	\xintSeries	202	.9	\xintFxFtPowerSeries	206
.4	\xintiSeries	202	.10	\xintFxFtPowerSeriesX	207
.5	\xintPowerSeries	203	.11	\xintFloatPowerSeries	207
.6	\xintPowerSeriesX	204	.12	\xintFloatPowerSeriesX	209

The commenting is currently (2014/10/28) 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.

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2 \catcode13=5 % ^^M
3 \endlinechar=13 %
4 \catcode123=1 % {
5 \catcode125=2 % }
6 \catcode64=11 % @
7 \catcode35=6 % #
8 \catcode44=12 % ,
9 \catcode45=12 % -
10 \catcode46=12 % .
11 \catcode58=12 % :
12 \let\z\endgroup
13 \expandafter\let\expandafter\x\csname ver@xintseries.sty\endcsname
14 \expandafter\let\expandafter\w\csname ver@xintfrac.sty\endcsname
15 \expandafter
16 \ifx\csname PackageInfo\endcsname\relax
17 \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
18 \else
19 \def\y#1#2{\PackageInfo{#1}{#2}}%
20 \fi
21 \expandafter
22 \ifx\csname numexpr\endcsname\relax
23 \y{xintseries}{\numexpr not available, aborting input}%
24 \aftergroup\endinput
25 \else
26 \ifx\x\relax % plain-TeX, first loading of xintseries.sty
27 \ifx\w\relax % but xintfrac.sty not yet loaded.
28 \def\z{\endgroup\input xintfrac.sty\relax}%
29 \fi
30 \else
31 \def\empty {}%
32 \ifx\x\empty % LaTeX, first loading,
33 % variable is initialized, but \ProvidesPackage not yet seen
34 \ifx\w\relax % xintfrac.sty not yet loaded.
35 \def\z{\endgroup\RequirePackage{xintfrac}}%
36 \fi
37 \else
38 \aftergroup\endinput % xintseries already loaded.
39 \fi
40 \fi
41 \fi
42 \z%
43 \XINTsetupcatcodes% defined in xintkernel.sty

```

8.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintseries}%
46 [2014/10/28 v1.1 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.

```

47 \def\xintSeries {\romannumeral0\xintseries }%
48 \def\xintseries #1#2%
49 {%
50   \expandafter\XINT_series\expandafter
51   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
52 }%
53 \def\XINT_series #1#2#3%
54 {%
55   \ifnum #2<#1
56     \xint_afterfi { 0/1[0]}%
57   \else
58     \xint_afterfi {\XINT_series_loop {#1}{0}{#2}{#3}}%
59   \fi
60 }%
61 \def\XINT_series_loop #1#2#3#4%
62 {%
63   \ifnum #3>#1 \else \XINT_series_exit \fi
64   \expandafter\XINT_series_loop\expandafter
65   {\the\numexpr #1+1\expandafter }\expandafter
66   {\romannumeral0\xintadd {#2}{#4{#1}}}%
67   {#3}{#4}%
68 }%
69 \def\XINT_series_exit \fi #1#2#3#4#5#6#7#8%
70 {%
71   \fi\xint_gobble_ii #6%
72 }%

```

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.

```

73 \def\xintiSeries {\romannumeral0\xintiseries }%
74 \def\xintiseries #1#2%
75 {%
76   \expandafter\XINT_iseries\expandafter
77   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
78 }%
79 \def\XINT_iseries #1#2#3%
80 {%
81   \ifnum #2<#1
82     \xint_afterfi { 0}%
83   \else
84     \xint_afterfi {\XINT_iseries_loop {#1}{0}{#2}{#3}}%
85   \fi
86 }%

```

```

87 \def\XINT_iseries_loop #1#2#3#4%
88 {%
89   \ifnum #3>#1 \else \XINT_iseries_exit \fi
90   \expandafter\XINT_iseries_loop\expandafter
91   {\the\numexpr #1+1\expandafter }\expandafter
92   {\romannumeral0\xintiadd {#2}{#4{#1}}}%
93   {#3}{#4}%
94 }%
95 \def\XINT_iseries_exit \fi #1#2#3#4#5#6#7#8%
96 {%
97   \fi\xint_gobble_ii #6%
98 }%

```

8.5 \xintPowerSeries

The 1.03 version was very lame and created a build-up of denominators. 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.

```

99 \def\xintPowerSeries {\romannumeral0\xintpowerseries }%
100 \def\xintpowerseries #1#2%
101 {%
102   \expandafter\XINT_powseries\expandafter
103   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
104 }%
105 \def\XINT_powseries #1#2#3#4%
106 {%
107   \ifnum #2<#1
108     \xint_afterfi { 0/1[0]}%
109   \else
110     \xint_afterfi
111     {\XINT_powseries_loop_i {#3{#2}}{#1}{#2}{#3}{#4}}%
112   \fi
113 }%
114 \def\XINT_powseries_loop_i #1#2#3#4#5%
115 {%
116   \ifnum #3>#2 \else\XINT_powseries_exit_i\fi
117   \expandafter\XINT_powseries_loop_ii\expandafter
118   {\the\numexpr #3-1\expandafter}\expandafter
119   {\romannumeral0\xintmul {#1}{#5}}{#2}{#4}{#5}%
120 }%
121 \def\XINT_powseries_loop_ii #1#2#3#4%
122 {%
123   \expandafter\XINT_powseries_loop_i\expandafter
124   {\romannumeral0\xintadd {#4{#1}}{#2}}{#3}{#1}{#4}%
125 }%
126 \def\XINT_powseries_exit_i\fi #1#2#3#4#5#6#7#8#9%
127 {%
128   \fi \XINT_powseries_exit_ii #6{#7}%
129 }%
130 \def\XINT_powseries_exit_ii #1#2#3#4#5#6%

```

```

131 {%
132   \xintmul{\xintPow {#5}{#6}}{#4}%
133 }%

```

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.

```

134 \def\xintPowerSeriesX {\romannumeral0\xintpowerseriesx }%
135 \def\xintpowerseriesx #1#2%
136 {%
137   \expandafter\XINT_powseriesx\expandafter
138   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
139 }%
140 \def\XINT_powseriesx #1#2#3#4%
141 {%
142   \ifnum #2<#1
143     \xint_afterfi { 0/1[0]}%
144   \else
145     \xint_afterfi
146     {\expandafter\XINT_powseriesx_pre\expandafter
147      {\romannumeral-'0#4}{#1}{#2}{#3}%
148     }%
149   \fi
150 }%
151 \def\XINT_powseriesx_pre #1#2#3#4%
152 {%
153   \XINT_powseries_loop_i {#4}{#3}{#2}{#3}{#4}{#1}%
154 }%

```

8.7 \xintRationalSeries

This computes $F(a) + \dots + F(b)$ 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.

```

155 \def\xintRationalSeries {\romannumeral0\xintratseries }%
156 \def\xintratseries #1#2%
157 {%
158   \expandafter\XINT_ratseries\expandafter
159   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
160 }%
161 \def\XINT_ratseries #1#2#3#4%
162 {%
163   \ifnum #2<#1
164     \xint_afterfi { 0/1[0]}%

```

```

165 \else
166   \xint_afterfi
167   {\XINT_ratseries_loop {#2}{1}{#1}{#4}{#3}}%
168 \fi
169 }%
170 \def\XINT_ratseries_loop #1#2#3#4%
171 {%
172   \ifnum #1>#3 \else\XINT_ratseries_exit_i\fi
173   \expandafter\XINT_ratseries_loop\expandafter
174   {\the\numexpr #1-1\expandafter}\expandafter
175   {\romannumeral0\xintadd {1}{\xintMul {#2}{#4{#1}}}}{#3}{#4}%
176 }%
177 \def\XINT_ratseries_exit_i\fi #1#2#3#4#5#6#7#8%
178 {%
179   \fi \XINT_ratseries_exit_ii #6%
180 }%
181 \def\XINT_ratseries_exit_ii #1#2#3#4#5%
182 {%
183   \XINT_ratseries_exit_iii #5%
184 }%
185 \def\XINT_ratseries_exit_iii #1#2#3#4%
186 {%
187   \xintmul{#2}{#4}%
188 }%

```

8.8 \xintRationalSeriesX

a,b,initial,radiofunction,x

This computes $F(a,x)+\dots+F(b,x)$ 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.

```

189 \def\xintRationalSeriesX {\romannumeral0\xintratseriesx }%
190 \def\xintratseriesx #1#2%
191 {%
192   \expandafter\XINT_ratseriesx\expandafter
193   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
194 }%
195 \def\XINT_ratseriesx #1#2#3#4#5%
196 {%
197   \ifnum #2<#1
198     \xint_afterfi { 0/1[0]}%
199   \else
200     \xint_afterfi
201     {\expandafter\XINT_ratseriesx_pre\expandafter
202      {\romannumeral-'0#5}{#2}{#1}{#4}{#3}%
203     }%
204   \fi
205 }%
206 \def\XINT_ratseriesx_pre #1#2#3#4#5%

```

```

207 {%
208   \XINT_ratseries_loop {#2}{1}{#3}{#4{#1}}{#5{#1}}%
209 }%

```

8.9 \xintFxpPowerSeries

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.

```

210 \def\xintFxpPowerSeries {\romannumeral0\xintfxptpowerseries }%
211 \def\xintfxptpowerseries #1#2%
212 {%
213   \expandafter\XINT_fppowseries\expandafter
214   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
215 }%
216 \def\XINT_fppowseries #1#2#3#4#5%
217 {%
218   \ifnum #2<#1
219     \xint_afterfi { 0}%
220   \else
221     \xint_afterfi
222     {\expandafter\XINT_fppowseries_loop_pre\expandafter
223      {\romannumeral0\xinttrunc {#5}{\xintPow {#4}{#1}}}%
224      {#1}{#4}{#2}{#3}{#5}%
225     }%
226   \fi
227 }%
228 \def\XINT_fppowseries_loop_pre #1#2#3#4#5#6%
229 {%
230   \ifnum #4>#2 \else\XINT_fppowseries_dont_i \fi
231   \expandafter\XINT_fppowseries_loop_i\expandafter
232   {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
233   {\romannumeral0\xintitrunc {#6}{\xintMul {#5}{#2}}{#1}}%
234   {#1}{#3}{#4}{#5}{#6}%
235 }%
236 \def\XINT_fppowseries_dont_i \fi\expandafter\XINT_fppowseries_loop_i
237   {\fi \expandafter\XINT_fppowseries_dont_ii }%
238 \def\XINT_fppowseries_dont_ii #1#2#3#4#5#6#7{\xinttrunc {#7}{#2[-#7]}}%
239 \def\XINT_fppowseries_loop_i #1#2#3#4#5#6#7%
240 {%
241   \ifnum #5>#1 \else \XINT_fppowseries_exit_i \fi
242   \expandafter\XINT_fppowseries_loop_ii\expandafter
243   {\romannumeral0\xinttrunc {#7}{\xintMul {#3}{#4}}}%
244   {#1}{#4}{#2}{#5}{#6}{#7}%
245 }%
246 \def\XINT_fppowseries_loop_ii #1#2#3#4#5#6#7%
247 {%
248   \expandafter\XINT_fppowseries_loop_i\expandafter
249   {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
250   {\romannumeral0\xintiadd {#4}{\xintitrunc {#7}{\xintMul {#6}{#2}}{#1}}}%
251   {#1}{#3}{#5}{#6}{#7}%

```

```

252 }%
253 \def\XINT_fppowseries_exit_i\fi\expandafter\XINT_fppowseries_loop_ii
254   {\fi \expandafter\XINT_fppowseries_exit_ii }%
255 \def\XINT_fppowseries_exit_ii #1#2#3#4#5#6#7%
256 {%
257   \xinttrunc {#7}
258   {\xintiiadd {#4}{\xintiTrunc {#7}{\xintMul {#6{#2}}{#1}}}{-#7}}%
259 }%

```

8.10 \xintFxFtPowerSeriesX

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.

```

260 \def\xintFxFtPowerSeriesX {\romannumeral0\xintfxptpowerseriesx }%
261 \def\xintfxptpowerseriesx #1#2%
262 {%
263   \expandafter\XINT_fppowseriesx\expandafter
264   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
265 }%
266 \def\XINT_fppowseriesx #1#2#3#4#5%
267 {%
268   \ifnum #2<#1
269     \xint_afterfi { 0}%
270   \else
271     \xint_afterfi
272     {\expandafter \XINT_fppowseriesx_pre \expandafter
273     {\romannumeral-‘0#4}{#1}{#2}{#3}{#5}%
274     }%
275   \fi
276 }%
277 \def\XINT_fppowseriesx_pre #1#2#3#4#5%
278 {%
279   \expandafter\XINT_fppowseries_loop_pre\expandafter
280   {\romannumeral0\xinttrunc {#5}{\xintPow {#1}{#2}}}%
281   {#2}{#1}{#3}{#4}{#5}%
282 }%

```

8.11 \xintFloatPowerSeries

1.08a. I still have to re-visit `\xintFxFtPowerSeries`; temporarily I just adapted the code to the case of floats.

```

283 \def\xintFloatPowerSeries {\romannumeral0\xintfloatpowerseries }%
284 \def\xintfloatpowerseries #1{\XINT_flpowseries_chkopt #1\xint_relax }%
285 \def\XINT_flpowseries_chkopt #1%
286 {%
287   \ifx [#1\expandafter\XINT_flpowseries_opt
288   \else\expandafter\XINT_flpowseries_noopt
289   \fi
290   #1%

```

```

291 }%
292 \def\XINT_flpowseries_noopt #1\xint_relax #2%
293 {%
294   \expandafter\XINT_flpowseries\expandafter
295   {\the\numexpr #1\expandafter}\expandafter
296   {\the\numexpr #2}\XINTdigits
297 }%
298 \def\XINT_flpowseries_opt [\xint_relax #1]#2#3%
299 {%
300   \expandafter\XINT_flpowseries\expandafter
301   {\the\numexpr #2\expandafter}\expandafter
302   {\the\numexpr #3\expandafter}{\the\numexpr #1}%
303 }%
304 \def\XINT_flpowseries #1#2#3#4#5%
305 {%
306   \ifnum #2<#1
307     \xint_afterfi { 0.e0}%
308   \else
309     \xint_afterfi
310     {\expandafter\XINT_flpowseries_loop_pre\expandafter
311      {\romannumeral0\XINTinfloatpow [#3]{#5}{#1}}%
312      {#1}{#5}{#2}{#4}{#3}%
313     }%
314   \fi
315 }%
316 \def\XINT_flpowseries_loop_pre #1#2#3#4#5#6%
317 {%
318   \ifnum #4>#2 \else\XINT_flpowseries_dont_i \fi
319   \expandafter\XINT_flpowseries_loop_i\expandafter
320   {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
321   {\romannumeral0\XINTinfloatmul [#6]{#5}{#2}}{#1}%
322   {#1}{#3}{#4}{#5}{#6}%
323 }%
324 \def\XINT_flpowseries_dont_i \fi\expandafter\XINT_flpowseries_loop_i
325   {\fi \expandafter\XINT_flpowseries_dont_ii }%
326 \def\XINT_flpowseries_dont_ii #1#2#3#4#5#6#7{\xintfloat [#7]{#2}}%
327 \def\XINT_flpowseries_loop_i #1#2#3#4#5#6#7%
328 {%
329   \ifnum #5>#1 \else \XINT_flpowseries_exit_i \fi
330   \expandafter\XINT_flpowseries_loop_ii\expandafter
331   {\romannumeral0\XINTinfloatmul [#7]{#3}{#4}}%
332   {#1}{#4}{#2}{#5}{#6}{#7}%
333 }%
334 \def\XINT_flpowseries_loop_ii #1#2#3#4#5#6#7%
335 {%
336   \expandafter\XINT_flpowseries_loop_i\expandafter
337   {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
338   {\romannumeral0\XINTinfloatadd [#7]{#4}%
339    {\XINTinfloatmul [#7]{#6}{#2}}{#1}}%
340   {#1}{#3}{#5}{#6}{#7}%
341 }%
342 \def\XINT_flpowseries_exit_i\fi\expandafter\XINT_flpowseries_loop_ii

```



```

343   {\fi \expandafter\XINT_flpowseries_exit_ii }%
344 \def\XINT_flpowseries_exit_ii #1#2#3#4#5#6#7%
345 {%
346   \xintfloatadd [#7]{#4}{\XINTinfloatmul [#7]{#6{#2}}{#1}}%
347 }%

```

8.12 \xintFloatPowerSeriesX

1.08a

```

348 \def\xintFloatPowerSeriesX {\romannumeral0\xintfloatpowerseriesx }%
349 \def\xintfloatpowerseriesx #1{\XINT_flpowseriesx_chkopt #1\xint_relax }%
350 \def\XINT_flpowseriesx_chkopt #1%
351 {%
352   \ifx [#1\expandafter\XINT_flpowseriesx_opt
353     \else\expandafter\XINT_flpowseriesx_noopt
354   \fi
355   #1%
356 }%
357 \def\XINT_flpowseriesx_noopt #1\xint_relax #2%
358 {%
359   \expandafter\XINT_flpowseriesx\expandafter
360   {\the\numexpr #1\expandafter}\expandafter
361   {\the\numexpr #2}\XINTdigits
362 }%
363 \def\XINT_flpowseriesx_opt [\xint_relax #1]#2#3%
364 {%
365   \expandafter\XINT_flpowseriesx\expandafter
366   {\the\numexpr #2\expandafter}\expandafter
367   {\the\numexpr #3\expandafter}{\the\numexpr #1}%
368 }%
369 \def\XINT_flpowseriesx #1#2#3#4#5%
370 {%
371   \ifnum #2<#1
372     \xint_afterfi { 0.e0}%
373   \else
374     \xint_afterfi
375     {\expandafter \XINT_flpowseriesx_pre \expandafter
376      {\romannumeral- '0#5}{#1}{#2}{#4}{#3}%
377      }%
378   \fi
379 }%
380 \def\XINT_flpowseriesx_pre #1#2#3#4#5%
381 {%
382   \expandafter\XINT_flpowseries_loop_pre\expandafter
383   {\romannumeral0\XINTinfloatpow [#5]{#1}{#2}}%
384   {#2}{#1}{#3}{#4}{#5}%
385 }%
386 \XINT_restorecatcodes_endinput%

```

9 Package *xintcfrac* implementation

1	Catcodes, ε -TeX and reload detection	210	2	Package identification	211
---	---	-----	---	------------------------	-----

.3	<code>\xintCFrac</code>	211	.17	<code>\xintCtoCv, \xintCstoCv</code>	223
.4	<code>\xintGCFrac</code>	212	.18	<code>\xintiCstoCv</code>	224
.5	<code>\xintGGCFrac</code>	214	.19	<code>\xintGctoCv</code>	224
.6	<code>\xintGctoGCx</code>	215	.20	<code>\xintiGctoCv</code>	226
.7	<code>\xintFtoCs</code>	215	.21	<code>\xintFtoCv</code>	227
.8	<code>\xintFtoCx</code>	216	.22	<code>\xintFtoCCv</code>	227
.9	<code>\xintFtoC</code>	216	.23	<code>\xintCntoF</code>	227
.10	<code>\xintFtoGC</code>	217	.24	<code>\xintGCntoF</code>	228
.11	<code>\xintFGtoC</code>	217	.25	<code>\xintCntoCs</code>	229
.12	<code>\xintFtoCC</code>	218	.26	<code>\xintCntoGC</code>	229
.13	<code>\xintCtoF, \xintCstoF</code>	219	.27	<code>\xintGCntoGC</code>	230
.14	<code>\xintiCstoF</code>	220	.28	<code>\xintCstoGC</code>	231
.15	<code>\xintGctoF</code>	220	.29	<code>\xintGctoGC</code>	231
.16	<code>\xintiGctoF</code>	222			

The commenting is currently (2014/10/28) 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`. All uses of `\W` and many instances of `\Z` as delimiters removed, this was in some cases not very safe (for example in the treatment of the optional arguments to some macros). Actually I have also replaced everywhere else in the bundle the use of `\Z` in the treatment of macros with optional arguments with the safer `\xint_relax` (the more recent `\xintSeq` already used `\xint_bye`).

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.

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2 \catcode13=5 % ^^M
3 \endlinechar=13 %
4 \catcode123=1 % {
5 \catcode125=2 % }
6 \catcode64=11 % @
7 \catcode35=6 % #
8 \catcode44=12 % ,
9 \catcode45=12 % -
10 \catcode46=12 % .
11 \catcode58=12 % :
12 \let\z\endgroup
13 \expandafter\let\expandafter\x\csname ver@xintcfrac.sty\endcsname
14 \expandafter\let\expandafter\w\csname ver@xintfrac.sty\endcsname
15 \expandafter
16 \ifx\csname PackageInfo\endcsname\relax
17 \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
18 \else
19 \def\y#1#2{\PackageInfo{#1}{#2}}%
20 \fi
21 \expandafter
22 \ifx\csname numexpr\endcsname\relax
23 \y{xintcfrac}{\numexpr not available, aborting input}%
24 \aftergroup\endinput
25 \else

```

```

26 \ifx\x\relax % plain-TeX, first loading of xintcfrac.sty
27 \ifx\w\relax % but xintfrac.sty not yet loaded.
28 \def\z{\endgroup\input xintfrac.sty\relax}%
29 \fi
30 \else
31 \def\empty {}%
32 \ifx\x\empty % LaTeX, first loading,
33 % variable is initialized, but \ProvidesPackage not yet seen
34 \ifx\w\relax % xintfrac.sty not yet loaded.
35 \def\z{\endgroup\RequirePackage{xintfrac}}%
36 \fi
37 \else
38 \aftergroup\endinput % xintcfrac already loaded.
39 \fi
40 \fi
41 \fi
42 \z%
43 \XINTsetupcatcodes% defined in xintkernel.sty

```

9.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintcfrac}%
46 [2014/10/28 v1.1 Expandable continued fractions with xint package (jfb)]%

```

9.3 *\xintCFrac*

```

47 \def\xintCFrac {\romannumeral0\xintcfrac }%
48 \def\xintcfrac #1%
49 {%
50 \XINT_cfrac_opt_a #1\xint_relax
51 }%
52 \def\XINT_cfrac_opt_a #1%
53 {%
54 \ifx[#1\XINT_cfrac_opt_b\fi \XINT_cfrac_noopt #1%
55 }%
56 \def\XINT_cfrac_noopt #1\xint_relax
57 {%
58 \expandafter\XINT_cfrac_A\romannumeral0\xintraawithzeros {#1}\Z
59 \relax\relax
60 }%
61 \def\XINT_cfrac_opt_b\fi\XINT_cfrac_noopt [\xint_relax #1]%
62 {%
63 \fi\csname XINT_cfrac_opt#1\endcsname
64 }%
65 \def\XINT_cfrac_optl #1%
66 {%
67 \expandafter\XINT_cfrac_A\romannumeral0\xintraawithzeros {#1}\Z
68 \relax\hfill
69 }%
70 \def\XINT_cfrac_optc #1%
71 {%
72 \expandafter\XINT_cfrac_A\romannumeral0\xintraawithzeros {#1}\Z
73 \relax\relax

```

```

74 }%
75 \def\XINT_cfrac_optr #1%
76 {%
77   \expandafter\XINT_cfrac_A\romannumeral0\xintraewithzeros {#1}\Z
78   \hfill\relax
79 }%
80 \def\XINT_cfrac_A #1/#2\Z
81 {%
82   \expandafter\XINT_cfrac_B\romannumeral0\xintiidivision {#1}{#2}{#2}%
83 }%
84 \def\XINT_cfrac_B #1#2%
85 {%
86   \XINT_cfrac_C #2\Z {#1}%
87 }%
88 \def\XINT_cfrac_C #1%
89 {%
90   \xint_gob_til_zero #1\XINT_cfrac_integer 0\XINT_cfrac_D #1%
91 }%
92 \def\XINT_cfrac_integer 0\XINT_cfrac_D 0#1\Z #2#3#4#5{ #2}%
93 \def\XINT_cfrac_D #1\Z #2#3{\XINT_cfrac_loop_a {#1}{#3}{#1}{#2}}}%
94 \def\XINT_cfrac_loop_a
95 {%
96   \expandafter\XINT_cfrac_loop_d\romannumeral0\XINT_div_prepare
97 }%
98 \def\XINT_cfrac_loop_d #1#2%
99 {%
100   \XINT_cfrac_loop_e #2.{#1}%
101 }%
102 \def\XINT_cfrac_loop_e #1%
103 {%
104   \xint_gob_til_zero #1\xint_cfrac_loop_exit0\XINT_cfrac_loop_f #1%
105 }%
106 \def\XINT_cfrac_loop_f #1.#2#3#4%
107 {%
108   \XINT_cfrac_loop_a {#1}{#3}{#1}{#2#4}%
109 }%
110 \def\xint_cfrac_loop_exit0\XINT_cfrac_loop_f #1.#2#3#4#5#6%
111   {\XINT_cfrac_T #5#6{#2}#4\Z }%
112 \def\XINT_cfrac_T #1#2#3#4%
113 {%
114   \xint_gob_til_Z #4\XINT_cfrac_end\Z\XINT_cfrac_T #1#2{#4+\cfrac{#11#2}{#3}}}%
115 }%
116 \def\XINT_cfrac_end\Z\XINT_cfrac_T #1#2#3%
117 {%
118   \XINT_cfrac_end_b #3%
119 }%
120 \def\XINT_cfrac_end_b \Z+\cfrac#1#2{ #2}%

```

9.4 *\xintGCFrac*

```

121 \def\xintGCFrac {\romannumeral0\xintgcfrac }%
122 \def\xintgcfrac #1{\XINT_gcfrac_opt_a #1\xint_relax }%
123 \def\XINT_gcfrac_opt_a #1%
124 {%

```

```

125 \ifx[#1\XINT_gcfrac_opt_b\fi \XINT_gcfrac_noopt #1%
126 }%
127 \def\XINT_gcfrac_noopt #1\xint_relax
128 {%
129 \XINT_gcfrac #1+\xint_relax/\relax\relax
130 }%
131 \def\XINT_gcfrac_opt_b\fi\XINT_gcfrac_noopt [\xint_relax #1]%
132 {%
133 \fi\csname XINT_gcfrac_opt#1\endcsname
134 }%
135 \def\XINT_gcfrac_optl #1%
136 {%
137 \XINT_gcfrac #1+\xint_relax/\relax\hfill
138 }%
139 \def\XINT_gcfrac_optc #1%
140 {%
141 \XINT_gcfrac #1+\xint_relax/\relax\relax
142 }%
143 \def\XINT_gcfrac_optr #1%
144 {%
145 \XINT_gcfrac #1+\xint_relax/\hfill\relax
146 }%
147 \def\XINT_gcfrac
148 {%
149 \expandafter\XINT_gcfrac_enter\romannumeral-'0%
150 }%
151 \def\XINT_gcfrac_enter {\XINT_gcfrac_loop {}}%
152 \def\XINT_gcfrac_loop #1#2+#3/%
153 {%
154 \xint_gob_til_xint_relax #3\XINT_gcfrac_endloop\xint_relax
155 \XINT_gcfrac_loop {{#3}{#2}{#1}}%
156 }%
157 \def\XINT_gcfrac_endloop\xint_relax\XINT_gcfrac_loop #1#2#3%
158 {%
159 \XINT_gcfrac_T #2#3#1\xint_relax\xint_relax
160 }%
161 \def\XINT_gcfrac_T #1#2#3#4{\XINT_gcfrac_U #1#2{\xintFrac{#4}}}%
162 \def\XINT_gcfrac_U #1#2#3#4#5%
163 {%
164 \xint_gob_til_xint_relax #5\XINT_gcfrac_end\xint_relax\XINT_gcfrac_U
165 #1#2{\xintFrac{#5}}%
166 \ifcase\xintSgn{#4}
167 +\or+\else-\fi
168 \cfrac{#1\xintFrac{\xintAbs{#4}}{#2}{#3}}%
169 }%
170 \def\XINT_gcfrac_end\xint_relax\XINT_gcfrac_U #1#2#3%
171 {%
172 \XINT_gcfrac_end_b #3%
173 }%
174 \def\XINT_gcfrac_end_b #1\cfrac#2#3{ #3}%

```

9.5 *\xintGGCFrac*

New with 1.09m

```

175 \def\xintGGCFrac {\romannumeral0\xintggcfrac }%
176 \def\xintggcfrac #1{\XINT_ggcfrac_opt_a #1\xint_relax }%
177 \def\XINT_ggcfrac_opt_a #1%
178 {%
179   \ifx[#1\XINT_ggcfrac_opt_b\fi \XINT_ggcfrac_noopt #1%
180 }%
181 \def\XINT_ggcfrac_noopt #1\xint_relax
182 {%
183   \XINT_ggcfrac #1+\xint_relax/\relax\relax
184 }%
185 \def\XINT_ggcfrac_opt_b\fi\XINT_ggcfrac_noopt [\xint_relax #1]%
186 {%
187   \fi\csname XINT_ggcfrac_opt#1\endcsname
188 }%
189 \def\XINT_ggcfrac_optl #1%
190 {%
191   \XINT_ggcfrac #1+\xint_relax/\relax\hfill
192 }%
193 \def\XINT_ggcfrac_optc #1%
194 {%
195   \XINT_ggcfrac #1+\xint_relax/\relax\relax
196 }%
197 \def\XINT_ggcfrac_optr #1%
198 {%
199   \XINT_ggcfrac #1+\xint_relax/\hfill\relax
200 }%
201 \def\XINT_ggcfrac
202 {%
203   \expandafter\XINT_ggcfrac_enter\romannumeral-‘0%
204 }%
205 \def\XINT_ggcfrac_enter {\XINT_ggcfrac_loop {}}%
206 \def\XINT_ggcfrac_loop #1#2+#3/%
207 {%
208   \xint_gob_til_xint_relax #3\XINT_ggcfrac_endloop\xint_relax
209   \XINT_ggcfrac_loop {{#3}{#2}{#1}}%
210 }%
211 \def\XINT_ggcfrac_endloop\xint_relax\XINT_ggcfrac_loop #1#2#3%
212 {%
213   \XINT_ggcfrac_T #2#3#1\xint_relax\xint_relax
214 }%
215 \def\XINT_ggcfrac_T #1#2#3#4{\XINT_ggcfrac_U #1#2{#4}}%
216 \def\XINT_ggcfrac_U #1#2#3#4#5%
217 {%
218   \xint_gob_til_xint_relax #5\XINT_ggcfrac_end\xint_relax\XINT_ggcfrac_U
219   #1#2{#5+\cfrac{#1#4#2}{#3}}%
220 }%
221 \def\XINT_ggcfrac_end\xint_relax\XINT_ggcfrac_U #1#2#3%
222 {%
223   \XINT_ggcfrac_end_b #3%

```

```

224 }%
225 \def\XINT_ggcfrac_end_b #1\cfrac#2#3{ #3}%

```

9.6 \xintGctoGCx

```

226 \def\xintGctoGCx {\romannumeral0\xintgctogcx }%
227 \def\xintgctogcx #1#2#3%
228 {%
229   \expandafter\XINT_gctgcx_start\expandafter {\romannumeral-‘0#3}{#1}{#2}%
230 }%
231 \def\XINT_gctgcx_start #1#2#3{\XINT_gctgcx_loop_a {}{#2}{#3}#1+\xint_relax/}%
232 \def\XINT_gctgcx_loop_a #1#2#3#4+#5/%
233 {%
234   \xint_gob_til_xint_relax #5\XINT_gctgcx_end\xint_relax
235   \XINT_gctgcx_loop_b {#1{#4}}{#2{#5}#3}{#2}{#3}%
236 }%
237 \def\XINT_gctgcx_loop_b #1#2%
238 {%
239   \XINT_gctgcx_loop_a {#1#2}%
240 }%
241 \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.

```

242 \def\xintFtoCs {\romannumeral0\xintftocs }%
243 \def\xintftocs #1%
244 {%
245   \expandafter\XINT_ftc_A\romannumeral0\xintrawwithzeros {#1}\Z
246 }%
247 \def\XINT_ftc_A #1/#2\Z
248 {%
249   \expandafter\XINT_ftc_B\romannumeral0\xintiidivision {#1}{#2}{#2}%
250 }%
251 \def\XINT_ftc_B #1#2%
252 {%
253   \XINT_ftc_C #2.{#1}%
254 }%
255 \def\XINT_ftc_C #1%
256 {%
257   \xint_gob_til_zero #1\XINT_ftc_integer 0\XINT_ftc_D #1%
258 }%
259 \def\XINT_ftc_integer 0\XINT_ftc_D 0#1.#2#3{ #2}%
260 \def\XINT_ftc_D #1.#2#3{\XINT_ftc_loop_a {#1}{#3}{#1}{#2, }}% 1.09m adds a space
261 \def\XINT_ftc_loop_a
262 {%
263   \expandafter\XINT_ftc_loop_d\romannumeral0\XINT_div_prepare
264 }%
265 \def\XINT_ftc_loop_d #1#2%
266 {%
267   \XINT_ftc_loop_e #2.{#1}%
268 }%
269 \def\XINT_ftc_loop_e #1%

```

```

270 {%
271   \xint_gob_til_zero #1\xint_ftc_loop_exit0\XINT_ftc_loop_f #1%
272 }%
273 \def\XINT_ftc_loop_f #1.#2#3#4%
274 {%
275   \XINT_ftc_loop_a {#1}{#3}{#1}{#4#2, }% 1.09m has an added space here
276 }%
277 \def\xint_ftc_loop_exit0\XINT_ftc_loop_f #1.#2#3#4{ #4#2}%

```

9.8 \xintFtoCx

```

278 \def\xintFtoCx {\romannumeral0\xintftocx }%
279 \def\xintftocx #1#2%
280 {%
281   \expandafter\XINT_ftcx_A\romannumeral0\xintrawwithzeros {#2}\Z {#1}%
282 }%
283 \def\XINT_ftcx_A #1/#2\Z
284 {%
285   \expandafter\XINT_ftcx_B\romannumeral0\xintiidivision {#1}{#2}{#2}%
286 }%
287 \def\XINT_ftcx_B #1#2%
288 {%
289   \XINT_ftcx_C #2.{#1}%
290 }%
291 \def\XINT_ftcx_C #1%
292 {%
293   \xint_gob_til_zero #1\XINT_ftcx_integer 0\XINT_ftcx_D #1%
294 }%
295 \def\XINT_ftcx_integer 0\XINT_ftcx_D 0#1.#2#3#4{ #2}%
296 \def\XINT_ftcx_D #1.#2#3#4{\XINT_ftcx_loop_a {#1}{#3}{#1}{#2#4}{#4}}%
297 \def\XINT_ftcx_loop_a
298 {%
299   \expandafter\XINT_ftcx_loop_d\romannumeral0\XINT_div_prepare
300 }%
301 \def\XINT_ftcx_loop_d #1#2%
302 {%
303   \XINT_ftcx_loop_e #2.{#1}%
304 }%
305 \def\XINT_ftcx_loop_e #1%
306 {%
307   \xint_gob_til_zero #1\xint_ftcx_loop_exit0\XINT_ftcx_loop_f #1%
308 }%
309 \def\XINT_ftcx_loop_f #1.#2#3#4#5%
310 {%
311   \XINT_ftcx_loop_a {#1}{#3}{#1}{#4#2#5}{#5}%
312 }%
313 \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 {}`


```

314 \def\xintFtoC {\romannumeral0\xintftoc}%
315 \def\xintftoc {\xintftocx {}}%

```

9.10 `\xintFtoGC`

```

316 \def\xintFtoGC {\romannumeral0\xintftogc}%
317 \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.

```

318 \def\xintFGtoC {\romannumeral0\xintfgtoc}%
319 \def\xintfgtoc#1%
320 {%
321   \expandafter\XINT_fgtc_a\romannumeral0\xintraewithzeros {#1}\Z
322 }%
323 \def\XINT_fgtc_a #1/#2\Z #3%
324 {%
325   \expandafter\XINT_fgtc_b\romannumeral0\xintraewithzeros {#3}\Z #1/#2\Z { }%
326 }%
327 \def\XINT_fgtc_b #1/#2\Z
328 {%
329   \expandafter\XINT_fgtc_c\romannumeral0\xintiidivision {#1}{#2}{#2}%
330 }%
331 \def\XINT_fgtc_c #1#2#3#4/#5\Z
332 {%
333   \expandafter\XINT_fgtc_d\romannumeral0\xintiidivision
334     {#4}{#5}{#5}{#1}{#2}{#3}%
335 }%
336 \def\XINT_fgtc_d #1#2#3#4#5#6#7%
337 {%
338   \xintifEq {#1}{#4}{\XINT_fgtc_da {#1}{#2}{#3}{#4}}%
339     {\xint_thirdofthree}%
340 }%
341 \def\XINT_fgtc_da #1#2#3#4#5#6#7%
342 {%
343   \XINT_fgtc_e {#2}{#5}{#3}{#6}{#7}{#1}}%
344 }%
345 \def\XINT_fgtc_e #1%
346 {%
347   \xintifZero {#1}{\expandafter\xint_firstofone\xint_gobble_iii}%
348     {\XINT_fgtc_f {#1}}%
349 }%
350 \def\XINT_fgtc_f #1#2%
351 {%
352   \xintifZero {#2}{\xint_thirdofthree}{\XINT_fgtc_g {#1}{#2}}%
353 }%
354 \def\XINT_fgtc_g #1#2#3%
355 {%
356   \expandafter\XINT_fgtc_h\romannumeral0\XINT_div_prepare {#1}{#3}{#1}{#2}%
357 }%
358 \def\XINT_fgtc_h #1#2#3#4#5%

```

```

359 {%
360   \expandafter\XINT_fgtd\romannumeral0\XINT_div_prepare
361   {#4}{#5}{#4}{#1}{#2}{#3}%
362 }%

```

9.12 \xintFtoCC

```

363 \def\xintFtoCC {\romannumeral0\xintftocc }%
364 \def\xintftocc #1%
365 {%
366   \expandafter\XINT_ftcc_A\expandafter {\romannumeral0\xintrawwithzeros {#1}}%
367 }%
368 \def\XINT_ftcc_A #1%
369 {%
370   \expandafter\XINT_ftcc_B
371   \romannumeral0\xintrawwithzeros {\xintAdd {1/2[0]}{#1[0]}}\Z {#1[0]}%
372 }%
373 \def\XINT_ftcc_B #1/#2\Z
374 {%
375   \expandafter\XINT_ftcc_C\expandafter {\romannumeral0\xintiiquo {#1}{#2}}%
376 }%
377 \def\XINT_ftcc_C #1#2%
378 {%
379   \expandafter\XINT_ftcc_D\romannumeral0\xintsub {#2}{#1}\Z {#1}%
380 }%
381 \def\XINT_ftcc_D #1%
382 {%
383   \xint_UDzerominusfork
384   #1-\XINT_ftcc_integer
385   0#1\XINT_ftcc_En
386   0-{\XINT_ftcc_Ep #1}%
387   \krof
388 }%
389 \def\XINT_ftcc_Ep #1\Z #2%
390 {%
391   \expandafter\XINT_ftcc_loop_a\expandafter
392   {\romannumeral0\xintdiv {1[0]}{#1}}{#2+1/}%
393 }%
394 \def\XINT_ftcc_En #1\Z #2%
395 {%
396   \expandafter\XINT_ftcc_loop_a\expandafter
397   {\romannumeral0\xintdiv {1[0]}{#1}}{#2+-1/}%
398 }%
399 \def\XINT_ftcc_integer #1\Z #2{ #2}%
400 \def\XINT_ftcc_loop_a #1%
401 {%
402   \expandafter\XINT_ftcc_loop_b
403   \romannumeral0\xintrawwithzeros {\xintAdd {1/2[0]}{#1}}\Z {#1}%
404 }%
405 \def\XINT_ftcc_loop_b #1/#2\Z
406 {%
407   \expandafter\XINT_ftcc_loop_c\expandafter
408   {\romannumeral0\xintiiquo {#1}{#2}}%

```

```

409 }%
410 \def\XINT_ftcc_loop_c #1#2%
411 {%
412   \expandafter\XINT_ftcc_loop_d
413   \romannumeral0\xintsub {#2}{#1[0]}\Z {#1}%
414 }%
415 \def\XINT_ftcc_loop_d #1%
416 {%
417   \xint_UDzerominusfork
418   #1-\XINT_ftcc_end
419   0#1\XINT_ftcc_loop_N
420   0-\XINT_ftcc_loop_P #1}%
421 \krof
422 }%
423 \def\XINT_ftcc_end #1\Z #2#3{ #3#2}%
424 \def\XINT_ftcc_loop_P #1\Z #2#3%
425 {%
426   \expandafter\XINT_ftcc_loop_a\expandafter
427   {\romannumeral0\xintdiv {1[0]}\{#1}\{#3#2+1/}%
428 }%
429 \def\XINT_ftcc_loop_N #1\Z #2#3%
430 {%
431   \expandafter\XINT_ftcc_loop_a\expandafter
432   {\romannumeral0\xintdiv {1[0]}\{#1}\{#3#2+-1/}%
433 }%

```

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.

```

434 \def\xintCstoF {\romannumeral0\xintcstof }%
435 \def\xintcstof #1%
436 {%
437   \expandafter\XINT_ctf_prep \romannumeral0\xintcsvtolist{#1}\xint_relax
438 }%
439 \def\xintCtoF {\romannumeral0\xintctof }%
440 \def\xintctof #1%
441 {%
442   \expandafter\XINT_ctf_prep \romannumeral-‘0#1\xint_relax
443 }%
444 \def\XINT_ctf_prep
445 {%
446   \XINT_ctf_loop_a 1001%
447 }%
448 \def\XINT_ctf_loop_a #1#2#3#4#5%
449 {%
450   \xint_gob_til_xint_relax #5\XINT_ctf_end\xint_relax
451   \expandafter\XINT_ctf_loop_b
452   \romannumeral0\xintraewithzeros {#5}.\{#1}\{#2}\{#3}\{#4}%
453 }%
454 \def\XINT_ctf_loop_b #1/#2.#3#4#5#6%
455 {%

```

```

456 \expandafter\XINT_ctf_loop_c\expandafter
457 {\romannumeral0\XINT_mul_fork #2\Z #4\Z }%
458 {\romannumeral0\XINT_mul_fork #2\Z #3\Z }%
459 {\romannumeral0\xintiiadd {\XINT_Mul {#2}{#6}}{\XINT_Mul {#1}{#4}}}%
460 {\romannumeral0\xintiiadd {\XINT_Mul {#2}{#5}}{\XINT_Mul {#1}{#3}}}%
461 }%
462 \def\XINT_ctf_loop_c #1#2%
463 {%
464 \expandafter\XINT_ctf_loop_d\expandafter {\expandafter{#2}{#1}}%
465 }%
466 \def\XINT_ctf_loop_d #1#2%
467 {%
468 \expandafter\XINT_ctf_loop_e\expandafter {\expandafter{#2}{#1}}%
469 }%
470 \def\XINT_ctf_loop_e #1#2%
471 {%
472 \expandafter\XINT_ctf_loop_a\expandafter{#2}{#1}%
473 }%
474 \def\XINT_ctf_end #1.#2#3#4#5{\xintrawithzeros {#2/#3}}% 1.09b removes [0]

```

9.14 *\xintiCstoF*

```

475 \def\xintiCstoF {\romannumeral0\xinticstof }%
476 \def\xinticstof #1%
477 {%
478 \expandafter\XINT_icstf_prep \romannumeral-‘0#1,\xint_relax,%
479 }%
480 \def\XINT_icstf_prep
481 {%
482 \XINT_icstf_loop_a 1001%
483 }%
484 \def\XINT_icstf_loop_a #1#2#3#4#5,%
485 {%
486 \xint_gob_til_xint_relax #5\XINT_icstf_end\xint_relax
487 \expandafter
488 \XINT_icstf_loop_b \romannumeral-‘0#5.{#1}{#2}{#3}{#4}%
489 }%
490 \def\XINT_icstf_loop_b #1.#2#3#4#5%
491 {%
492 \expandafter\XINT_icstf_loop_c\expandafter
493 {\romannumeral0\xintiiadd {#5}{\XINT_Mul {#1}{#3}}}%
494 {\romannumeral0\xintiiadd {#4}{\XINT_Mul {#1}{#2}}}%
495 {#2}{#3}%
496 }%
497 \def\XINT_icstf_loop_c #1#2%
498 {%
499 \expandafter\XINT_icstf_loop_a\expandafter {#2}{#1}%
500 }%
501 \def\XINT_icstf_end#1.#2#3#4#5{\xintrawithzeros {#2/#3}}% 1.09b removes [0]

```

9.15 *\xintGctoF*

```

502 \def\xintGctoF {\romannumeral0\xintgctof }%
503 \def\xintgctof #1%

```

```

504 {%
505   \expandafter\XINT_gctf_prep \romannumeral-‘0#1+\xint_relax/%
506 }%
507 \def\XINT_gctf_prep
508 {%
509   \XINT_gctf_loop_a 1001%
510 }%
511 \def\XINT_gctf_loop_a #1#2#3#4#5+%
512 {%
513   \expandafter\XINT_gctf_loop_b
514   \romannumeral0\xintraawithzeros {#5}.{#1}{#2}{#3}{#4}%
515 }%
516 \def\XINT_gctf_loop_b #1/#2.#3#4#5#6%
517 {%
518   \expandafter\XINT_gctf_loop_c\expandafter
519   {\romannumeral0\XINT_mul_fork #2\Z #4\Z }%
520   {\romannumeral0\XINT_mul_fork #2\Z #3\Z }%
521   {\romannumeral0\xintiiadd {\XINT_Mul {#2}{#6}}{\XINT_Mul {#1}{#4}}}%
522   {\romannumeral0\xintiiadd {\XINT_Mul {#2}{#5}}{\XINT_Mul {#1}{#3}}}%
523 }%
524 \def\XINT_gctf_loop_c #1#2%
525 {%
526   \expandafter\XINT_gctf_loop_d\expandafter {\expandafter{#2}{#1}}%
527 }%
528 \def\XINT_gctf_loop_d #1#2%
529 {%
530   \expandafter\XINT_gctf_loop_e\expandafter {\expandafter{#2}{#1}}%
531 }%
532 \def\XINT_gctf_loop_e #1#2%
533 {%
534   \expandafter\XINT_gctf_loop_f\expandafter {\expandafter{#2}{#1}}%
535 }%
536 \def\XINT_gctf_loop_f #1#2/%
537 {%
538   \xint_gob_til_xint_relax #2\XINT_gctf_end\xint_relax
539   \expandafter\XINT_gctf_loop_g
540   \romannumeral0\xintraawithzeros {#2}.#1%
541 }%
542 \def\XINT_gctf_loop_g #1/#2.#3#4#5#6%
543 {%
544   \expandafter\XINT_gctf_loop_h\expandafter
545   {\romannumeral0\XINT_mul_fork #1\Z #6\Z }%
546   {\romannumeral0\XINT_mul_fork #1\Z #5\Z }%
547   {\romannumeral0\XINT_mul_fork #2\Z #4\Z }%
548   {\romannumeral0\XINT_mul_fork #2\Z #3\Z }%
549 }%
550 \def\XINT_gctf_loop_h #1#2%
551 {%
552   \expandafter\XINT_gctf_loop_i\expandafter {\expandafter{#2}{#1}}%
553 }%
554 \def\XINT_gctf_loop_i #1#2%
555 {%

```

```

556 \expandafter\XINT_gctf_loop_j\expandafter {\expandafter{#2}#1}%
557 }%
558 \def\XINT_gctf_loop_j #1#2%
559 {%
560 \expandafter\XINT_gctf_loop_a\expandafter {#2}#1%
561 }%
562 \def\XINT_gctf_end #1.#2#3#4#5{\xintrawithzeros {#2/#3}}% 1.09b removes [0]

```

9.16 \xintiGctoF

```

563 \def\xintiGctoF {\romannumeral0\xintigctof }%
564 \def\xintigctof #1%
565 {%
566 \expandafter\XINT_igctf_prep \romannumeral-‘0#1+\xint_relax/%
567 }%
568 \def\XINT_igctf_prep
569 {%
570 \XINT_igctf_loop_a 1001%
571 }%
572 \def\XINT_igctf_loop_a #1#2#3#4#5+%
573 {%
574 \expandafter\XINT_igctf_loop_b
575 \romannumeral-‘0#5.{#1}{#2}{#3}{#4}%
576 }%
577 \def\XINT_igctf_loop_b #1.#2#3#4#5%
578 {%
579 \expandafter\XINT_igctf_loop_c\expandafter
580 {\romannumeral0\xintiiadd {#5}{\XINT_Mul {#1}{#3}}}%
581 {\romannumeral0\xintiiadd {#4}{\XINT_Mul {#1}{#2}}}%
582 {#2}{#3}%
583 }%
584 \def\XINT_igctf_loop_c #1#2%
585 {%
586 \expandafter\XINT_igctf_loop_f\expandafter {\expandafter{#2}{#1}}%
587 }%
588 \def\XINT_igctf_loop_f #1#2#3#4/%
589 {%
590 \xint_gob_til_xint_relax #4\XINT_igctf_end\xint_relax
591 \expandafter\XINT_igctf_loop_g
592 \romannumeral-‘0#4.{#2}{#3}#1%
593 }%
594 \def\XINT_igctf_loop_g #1.#2#3%
595 {%
596 \expandafter\XINT_igctf_loop_h\expandafter
597 {\romannumeral0\XINT_mul_fork #1\Z #3\Z }%
598 {\romannumeral0\XINT_mul_fork #1\Z #2\Z }%
599 }%
600 \def\XINT_igctf_loop_h #1#2%
601 {%
602 \expandafter\XINT_igctf_loop_i\expandafter {#2}{#1}%
603 }%
604 \def\XINT_igctf_loop_i #1#2#3#4%
605 {%
606 \XINT_igctf_loop_a {#3}{#4}{#1}{#2}%

```

```
607 }%
608 \def\XINT_igctf_end #1.#2#3#4#5{\xintrawithzeros {#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.

```
609 \def\xintCstoCv {\romannumeral0\xintcstocv }%
610 \def\xintcstocv #1%
611 {%
612   \expandafter\XINT_ctcv_prep\romannumeral0\xintcsvtolist{#1}\xint_relax
613 }%
614 \def\xintCtoCv {\romannumeral0\xintctocv }%
615 \def\xintctocv #1%
616 {%
617   \expandafter\XINT_ctcv_prep\romannumeral-‘0#1\xint_relax
618 }%
619 \def\XINT_ctcv_prep
620 {%
621   \XINT_ctcv_loop_a {}1001%
622 }%
623 \def\XINT_ctcv_loop_a #1#2#3#4#5#6%
624 {%
625   \xint_gob_til_xint_relax #6\XINT_ctcv_end\xint_relax
626   \expandafter\XINT_ctcv_loop_b
627   \romannumeral0\xintrawithzeros {#6}.{#2}{#3}{#4}{#5}{#1}%
628 }%
629 \def\XINT_ctcv_loop_b #1/#2.#3#4#5#6%
630 {%
631   \expandafter\XINT_ctcv_loop_c\expandafter
632   {\romannumeral0\XINT_mul_fork #2\Z #4\Z }%
633   {\romannumeral0\XINT_mul_fork #2\Z #3\Z }%
634   {\romannumeral0\xintiiadd {\XINT_Mul {#2}{#6}}{\XINT_Mul {#1}{#4}}}%
635   {\romannumeral0\xintiiadd {\XINT_Mul {#2}{#5}}{\XINT_Mul {#1}{#3}}}%
636 }%
637 \def\XINT_ctcv_loop_c #1#2%
638 {%
639   \expandafter\XINT_ctcv_loop_d\expandafter {\expandafter{#2}{#1}}%
640 }%
641 \def\XINT_ctcv_loop_d #1#2%
642 {%
643   \expandafter\XINT_ctcv_loop_e\expandafter {\expandafter{#2}{#1}}%
644 }%
645 \def\XINT_ctcv_loop_e #1#2%
646 {%
647   \expandafter\XINT_ctcv_loop_f\expandafter{#2}{#1}%
648 }%
649 \def\XINT_ctcv_loop_f #1#2#3#4#5%
650 {%
651   \expandafter\XINT_ctcv_loop_g\expandafter
652   {\romannumeral0\xintrawithzeros {#1/#2}}{#5}{#1}{#2}{#3}{#4}%
653 }%
```

```

654 \def\XINT_ctcv_loop_g #1#2{\XINT_ctcv_loop_a {#2{#1}}}% 1.09b removes [0]
655 \def\XINT_ctcv_end #1.#2#3#4#5#6{ #6}%

```

9.18 \xintiCstoCv

```

656 \def\xintiCstoCv {\romannumeral0\xinticstocv }%
657 \def\xinticstocv #1%
658 {%
659   \expandafter\XINT_icstcv_prep \romannumeral-‘0#1,\xint_relax,%
660 }%
661 \def\XINT_icstcv_prep
662 {%
663   \XINT_icstcv_loop_a {}1001%
664 }%
665 \def\XINT_icstcv_loop_a #1#2#3#4#5#6,%
666 {%
667   \xint_gob_til_xint_relax #6\XINT_icstcv_end\xint_relax
668   \expandafter
669   \XINT_icstcv_loop_b \romannumeral-‘0#6.{#2}{#3}{#4}{#5}{#1}%
670 }%
671 \def\XINT_icstcv_loop_b #1.#2#3#4#5%
672 {%
673   \expandafter\XINT_icstcv_loop_c\expandafter
674   {\romannumeral0\xintiadd {#5}{\XINT_Mul {#1}{#3}}}%
675   {\romannumeral0\xintiadd {#4}{\XINT_Mul {#1}{#2}}}%
676   {{#2}{#3}}}%
677 }%
678 \def\XINT_icstcv_loop_c #1#2%
679 {%
680   \expandafter\XINT_icstcv_loop_d\expandafter {#2}{#1}%
681 }%
682 \def\XINT_icstcv_loop_d #1#2%
683 {%
684   \expandafter\XINT_icstcv_loop_e\expandafter
685   {\romannumeral0\xintraewithzeros {#1/#2}}{#1}{#2}}%
686 }%
687 \def\XINT_icstcv_loop_e #1#2#3#4{\XINT_icstcv_loop_a {#4{#1}}#2#3}%
688 \def\XINT_icstcv_end #1.#2#3#4#5#6{ #6}% 1.09b removes [0]

```

9.19 \xintGctoCv

```

689 \def\xintGctoCv {\romannumeral0\xintgctocv }%
690 \def\xintgctocv #1%
691 {%
692   \expandafter\XINT_gctcv_prep \romannumeral-‘0#1+\xint_relax/%
693 }%
694 \def\XINT_gctcv_prep
695 {%
696   \XINT_gctcv_loop_a {}1001%
697 }%
698 \def\XINT_gctcv_loop_a #1#2#3#4#5#6+%
699 {%
700   \expandafter\XINT_gctcv_loop_b
701   \romannumeral0\xintraewithzeros {#6}.{#2}{#3}{#4}{#5}{#1}%

```



```

702 }%
703 \def\XINT_gctcv_loop_b #1/#2.#3#4#5#6%
704 {%
705   \expandafter\XINT_gctcv_loop_c\expandafter
706   {\romannumeral0\XINT_mul_fork #2\Z #4\Z }%
707   {\romannumeral0\XINT_mul_fork #2\Z #3\Z }%
708   {\romannumeral0\xintiiadd {\XINT_Mul {#2}{#6}}{\XINT_Mul {#1}{#4}}}%
709   {\romannumeral0\xintiiadd {\XINT_Mul {#2}{#5}}{\XINT_Mul {#1}{#3}}}%
710 }%
711 \def\XINT_gctcv_loop_c #1#2%
712 {%
713   \expandafter\XINT_gctcv_loop_d\expandafter {\expandafter{#2}{#1}}%
714 }%
715 \def\XINT_gctcv_loop_d #1#2%
716 {%
717   \expandafter\XINT_gctcv_loop_e\expandafter {\expandafter{#2}{#1}}%
718 }%
719 \def\XINT_gctcv_loop_e #1#2%
720 {%
721   \expandafter\XINT_gctcv_loop_f\expandafter {#2}#1%
722 }%
723 \def\XINT_gctcv_loop_f #1#2%
724 {%
725   \expandafter\XINT_gctcv_loop_g\expandafter
726   {\romannumeral0\xintraewithzeros {#1/#2}}{\{#1}{#2}}%
727 }%
728 \def\XINT_gctcv_loop_g #1#2#3#4%
729 {%
730   \XINT_gctcv_loop_h {#4{#1}}{#2#3}% 1.09b removes [0]
731 }%
732 \def\XINT_gctcv_loop_h #1#2#3/%
733 {%
734   \xint_gob_til_xint_relax #3\XINT_gctcv_end\xint_relax
735   \expandafter\XINT_gctcv_loop_i
736   \romannumeral0\xintraewithzeros {#3}.#2{#1}%
737 }%
738 \def\XINT_gctcv_loop_i #1/#2.#3#4#5#6%
739 {%
740   \expandafter\XINT_gctcv_loop_j\expandafter
741   {\romannumeral0\XINT_mul_fork #1\Z #6\Z }%
742   {\romannumeral0\XINT_mul_fork #1\Z #5\Z }%
743   {\romannumeral0\XINT_mul_fork #2\Z #4\Z }%
744   {\romannumeral0\XINT_mul_fork #2\Z #3\Z }%
745 }%
746 \def\XINT_gctcv_loop_j #1#2%
747 {%
748   \expandafter\XINT_gctcv_loop_k\expandafter {\expandafter{#2}{#1}}%
749 }%
750 \def\XINT_gctcv_loop_k #1#2%
751 {%
752   \expandafter\XINT_gctcv_loop_l\expandafter {\expandafter{#2}#1}%
753 }%

```

```

754 \def\XINT_gctcv_loop_l #1#2%
755 {%
756   \expandafter\XINT_gctcv_loop_m\expandafter {\expandafter{#2}#1}%
757 }%
758 \def\XINT_gctcv_loop_m #1#2{\XINT_gctcv_loop_a {#2}#1}%
759 \def\XINT_gctcv_end #1.#2#3#4#5#6{ #6}%

```

9.20 \xintiGctoCv

```

760 \def\xintiGctoCv {\romannumeral0\xintigctocv }%
761 \def\xintigctocv #1%
762 {%
763   \expandafter\XINT_igctcv_prep \romannumeral-‘0#1+\xint_relax/%
764 }%
765 \def\XINT_igctcv_prep
766 {%
767   \XINT_igctcv_loop_a {}1001%
768 }%
769 \def\XINT_igctcv_loop_a #1#2#3#4#5#6+%
770 {%
771   \expandafter\XINT_igctcv_loop_b
772   \romannumeral-‘0#6.{#2}{#3}{#4}{#5}{#1}%
773 }%
774 \def\XINT_igctcv_loop_b #1.#2#3#4#5%
775 {%
776   \expandafter\XINT_igctcv_loop_c\expandafter
777   {\romannumeral0\xintiiadd {#5}{\XINT_Mul {#1}{#3}}}%
778   {\romannumeral0\xintiiadd {#4}{\XINT_Mul {#1}{#2}}}%
779   {{#2}{#3}}}%
780 }%
781 \def\XINT_igctcv_loop_c #1#2%
782 {%
783   \expandafter\XINT_igctcv_loop_f\expandafter {\expandafter{#2}{#1}}%
784 }%
785 \def\XINT_igctcv_loop_f #1#2#3#4/%
786 {%
787   \xint_gob_til_xint_relax #4\XINT_igctcv_end_a\xint_relax
788   \expandafter\XINT_igctcv_loop_g
789   \romannumeral-‘0#4.#1#2{#3}%
790 }%
791 \def\XINT_igctcv_loop_g #1.#2#3#4#5%
792 {%
793   \expandafter\XINT_igctcv_loop_h\expandafter
794   {\romannumeral0\XINT_mul_fork #1\Z #5\Z }%
795   {\romannumeral0\XINT_mul_fork #1\Z #4\Z }%
796   {{#2}{#3}}}%
797 }%
798 \def\XINT_igctcv_loop_h #1#2%
799 {%
800   \expandafter\XINT_igctcv_loop_i\expandafter {\expandafter{#2}{#1}}%
801 }%
802 \def\XINT_igctcv_loop_i #1#2{\XINT_igctcv_loop_k #2{#2#1}}%
803 \def\XINT_igctcv_loop_k #1#2%
804 {%

```

```

805 \expandafter\XINT_igctcv_loop_l\expandafter
806 {\romannumeral0\xintraewithzeros {#1/#2}}}%
807 }%
808 \def\XINT_igctcv_loop_l #1#2#3{\XINT_igctcv_loop_a {#3{#1}}#2}%1.09i removes [0]
809 \def\XINT_igctcv_end_a #1.#2#3#4#5%
810 {%
811 \expandafter\XINT_igctcv_end_b\expandafter
812 {\romannumeral0\xintraewithzeros {#2/#3}}}%
813 }%
814 \def\XINT_igctcv_end_b #1#2{ #2{#1}}% 1.09b removes [0]

```

9.21 \xintFtoCv

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

```

815 \def\xintFtoCv {\romannumeral0\xintftocv }%
816 \def\xintftocv #1%
817 {%
818 \xinticstocv {\xintFtoCs {#1}}}%
819 }%

```

9.22 \xintFtoCCv

```

820 \def\xintFtoCCv {\romannumeral0\xintftoccv }%
821 \def\xintftoccv #1%
822 {%
823 \xintigctocv {\xintFtoCC {#1}}}%
824 }%

```

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.

```

825 \def\xintCntoF {\romannumeral0\xintcntof }%
826 \def\xintcntof #1%
827 {%
828 \expandafter\XINT_cntf\expandafter {\the\numexpr #1}%
829 }%
830 \def\XINT_cntf #1#2%
831 {%
832 \ifnum #1>\xint_c_
833 \xint_afterfi {\expandafter\XINT_cntf_loop\expandafter
834 {\the\numexpr #1-1\expandafter}\expandafter
835 {\romannumeral-‘0#2{#1}}{#2}}}%
836 \else
837 \xint_afterfi
838 {\ifnum #1=\xint_c_
839 \xint_afterfi {\expandafter\space \romannumeral-‘0#2{0}}}%
840 \else \xint_afterfi { }% 1.09m now returns nothing.
841 \fi}%
842 \fi
843 }%
844 \def\XINT_cntf_loop #1#2#3%

```

```

845 {%
846   \ifnum #1>\xint_c_ \else \XINT_cntf_exit \fi
847   \expandafter\XINT_cntf_loop\expandafter
848   {\the\numexpr #1-1\expandafter }\expandafter
849   {\romannumeral0\xintadd {\xintDiv {1[0]}\{#2}\{#3\{#1}\}}%
850   {\#3}%
851 }%
852 \def\XINT_cntf_exit \fi
853   \expandafter\XINT_cntf_loop\expandafter
854   #1\expandafter #2#3%
855 {%
856   \fi\xint_gobble_ii #2%
857 }%

```

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.

```

858 \def\xintGCntoF {\romannumeral0\xintgcntof }%
859 \def\xintgcntof #1%
860 {%
861   \expandafter\XINT_gcntf\expandafter {\the\numexpr #1}%
862 }%
863 \def\XINT_gcntf #1#2#3%
864 {%
865   \ifnum #1>\xint_c_
866     \xint_afterfi {\expandafter\XINT_gcntf_loop\expandafter
867                   {\the\numexpr #1-1\expandafter}\expandafter
868                   {\romannumeral-'\0#2\{#1}\{#2}\{#3}\}}%
869   \else
870     \xint_afterfi
871     {\ifnum #1=\xint_c_
872       \xint_afterfi {\expandafter\space\romannumeral-'\0#2\{0}\}%
873       \else \xint_afterfi { }% 1.09m now returns nothing rather than 0/1[0]
874       \fi}%
875   \fi
876 }%
877 \def\XINT_gcntf_loop #1#2#3#4%
878 {%
879   \ifnum #1>\xint_c_ \else \XINT_gcntf_exit \fi
880   \expandafter\XINT_gcntf_loop\expandafter
881   {\the\numexpr #1-1\expandafter }\expandafter
882   {\romannumeral0\xintadd {\xintDiv {\#4\{#1}\{#2}\{#3\{#1}\}}%
883   {\#3}\{#4}%
884 }%
885 \def\XINT_gcntf_exit \fi
886   \expandafter\XINT_gcntf_loop\expandafter
887   #1\expandafter #2#3#4%
888 {%
889   \fi\xint_gobble_ii #2%
890 }%

```

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.

```

891 \def\xintCntoCs {\romannumeral0\xintcntocs }%
892 \def\xintcntocs #1%
893 {%
894   \expandafter\XINT_cntcs\expandafter {\the\numexpr #1}%
895 }%
896 \def\XINT_cntcs #1#2%
897 {%
898   \ifnum #1<0
899     \xint_afterfi { }% 1.09i: a 0/1[0] was here, now the macro returns nothing
900   \else
901     \xint_afterfi {\expandafter\XINT_cntcs_loop\expandafter
902                   {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
903                   {\romannumeral-'0#2{#1}}{#2}}% produced coeff not braced
904   \fi
905 }%
906 \def\XINT_cntcs_loop #1#2#3%
907 {%
908   \ifnum #1>-\xint_c_i \else \XINT_cntcs_exit \fi
909   \expandafter\XINT_cntcs_loop\expandafter
910   {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
911   {\romannumeral-'0#3{#1}, #2}{#3}% space added, 1.09m
912 }%
913 \def\XINT_cntcs_exit \fi
914   \expandafter\XINT_cntcs_loop\expandafter
915   #1\expandafter #2#3%
916 {%
917   \fi\XINT_cntcs_exit_b #2%
918 }%
919 \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.

```

920 \def\xintCntoGC {\romannumeral0\xintcntogc }%
921 \def\xintcntogc #1%
922 {%
923   \expandafter\XINT_cntgc\expandafter {\the\numexpr #1}%
924 }%
925 \def\XINT_cntgc #1#2%
926 {%
927   \ifnum #1<0

```

```

928 \xint_afterfi { }% 1.09i there was as strange 0/1[0] here, removed
929 \else
930 \xint_afterfi {\expandafter\xINT_cntgc_loop\expandafter
931               {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
932               {\expandafter{\romannumeral-'0#2{#1}}{#2}}}%
933 \fi
934 }%
935 \def\xINT_cntgc_loop #1#2#3%
936 {%
937   \ifnum #1>-\xint_c_i \else \XINT_cntgc_exit \fi
938   \expandafter\xINT_cntgc_loop\expandafter
939   {\the\numexpr #1-\xint_c_i\expandafter }\expandafter
940   {\expandafter{\romannumeral-'0#3{#1}}+1/#2}{#3}%
941 }%
942 \def\xINT_cntgc_exit \fi
943 \expandafter\xINT_cntgc_loop\expandafter
944 #1\expandafter #2#3%
945 {%
946   \fi\xINT_cntgc_exit_b #2%
947 }%
948 \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.

```

949 \def\xintGCntoGC {\romannumeral0\xintgcntogc }%
950 \def\xintgcntogc #1%
951 {%
952   \expandafter\xINT_gcntgc\expandafter {\the\numexpr #1}%
953 }%
954 \def\xINT_gcntgc #1#2#3%
955 {%
956   \ifnum #1<0
957     \xint_afterfi { }% 1.09i now returns nothing
958   \else
959     \xint_afterfi {\expandafter\xINT_gcntgc_loop\expandafter
960                   {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
961                   {\expandafter{\romannumeral-'0#2{#1}}{#2}{#3}}}%
962   \fi
963 }%
964 \def\xINT_gcntgc_loop #1#2#3#4%
965 {%
966   \ifnum #1>-\xint_c_i \else \XINT_gcntgc_exit \fi
967   \expandafter\xINT_gcntgc_loop_b\expandafter
968   {\expandafter{\romannumeral-'0#4{#1}}/#2}{#3{#1}}{#1}{#3}{#4}%
969 }%
970 \def\xINT_gcntgc_loop_b #1#2#3%
971 {%
972   \expandafter\xINT_gcntgc_loop\expandafter
973   {\the\numexpr #3-\xint_c_i \expandafter}\expandafter
974   {\expandafter{\romannumeral-'0#2}{#1}}%

```

```

975 }%
976 \def\XINT_gcntgc_exit \fi
977   \expandafter\XINT_gcntgc_loop_b\expandafter #1#2#3#4#5%
978 {%
979   \fi\XINT_gcntgc_exit_b #1%
980 }%
981 \def\XINT_gcntgc_exit_b #1/{ }%

```

9.28 \xintCstoGC

```

982 \def\xintCstoGC {\romannumeral0\xintcstogc }%
983 \def\xintcstogc #1%
984 {%
985   \expandafter\XINT_cstc_prep \romannumeral-‘0#1,\xint_relax,%
986 }%
987 \def\XINT_cstc_prep #1,{\XINT_cstc_loop_a {{#1}}}%
988 \def\XINT_cstc_loop_a #1#2,%
989 {%
990   \xint_gob_til_xint_relax #2\XINT_cstc_end\xint_relax
991   \XINT_cstc_loop_b {#1}{#2}%
992 }%
993 \def\XINT_cstc_loop_b #1#2{\XINT_cstc_loop_a {#1+1/{#2}}}%
994 \def\XINT_cstc_end\xint_relax\XINT_cstc_loop_b #1#2{ #1}%

```

9.29 \xintGctoGC

```

995 \def\xintGctoGC {\romannumeral0\xintgctogc }%
996 \def\xintgctogc #1%
997 {%
998   \expandafter\XINT_gctgc_start \romannumeral-‘0#1+\xint_relax/%
999 }%
1000 \def\XINT_gctgc_start {\XINT_gctgc_loop_a {}}%
1001 \def\XINT_gctgc_loop_a #1#2+#3/%
1002 {%
1003   \xint_gob_til_xint_relax #3\XINT_gctgc_end\xint_relax
1004   \expandafter\XINT_gctgc_loop_b\expandafter
1005   {\romannumeral-‘0#2}{#3}{#1}%
1006 }%
1007 \def\XINT_gctgc_loop_b #1#2%
1008 {%
1009   \expandafter\XINT_gctgc_loop_c\expandafter
1010   {\romannumeral-‘0#2}{#1}%
1011 }%
1012 \def\XINT_gctgc_loop_c #1#2#3%
1013 {%
1014   \XINT_gctgc_loop_a {#3{#2}+{#1}}/%
1015 }%
1016 \def\XINT_gctgc_end\xint_relax\expandafter\XINT_gctgc_loop_b
1017 {%
1018   \expandafter\XINT_gctgc_end_b
1019 }%
1020 \def\XINT_gctgc_end_b #1#2#3{ #3{#1}}%
1021 \XINT_restorecatcodes_endinput%

```

10 Package `xintexpr` implementation

Contents

10.1	Catcodes, ε -TeX and reload detection	236
10.2	Package identification	238
10.3	Locking and unlocking	238
10.4	<code>\XINT_expr_wrap</code> , <code>\XINT_iiexpr_wrap</code>	238
10.5	<code>\XINT_protectii</code> , <code>\XINT_expr_usethe</code>	238
10.6	<code>\XINT_expr_print</code> , <code>\XINT_iiexpr_print</code> , <code>\XINT_boolexpr_print</code>	238
10.7	<code>\xintexpr</code> , <code>\xintiexpr</code> , <code>\xintfloatexpr</code> , <code>\xintiifexpr</code> , <code>\xinttheexpr</code> , etc...	238
10.8	<code>\xintthe</code>	239
10.9	<code>\xintthecoords</code>	239
10.10	<code>\xintbareeval</code> , <code>\xintbarefloateval</code> , <code>\xintbareiieval</code>	239
10.11	<code>\xinteval</code> , <code>\xintiieval</code>	239
10.12	<code>\xintieval</code> , <code>\XINT_iexpr_wrap</code>	239
10.13	<code>\xintfloateval</code> , <code>\XINT_flexpr_wrap</code> , <code>\XINT_flexpr_print</code>	240
10.14	<code>\xintboolexpr</code> , <code>\xinttheboolexpr</code>	240
10.15	<code>\xintifboolexpr</code> , <code>\xintifboolfloatexpr</code> , <code>\xintifbooliiexpr</code>	240
10.16	Macros handling csv lists on output (for <code>\XINT_expr_print</code> et al. routines)	241
10.16.1	<code>\XINT::_end</code>	241
10.16.2	<code>\xintCSV::csv</code>	241
10.16.3	<code>\xintSPRaw</code> , <code>\xintSPRaw::csv</code>	241
10.16.4	<code>\xintIsTrue::csv</code>	241
10.16.5	<code>\xintRound::csv</code>	242
10.16.6	<code>\XINTinFloat::csv</code>	242
10.16.7	<code>\xintPFloat::csv</code>	242
10.17	<code>\XINT_expr_getnext</code> : fetching some number then an operator	243
10.18	The integer or decimal number or hexa-decimal number or function name or variable name or special hacky things big parser	243
10.18.1	Integral part	244
10.18.2	Fractional part	244
10.18.3	Scientific notation	245
10.18.4	Hexadecimal numbers	246
10.18.5	Function and variable names	247
10.19	<code>\XINT_expr_gettop</code> : finding the next operator or closing parenthesis or end of expression	248
10.20	Opening and closing parentheses, square brackets for lists, the <code>^C</code> for omit and abort within seq or rseq	249
10.21	The <code> </code> , <code> </code> , <code>&</code> , <code>&&</code> , <code><</code> , <code>></code> , <code>=</code> , <code>==</code> , <code><=</code> , <code>>=</code> , <code>!=</code> , <code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>^</code> , <code>**</code> , <code>//</code> , <code>/:</code> , <code>...</code> , <code>..[</code> , <code>]</code> , <code>[[</code> , <code>]]</code> , <code>[:</code> , <code>:</code> , <code>^C</code> , and <code>++</code> operators	251
10.21.1	The <code>]+</code> , <code>]-</code> , <code>]*</code> , <code>]/</code> , <code>]^</code> , <code>+]</code> , <code>-]</code> , <code>*]</code> , <code>/]</code> , and <code>^[</code> list operators	252
10.21.2	The <code>'and'</code> , <code>'or'</code> , <code>'xor'</code> , and <code>'mod'</code> as infix operator words	254
10.21.3	List selectors: <code>[list][N]</code> , <code>[list][:b]</code> , <code>[list][a:]</code> , <code>[list][a:b]</code>	255
10.22	Macros for a..b list generation	258
10.22.1	<code>\xintSeq::csv</code>	258
10.22.2	<code>\xintiiSeq::csv</code>	259
10.23	Macros for a..[d]..b list generation	260
10.23.1	<code>\xintSeqA::csv</code> , <code>\xintiiSeqA::csv</code> , <code>\XINTinFloatSeqA::csv</code>	260
10.23.2	<code>\xintSeqB::csv</code>	260
10.23.3	<code>\xintiiSeqB::csv</code>	260
10.23.4	<code>\XINTinFloatSeqB::csv</code>	261

Contents

10.24	The comma as binary operator	261
10.25	The minus as prefix operator of variable precedence level	262
10.26	? as two-way and ?? as three-way conditionals with braced branches	263
10.27	! as postfix factorial operator	263
10.28	The A/B[N] mechanism	263
10.29	For variables	264
10.29.1	Defining variables	264
10.29.2	Letters as dummy variables; the nil list	264
10.29.3	The omit and abort constructs	265
10.29.4	The @, @1, @2, @3, @4, @@, @@(1), ..., @@@, @@@(1), ... for recursion	265
10.30	For functions	266
10.31	The bool, togl, protect, unknown, and break "functions"	266
10.32	seq and the implementation of dummy variables	266
10.32.1	\XINT_expr_onlitteral_seq	267
10.32.2	\XINT_expr_onlitteral_seq_a	267
10.32.3	\XINT_isbalanced_a for \XINT_expr_onlitteral_seq_a	267
10.32.4	\XINT_allexpr_func_seqx, \XINT_allexpr_func_subx	268
10.32.5	break, abort, omit within seq	268
10.32.6	\XINT_expr_seq:_A	269
10.32.7	add and mul, , \XINT_expr_onlitteral_add, \XINT_expr_onlitteral_mul	269
10.32.8	\XINT_expr_func_opx, \XINT_flexpr_func_opx, \XINT_iiexpr_func_opx	269
10.32.9	\XINT_expr_op:_a,	269
10.32.10	subs, \XINT_expr_onlitteral_subs	270
10.33	rseq	270
10.33.1	\XINT_expr_rseqx	270
10.33.2	\XINT_expr_rseqy	271
10.33.3	\XINT_expr_rseq:_a etc...	271
10.33.4	\XINT_expr_rseq:_A etc...	271
10.34	rrseq	272
10.34.1	\XINT_expr_rrseqx	272
10.34.2	\XINT_expr_rrseqy	272
10.34.3	\XINT_expr_rrseq:_a etc...	272
10.34.4	\XINT_expr_rrseq:_A etc...	273
10.35	iter	273
10.35.1	\XINT_expr_iterx	274
10.35.2	\XINT_expr_ityy	274
10.35.3	\XINT_expr_iter:_a etc...	274
10.35.4	\XINT_expr_iter:_A etc...	275
10.36	Macros handling csv lists for functions with multiple comma separated arguments in expressions	275
10.36.1	\xintANDof:csv	275
10.36.2	\xintORof:csv	276
10.36.3	\xintXORof:csv	276
10.36.4	Generic csv routine	276
10.36.5	\xintMaxof:csv, \xintiiMaxof:csv	276
10.36.6	\xintMinof:csv, \xintiiMinof:csv	277
10.36.7	\xintSum:csv, cshxintiiSum:csv	277
10.36.8	\xintPrd:csv, \xintiiPrd:csv	277
10.36.9	\xintGCDof:csv, \xintLCMof:csv	277
10.36.10	\XINTinFloatdigits, \XINTinFloatSqrtdigits	277
10.36.11	\XINTinFloatMaxof:csv, \XINTinFloatMinof:csv	277
10.36.12	\XINTinFloatSum:csv, \XINTinFloatPrd:csv	278

10.37	The num, reduce, abs, sgn, frac, floor, ceil, sqr, sqrt, float, round, trunc, mod, quo, rem, gcd, lcm, max, min, '+', '*', '?', '!', not, all, any, xor, if, ifsgn, first, last, even, odd, and reversed functions	278
10.38	f-expandable versions of the SeqB::csv routines, for \xintNewExpr	283
10.38.1	\xintSeqB:f:csv	283
10.38.2	\xintiiSeqB:f:csv	284
10.38.3	\XINTinFloatSeqB:f:csv	285
10.39	\xintNewExpr, \xintNewIExpr, \xintNewFloatExpr, \xintNewIIExpr	285
10.39.1	\xintApply::csv	285
10.39.2	\xintApply:::csv	286
10.39.3	\XINT_expr_RApply::csv, \XINT_expr_LApply:::csv, \XINT_expr_RLApply:::csv	286
10.39.4	Mysterious stuff	286

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 \xintiiexpr 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).

novelties They are quite a few.

- *\xintiiexpr*, *\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,
- `\xinthecoords` 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 sequences, with tacit multiplication. It has also made small improvements. (speed gains in macro expansions in quite a few places.)

Also, 1.09i implements `\xintiexpr`, `\xinttheiexpr`. 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 `\xintiexpr`, 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 1.1 of [2014/10/28].

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.

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2 \catcode13=5 % ^^M
3 \endlinechar=13 %
4 \catcode123=1 % {
5 \catcode125=2 % }
6 \catcode64=11 % @
7 \catcode35=6 % #
8 \catcode44=12 % ,
9 \catcode45=12 % -
10 \catcode46=12 % .
11 \catcode58=12 % :
12 \def\z {\endgroup}%
13 \expandafter\let\expandafter\x\csname ver@xintexpr.sty\endcsname
14 \expandafter\let\expandafter\w\csname ver@xintfrac.sty\endcsname
15 \expandafter\let\expandafter\t\csname ver@xinttools.sty\endcsname
16 \expandafter
17 \ifx\csname PackageInfo\endcsname\relax
18 \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
19 \else
20 \def\y#1#2{\PackageInfo{#1}{#2}}%
21 \fi
22 \expandafter
23 \ifx\csname numexpr\endcsname\relax
24 \y{xintexpr}{\numexpr not available, aborting input}%
25 \aftergroup\endinput
26 \else
27 \ifx\x\relax % plain-TeX, first loading of xintexpr.sty
28 \ifx\w\relax % but xintfrac.sty not yet loaded.
29 \expandafter\def\expandafter\z\expandafter
30 {\z\input xintfrac.sty\relax}%
31 \fi
32 \ifx\t\relax % but xinttools.sty not yet loaded.
33 \expandafter\def\expandafter\z\expandafter
34 {\z\input xinttools.sty\relax}%
35 \fi
36 \else
37 \def\empty {}%
38 \ifx\x\empty % LaTeX, first loading,
39 % variable is initialized, but \ProvidesPackage not yet seen
40 \ifx\w\relax % xintfrac.sty not yet loaded.
41 \expandafter\def\expandafter\z\expandafter
42 {\z\RequirePackage{xintfrac}}%
43 \fi
44 \ifx\t\relax % xinttools.sty not yet loaded.
45 \expandafter\def\expandafter\z\expandafter
46 {\z\RequirePackage{xinttools}}%
47 \fi
48 \else
49 \aftergroup\endinput % xintexpr already loaded.
50 \fi
51 \fi

```

```
52 \fi
53 \z%
54 \XINTsetupcatcodes%
```

10.2 Package identification

```
55 \XINT_providespackage
56 \ProvidesPackage{xintexpr}%
57 [2014/10/28 v1.1 Expandable expression parser (jfb)]%
```

10.3 Locking and unlocking

```
58 \def\xint_gob_til_! #1!{% this ! has catcode 11
59 \edef\XINT_expr_lockscan#1!{\noexpand\expandafter\space\noexpand\csname .=#1\endcsname }%
60 \edef\XINT_expr_lockit #1{\noexpand\expandafter\space\noexpand\csname .=#1\endcsname }%
61 \def\XINT_expr_inintpart #1!{\XINT_num{#1}}%
62 \def\XINT_expr_infracpart #1e#2!{#1![\the\numexpr#2-\xintLength{#1}]!}%
63 \def\XINT_expr_inexppart e#1!{![\the\numexpr #1]!}%
64 % je dois réfléchir si je dois bloquer expansion après unlock_a, à cause de nil.
65 \def\XINT_expr_unlock {\expandafter\XINT_expr_unlock_a\string }%
66 \def\XINT_expr_unlock_a #1.={%
67 \def\XINT_expr_unexpectedtoken {\xintError:ignored }%
68 \let\XINT_expr_done\space
```

10.4 \XINT_expr_wrap, \XINT_iiexpr_wrap

```
69 \def\XINT_expr_wrap { !\XINT_expr_usethe\XINT_protectii\XINT_expr_print }%
70 \def\XINT_iiexpr_wrap { !\XINT_expr_usethe\XINT_protectii\XINT_iiexpr_print }%
```

10.5 \XINT_protectii, \XINT_expr_usethe

```
71 \def\XINT_protectii #1{\noexpand\XINT_protectii\noexpand #1\noexpand }%
72 \protected\def\XINT_expr_usethe\XINT_protectii {\xintError:missing_xintthe!}%
73 % \catcode'. 11 \catcode32=11 \catcode'=11\relax%
74 % \def\XINT_expr_nil{!\XINT_expr_usethe\XINT_protectii\XINT_expr_unlock\.= }%
75 % \catcode'=12\catcode32=10\relax\catcode'. 12
```

10.6 \XINT_expr_print, \XINT_iiexpr_print, \XINT_boolexpr_print

See also the `\XINT_flexpr_print` which is special, below.

```
76 \def\XINT_expr_print #1{\xintSPRaw::csv {\XINT_expr_unlock #1}}%
77 \def\XINT_iiexpr_print #1{\xintCSV::csv {\XINT_expr_unlock #1}}%
78 \def\XINT_boolexpr_print #1{\xintIsTrue::csv {\XINT_expr_unlock #1}}%
```

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

```
79 \def\xintexpr {\romannumeral0\xinteval }%
80 \def\xintiexpr {\romannumeral0\xintieval }%
81 \def\xintfloatexpr {\romannumeral0\xintfloateval }%
82 \def\xintiexpr {\romannumeral0\xintiieval }%
83 \def\xinttheexpr
84 {\romannumeral-'\0\expandafter\XINT_expr_print\romannumeral0\xintbareeval }%
85 \def\xinttheiexpr {\romannumeral-'\0\xintthe\xintiexpr }%
86 \def\xintthefloatexpr {\romannumeral-'\0\xintthe\xintfloatexpr }%
87 \def\xinttheiexpr
```

```
88 {\romannumeral-‘0\expandafter\XINT_iexpr_print\romannumeral0\xintbareiieval}%
89 % \let\xintnumexpr \xintiexpr % was deprecated, now obsolete with 1.1
90 % \let\xintthenumexpr\xinttheiexpr % was deprecated, now obsolete with 1.1
```

10.8 \xintthe

```
91 \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 craziness 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!

```
92 \def\xintthecoords #1{\romannumeral-‘0\expandafter\expandafter\expandafter
93     \XINT_thecoords_a
94     \expandafter\xint_gobble_iii\romannumeral0#1}%
95 \def\XINT_thecoords_a #1#2 #1=print macro, indispensable for scientific notation
96     {\expandafter\XINT_expr_unlock\csname.=\expandafter\XINT_thecoords_b
97         \romannumeral-‘0#1#2,!,!,^\\endcsname}%
98 \def\XINT_thecoords_b #1#2,#3#4,%
99     {\xint_gob_til_! #3\XINT_thecoords_c ! (#1#2, #3#4)\XINT_thecoords_b}%
100 \def\XINT_thecoords_c #1^{}%%
```

10.10 \xintbareeval, \xintbarefloateval, \xintbareiieval

```
101 \def\xintbareeval
102     {\expandafter\XINT_expr_until_end_a\romannumeral-‘0\XINT_expr_getnext}%
103 \def\xintbarefloateval
104     {\expandafter\XINT_flexpr_until_end_a\romannumeral-‘0\XINT_expr_getnext}%
105 \def\xintbareiieval
106     {\expandafter\XINT_iexpr_until_end_a\romannumeral-‘0\XINT_expr_getnext}%%
```

10.11 \xinteval, \xintiieval

```
107 \def\xinteval {\expandafter\XINT_expr_wrap\romannumeral0\xintbareeval}%
108 \def\xintiieval {\expandafter\XINT_iexpr_wrap\romannumeral0\xintbareiieval}%
```

10.12 \xintieval, \XINT_iexpr_wrap

Optional argument since 1.1

```
109 \def\xintieval #1%
110     {\ifx [#1\expandafter\XINT_iexpr_withopt\else\expandafter\XINT_iexpr_noopt \fi #1}%
111 \def\XINT_iexpr_noopt
112     {\expandafter\XINT_iexpr_wrap \expandafter 0\romannumeral0\xintbareeval}%
113 \def\XINT_iexpr_withopt [#1]%
114 {%
115     \expandafter\XINT_iexpr_wrap\expandafter
116     {\the\numexpr \xint_zapspace #1 \xint_bye\xint_bye\expandafter}%%
```



```

117 \romannumeral0\xintbareeval
118 }%
119 \def\xINT_iexpr_wrap #1#2%
120 {%
121 \expandafter\xINT_expr_wrap
122 \csname .=\xintRound::csv {#1}{\XINT_expr_unlock #2}\endcsname
123 }%

```

10.13 \xintfloateval, \XINT_flexpr_wrap, \XINT_flexpr_print

Optional argument since 1.1

```

124 \def\xintfloateval #1%
125 {%
126 \ifx [#1\expandafter\xINT_flexpr_withopt_a\else\expandafter\xINT_flexpr_noopt
127 \fi #1%
128 }%
129 \def\xINT_flexpr_noopt
130 {%
131 \expandafter\xINT_flexpr_withopt_b\expandafter\xinttheDigits
132 \romannumeral0\xintbarefloateval
133 }%
134 \def\xINT_flexpr_withopt_a [#1]%
135 {%
136 \expandafter\xINT_flexpr_withopt_b\expandafter
137 {\the\numexpr\xint_zapspaces #1 \xint_bye\xint_bye\expandafter}%
138 \romannumeral0\xintbarefloateval
139 }%
140 \def\xINT_flexpr_withopt_b #1#2%
141 {%
142 \expandafter\xINT_flexpr_wrap\csname .;#1.=% ; and not : as before b'cause NewExpr
143 \XINTinFloat::csv {#1}{\XINT_expr_unlock #2}\endcsname
144 }%
145 \def\xINT_flexpr_wrap { !\XINT_expr_usethe\xINT_protectii\xINT_flexpr_print }%
146 \def\xINT_flexpr_print #1%
147 {%
148 \expandafter\xintPFloat::csv
149 \romannumeral-'\0\expandafter\xINT_expr_unlock_sp\string #1!%
150 }%
151 \catcode': 12
152 \def\xINT_expr_unlock_sp #1.;#2.=#3!{{#2}{#3}}%
153 \catcode': 11

```

10.14 \xintboolexpr, \xinttheboolexpr

```

154 \def\xintboolexpr {\romannumeral0\expandafter\expandafter\expandafter
155 \XINT_boolexpr_done \expandafter\xint_gobble_iv\romannumeral0\xinteval }%
156 \def\xinttheboolexpr {\romannumeral-'\0\expandafter\expandafter\expandafter
157 \XINT_boolexpr_print\expandafter\xint_gobble_iv\romannumeral0\xinteval }%
158 \def\xINT_boolexpr_done { !\XINT_expr_usethe\xINT_protectii\xINT_boolexpr_print }%

```

10.15 \xintifboolexpr, \xintifboolfloatexpr, \xintifboolliexpr

Do not work with comma separated expressions.


```

159 \def\xintifboolexpr      #1{\romannumeral0\xintifnotzero {\xinttheexpr #1\relax}}%
160 \def\xintifboolfloatexpr #1{\romannumeral0\xintifnotzero {\xintthefloatexpr #1\relax}}%
161 \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`

```

162 \def\XINT::_end #1,#2{\xint_gobble_i #2}%

```

10.16.2 `\xintCSV::csv`

pour `\xinttheiiexpr`

```

163 \def\xintCSV::csv #1{\expandafter\XINT_csv::_a\romannumeral-‘0#1,^,%}
164 \def\XINT_csv::_a {\XINT_csv::_b {}}%
165 \def\XINT_csv::_b #1#2,{\XINT_csv::_c #2,{#1}}%
166 \def\XINT_csv::_c #1{\if ^#1\expandafter\XINT::_end\fi\XINT_csv::_d #1}%
167 \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.

```

168 \def\xintSPRaw    {\romannumeral0\xintspraw}%
169 \def\xintspraw    #1{\expandafter\XINT_spraw\romannumeral-‘0#1[\W]}%
170 \def\XINT_spraw #1[#2#3]{\xint_gob_til_W #2\XINT_spraw_a\W\XINT_spraw_p #1[#2#3]}%
171 \def\XINT_spraw_a\W\XINT_spraw_p #1[\W]{ #1}%
172 \def\XINT_spraw_p #1[\W]{\xintpraw {#1}}%
173 \def\xintSPRaw::csv #1{\romannumeral0\expandafter\XINT_spraw::_a\romannumeral-‘0#1,^,%}
174 \def\XINT_spraw::_a {\XINT_spraw::_b {}}%
175 \def\XINT_spraw::_b #1#2,{\XINT_spraw::_c #2,{#1}}%
176 \def\XINT_spraw::_c #1{\if ,#1\xint_dothis\XINT_spraw::_e\fi
177     \if ^#1\xint_dothis\XINT::_end\fi
178     \xint_orthat\XINT_spraw::_d #1}%
179 \def\XINT_spraw::_d #1,{\expandafter\XINT_spraw::_e\romannumeral0\XINT_spraw #1[\W],}%
180 \def\XINT_spraw::_e #1,#2{\XINT_spraw::_b {#2, #1}}%

```

10.16.4 `\xintIsTrue::csv`

```

181 \def\xintIsTrue::csv #1{\romannumeral0\expandafter\XINT_istrue::_a\romannumeral-‘0#1,^,%}
182 \def\XINT_istrue::_a {\XINT_istrue::_b {}}%
183 \def\XINT_istrue::_b #1#2,{\XINT_istrue::_c #2,{#1}}%
184 \def\XINT_istrue::_c #1{\if ,#1\xint_dothis\XINT_istrue::_e\fi
185     \if ^#1\xint_dothis\XINT::_end\fi
186     \xint_orthat\XINT_istrue::_d #1}%

```

```
187 \def\XINT_istrue::_d #1,{\expandafter\XINT_istrue::_e\romannumeral0\xintisnotzero {#1},}%
188 \def\XINT_istrue::_e #1,#2{\XINT_istrue::_b {#2, #1}}%
```

10.16.5 \xintRound::csv

Pour `\xintexpr` 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`.

```
189 \def\XINT_:::_end #1,#2#3{\xint_gobble_i #3}%
190 \def\xintRound::csv #1#2{\romannumeral0\expandafter\XINT_round::_b\expandafter
191   {\the\numexpr#1\expandafter}\expandafter{\expandafter}\romannumeral-‘0#2,^,}%
192 \def\XINT_round::_b #1#2#3,{\XINT_round::_c #3,{#1}{#2}}%
193 \def\XINT_round::_c #1{\if ,#1\xint_dothis\XINT_round::_e\fi
194   \if ^#1\xint_dothis\XINT_:::_end\fi
195   \xint_orthat\XINT_round::_d #1}%
196 \def\XINT_round::_d #1,#2{%
197   \expandafter\XINT_round::_e\romannumeral0\ifnum#2>\xint_c_
198   \expandafter\xintround\else\expandafter\xintiround\fi {#2}{#1},{#2}}%
199 \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`.

```
200 \def\XINTinFloat::csv #1#2{\romannumeral0\expandafter\XINT_infloat::_b\expandafter
201   {\the\numexpr #1\expandafter}\expandafter{\expandafter}\romannumeral-‘0#2,^,}%
202 \def\XINT_infloat::_b #1#2#3,{\XINT_infloat::_c #3,{#1}{#2}}%
203 \def\XINT_infloat::_c #1{\if ,#1\xint_dothis\XINT_infloat::_e\fi
204   \if ^#1\xint_dothis\XINT_:::_end\fi
205   \xint_orthat\XINT_infloat::_d #1}%
206 \def\XINT_infloat::_d #1,#2%
207   {\expandafter\XINT_infloat::_e\romannumeral0\XINTinfloat [#2]{#1},{#2}}%
208 \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`

```
209 \def\xintPFloat::csv #1#2{\romannumeral0\expandafter\XINT_pfloat::_b\expandafter
210   {\the\numexpr #1\expandafter}\expandafter{\expandafter}\romannumeral-‘0#2,^,}%
211 \def\XINT_pfloat::_b #1#2#3,{\XINT_pfloat::_c #3,{#1}{#2}}%
212 \def\XINT_pfloat::_c #1{\if ,#1\xint_dothis\XINT_pfloat::_e\fi
213   \if ^#1\xint_dothis\XINT_:::_end\fi
214   \xint_orthat\XINT_pfloat::_d #1}%
215 \def\XINT_pfloat::_d #1,#2%
216   {\expandafter\XINT_pfloat::_e\romannumeral0\XINT_pfloat_opt [\xint_relax #2]{#1},{#2}}%
217 \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.

```

218 \def\XINT_expr_getnext #1%
219 {%
220   \expandafter\XINT_expr_getnext_a\romannumeral-‘0#1%
221 }%
222 \def\XINT_expr_getnext_a #1%
223 {% screens out sub-expressions and \count or \dimen registers/variables
224   \xint_gob_til_! #1\XINT_expr_subexpr !% recall this ! has catcode 11
225   \ifcat\relax#1% \count or \numexpr etc... token or count, dimen, skip cs
226     \expandafter\XINT_expr_countetc
227   \else
228     \expandafter\expandafter\expandafter\XINT_expr_getnextfork\expandafter\string
229   \fi
230   #1%
231 }%
232 \def\XINT_expr_subexpr !#1\fi !{\expandafter\XINT_expr_getop\xint_gobble_iii }%
233 \def\XINT_expr_countetc #1%
234 {%
235   \ifx\count#1\else\ifx#1\dimen\else\ifx#1\numexpr\else\ifx#1\dimexpr\else
236   \ifx\skip#1\else\ifx\glueexpr#1\else\ifx\fontdimen#1\else
237     \XINT_expr_unpackvar
238   \fi\fi\fi\fi\fi\fi\fi
239   \expandafter\XINT_expr_getnext\number #1%
240 }%
241 \def\XINT_expr_unpackvar\fi\fi\fi\fi\fi\fi\fi\expandafter\XINT_expr_getnext\number #1%
242   {\fi\fi\fi\fi\fi\fi\fi\expandafter\XINT_expr_getop\csname .=#1#2\endcsname }%
243 \begingroup
244 \lccode‘*’=#
245 \lowercase{\endgroup
246 \def\XINT_expr_getnextfork #1{%
247   \if#1*\xint_dothis {\XINT_expr_scan_macropar *}\fi
248   \if#1[\xint_dothis {\xint_c_xviii ({} }\fi
249   \if#1+\xint_dothis \XINT_expr_getnext \fi
250   \if#1.\xint_dothis {\XINT_expr_scandec_II\XINT_expr_infracpart}\fi
251   \if#1-\xint_dothis -\fi
252   \if#1(\xint_dothis {\xint_c_xviii ({} }\fi
253   \xint_orthat {\XINT_expr_scan_nbr_or_func #1}%
254 }}%
255 \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

```

256 \catcode96 11 % ‘
257 \def\XINT_expr_scan_nbr_or_func #1% this #1 has necessarily here catcode 12
258 {%
259   \if "#1\xint_dothis \XINT_expr_scanhex_I\fi
260   \if ‘#1\xint_dothis {\XINT_expr_onlitteral_‘}\fi

```

```

261 \ifnum \xint_c_ix<1#1 \xint_dothis \XINT_expr_scandec_I\fi
262 \xint_orthat \XINT_expr_scanfunc #1%
263 }%
264 \catcode96 12 % ‘
265 \def\XINT_expr_scandec_I
266 {%
267 \expandafter\XINT_expr_getop\romannumeral-‘0\expandafter
268 \XINT_expr_lockscan\romannumeral0\expandafter\XINT_expr_inintpart
269 \romannumeral-‘0\XINT_expr_scanintpart_b
270 }%
271 \def\XINT_expr_scandec_II
272 {%
273 \expandafter\XINT_expr_getop\romannumeral-‘0\expandafter
274 \XINT_expr_lockscan\romannumeral0\expandafter\XINT_expr_inintpart
275 \romannumeral-‘0\XINT_expr_scanfracpart_b
276 }%

10.18.1 Integral part

277 \def\XINT_expr_scanintpart_a #1%
278 {% careful that ! has catcode letter here
279 \ifcat \relax #1\xint_dothis{!!#1}\fi % stops the scan
280 \if e#1\xint_dothis{\expandafter\XINT_expr_inexppart
281 \romannumeral-‘0\XINT_expr_scanexppart_a e}\fi
282 \if E#1\xint_dothis{\expandafter\XINT_expr_inexppart
283 \romannumeral-‘0\XINT_expr_scanexppart_a e}\fi
284 % \if @#1\xint_dothis{!*#1}\fi % tacit multiplication later
285 % \if _#1\xint_dothis{!*#1}\fi % tacit multiplication for variables
286 \ifcat a#1\xint_dothis{!!*#1}\fi % includes subexpressions (#1!= letter)
287 \xint_orthat {\expandafter\XINT_expr_scanintpart_aa\string #1}%
288 }%
289 \def\XINT_expr_scanintpart_aa #1%
290 {%
291 \if .#1\xint_dothis\XINT_expr_scandec_transition\fi
292 \ifnum \xint_c_ix<1#1 \xint_dothis\XINT_expr_scanintpart_b\fi
293 \xint_orthat {!!}#1%
294 }%
295 \def\XINT_expr_scanintpart_b #1#2%
296 {%
297 \expandafter #1\romannumeral-‘0\expandafter
298 \XINT_expr_scanintpart_a\romannumeral-‘0#2%
299 }%
300 \def\XINT_expr_scandec_transition .#1%
301 {%
302 \expandafter\XINT_expr_scandec_trans_a\romannumeral-‘0#1%
303 }%
304 \def\XINT_expr_scandec_trans_a #1%
305 {%
306 \if .#1\xint_dothis{!!..}\fi
307 \xint_orthat {\expandafter\XINT_expr_infracpart
308 \romannumeral-‘0\XINT_expr_scanfracpart_a #1}%
309 }%

```

10.18.2 Fractional part

```

310 \def\XINT_expr_scanfracpart_a #1%
311 {%
312   \ifcat \relax #1\xint_dothis{e!#1}\fi % stops the scan
313   \if e#1\xint_dothis{\XINT_expr_scanexppart_a e}\fi
314   \if E#1\xint_dothis{\XINT_expr_scanexppart_a e}\fi
315   \ifcat a#1\xint_dothis{e!*#1}\fi % and also the case of subexpressions (!)
316   \xint_orthat {\expandafter\XINT_expr_scanfracpart_aa\string #1}%
317 }%
318 \def\XINT_expr_scanfracpart_aa #1%
319 {%
320   \ifnum \xint_c_ix<1#1
321     \expandafter\XINT_expr_scanfracpart_b
322   \else
323     \xint_afterfi {e!}%
324   \fi
325   #1%
326 }%
327 \def\XINT_expr_scanfracpart_b #1#2%
328 {%
329   \expandafter #1\romannumeral-'0\expandafter
330   \XINT_expr_scanfracpart_a\romannumeral-'0#2%
331 }%

```

10.18.3 Scientific notation

```

332 \def\XINT_expr_scanexppart_a #1#2%
333 {%
334   \expandafter #1\romannumeral-'0\expandafter
335   \XINT_expr_scanexppart_b\romannumeral-'0#2%
336 }%
337 \def\XINT_expr_scanexppart_b #1%
338 {%
339   \ifcat \relax #1\xint_dothis{0!#1}\fi % stops the scan (incorrect syntax)
340   \ifcat a#1\xint_dothis{0!*#1}\fi % idem
341   \if +#1\xint_dothis {\XINT_expr_scanexppart_a +}\fi
342   \if -#1\xint_dothis {\XINT_expr_scanexppart_a -}\fi
343   \xint_orthat {\expandafter\XINT_expr_scanexppart_c\string #1}%
344 }%
345 \def\XINT_expr_scanexppart_c #1%
346 {%
347   \ifnum \xint_c_ix<1#1
348     \expandafter\XINT_expr_scanexppart_d
349   \else
350     \expandafter !%
351   \fi
352   #1%
353 }%
354 \def\XINT_expr_scanexppart_d #1#2%
355 {%
356   \expandafter #1\romannumeral-'0\expandafter
357   \XINT_expr_scanexppart_e\romannumeral-'0#2%
358 }%
359 \def\XINT_expr_scanexppart_e #1%
360 {%

```

```

361 \ifcat \relax #1\xint_dothis{!#1}\fi % stops the scan
362 \ifcat a#1\xint_dothis{!*#1}\fi % idem
363 \xint_orthat {\expandafter\XINT_expr_scanexppart_f\string #1}%
364 }%
365 \def\XINT_expr_scanexppart_f #1%
366 {%
367 \ifnum \xint_c_ix<1#1
368 \expandafter\XINT_expr_scanexppart_d
369 \else
370 \expandafter !%
371 \fi
372 #1%
373 }%

```

10.18.4 Hexadecimal numbers

```

374 \def\XINT_expr_scanhex_I #1%
375 {%
376 \expandafter\XINT_expr_getop\romannumeral-'0\expandafter
377 \XINT_expr_lockscan\expandafter\XINT_expr_inhex
378 \romannumeral-'0\XINT_expr_scanhexI_a
379 }%
380 \def\XINT_expr_inhex #1.#2#3;% expanded inside \csname..\endcsname
381 {%
382 \if#2I\xintHexToDec{#1}%
383 \else
384 \xintiiMul{\xintiiPow{625}{\xintLength{#3}}}{\xintHexToDec{#1#3}}%
385 [\the\numexpr-4*\xintLength{#3}]%
386 \fi
387 }%
388 \def\XINT_expr_scanhexI_a #1%
389 {%
390 \ifcat #1\relax\xint_dothis{.I;!#1}\fi
391 \ifx !#1\xint_dothis{.I;!*!}\fi % tacit multiplication
392 \xint_orthat {\expandafter\XINT_expr_scanhexI_aa\string #1}%
393 }%
394 \def\XINT_expr_scanhexI_aa #1%
395 {%
396 \if\ifnum'#1>' /
397 \ifnum'#1>' 9
398 \ifnum'#1>' @
399 \ifnum'#1>' F
400 0\else1\fi\else0\fi\else1\fi\else0\fi 1%
401 \expandafter\XINT_expr_scanhexI_b
402 \else
403 \if .#1%
404 \expandafter\xint_firstoftwo
405 \else % gather what we got so far, leave catcode 12 #1 in stream
406 \expandafter\xint_secondoftwo
407 \fi
408 {\expandafter\XINT_expr_scanhex_transition}%
409 {\xint_afterfi {.I;!}}%
410 \fi
411 #1%

```

```

412 }%
413 \def\XINT_expr_scanhexI_b #1#2%
414 {%
415     \expandafter #1\romannumeral-'0\expandafter
416     \XINT_expr_scanhexI_a\romannumeral-'0#2%
417 }%
418 \def\XINT_expr_scanhex_transition .#1%
419 {%
420     \expandafter.\expandafter.\romannumeral-'0\expandafter
421     \XINT_expr_scanhexII_a\romannumeral-'0#1%
422 }%
423 \def\XINT_expr_scanhexII_a #1%
424 {%
425     \ifcat #1\relax\xint_dothis{;!#1}\fi
426     \ifx      !#1\xint_dothis{;!*!}\fi % tacit multiplication
427     \xint_orthat {\expandafter\XINT_expr_scanhexII_aa\string #1}%
428 }%
429 \def\XINT_expr_scanhexII_aa #1%
430 {%
431     \if\ifnum'#1>' /
432     \ifnum'#1>'9
433     \ifnum'#1>'@
434     \ifnum'#1>'F
435     0\else1\fi\else0\fi\else1\fi\else0\fi 1%
436     \expandafter\XINT_expr_scanhexII_b
437     \else
438     \xint_afterfi {;!}%
439     \fi
440     #1%
441 }%
442 \def\XINT_expr_scanhexII_b #1#2%
443 {%
444     \expandafter #1\romannumeral-'0\expandafter
445     \XINT_expr_scanhexII_a\romannumeral-'0#2%
446 }%

```

10.18.5 Function and variable names

```

447 \def\XINT_expr_scanfunc
448 {%
449     \expandafter\XINT_expr_func\romannumeral-'0\XINT_expr_scanfunc_a
450 }%
451 \def\XINT_expr_scanfunc_a #1#2%
452 {%
453     \expandafter #1\romannumeral-'0\expandafter\XINT_expr_scanfunc_b\romannumeral-'0#2%
454 }%
455 \def\XINT_expr_scanfunc_b #1%
456 {%
457     \ifx !#1\xint_dothis{\xint_firstoftwo{(_*!)}\fi
458     \ifcat \relax#1\xint_dothis{(_}\fi
459     \if (#1\xint_dothis{\xint_firstoftwo{(')}\fi
460     \if _#1\xint_dothis \XINT_expr_scanfunc_a \fi
461     \if @#1\xint_dothis \XINT_expr_scanfunc_a \fi
462     \ifnum \xint_c_ix<1\string#1 \xint_dothis \XINT_expr_scanfunc_a \fi

```

```

463 \ifcat a#1\xint_dothis \XINT_expr_scanfunc_a \fi
464 \xint_orthat {(_}%
465   #1%
466 }%
467 \def\xint_expr_func #1(#2%
468 {% #2=' pour une fonction, #2=_ pour une variable
469   \if #2'\ifcsname XINT_expr_var_#1\endcsname
470     \expandafter\expandafter\expandafter\xint_thirdofthree
471   \fi\fi
472   \xint_firstoftwo {\xint_c_xviii #2{#1}}{\xint_c_xviii _{#1}}*}%
473 }%

```

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

Release 1.1 implements multi-character operators.

```

474 \def\xint_expr_getop #1#2% this #1 is the current locked computed value
475 {%
476   \expandafter\xint_expr_getop_a\expandafter #1\romannumeral-'0#2%
477 }%
478 \catcode'* 11
479 \def\xint_expr_getop_a #1#2%
480 {%
481   \ifx   \relax #2\xint_dothis\xint_firstofthree\fi
482   \ifcat \relax #2\xint_dothis\xint_secondofthree\fi
483   \if    _#2\xint_dothis      \xint_secondofthree\fi
484   \if    @#2\xint_dothis      \xint_secondofthree\fi
485   \if    (#2\xint_dothis      \xint_secondofthree\fi
486   \ifx   !#2\xint_dothis      \xint_secondofthree\fi
487   \xint_orthat \xint_thirdofthree
488   {\XINT_expr_foundend #1}%
489   {\XINT_expr_precedence_* *#1#2}% tacit multiplication
490   {\XINT_expr_getop_b #2#1}%
491 }%
492 \catcode'* 12
493 \def\xint_expr_foundend {\xint_c_ \relax }% \relax is a place holder here.
494 \def\xint_expr_getop_b #1%
495 {% ? and : a special syntax in \xintexpr as they are
496 % followed by braced arguments, and thus we must intercept them here.
497 % I wanted to change this but now I don't have time to think about it.
498 % 1.1 removes : as logic operator. Replaced by ??.
499   \if '#1\xint_dothis{\XINT_expr_binopwrdrd }\fi
500   \if ?#1\xint_dothis{\XINT_expr_precedence_? ?}\fi
501 %   \if :#1\xint_dothis{\XINT_expr_precedence_: :}\fi % must preserve braces!
502   \xint_orthat      {\XINT_expr_scanop_a #1}%
503 }%
504 \def\xint_expr_binopwrdrd #1#2'\{\expandafter\xint_expr_foundop_a
505   \csname XINT_expr_itself_\xint_zapspaces #2 \xint_bye\xint_bye\endcsname #1}%
506 \def\xint_expr_scanop_a #1#2#3%
507   {\expandafter\xint_expr_scanop_b\expandafter #1\expandafter #2\romannumeral-'0#3}%
508 \def\xint_expr_scanop_b #1#2#3%
509 {%

```



```

510 \ifcat#3\relax\xint_dothis{\XINT_expr_foundop_a #1#2#3}\fi
511 \ifcsname XINT_expr_itself_#1#3\endcsname
512 \xint_dothis
513     {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#3\endcsname #2}\fi
514 \xint_orthat {\XINT_expr_foundop_a #1#2#3}%
515 }%
516 \def\XINT_expr_scanop_c #1#2#3%
517 {%
518     \expandafter\XINT_expr_scanop_d\expandafter #1\expandafter #2\romannumeral-`0#3%
519 }%
520 \def\XINT_expr_scanop_d #1#2#3%
521 {%
522     \ifcat#3\relax \xint_dothis{\XINT_expr_foundop #1#2#3}\fi
523     \ifcsname XINT_expr_itself_#1#3\endcsname
524     \xint_dothis
525         {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#3\endcsname #2}\fi
526     \xint_orthat {\csname XINT_expr_precedence_#1\endcsname #1#2#3}%
527 }%
528 \def\XINT_expr_foundop_a #1%
529 {%
530     \ifcsname XINT_expr_precedence_#1\endcsname
531     \csname XINT_expr_precedence_#1\endcsname\expandafter\endcsname
532     \expandafter #1%
533     \else
534     \xint_afterfi{\XINT_expr_unknown_operator {#1}\XINT_expr_getop}%
535     \fi
536 }%
537 \def\XINT_expr_unknown_operator #1{\xintError:removed \xint_gobble_i {#1}}%
538 \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

```

539 \catcode') 11
540 \def\XINT_tmpa #1#2#3#4% (avant #4#5)
541 {%
542     \def#1##1%
543     {%
544         \xint_UDsignfork
545             ##1{\expandafter#1\romannumeral-`0#3}%
546             -{#2##1}%
547         \krof
548     }%
549     \def#2##1##2%
550     {%
551         \ifcase ##1\xint_afterfi
552         {\ifx\XINT_expr_itself_`^C ##2\xint_dothis
553         {\expandafter#1\romannumeral-`0\expandafter\XINT_expr_getnext\xint_gobble_i}\fi
554         \xint_orthat \XINT_expr_done }%
555         \or\xint_afterfi{\XINT_expr_extra_)
556             \expandafter #1\romannumeral-`0\XINT_expr_getop }%
557     \else

```

```

558     \xint_afterfi{\expandafter#1\romannumeral-'0\csname XINT_#4_op_##2\endcsname }%
559     \fi
560 }%
561 }%
562 \def\xintexpr_extra_){\xintError:removed }%
563 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
564     \expandafter\xint_tmpa
565     \csname XINT_#1_until_end_a\expandafter\endcsname
566     \csname XINT_#1_until_end_b\expandafter\endcsname
567     \csname XINT_#1_op_-vi\endcsname
568     {#1}%
569 }%
570 \def\xint_tmpa #1#2#3#4#5#6%
571 {%
572     \def #1##1{\expandafter #3\romannumeral-'0\xintexpr_getnext }%
573     \def #2{\expandafter #3\romannumeral-'0\xintexpr_getnext }%
574     \def #3##1{\xint_UDsignfork
575         ##1{\expandafter #3\romannumeral-'0#5}%
576         -{#4##1}%
577         \krof }%
578     \def #4##1##2{\ifcase ##1%
579         \xint_afterfi{\ifx\xintexpr_itself^C ##2\xint_dothis{\xint_c_ ##2}\fi
580             \xint_orthat\xintexpr_missing_} }%
581         \or \csname XINT_#6_op_##2\expandafter\endcsname
582         \else
583         \xint_afterfi{\expandafter #3\romannumeral-'0\csname XINT_#6_op_##2\endcsname }%
584         \fi
585     }%
586 }%
587 \def\xintexpr_missing_){\xintError:inserted \xint_c_ \xintexpr_done }%
588 \catcode') 12
589 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
590     \expandafter\xint_tmpa
591     \csname XINT_#1_op_(\expandafter\endcsname
592     \csname XINT_#1_oparen\expandafter\endcsname
593     \csname XINT_#1_until_)_a\expandafter\endcsname
594     \csname XINT_#1_until_)_b\expandafter\endcsname
595     \csname XINT_#1_op_-vi\endcsname
596     {#1}%
597 }%
598 \expandafter\let\csname XINTexpr_precedence_)\endcsname\xint_c_i
599 \expandafter\let\csname XINTexpr_precedence_]\endcsname\xint_c_i
600 \expandafter\let\csname XINTexpr_precedence_;\endcsname\xint_c_i
601 \let\xintexpr_precedence_a \xint_c_xviii
602 \expandafter\let\csname XINTexpr_precedence^C\endcsname \xint_c_
603 \expandafter\let\csname XINTexpr_precedence_++)\endcsname \xint_c_i
604 \catcode'. 11 \catcode'=' 11 \catcode'+ 11
605 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
606     \expandafter\let\csname XINT_#1_op_)\endcsname \XINTexpr_getop
607     \expandafter\let\csname XINT_#1_op_;\endcsname \space
608     \expandafter\def\csname XINT_#1_op_]\endcsname ##1{\XINTexpr_getop ##1a}%
609     \expandafter\let\csname XINT_#1_op_a\endcsname \XINTexpr_getop

```

```

610 \expandafter\def\csname XINT_#1_op_++\endcsname ##1##2\relax
611 {\expandafter\XINT_expr_founded\expandafter
612   {\expandafter\.=+\xintiCeil{\XINT_expr_unlock ##1}}}%
613 }%
614 \catcode'. 12 \catcode'= 12 \catcode'+ 12

```

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

```

615 \xintFor* #1 in {{=}{<=}{>=}{!=}{&&}{||}{**}{//}{/}{:}{..}{.}{.}{.}{.}}%
616   {+}{-}{*}{/}{^}{a+}{a-}{a*}{a/}{a**}{a^}%
617   {}{[]}{[:]{:}{^C}{++}{++}}%
618 \do {\expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}}%
619 \subsubsection{unexpanded{unexpanded{The \detokenize{|, &, xor, <, >, =,
620 <=, >=, !=, //, /:, .., ..[, and ].. operators}}}}
621 % \begin{macrocode}
622 \def\XINT_tmpc #1#2#3#4#5#6#7#8%
623 {%
624 \def #1##1% \XINT_expr_op_<op> ou flexpr ou iiexpr
625 {% keep value, get next number and operator, then do until
626 \expandafter #2\expandafter ##1%
627 \romannumeral-'0\expandafter\XINT_expr_getnext }%
628 \def #2##1##2% \XINT_expr_until_<op>_a ou flexpr ou iiexpr
629 {\xint_UDsignfork ##2{\expandafter #2\expandafter ##1\romannumeral-'0#4}%
630 -{#3##1##2}%
631 \krof }%
632 \def #3##1##2##3##4% \XINT_expr_until_<op>_b ou flexpr ou iiexpr
633 {% either execute next operation now, or first do next (possibly unary)
634 \ifnum ##2>#5%
635 \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral-'0%
636 \csname XINT_#8_op_##3\endcsname {##4}}%
637 \else \xint_afterfi {\expandafter ##2\expandafter ##3%
638 \csname .=#6{\XINT_expr_unlock ##1}{\XINT_expr_unlock ##4}\endcsname }%
639 \fi }%
640 \let #7#5%
641 }%
642 \def\XINT_tmpb #1#2#3#4#5#6%
643 {%
644 \expandafter\XINT_tmpc
645 \csname XINT_#1_op_#3\endcsname
646 \csname XINT_#1_until_#3_a\endcsname
647 \csname XINT_#1_until_#3_b\endcsname
648 \csname XINT_#1_op_#5\endcsname
649 \csname xint_c_#4\endcsname
650 \csname #2#6\endcsname
651 \csname XINT_expr_precedence_#3\endcsname {#1}%
652 }%
653 \xintFor #1 in {expr, flexpr, iiexpr} \do {%
654 \def\XINT_tmpa ##1{\XINT_tmpb {#1}{xint}##1}%
655 \xintApplyInline {\XINT_tmpa }{%
656   {{iii}{vi}{OR}}%
657   {{iv}{vi}{AND}}%
658   {{xor}{iii}{vi}{XOR}}%

```

```

659   {<{v}{vi}{Lt}}}%
660   {>{v}{vi}{Gt}}}%
661   {={v}{vi}{Eq}}}%
662   {{<={v}{vi}{LtorEq}}}%
663   {{>={v}{vi}{GtorEq}}}%
664   {{!=}{v}{vi}{Neq}}}%
665   {{..}{iii}{vi}{Seq::csv}}% will get redefined to use \xintiiSeq::csv in xintiiexpr
666   {{//}{vii}{vii}{DivTrunc}}% will get redefined for xintiiexpr
667   {{/:}{vii}{vii}{Mod}}}%
668   }%
669 }%
670 \def\XINT_tmpa #1{\XINT_tmpb {expr}{xint}#1}%
671 \xintApplyInline {\XINT_tmpa }{%
672   {+{vi}{vi}{Add}}}%
673   {-{vi}{vi}{Sub}}}%
674   {*{vii}{vii}{Mul}}}%
675   {/{vii}{vii}{Div}}}%
676   {^{viii}{viii}{Pow}}}%
677   {{..}{iii}{vi}{SeqA::csv}}%
678   {{}.{iii}{vi}{SeqB::csv}}}%
679 }%
680 \def\XINT_tmpa #1{\XINT_tmpb {flexpr}{XINTinFloat}#1}%
681 \xintApplyInline {\XINT_tmpa }{%
682   {+{vi}{vi}{Add}}}%
683   {-{vi}{vi}{Sub}}}%
684   {*{vii}{vii}{Mul}}}%
685   {/{vii}{vii}{Div}}}%
686   {^{viii}{viii}{Power}}}%
687   {{..}{iii}{vi}{SeqA::csv}}%
688   {{}.{iii}{vi}{SeqB::csv}}}%
689 }%
690 \def\XINT_tmpa #1{\XINT_tmpb {iiexpr}{xint}#1}%
691 \xintApplyInline {\XINT_tmpa }{%
692   {+{vi}{vi}{iiAdd}}}%
693   {-{vi}{vi}{iiSub}}}%
694   {*{vii}{vii}{iiMul}}}%
695   {/{vii}{vii}{iiDivRound}}% CHANGED IN 1.1! PREVIOUSLY DID EUCLIDEAN QUOTIENT
696   {^{viii}{viii}{iiPow}}}%
697   {{..}{iii}{vi}{iiSeqA::csv}}%
698   {{}.{iii}{vi}{iiSeqB::csv}}%
699   {{..}{iii}{vi}{iiSeq::csv}}%
700   {{//}{vii}{vii}{iiDivTrunc}}%
701   {{/:}{vii}{vii}{iiMod}}}%
702 }%

```

10.21.1 The], -,], *, /,], ^, +[, -[, *, /[, and ^[list operators

\XINT_expr_binop_inline_b

```

703 \def\XINT_expr_binop_inline_a
704   {\expandafter\xint_gobble_i\romannumeral-'0\XINT_expr_binop_inline_b }%
705 \def\XINT_expr_binop_inline_b #1#2,{\XINT_expr_binop_inline_c #2,{#1}}%
706 \def\XINT_expr_binop_inline_c #1{%
707   \if ,#1\xint_dothis\XINT_expr_binop_inline_e\fi
708   \if ^#1\xint_dothis\XINT_expr_binop_inline_end\fi

```

```

709 \xint_orthat\XINT_expr_binop_inline_d #1}%
710 \def\XINT_expr_binop_inline_d #1,#2{,#2{#1}\XINT_expr_binop_inline_b {#2}}%
711 \def\XINT_expr_binop_inline_e #1,#2{,\XINT_expr_binop_inline_b {#2}}%
712 \def\XINT_expr_binop_inline_end #1,#2{}%
713 \def\XINT_tmpc #1#2#3#4#5#6#7#8%
714 {%
715 \def #1##1% \XINT_expr_op_<op> ou flexpr ou iiexpr
716 {% keep value, get next number and operator, then do until
717 \expandafter #2\expandafter ##1%
718 \romannumeral-'0\expandafter\XINT_expr_getnext }%
719 \def #2##1##2% \XINT_expr_until_<op>_a ou flexpr ou iiexpr
720 {\xint_UDsignfork ##2{\expandafter #2\expandafter ##1\romannumeral-'0#4}%
721 -{#3##1##2}}%
722 \krof }%
723 \def #3##1##2##3##4% \XINT_expr_until_<op>_b ou flexpr ou iiexpr
724 {% either execute next operation now, or first do next (possibly unary)
725 \ifnum ##2>#5%
726 \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral-'0%
727 \csname XINT_#8_op_##3\endcsname {##4}}%
728 \else \xint_afterfi {\expandafter ##2\expandafter ##3%
729 \csname .=\expandafter\XINT_expr_binop_inline_a\expandafter
730 {\expandafter\expandafter\expandafter#6\expandafter
731 \xint_exchangetwo_keepbraces\expandafter
732 {\expandafter\XINT_expr_unlock\expandafter ##4\expandafter}\expandafter}%
733 \romannumeral-'0\XINT_expr_unlock ##1,^,\endcsname }%
734 \fi }%
735 \let #7#5%
736 }%
737 \def\XINT_tmpb #1#2#3#4%
738 {%
739 \expandafter\XINT_tmpc
740 \csname XINT_#1_op_#2\expandafter\endcsname
741 \csname XINT_#1_until_#2_a\expandafter\endcsname
742 \csname XINT_#1_until_#2_b\expandafter\endcsname
743 \csname XINT_#1_op_-#3\expandafter\endcsname
744 \csname xint_c_#3\expandafter\endcsname
745 \csname #4\expandafter\endcsname
746 \csname XINT_expr_precedence_#2\endcsname {#1}%
747 }%
748 \xintApplyInline {\expandafter\XINT_tmpb \xint_firstofone}{%
749 {{expr}{a+}{vi}{xintAdd}}}%
750 {{expr}{a-}{vi}{xintSub}}}%
751 {{expr}{a*}{vii}{xintMul}}}%
752 {{expr}{a/}{vii}{xintDiv}}}%
753 {{expr}{a^}{viii}{xintPow}}}%
754 {{iiexpr}{a+}{vi}{xintiiAdd}}}%
755 {{iiexpr}{a-}{vi}{xintiiSub}}}%
756 {{iiexpr}{a*}{vii}{xintiiMul}}}%
757 {{iiexpr}{a/}{vii}{xintiiDivRound}}}%
758 {{iiexpr}{a^}{viii}{xintiiPow}}}%
759 {{flexpr}{a+}{vi}{XINTinFloatAdd}}}%
760 {{flexpr}{a-}{vi}{XINTinFloatSub}}}%

```

```

761  {{flexpr}{a*}{vii}{XINTinFloatMul}}}%
762  {{flexpr}{a/}{vii}{XINTinFloatDiv}}}%
763  {{flexpr}{a^}{viii}{XINTinFloatPower}}}%
764 }%
765 \def\XINT_tmpc #1#2#3#4#5#6#7%
766 {%
767   \def #1##1{\expandafter#2\expandafter##1\romannumeral-'0%
768     \expandafter #3\romannumeral-'0\XINT_expr_getnext }%
769   \def #2##1##2##3##4%
770   {% either execute next operation now, or first do next (possibly unary)
771     \ifnum ##2>#4%
772     \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral-'0%
773       \csname XINT_#7_op_##3\endcsname {##4}}}%
774     \else \xint_afterfi {\expandafter ##2\expandafter ##3%
775       \csname .=\expandafter\XINT_expr_binop_inline_a\expandafter
776         {\expandafter#5\expandafter
777           {\expandafter\XINT_expr_unlock\expandafter ##1\expandafter}\expandafter}%
778         \romannumeral-'0\XINT_expr_unlock ##4,^,\endcsname }%
779     \fi }%
780   \let #6#4%
781 }%
782 \def\XINT_tmpb #1#2#3#4%
783 {%
784   \expandafter\XINT_tmpc
785   \csname XINT_#1_op_#2\expandafter\endcsname
786   \csname XINT_#1_until_#2\expandafter\endcsname
787   \csname XINT_#1_until_)_a\expandafter\endcsname
788   \csname xint_c_#3\expandafter\endcsname
789   \csname #4\expandafter\endcsname
790   \csname XINT_expr_precedence_#2\endcsname {#1}%
791 }%
792 \xintApplyInline {\expandafter\XINT_tmpb\xint_firstofone }{%
793   {{expr}{+}{vi}{xintAdd}}}%
794   {{expr}{-}{vi}{xintSub}}}%
795   {{expr}{*}{vii}{xintMul}}}%
796   {{expr}{/}{vii}{xintDiv}}}%
797   {{expr}{^}{viii}{xintPow}}}%
798   {{iiexpr}{+}{vi}{xintiiAdd}}}%
799   {{iiexpr}{-}{vi}{xintiiSub}}}%
800   {{iiexpr}{*}{vii}{xintiiMul}}}%
801   {{iiexpr}{/}{vii}{xintiiDivRound}}}%
802   {{iiexpr}{^}{viii}{xintiiPow}}}%
803   {{flexpr}{+}{vi}{XINTinFloatAdd}}}%
804   {{flexpr}{-}{vi}{XINTinFloatSub}}}%
805   {{flexpr}{*}{vii}{XINTinFloatMul}}}%
806   {{flexpr}{/}{vii}{XINTinFloatDiv}}}%
807   {{flexpr}{^}{viii}{XINTinFloatPower}}}%
808 }%

```

10.21.2 The 'and', 'or', 'xor', and 'mod' as infix operator words

```

809 \xintFor #1 in {and,or,xor,mod} \do {%
810   \expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}}%

```

```

811 \expandafter\let\csname XINT_expr_precedence_and\expandafter\endcsname
812       \csname XINT_expr_precedence_&\endcsname
813 \expandafter\let\csname XINT_expr_precedence_or\expandafter\endcsname
814       \csname XINT_expr_precedence_|\endcsname
815 \expandafter\let\csname XINT_expr_precedence_mod\expandafter\endcsname
816       \csname XINT_expr_precedence_/\endcsname
817 \xintFor #1 in {expr, flexpr, iiexpr} \do {%
818   \expandafter\let\csname XINT_#1_op_and\expandafter\endcsname
819       \csname XINT_#1_op_&\endcsname
820   \expandafter\let\csname XINT_#1_op_or\expandafter\endcsname
821       \csname XINT_#1_op_|\endcsname
822   \expandafter\let\csname XINT_#1_op_mod\expandafter\endcsname
823       \csname XINT_#1_op_/\endcsname
824 }%
825 % \subsubsection{unexpanded{unexpanded{The \detokenize{||, &&, **, **[, ]**}
826 % operators as synonyms}}}
827 \expandafter\let\csname XINT_expr_precedence_==\expandafter\endcsname
828       \csname XINT_expr_precedence_=\endcsname
829 \expandafter\let\csname XINT_expr_precedence_&&\expandafter\endcsname
830       \csname XINT_expr_precedence_&\endcsname
831 \expandafter\let\csname XINT_expr_precedence_||\expandafter\endcsname
832       \csname XINT_expr_precedence_|\endcsname
833 \expandafter\let\csname XINT_expr_precedence_**\expandafter\endcsname
834       \csname XINT_expr_precedence_^\endcsname
835 \expandafter\let\csname XINT_expr_precedence_a**\expandafter\endcsname
836       \csname XINT_expr_precedence_a^\endcsname
837 \expandafter\let\csname XINT_expr_precedence_**[\expandafter\endcsname
838       \csname XINT_expr_precedence_^[ \endcsname
839 \xintFor #1 in {expr, flexpr, iiexpr} \do {%
840   \expandafter\let\csname XINT_#1_op_==\expandafter\endcsname
841       \csname XINT_#1_op_=\endcsname
842   \expandafter\let\csname XINT_#1_op_&&\expandafter\endcsname
843       \csname XINT_#1_op_&\endcsname
844   \expandafter\let\csname XINT_#1_op_||\expandafter\endcsname
845       \csname XINT_#1_op_|\endcsname
846   \expandafter\let\csname XINT_#1_op_**\expandafter\endcsname
847       \csname XINT_#1_op_^\endcsname
848   \expandafter\let\csname XINT_#1_op_a**\expandafter\endcsname
849       \csname XINT_#1_op_a^\endcsname
850   \expandafter\let\csname XINT_#1_op_**[\expandafter\endcsname
851       \csname XINT_#1_op_^[ \endcsname
852 }%

```

10.21.3 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.

```

853 \def\XINT_tmpa #1#2#3#4#5#6%

```

```

854 {%
855   \def #1##1% \XINT_expr_op_][
856   {%
857     \expandafter #2\expandafter ##1\romannumeral-'0\XINT_expr_getnext
858   }%
859   \def #2##1##2% \XINT_expr_until_][_a
860   {\xint_UDsignfork
861     ##2{\expandafter #2\expandafter ##1\romannumeral-'0#4}%
862     -{##3##1##2}%
863   \krof }%
864   \def #3##1##2##3##4% \XINT_expr_until_][_b
865   {%
866     \ifnum ##2>\xint_c_ii
867       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral-'0%
868         \csname XINT_#6_op_##3\endcsname {##4}}%
869     \else
870       \xint_afterfi
871       {\expandafter ##2\expandafter ##3\csname
872         .=\expandafter\xintListSel:csv \romannumeral-'0\XINT_expr_unlock ##4;%
873         \XINT_expr_unlock ##1;\endcsname % unlock for \xintNewExpr
874       }%
875     \fi
876   }%
877   \let #5\xint_c_ii
878 }%
879 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
880 \expandafter\XINT_tmpa
881   \csname XINT_#1_op_][\expandafter\endcsname
882   \csname XINT_#1_until_][_a\expandafter\endcsname
883   \csname XINT_#1_until_][_b\expandafter\endcsname
884   \csname XINT_#1_op_-vi\expandafter\endcsname
885   \csname XINT_expr_precedence_][\endcsname {#1}%
886 }%
887 \def\XINT_tmpa #1#2#3#4#5#6%
888 {%
889   \def #1##1% \XINT_expr_op_:
890   {%
891     \expandafter #2\expandafter ##1\romannumeral-'0\XINT_expr_getnext
892   }%
893   \def #2##1##2% \XINT_expr_until_:_a
894   {\xint_UDsignfork
895     ##2{\expandafter #2\expandafter ##1\romannumeral-'0#4}%
896     -{##3##1##2}%
897   \krof }%
898   \def #3##1##2##3##4% \XINT_expr_until_:_b
899   {%
900     \ifnum ##2>\xint_c_iii
901       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral-'0%
902         \csname XINT_#6_op_##3\endcsname {##4}}%
903     \else
904       \xint_afterfi
905       {\expandafter ##2\expandafter ##3\csname

```



```

906      .:=\xintiiifSgn{\XINT_expr_unlock ##1}NPP.% : and dots for expansion
907      \xintiiifSgn{\XINT_expr_unlock ##4}NPP.% in \xintNewExpr context
908 % reason for \xintNum is a/1[x] format, but -0.5 will not work, seen <0, but 0 after
909      \xintNum{\XINT_expr_unlock ##1};\xintNum{\XINT_expr_unlock ##4}\endcsname
910      }%
911      \fi
912      }%
913      \let #5\xint_c_iii
914 }%
915 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
916 \expandafter\XINT_tmpa
917   \csname XINT_#1_op_:\expandafter\endcsname
918   \csname XINT_#1_until:_a\expandafter\endcsname
919   \csname XINT_#1_until:_b\expandafter\endcsname
920   \csname XINT_#1_op_-vi\expandafter\endcsname
921   \csname XINT_expr_precedence_:\endcsname {#1}%
922 }%
923 \catcode'[ 11 \catcode' 11
924 \let\XINT_expr_precedence_:\xint_c_iii
925 \def\XINT_expr_op_:#1{\expandafter\xint_c_i\expandafter }%
926   \csname .:=\xintiiifSgn{\XINT_expr_unlock #1}npp\XINT_expr_unlock #1\endcsname }%
927 \let\XINT_flexpr_op_:\XINT_expr_op_:]
928 \let\XINT_iiexpr_op_:\XINT_expr_op_:]
929 \let\XINT_expr_precedence_[: \xint_c_iii
930 \edef\XINT_expr_op_[: #1{\xint_c_ii \expandafter\noexpand
931   \csname XINT_expr_itself_]\endcsname #10\string :}%
932 % : must be catcode 12, else will be mistaken for start of variable by expression parser
933 \let\XINT_flexpr_op_[: \XINT_expr_op_[:
934 \let\XINT_iiexpr_op_[: \XINT_expr_op_[:
935 \catcode'[ 12 \catcode' 12
936 \def\xintListSel:csv #1{% these complications are due to \xintNewExpr matters
937   \if ]\noexpand#1\xint_dothis{\expandafter\XINT_listsel:_s\romannumeral-'0}\fi
938   \if : \noexpand#1\xint_dothis{\XINT_listsel:_:}\fi
939   \xint_orthat {\XINT_listsel:_nth #1}%
940 }%
941 \def\XINT_listsel:_s #1{\if p#1\expandafter\XINT_listsel:_trim\else
942   \expandafter\XINT_listsel:_keep\fi }%
943 \def\XINT_listsel:_: #1.#2.{\csname XINT_listsel:_#1#2\endcsname }%
944 \def\XINT_listsel:_trim #1;#2;%
945   {\xintListWithSep,{\xintTrim {\xintNum{#1}}{\xintCSVtoListNonStripped{#2}}}}%
946 \def\XINT_listsel:_keep #1;#2;%
947   {\xintListWithSep,{\xintKeep {\xintNum{#1}}{\xintCSVtoListNonStripped{#2}}}}%
948 \def\XINT_listsel:_nth#1;#2;%
949   {\xintNthElt {\xintNum{#1}}{\xintCSVtoListNonStripped{#2}}}%
950 \def\XINT_listsel:_PP #1;#2;#3;%
951   {\xintListWithSep,%
952     {\xintTrim {\xintNum{#1}}%
953       {\xintKeep {\xintNum{#2}}%
954         {\xintCSVtoListNonStripped{#3}}}%
955     }%
956   }%
957   }%

```

```

958 \def\XINT_listsel:_NN #1;#2;#3;%
959   {\xintListWithSep,%
960     {\xintTrim {\xintNum{#2}}%
961       {\xintKeep {\xintNum{#1}}%
962         {\xintCSVtoListNonStripped{#3}}%
963       }%
964     }%
965   }%
966 \def\XINT_listsel:_NP #1;#2;#3;%
967   {\expandafter\XINT_listsel:_NP_a \the\numexpr #1+%
968     \xintNthElt{0}{\xintCSVtoListNonStripped{#3}};#2;#3;%}
969 \def\XINT_listsel:_NP_a #1#2;{\if -#1\expandafter\XINT_listsel:_OP\fi
970   \XINT_listsel:_PP #1#2;%}
971 \def\XINT_listsel:_OP\XINT_listsel:_PP #1;{\XINT_listsel:_PP 0;%}
972 \def\XINT_listsel:_PN #1;#2;#3;%
973   {\expandafter\XINT_listsel:_PN_a \the\numexpr #2+%
974     \xintNthElt{0}{\xintCSVtoListNonStripped{#3}};#1;#3;%}
975 \def\XINT_listsel:_PN_a #1#2;#3;{\if -#1\expandafter\XINT_listsel:_PO\fi
976   \XINT_listsel:_PP #3;#1#2;%}
977 \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.

```

978 \def\xintSeq::csv {\romannumeral0\xintseq::csv}%
979 \def\xintseq::csv #1#2%
980 {%
981   \expandafter\XINT_seq::csv\expandafter
982     {\the\numexpr \xintiCeil{#1}\expandafter}\expandafter
983     {\the\numexpr \xintiFloor{#2}}%
984 }%
985 \def\XINT_seq::csv #1#2%
986 {%
987   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
988     \expandafter\XINT_seq::csv_z
989   \or
990     \expandafter\XINT_seq::csv_p
991   \else
992     \expandafter\XINT_seq::csv_n
993   \fi
994   {#2}{#1}%
995 }%
996 \def\XINT_seq::csv_z #1#2{ #1/1[0]}%
997 \def\XINT_seq::csv_p #1#2%
998 {%

```

```

999   \ifnum #1>#2
1000   \expandafter\expandafter\expandafter\XINT_seq::csv_p
1001   \else
1002   \expandafter\XINT_seq::csv_e
1003   \fi
1004   \expandafter{\the\numexpr #1-\xint_c_i}{#2},#1/1[0]%
1005 }%
1006 \def\XINT_seq::csv_n #1#2%
1007 {%
1008   \ifnum #1<#2
1009   \expandafter\expandafter\expandafter\XINT_seq::csv_n
1010   \else
1011   \expandafter\XINT_seq::csv_e
1012   \fi
1013   \expandafter{\the\numexpr #1+\xint_c_i}{#2},#1/1[0]%
1014 }%
1015 \def\XINT_seq::csv_e #1,{ }%

```

10.22.2 \xintiiseq::csv

```

1016 \def\xintiiseq::csv {\romannumeral0\xintiiseq::csv }%
1017 \def\xintiiseq::csv #1#2%
1018 {%
1019   \expandafter\XINT_iiseq::csv\expandafter
1020   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
1021 }%
1022 \def\XINT_iiseq::csv #1#2%
1023 {%
1024   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
1025   \expandafter\XINT_iiseq::csv_z
1026   \or
1027   \expandafter\XINT_iiseq::csv_p
1028   \else
1029   \expandafter\XINT_iiseq::csv_n
1030   \fi
1031   {#2}{#1}%
1032 }%
1033 \def\XINT_iiseq::csv_z #1#2{ #1}%
1034 \def\XINT_iiseq::csv_p #1#2%
1035 {%
1036   \ifnum #1>#2
1037   \expandafter\expandafter\expandafter\XINT_iiseq::csv_p
1038   \else
1039   \expandafter\XINT_seq::csv_e
1040   \fi
1041   \expandafter{\the\numexpr #1-\xint_c_i}{#2},#1%
1042 }%
1043 \def\XINT_iiseq::csv_n #1#2%
1044 {%
1045   \ifnum #1<#2
1046   \expandafter\expandafter\expandafter\XINT_iiseq::csv_n
1047   \else
1048   \expandafter\XINT_seq::csv_e

```

```

1049 \fi
1050 \expandafter{\the\numexpr #1+\xint_c_i}{#2},#1%
1051 }%
1052 \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*

```

1053 \def\xintSeqA::csv #1%
1054 {\expandafter\xINT_seq::csv\expandafter{\romannumeral0\xintra {#1}}}%
1055 \def\xINT_seq::csv #1#2{\expandafter\xINT_seq::csv_a \romannumeral0\xintra {#2};#1;}%
1056 \def\xintiiSeqA::csv #1#2{\XINT_iiseq::csv #1#2}%
1057 \def\xINT_iiseq::csv #1#2#3#4{\expandafter\xINT_seq::csv_a
1058 \romannumeral-‘0\expandafter \XINT_expr_unlock\expandafter#4%
1059 \expandafter;\romannumeral-‘0\XINT_expr_unlock #2;}%
1060 \def\xINTinFloatSeqA::csv #1{\expandafter\xINT_flseq::csv\expandafter
1061 {\romannumeral0\XINTinfloat [\XINTdigits]{#1}}}%
1062 \def\xINT_flseq::csv #1#2%
1063 {\expandafter\xINT_seq::csv_a\romannumeral0\XINTinfloat [\XINTdigits]{#2};#1;}%
1064 \def\xINT_seq::csv_a #1{\xint_UDzerominusfork
1065 #1-{z}%
1066 0#1{n}%
1067 0-{p}%
1068 \krof #1}%

```

10.23.2 *\xintSeqB::csv*

```

1069 \def\xintSeqB::csv #1#2%
1070 {\expandafter\xINT_seqb::csv \expandafter{\romannumeral0\xintra{#2}}{#1}}%
1071 \def\xINT_seqb::csv #1#2{\expandafter\xINT_seqb::csv_a\romannumeral-‘0#2#1!}%
1072 \def\xINT_seqb::csv_a #1#2;#3;#4!\expandafter\xINT_expr_seq_empty?
1073 \romannumeral0\csname XINT_seqb::csv_#1\endcsname {#3}{#4}{#2}}%
1074 \def\xINT_seqb::csv_p #1#2#3%
1075 {%
1076 \xintifCmp {#1}{#2}{, #1\expandafter\xINT_seqb::csv_p\expandafter}%
1077 {, #1\xint_gobble_iii}{\xint_gobble_iii}%
1078 % \romannumeral0 stopped by \endcsname, XINT_expr_seq_empty? constructs "nil".
1079 {\romannumeral0\xintadd {#3}{#1}}{#2}{#3}%
1080 }%
1081 \def\xINT_seqb::csv_n #1#2#3%
1082 {%
1083 \xintifCmp {#1}{#2}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1084 {, #1\expandafter\xINT_seqb::csv_n\expandafter}%
1085 {\romannumeral0\xintadd {#3}{#1}}{#2}{#3}%
1086 }%
1087 \def\xINT_seqb::csv_z #1#2#3{, #1}%

```

10.23.3 *\xintiiSeqB::csv*

```

1088 \def\xintiiSeqB::csv #1#2{\XINT_iiseqb::csv #1#2}%
1089 \def\xINT_iiseqb::csv #1#2#3#4%

```

```

1090   {\expandafter\XINT_iiseqb::csv_a
1091     \romannumeral-‘0\expandafter \XINT_expr_unlock\expandafter#2%
1092     \romannumeral-‘0\XINT_expr_unlock #4!}%
1093 \def\XINT_iiseqb::csv_a #1#2;#3;#4!{\expandafter\XINT_expr_seq_empty?
1094   \romannumeral-‘0\csname XINT_iiseqb::csv_#1\endcsname {#3}{#4}{#2}}%
1095 \def\XINT_iiseqb::csv_p #1#2#3%
1096 {%
1097   \xintSgnFork{\XINT_Cmp {#1}{#2}}{, #1\expandafter\XINT_iiseqb::csv_p\expandafter}%
1098   {, #1\xint_gobble_iii}{\xint_gobble_iii}%
1099   {\romannumeral0\xintiadd {#3}{#1}}{#2}{#3}%
1100 }%
1101 \def\XINT_iiseqb::csv_n #1#2#3%
1102 {%
1103   \xintSgnFork{\XINT_Cmp {#1}{#2}}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1104   {, #1\expandafter\XINT_iiseqb::csv_n\expandafter}%
1105   {\romannumeral0\xintiadd {#3}{#1}}{#2}{#3}%
1106 }%
1107 \def\XINT_iiseqb::csv_z #1#2#3{, #1}%

```

10.23.4 \XINTinFloatSeqB::csv

```

1108 \def\XINTinFloatSeqB::csv #1#2{\expandafter\XINT_flseqb::csv \expandafter
1109   {\romannumeral0\XINTinfloat [\XINTdigits]{#2}}{#1}}%
1110 \def\XINT_flseqb::csv #1#2{\expandafter\XINT_flseqb::csv_a\romannumeral-‘0#2#1!}%
1111 \def\XINT_flseqb::csv_a #1#2;#3;#4!{\expandafter\XINT_expr_seq_empty?
1112   \romannumeral-‘0\csname XINT_flseqb::csv_#1\endcsname {#3}{#4}{#2}}%
1113 \def\XINT_flseqb::csv_p #1#2#3%
1114 {%
1115   \xintifCmp {#1}{#2}{, #1\expandafter\XINT_flseqb::csv_p\expandafter}%
1116   {, #1\xint_gobble_iii}{\xint_gobble_iii}%
1117   {\romannumeral0\XINTinfloatadd {#3}{#1}}{#2}{#3}%
1118 }%
1119 \def\XINT_flseqb::csv_n #1#2#3%
1120 {%
1121   \xintifCmp {#1}{#2}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1122   {, #1\expandafter\XINT_flseqb::csv_n\expandafter}%
1123   {\romannumeral0\XINTinfloatadd {#3}{#1}}{#2}{#3}%
1124 }%
1125 \def\XINT_flseqb::csv_z #1#2#3{, #1}%

```

10.24 The comma as binary operator

New with 1.09a.

```

1126 \def\XINT_tmpa #1#2#3#4#5#6%
1127 {%
1128   \def #1##1% \XINT_expr_op_,
1129   {%
1130     \expandafter #2\expandafter ##1\romannumeral-‘0\XINT_expr_getnext
1131   }%
1132   \def #2##1##2% \XINT_expr_until_,_a
1133   {\xint_UDsignfork
1134     ##2{\expandafter #2\expandafter ##1\romannumeral-‘0#4}%
1135     -{#3##1##2}%
1136   \krof }%

```

```

1137 \def #3##1##2##3##4% \XINT_expr_until_,_b
1138 {%
1139   \ifnum ##2>\xint_c_ii
1140     \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral-'0%
1141                   \csname XINT_#6_op_##3\endcsname {##4}}%
1142   \else
1143     \xint_afterfi
1144     {\expandafter ##2\expandafter ##3%
1145     \csname .=\XINT_expr_unlock ##1,\XINT_expr_unlock ##4\endcsname }%
1146   \fi
1147 }%
1148 \let #5\xint_c_ii
1149 }%
1150 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1151 \expandafter\XINT_tmpa
1152   \csname XINT_#1_op_,\expandafter\endcsname
1153   \csname XINT_#1_until_,_a\expandafter\endcsname
1154   \csname XINT_#1_until_,_b\expandafter\endcsname
1155   \csname XINT_#1_op_-vi\expandafter\endcsname
1156   \csname XINT_expr_precedence_,\endcsname {#1}%
1157 }%

```

10.25 The minus as prefix operator of variable precedence level

```

1158 \def\XINT_tmpa #1#2#3%
1159 {%
1160   \expandafter\XINT_tmppb
1161   \csname XINT_#1_op_-#3\expandafter\endcsname
1162   \csname XINT_#1_until_-#3_a\expandafter\endcsname
1163   \csname XINT_#1_until_-#3_b\expandafter\endcsname
1164   \csname xint_c_#3\endcsname {#1}#2%
1165 }%
1166 \def\XINT_tmppb #1#2#3#4#5#6%
1167 {%
1168   \def #1% \XINT_expr_op_-<level>
1169   {% get next number+operator then switch to _until macro
1170     \expandafter #2\romannumeral-'0\XINT_expr_getnext
1171   }%
1172   \def #2##1% \XINT_expr_until_-<l>_a
1173   {\xint_UDsignfork
1174     ##1{\expandafter #2\romannumeral-'0#1}%
1175     -{##3##1}%
1176   \krof }%
1177   \def #3##1##2##3% \XINT_expr_until_-<l>_b
1178   {% _until tests precedence level with next op, executes now or postpones
1179     \ifnum ##1>#4%
1180       \xint_afterfi {\expandafter #2\romannumeral-'0%
1181                     \csname XINT_#5_op_##2\endcsname {##3}}%
1182     \else
1183       \xint_afterfi {\expandafter ##1\expandafter ##2%
1184                     \csname .=#6{\XINT_expr_unlock ##3}\endcsname }%
1185     \fi
1186   }%

```

```

1187 }%
1188 \xintApplyInline{\XINT_tmpa {expr}\xintOpp}{\vi}{vii}{viii}{ix}}%
1189 \xintApplyInline{\XINT_tmpa {fexpr}\xintOpp}{\vi}{vii}{viii}{ix}}%
1190 \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}

```

1191 \let\XINT_expr_precedence_? \xint_c_x
1192 \def\XINT_expr_op_? #1#2{\if ?#2\expandafter \XINT_expr_op_??\fi
1193      \XINT_expr_op_?a #1{#2}}%
1194 \def\XINT_expr_op_?a #1#2#3%
1195 {%
1196   \xintiiifNotZero{\XINT_expr_unlock #1}{\XINT_expr_getnext #2}{\XINT_expr_getnext #3}%
1197 }%
1198 \let\XINT_flexpr_op_?\XINT_expr_op_?
1199 \let\XINT_iiexpr_op_?\XINT_expr_op_?
1200 \def\XINT_expr_op_?? #1#2#3#4#5#6%
1201 {%
1202   \xintiiifSgn {\XINT_expr_unlock #2}{\XINT_expr_getnext #4}{\XINT_expr_getnext #5}%
1203   {\XINT_expr_getnext #6}%
1204 }%

```

10.27 ! as postfix factorial operator

As of 2014/10/28, not yet a float version of factorial. I must do it!

```

1205 \let\XINT_expr_precedence_! \xint_c_x
1206 \def\XINT_expr_op_! #1{\expandafter\XINT_expr_getop
1207      \csname .=\xintFac{\XINT_expr_unlock #1}\endcsname }%
1208 \let\XINT_flexpr_op_!\XINT_expr_op_!
1209 \def\XINT_iiexpr_op_! #1{\expandafter\XINT_expr_getop
1210      \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.

```

1211 \catcode'[ 11
1212 \catcode'* 11
1213 \let\XINT_expr_precedence_[ \xint_c_vii
1214 \def\XINT_expr_op_[ #1#2]{\expandafter\XINT_expr_getop
1215      \csname .=\xintE{\XINT_expr_unlock #1}%
1216      {\xint_zapspace #2 \xint_bye\xint_bye}\endcsname}%
1217 \def\XINT_iiexpr_op_[ #1#2]{\expandafter\XINT_expr_getop
1218      \csname .=\xintiiE{\XINT_expr_unlock #1}%
1219      {\xint_zapspace #2 \xint_bye\xint_bye}\endcsname}%

```

```

1220 \def\XINT_flexpr_op_[ #1#2]{\expandafter\XINT_expr_getop
1221         \csname .=\XINTinFloatE{\XINT_expr_unlock #1}%
1222         {\xint_zapspaces #2 \xint_bye\xint_bye}\endcsname}%
1223 \catcode'[ 12
1224 \catcode'* 12

```

10.29 For variables

```

1225 \def\XINT_expr_op__ #1% op__ with two _'s
1226     {%
1227         \ifcsname XINT_expr_var_#1\endcsname
1228         \expandafter\xint_firstoftwo
1229         \else
1230         \expandafter\xint_secondoftwo
1231         \fi
1232         {\expandafter\expandafter\expandafter\expandafter
1233         \expandafter\expandafter\expandafter
1234         \XINT_expr_getop\csname XINT_expr_var_#1\endcsname}%
1235         {\XINT_expr_unknown_variable {#1}%
1236         \expandafter\XINT_expr_getop\csname .:=0\endcsname}%
1237     }%
1238 \def\XINT_expr_unknown_variable #1{\xintError:removed \xint_gobble_i {#1}}%
1239 \let\XINT_flexpr_op__ \XINT_expr_op__
1240 \let\XINT_iiexpr_op__ \XINT_expr_op__

```

10.29.1 Defining variables

1.1 An active : character will be a pain and I almost 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 contain letters, digits, underscores, and must not start with a digit.

```

1241 \catcode': 12
1242 \def\xintdefvar #1:=#2;{\expandafter\odef
1243     \csname XINT_expr_var_\xint_zapspaces #1 \xint_bye\xint_bye\endcsname
1244     {\expandafter\empty\romannumeral0\xintbareeval #2\relax }}%
1245 \def\xintdefiivar #1:=#2;{\expandafter\odef
1246     \csname XINT_expr_var_\xint_zapspaces #1 \xint_bye\xint_bye\endcsname
1247     {\expandafter\empty\romannumeral0\xintbareiieval #2\relax }}%
1248 }%
1249 \def\xintdeffloatvar #1:=#2;{\expandafter\odef
1250     \csname XINT_expr_var_\xint_zapspaces #1 \xint_bye\xint_bye\endcsname
1251     {\expandafter\empty\romannumeral0\xintbarefloateval #2\relax }}%
1252 }%
1253 \catcode': 11

```

10.29.2 Letters as dummy variables; the nil list

```

1254 \def\XINT_tmpa #1%
1255 {%
1256     \expandafter\def\csname XINT_expr_var_#1\endcsname ##1\relax !#1##2%
1257     {\romannumeral0\XINT_expr_lockscan ##2!##1\relax !#1{##2}}%

```



```

1258 }%
1259 \xintApplyUnbraced \XINT_tmpa {abcdefghijklmnopqrstuvwxyz}%
1260 \xintApplyUnbraced \XINT_tmpa {ABCDEFGHIJKLMNOPQRSTUVWXYZ}%
1261 \expandafter\def\expandafter\XINT_expr_var_nil\expandafter
1262      {\expandafter\empty\csname . = \endcsname}%

```

10.29.3 The omit and abort constructs

```

1263 \catcode'. 11 \catcode'= 11
1264 \def\XINT_expr_var_omit #1\relax !{1^C!{}}{}}\relax !}% 24 juin
1265 \def\XINT_expr_var_abort #1\relax !{1^C!{}}{}}\relax !}% 25 juin
1266 \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!

```

1267 \catcode'? 3
1268 \def\XINT_expr_var_@ #1~#2{ #2#1~#2}%
1269 \expandafter\let\csname XINT_expr_var_@1\endcsname \XINT_expr_var_@
1270 \expandafter\def\csname XINT_expr_var_@2\endcsname #1~#2#3{ #3#1~#2#3}%
1271 \expandafter\def\csname XINT_expr_var_@3\endcsname #1~#2#3#4{ #4#1~#2#3#4}%
1272 \expandafter\def\csname XINT_expr_var_@4\endcsname #1~#2#3#4#5{ #5#1~#2#3#4#5}%
1273 \def\XINT_expr_func_@@ #1#2#3#4~#5?%
1274 {%
1275   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1276   {\xintNum{\XINT_expr_unlock#3}}{#5}#4~#5?%
1277 }%
1278 \def\XINT_expr_func_@@@ #1#2#3#4~#5~#6?%
1279 {%
1280   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1281   {\xintNum{\XINT_expr_unlock#3}}{#6}#4~#5~#6?%
1282 }%
1283 \def\XINT_expr_func_@@@@ #1#2#3#4~#5~#6~#7?%
1284 {%
1285   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1286   {\xintNum{\XINT_expr_unlock#3}}{#7}#4~#5~#6~#7?%
1287 }%
1288 \let\XINT_flexpr_func_@@\XINT_expr_func_@@
1289 \let\XINT_flexpr_func_@@@\XINT_expr_func_@@@
1290 \let\XINT_flexpr_func_@@@@\XINT_expr_func_@@@@
1291 \def\XINT_iiexpr_func_@@ #1#2#3#4~#5?%
1292 {%
1293   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1294   {\XINT_expr_unlock#3}{#5}#4~#5?%
1295 }%
1296 \def\XINT_iiexpr_func_@@@ #1#2#3#4~#5~#6?%
1297 {%
1298   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1299   {\XINT_expr_unlock#3}{#6}#4~#5~#6?%
1300 }%
1301 \def\XINT_iiexpr_func_@@@@ #1#2#3#4~#5~#6~#7?%
1302 {%

```

```

1303 \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1304 {\XINT_expr_unlock#3}{#7}#4~#5~#6~#7?%
1305 }%
1306 \catcode'? 11

```

10.30 For functions

```

1307 \def\XINT_tmpa #1#2#3{%
1308 \def #1##1% \XINT_expr_op_', #2=\XINT_expr_oparen
1309 {%
1310 \ifcsname XINT_expr_onlitteral_##1\endcsname
1311 \xint_dothis{\csname XINT_expr_onlitteral_##1\endcsname}\fi
1312 \ifcsname XINT_#3_func_##1\endcsname
1313 \xint_dothis{\expandafter\expandafter
1314 \csname XINT_#3_func_##1\endcsname\romannumeral-'0#2}\fi
1315 \xint_orthat{\XINT_expr_unknown_function {##1}%
1316 \expandafter\XINT_expr_func_unknown\romannumeral-'0#2}%
1317 }%
1318 }%
1319 \def\XINT_expr_unknown_function #1{\xintError:removed \xint_gobble_i {#1}}%
1320 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1321 \expandafter\XINT_tmpa
1322 \csname XINT_#1_op_'\expandafter\endcsname
1323 \csname XINT_#1_oparen\endcsname
1324 {#1}%
1325 }%
1326 \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.

```

1327 \def\XINT_expr_onlitteral_bool #1)%
1328 {\expandafter\XINT_expr_getop\csname .=\xintBool{#1}\endcsname }%
1329 \def\XINT_expr_onlitteral_togl #1)%
1330 {\expandafter\XINT_expr_getop\csname .=\xintToggle{#1}\endcsname }%
1331 \def\XINT_expr_onlitteral_protect #1)%
1332 {\expandafter\XINT_expr_getop\csname .=\detokenize{#1}\endcsname }%
1333 \def\XINT_expr_func_unknown #1#2#3{\expandafter #1\expandafter #2\csname .=\0\endcsname }%
1334 \def\XINT_expr_func_break #1#2#3%
1335 {\expandafter #1\expandafter #2\csname .=?\romannumeral-'0\XINT_expr_unlock #3\endcsname }%
1336 \let\XINT_flexpr_func_break \XINT_expr_func_break
1337 \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 \xintNew-Expr !)

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 iexpr.

10.32.1 \XINT_expr_onlitteral_seq

```
1338 \def\XINT_expr_onlitteral_seq
1339 {\expandafter\XINT_expr_onlitteral_seq_f\romannumeral-'\0\XINT_expr_onlitteral_seq_a {}}%
1340 \def\XINT_expr_onlitteral_seq_f #1#2{\xint_c_xviii '{seqx}#2)\relax #1}%
```

10.32.2 \XINT_expr_onlitteral_seq_a

```
1341 \def\XINT_expr_onlitteral_seq_a #1#2,%
1342 {% checks balancing of parentheses
1343   \ifcase\XINT_isbalanced_a \relax #1#2(\xint_bye)\xint_bye
1344     \expandafter\XINT_expr_onlitteral_seq_c
1345     \or\expandafter\XINT_expr_onlitteral_seq_b
1346     \else\expandafter\xintError:we_are_doomed
1347   \fi {#1#2},%
1348 }%
1349 \def\XINT_expr_onlitteral_seq_b #1,{\XINT_expr_onlitteral_seq_a {#1},}%
1350 \def\XINT_expr_onlitteral_seq_c #1,#2#3% #3 pour absorber le =
1351 {%
1352   \XINT_expr_onlitteral_seq_d {#2{#1}}}%
1353 }%
1354 \def\XINT_expr_onlitteral_seq_d #1#2#3)%
1355 {%
1356   \ifcase\XINT_isbalanced_a \relax #2#3(\xint_bye)\xint_bye
1357     \or\expandafter\XINT_expr_onlitteral_seq_e
1358     \else\expandafter\xintError:we_are_doomed
1359   \fi
1360   {#1}{#2#3}%
1361 }%
1362 \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.

```
1363 % use as \XINT_isbalanced_a \relax #1(\xint_bye)\xint_bye
1364 \def\XINT_isbalanced_a #1({\XINT_isbalanced_b #1)\xint_bye }%
1365 \def\XINT_isbalanced_b #1)#2%
1366   {\xint_bye #2\XINT_isbalanced_c\xint_bye\XINT_isbalanced_error }%
1367   if #2 is not \xint_bye, a ) was found, but there was no (. Hence error -> -1
1367 \def\XINT_isbalanced_error #1)\xint_bye {\m@ne}%
```

#2 was \xint_bye, was there a) in original #1?

```
1368 \def\xINT_isbalanced_c\xint_bye\xINT_isbalanced_error #1%
1369   {\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.

```
1370 \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.

```
1371 \def\xINT_isbalanced_d #1)#2%
1372   {\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.

```
1373 \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

```
1374 \def\xINT_expr_func_seqx #1#2{\XINT_allexpr_seqx \xintbareeval }%
1375 \def\xINT_flexpr_func_seqx #1#2{\XINT_allexpr_seqx \xintbarefloateval}%
1376 \def\xINT_iexpr_func_seqx #1#2{\XINT_allexpr_seqx \xintbareiieval }%
1377 \def\xINT_allexpr_seqx #1#2#3#4% #2 is the index list, fully evaluated and encapsulated
1378 {% #3 is the Latin letter serving as dummy variable, #4 is the expression to evaluate
1379   \expandafter \XINT_expr_getop
1380   \csname .=\expandafter\xINT_expr_seq:_aa
1381     \romannumeral-‘0\xINT_expr_unlock #2!{#1#4\relax !#3},^,\endcsname
1382 }%
1383 \def\xINT_expr_seq:_aa #1{\if +#1\expandafter\xINT_expr_seq:_A\else
1384   \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.

```
1385 \def\xINT_expr_seq:_a #1!#2{\expandafter\xINT_expr_seq_empty?
1386   \romannumeral0\xINT_expr_seq:_b {#2}#1}%
1387 \def\xINT_expr_seq:_b #1#2,{\XINT_expr_seq:_c #2,{#1}}%
1388 \def\xINT_expr_seq:_c #1{\if ,#1\xint_dothis\xINT_expr_seq:_noop\fi
1389   \if ^#1\xint_dothis\xINT_expr_seq:_end\fi
1390   \xint_orthat\xINT_expr_seq:_d #1}%
1391 \def\xINT_expr_seq:_d #1,#2{\expandafter\xINT_expr_seq:_e
1392   \romannumeral-‘0\expandafter\xINT_expr_unlock\romannumeral0#2{#1}{#2}}%
1393 \def\xINT_expr_seq:_e #1{\if #1^\xint_dothis\xINT_expr_seq:_abort\fi
1394   \if #1?\xint_dothis\xINT_expr_seq:_break\fi
1395   \if #1!\xint_dothis\xINT_expr_seq:_omit\fi
1396   \xint_orthat{\XINT_expr_seq:_goon #1}}%
1397 \def\xINT_expr_seq:_goon #1!#2#3#4{,#1\xINT_expr_seq:_b {#4}}%
1398 \def\xINT_expr_seq:_omit #1!#2#3#4{\XINT_expr_seq:_b {#4}}%
1399 \def\xINT_expr_seq:_abort #1!#2#3#4#5^,{ }%
```

```

1400 \def\XINT_expr_seq:_break #1!#2#3#4#5^,{, #1}%
1401 \def\XINT_expr_seq:_noop ,#1{\XINT_expr_seq:_b {#1}}%
1402 \def\XINT_expr_seq:_end ^,#1{% if all is omit, _empty? constructs "nil"
1403 \def\XINT_expr_seq_empty? #1{%
1404 \def\XINT_expr_seq_empty? ##1{\if ,##1\expandafter\xint_gobble_i\fi #1\endcsname }}%
1405 \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.

```

1406 \def\XINT_expr_seq:_A +#1!#2,^,%
1407 {\expandafter\XINT_expr_seq_empty?\romannumeral0\XINT_expr_seq:_D {#1}{#2}}%
1408 \def\XINT_expr_seq:_D #1#2{\expandafter\XINT_expr_seq:_E
1409 \romannumeral-'\0\expandafter\XINT_expr_unlock\romannumeral0#2{#1}{#2}}%
1410 \def\XINT_expr_seq:_E #1{\if #1^\xint_dothis\XINT_expr_seq:_Abort\fi
1411 \if #1?\xint_dothis\XINT_expr_seq:_Break\fi
1412 \if #1!\xint_dothis\XINT_expr_seq:_Omit\fi
1413 \xint_orthat{\XINT_expr_seq:_Goon #1}}%
1414 \def\XINT_expr_seq:_Goon #1!#2#3#4%
1415 {, #1\expandafter\XINT_expr_seq:_D\expandafter{\the\numexpr #3+\xint_c_i}{#4}}%
1416 \def\XINT_expr_seq:_Omit #1!#2#3#4%
1417 {\expandafter\XINT_expr_seq:_D\expandafter{\the\numexpr #3+\xint_c_i}{#4}}%
1418 \def\XINT_expr_seq:_Abort #1!#2#3#4}%
1419 \def\XINT_expr_seq:_Break #1!#2#3#4{, #1}%

```

10.32.7 add and mul, , \XINT_expr_onlitteral_add, \XINT_expr_onlitteral_mul

```

1420 \def\XINT_expr_onlitteral_add
1421 {\expandafter\XINT_expr_onlitteral_add_f\romannumeral-'\0\XINT_expr_onlitteral_seq_a {}}%
1422 \def\XINT_expr_onlitteral_add_f #1#2{\xint_c_xviii '{opx}#2)\relax #1+}%
1423 \def\XINT_expr_onlitteral_mul
1424 {\expandafter\XINT_expr_onlitteral_mul_f\romannumeral-'\0\XINT_expr_onlitteral_seq_a {}}%
1425 \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

```

1426 \expandafter\edef\csname XINT_expr_op:_+\endcsname
1427 {\noexpand\xint_gobble_v {}{}{}{} \expandafter\noexpand\csname .=\0\endcsname}%
1428 \expandafter\edef\csname XINT_expr_op:_*\endcsname
1429 {\noexpand\xint_gobble_v {}{}{}{} \expandafter\noexpand\csname .=\1\endcsname}%
1430 \def\XINT_expr_func_opx #1#2{\XINT_allexpr_opx \xintexpr}%
1431 \def\XINT_flexpr_func_opx #1#2{\XINT_allexpr_opx \xintfloatexpr}%
1432 \def\XINT_iiexpr_func_opx #1#2{\XINT_allexpr_opx \xintiexpr}%
1433 \def\XINT_allexpr_opx #1#2#3#4#5%
1434 {% au départ on avait op(#4,#3=#2 (évalué ici)) #3=la variable, #4=expression, #5=+ ou*.
1435 \expandafter\XINT_expr_getop\romannumeral0\expandafter\XINT_expr_op:_a
1436 \csname XINT_expr_op:_#5\expandafter\endcsname
1437 \romannumeral-'\0\XINT_expr_unlock #2!#5#1#3{#4}%
1438 }%

```

10.32.9 \XINT_expr_op:_a, ...

```

1439 \def\XINT_expr_op:_a #1#2!#3#4#5#6{\XINT_expr_op:_b {#1#4#3{#6\relax\relax !#5}}#2,^,%

```

```

1440 % #1=op_+ ou op_*, #2=liste, #3=+ou*, #4=\xintexpr, etc, #5=la var, #6=expression
1441 \def\xint_expr_op:_b #1#2,{\xint_expr_op:_c #2,#1}%
1442 \def\xint_expr_op:_c #1{\if ,#1\xint_dothis\xint_expr_op:_noop\fi
1443         \if ^#1\xint_dothis\xint_expr_op:_end\fi
1444         \xint_orthat\xint_expr_op:_d #1}%
1445 \def\xint_expr_op:_noop #1,#2#3#4#5{\xint_expr_op:_b {{#2}#3#4{#5}}}%
1446 \def\xint_expr_op:_d #1,#2#3#4#5%
1447 % #1=valeur, #2=partiel, #3=\xintexpr #4=+ ou *, #5 = expression
1448 {\expandafter\expandafter\expandafter\xint_expr_op:_e #3#2#4#3#5{#1}{#3#4{#5}}}%
1449 % #2=nom de la variable, #3=ancienne valeur variable,
1450 \def\xint_expr_op:_e !#1!#2#3#4{\xint_expr_op:_b {{!#1}#4}}%
1451 \def\xint_expr_op:_end ^,#1#2#3#4{\expandafter\expandafter\expandafter\space
1452         \expandafter\xint_gobble_iv #1}%

```

10.32.10 subs, \XINT_expr_onlitteral_subs

```

1453 \def\xint_expr_onlitteral_subs
1454 {\expandafter\xint_expr_onlitteral_subs_f\romannumeral-‘0\xint_expr_onlitteral_seq_a {}}%
1455 \def\xint_expr_onlitteral_subs_f #1#2{\xint_c_xviii ‘{subx}#2)\relax #1}%
1456 \def\xint_expr_func_subx #1#2{\XINT_allexpr_subx \xintbareeval }%
1457 \def\xint_flexpr_func_subx #1#2{\XINT_allexpr_subx \xintbarefloateval}%
1458 \def\xint_iexpr_func_subx #1#2{\XINT_allexpr_subx \xintbareiieval}%
1459 \def\xint_allexpr_subx #1#2#3#4% #2 is the value to assign to the dummy variable
1460 {% #3 is the dummy variable, #4 is the expression to evaluate
1461     \expandafter \XINT_expr_getop
1462     \csname .=\expandafter\xint_expr_subx:_a
1463         \romannumeral-‘0\xint_expr_unlock #2!{#1#4\relax !#3}\endcsname
1464 }%
1465 \def\xint_expr_subx:_a #1!#2% 10/25 that was a quick addition!
1466 {\expandafter\xint_expr_subx:_end \romannumeral0#2{#1}}%
1467 % attention, if one day I add a space in unlock, will need \romannumeral-‘0
1468 \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.

```

1469 \def\xint_expr_func_rseq {\XINT_allexpr_rseq \xintbareeval }%
1470 \def\xint_flexpr_func_rseq {\XINT_allexpr_rseq \xintbarefloateval }%
1471 \def\xint_iexpr_func_rseq {\XINT_allexpr_rseq \xintbareiieval }%
1472 \def\xint_allexpr_rseq #1#2%
1473 {%
1474     \expandafter\xint_expr_rseqx\expandafter #1\expandafter
1475     #2\romannumeral-‘0\xint_expr_onlitteral_seq_a {}}%
1476 }%

```

10.33.1 \XINT_expr_rseqx

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

```

1477 \def\xint_expr_rseqx #1#2#3#4%
1478 {%
1479     \expandafter\xint_expr_rseqx\romannumeral0#1(#4)\relax
1480     #2#3#1%

```

```

1481 }%
1482 % \def\XINT_expr_getlast #1,#2%
1483 % {%
1484 %     \if ^#2\xint_dothis{\XINT_expr_lockscan #1}\fi\xint_orthat{\XINT_expr_getlast #2}%
1485 % }%

```

10.33.2 \XINT_expr_rseq

```

1486 \def\XINT_expr_rseq #1#2#3#4#5% #1=valeurs pour variable (locked),
1487 % #2=toutes les valeurs initiales (csv,locked),
1488 % #3=variable, #4=expr,
1489 % #5=\xintbareeval ou \xintbarefloateval ou \xintbareiieval
1490 {%
1491     \expandafter \XINT_expr_getop
1492     \csname .=\XINT_expr_unlock #2%
1493     \expandafter\XINT_expr_rseq:_aa
1494         \romannumeral-`0\XINT_expr_unlock #1!{#5#4\relax !#3}#2,^,\endcsname
1495 }%
1496 \def\XINT_expr_rseq:_aa #1{\if +#1\expandafter\XINT_expr_rseq:_A\else
1497     \expandafter\XINT_expr_rseq:_a\fi #1}%

```

10.33.3 \XINT_expr_rseq:_a etc...

```

1498 \def\XINT_expr_rseq:_a #1!#2#3{\XINT_expr_rseq:_b #3{#2}#1}%
1499 \def\XINT_expr_rseq:_b #1#2#3,{\XINT_expr_rseq:_c #3,~#1{#2}}%
1500 \def\XINT_expr_rseq:_c #1{\if ,#1\xint_dothis\XINT_expr_rseq:_noop\fi
1501     \if ^#1\xint_dothis\XINT_expr_rseq:_end\fi
1502     \xint_orthat\XINT_expr_rseq:_d #1}%
1503 \def\XINT_expr_rseq:_d #1,~#2#3{\expandafter\XINT_expr_rseq:_e
1504     \romannumeral-`0\expandafter\XINT_expr_unlock\romannumeral0#3{#1}~#2{#3}}%
1505 \def\XINT_expr_rseq:_e #1{%
1506     \if ^#1\xint_dothis\XINT_expr_rseq:_abort\fi
1507     \if ?#1\xint_dothis\XINT_expr_rseq:_break\fi
1508     \if !#1\xint_dothis\XINT_expr_rseq:_omit\fi
1509     \xint_orthat{\XINT_expr_rseq:_goon #1}}%
1510 \def\XINT_expr_rseq:_goon #1!#2#3~#4#5{,#1\expandafter\XINT_expr_rseq:_b
1511     \romannumeral0\XINT_expr_lockit {#1}{#5}}%
1512 \def\XINT_expr_rseq:_omit #1!#2#3~{\XINT_expr_rseq:_b }%
1513 \def\XINT_expr_rseq:_abort #1!#2#3~#4#5#6^,{}%
1514 \def\XINT_expr_rseq:_break #1!#2#3~#4#5#6^,{,#1}%
1515 \def\XINT_expr_rseq:_noop ,~#1#2{\XINT_expr_rseq:_b #1{#2}}%
1516 \def\XINT_expr_rseq:_end ^,~#1#2{}% no nil for rseq

```

10.33.4 \XINT_expr_rseq:_A etc...

n++ for rseq

```

.
1517 \def\XINT_expr_rseq:_A +#1!#2#3,^,{\XINT_expr_rseq:_D {#1}#3{#2}}%
1518 \def\XINT_expr_rseq:_D #1#2#3{\expandafter\XINT_expr_rseq:_E
1519     \romannumeral-`0\expandafter\XINT_expr_unlock\romannumeral0#3{#1}~#2{#3}}%
1520 \def\XINT_expr_rseq:_E #1{\if #1^\xint_dothis\XINT_expr_rseq:_Abort\fi
1521     \if #1?\xint_dothis\XINT_expr_rseq:_Break\fi
1522     \if #1!\xint_dothis\XINT_expr_rseq:_Omit\fi
1523     \xint_orthat{\XINT_expr_rseq:_Goon #1}}%

```



```

1524 \def\XINT_expr_rseq:_Goon #1!#2#3~#4#5%
1525     {,#1\expandafter\XINT_expr_rseq:_D\expandafter{\the\numexpr #3+\xint_c_i\expandafter}%
1526     \romannumeral0\XINT_expr_lockit{#1}{#5}}}%
1527 \def\XINT_expr_rseq:_Omit #1!#2#3~%#4#5%
1528     {\expandafter\XINT_expr_rseq:_D\expandafter{\the\numexpr #3+\xint_c_i}}}%
1529 \def\XINT_expr_rseq:_Abort #1!#2#3~#4#5{}%
1530 \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.

```

1531 \def\XINT_expr_func_rrseq {\XINT_allexpr_rrseq \xintbareeval }%
1532 \def\XINT_flexpr_func_rrseq {\XINT_allexpr_rrseq \xintbarefloateval }%
1533 \def\XINT_iiexpr_func_rrseq {\XINT_allexpr_rrseq \xintbareiieval }%
1534 \def\XINT_allexpr_rrseq #1#2%
1535 {%
1536     \expandafter\XINT_expr_rrseq\expandafter #1\expandafter
1537     #2\romannumeral-‘0\XINT_expr_onlitteral_seq_a {}%
1538 }%

```

10.34.1 \XINT_expr_rrseqx

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

```

1539 \def\XINT_expr_rrseqx #1#2#3#4%
1540 {%
1541     \expandafter\XINT_expr_rrseq\romannumeral0#1(#4)\expandafter\relax
1542     \expandafter{\romannumeral0\xintapply \XINT_expr_lockit
1543         {\xintRevWithBraces{\xintCSVtoListNonStripped{\XINT_expr_unlock #2}}}}}%
1544     #2#3#1%
1545 }%

```

10.34.2 \XINT_expr_rrseqy

```

1546 \def\XINT_expr_rrseqy #1#2#3#4#5#6% #1=valeurs pour variable (locked),
1547     % #2=initial values (reversed, one (braced) token each)
1548     % #3=toutes les valeurs initiales (csv,locked),
1549     % #4=variable, #5=expr,
1550     % #6=\xintbareeval ou \xintbarefloateval ou \xintbareiieval
1551 {%
1552     \expandafter \XINT_expr_getop
1553     \csname .=\XINT_expr_unlock #3%
1554     \expandafter\XINT_expr_rrseq:_aa
1555         \romannumeral-‘0\XINT_expr_unlock #1!{#6#5\relax !#4}{#2},^,\endcsname
1556 }%
1557 \def\XINT_expr_rrseq:_aa #1{\if +#1\expandafter\XINT_expr_rrseq:_A\else
1558     \expandafter\XINT_expr_rrseq:_a\fi #1}%

```

10.34.3 \XINT_expr_rrseq:_a etc...

```

1559 \catcode‘? 3
1560 \def\XINT_expr_rrseq:_a #1!#2#3{\XINT_expr_rrseq:_b {#3}{#2}#1}%

```



```

1561 \def\XINT_expr_rrseq:_b #1#2#3,{\XINT_expr_rrseq:_c #3,~#1?{#2}}%
1562 \def\XINT_expr_rrseq:_c #1{\if ,#1\xint_dothis\XINT_expr_rrseq:_noop\fi
1563         \if ^#1\xint_dothis\XINT_expr_rrseq:_end\fi
1564         \xint_orthat\XINT_expr_rrseq:_d #1}%
1565 \def\XINT_expr_rrseq:_d #1,~#2?#3{\expandafter\XINT_expr_rrseq:_e
1566         \romannumeral-‘0\expandafter\XINT_expr_unlock\romannumeral0#3{#1}~#2?{#3}}%
1567 \def\XINT_expr_rrseq:_goon #1!#2#3~#4?#5{,#1\expandafter\XINT_expr_rrseq:_b\expandafter
1568         {\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
1569 \def\XINT_expr_rrseq:_omit #1!#2#3~{\XINT_expr_rrseq:_b }%
1570 \def\XINT_expr_rrseq:_abort #1!#2#3~#4?#5#6^,{,%
1571 \def\XINT_expr_rrseq:_break #1!#2#3~#4?#5#6^,{,#1}%
1572 \def\XINT_expr_rrseq:_noop ,~#1?#2{\XINT_expr_rrseq:_b {#1}{#2}}%
1573 \def\XINT_expr_rrseq:_end ^,~#1?#2{% No nil for rrseq.
1574 \catcode'? 11
1575 \def\XINT_expr_rrseq:_e #1{%
1576     \if ^#1\xint_dothis\XINT_expr_rrseq:_abort\fi
1577     \if ?#1\xint_dothis\XINT_expr_rrseq:_break\fi
1578     \if !#1\xint_dothis\XINT_expr_rrseq:_omit\fi
1579     \xint_orthat{\XINT_expr_rrseq:_goon #1}%
1580 }%

```

10.34.4 \XINT_expr_rrseq:_A etc...

n++ for rrseq

```

1581 \catcode'? 3
1582 \def\XINT_expr_rrseq:_A +#1!#2#3,^,{\XINT_expr_rrseq:_D {#1}{#3}{#2}}%
1583 \def\XINT_expr_rrseq:_D #1#2#3{\expandafter\XINT_expr_rrseq:_E
1584     \romannumeral-‘0\expandafter\XINT_expr_unlock\romannumeral0#3{#1}~#2?{#3}}%
1585 \def\XINT_expr_rrseq:_Goon #1!#2#3~#4?#5%
1586     {,#1\expandafter\XINT_expr_rrseq:_D\expandafter{\the\numexpr #3+\xint_c_i\expandafter}%
1587     \expandafter{\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
1588 \def\XINT_expr_rrseq:_Omit #1!#2#3~%#4?#5%
1589     {\expandafter\XINT_expr_rrseq:_D\expandafter{\the\numexpr #3+\xint_c_i}}%
1590 \def\XINT_expr_rrseq:_Abort #1!#2#3~#4?#5{%
1591 \def\XINT_expr_rrseq:_Break #1!#2#3~#4?#5{,#1}%
1592 \catcode'? 11
1593 \def\XINT_expr_rrseq:_E #1{\if #1^\xint_dothis\XINT_expr_rrseq:_Abort\fi
1594         \if #1?\xint_dothis\XINT_expr_rrseq:_Break\fi
1595         \if #1!\xint_dothis\XINT_expr_rrseq:_Omit\fi
1596         \xint_orthat{\XINT_expr_rrseq:_Goon #1}}%

```

10.35 iter

```

1597 \def\XINT_expr_func_iter {\XINT_allexpr_iter \xintbareeval }%
1598 \def\XINT_flexpr_func_iter {\XINT_allexpr_iter \xintbarefloateval }%
1599 \def\XINT_iiexpr_func_iter {\XINT_allexpr_iter \xintbareiieval }%
1600 \def\XINT_allexpr_iter #1#2%
1601 {%
1602     \expandafter\XINT_expr_iterx\expandafter #1\expandafter
1603     #2\romannumeral-‘0\XINT_expr_onlitteral_seq_a {}%
1604 }%

```

10.35.1 `\XINT_expr_iterx`

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

```
1605 \def\XINT_expr_iterx #1#2#3#4%
1606 {%
1607   \expandafter\XINT_expr_iter\romannumeral0#1(#4)\expandafter\relax
1608   \expandafter{\romannumeral0\xintapply \XINT_expr_lockit
1609     {\xintRevWithBraces{\xintCSVtoListNonStripped{\XINT_expr_unlock #2}}}}%
1610   #2#3#1%
1611 }%
```

10.35.2 `\XINT_expr_itery`

```
1612 \def\XINT_expr_itery #1#2#3#4#5#6% #1=valeurs pour variable (locked),
1613   % #2=initial values (reversed, one (braced) token each)
1614   % #3=toutes les valeurs initiales (csv,locked),
1615   % #4=variable, #5=expr,
1616   % #6=\xintbareeval ou \xintbarefloateval ou \xintbareiieval
1617 {%
1618   \expandafter \XINT_expr_getop
1619   \csname .=%
1620   \expandafter\XINT_expr_iter:_aa
1621     \romannumeral-‘0\XINT_expr_unlock #1!{#6#5\relax !#4}{#2},^,\endcsname
1622 }%
1623 \def\XINT_expr_iter:_aa #1{\if +#1\expandafter\XINT_expr_iter:_A\else
1624   \expandafter\XINT_expr_iter:_a\fi #1}%

```

10.35.3 `\XINT_expr_iter:_a` etc...

```
1625 \catcode‘? 3
1626 \def\XINT_expr_iter:_a #1!#2#3{\XINT_expr_iter:_b {#3}{#2}#1}%
1627 \def\XINT_expr_iter:_b #1#2#3,{\XINT_expr_iter:_c #3,~#1?{#2}}%
1628 \def\XINT_expr_iter:_c #1{\if ,#1\xint_dothis\XINT_expr_iter:_noop\fi
1629   \if ^#1\xint_dothis\XINT_expr_iter:_end\fi
1630   \xint_orthat\XINT_expr_iter:_d #1}%
1631 \def\XINT_expr_iter:_d #1,~#2?#3{\expandafter\XINT_expr_iter:_e
1632   \romannumeral-‘0\expandafter\XINT_expr_unlock\romannumeral0#3{#1}~#2?{#3}}%
1633 \def\XINT_expr_iter:_goon #1!#2#3~#4?#5{\expandafter\XINT_expr_iter:_b\expandafter
1634   {\romannumeral0\xinttrim{-1}}{\XINT_expr_lockit{#1}{#4}}{#5}}%
1635 \def\XINT_expr_iter:_omit #1!#2#3~{\XINT_expr_iter:_b }%
1636 \def\XINT_expr_iter:_abort #1!#2#3~#4?#5#6^,%
1637   {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1638     {\XINT_expr:_unlock}{\xintReverseOrder{#4\space}}}%
1639 \def\XINT_expr_iter:_break #1!#2#3~#4?#5#6^,%
1640   {\expandafter\xint_gobble_iv\romannumeral0\xintapplyunbraced
1641     {\XINT_expr:_unlock}{\xintReverseOrder{#4\space}},#1}%
1642 \def\XINT_expr_iter:_noop ,~#1?#2{\XINT_expr_iter:_b {#1}{#2}}%
1643 \def\XINT_expr_iter:_end ^,~#1?#2%
1644   {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1645     {\XINT_expr:_unlock}{\xintReverseOrder{#1\space}}}%
1646 \catcode‘? 11
1647 \def\XINT_expr_iter:_e #1{%
1648   \if ^#1\xint_dothis\XINT_expr_iter:_abort\fi

```

```

1649 \if ?#1\xint_dothis\XINT_expr_iter:_break\fi
1650 \if !#1\xint_dothis\XINT_expr_iter:_omit\fi
1651 \xint_orthat{\XINT_expr_iter:_goon #1}%
1652 }%
1653 \def\XINT_expr:_unlock #1{\XINT_expr_unlock #1}%

10.35.4 \XINT_expr_iter:_A etc...

n++ for iter
.
1654 \catcode'? 3
1655 \def\XINT_expr_iter:_A +#1!#2#3,^,{\XINT_expr_iter:_D {#1}{#3}{#2}}%
1656 \def\XINT_expr_iter:_D #1#2#3{\expandafter\XINT_expr_iter:_E
1657 \romannumeral-'0\expandafter\XINT_expr_unlock\romannumeral0#3{#1}~#2?{#3}}%
1658 \def\XINT_expr_iter:_Goon #1!#2#3~#4?#5%
1659 {\expandafter\XINT_expr_iter:_D\expandafter{\the\numexpr #3+\xint_c_i\expandafter}%
1660 \expandafter{\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
1661 \def\XINT_expr_iter:_Omit #1!#2#3~%#4?#5%
1662 {\expandafter\XINT_expr_iter:_D\expandafter{\the\numexpr #3+\xint_c_i}}%
1663 \def\XINT_expr_iter:_Abort #1!#2#3~#4?#5%
1664 {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1665 {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}}}%
1666 \def\XINT_expr_iter:_Break #1!#2#3~#4?#5%
1667 {\expandafter\xint_gobble_iv\romannumeral0\xintapplyunbraced
1668 {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}},#1}%
1669 \catcode'? 11
1670 \def\XINT_expr_iter:_E #1{\if #1^\xint_dothis\XINT_expr_iter:_Abort\fi
1671 \if #1?\xint_dothis\XINT_expr_iter:_Break\fi
1672 \if #1!\xint_dothis\XINT_expr_iter:_Omit\fi
1673 \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 roman-numeral in general, but in some cases they are, especially when involved in an `\xintNewExpr`. They will then be protected against expansion will 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.

```

1674 \def\xintANDof:csv #1{\expandafter\XINT_andof:_a\romannumeral-'0#1,,^}%
1675 \def\XINT_andof:_a #1{\if ,#1\expandafter\XINT_andof:_e
1676 \else\expandafter\XINT_andof:_c\fi #1}%
1677 \def\XINT_andof:_c #1,{\xintiiifNotZero {#1}{\XINT_andof:_a}{\XINT_andof:_no}}%
1678 \def\XINT_andof:_no #1^{0}%
1679 \def\XINT_andof:_e #1^{1}% works with empty list

```

10.36.2 `\xintORof:csv`

1.09a. For use by `\xintexpr`.

```
1680 \def\xintORof:csv #1{\expandafter\xINT_orof:_a\romannumeral-‘0#1,,^}%
1681 \def\xINT_orof:_a #1{\if ,#1\expandafter\xINT_orof:_e
1682         \else\expandafter\xINT_orof:_c\fi #1}%
1683 \def\xINT_orof:_c #1,{\xintiifNotZero{#1}{\xINT_orof:_yes}{\xINT_orof:_a}}%
1684 \def\xINT_orof:_yes #1^{1}%
1685 \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`).

```
1686 \def\xintXORof:csv #1{\expandafter\xINT_xorof:_a\expandafter 0\romannumeral-‘0#1,,^}%
1687 \def\xINT_xorof:_a #1#2,{\xINT_xorof:_b #2,#1}%
1688 \def\xINT_xorof:_b #1{\if ,#1\expandafter\xINT_xorof:_e
1689         \else\expandafter\xINT_xorof:_c\fi #1}%
1690 \def\xINT_xorof:_c #1,#2%
1691         {\xintiifNotZero {#1}{\if #20\xint_afterfi{\xINT_xorof:_a 1}%
1692                 \else\xint_afterfi{\xINT_xorof:_a 0}\fi}%
1693                 {\xINT_xorof:_a #2}%
1694         }%
1695 \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.

```
1696 \def\xINT_oncsv:_empty #1,^,#2{#2}%
1697 \def\xINT_oncsv:_end ^,#1#2#3#4{#1}%
1698 \def\xINT_oncsv:_a #1#2#3%
1699     {\if ,#3\expandafter\xINT_oncsv:_empty\else\expandafter\xINT_oncsv:_b\fi #1#2#3}%
1700 \def\xINT_oncsv:_b #1#2#3,%
1701     {\expandafter\xINT_oncsv:_c \expandafter{\romannumeral-‘0#2{#3}}#1#2}%
1702 \def\xINT_oncsv:_c #1#2#3#4,{\expandafter\xINT_oncsv:_d \romannumeral-‘0#4,{#1}{#2#3}%
1703 \def\xINT_oncsv:_d #1%
1704     {\if ^#1\expandafter\xINT_oncsv:_end\else\expandafter\xINT_oncsv:_e\fi #1}%
1705 \def\xINT_oncsv:_e #1,#2#3#4%
1706     {\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 `xintiexpr`, dans il faut bien sûr ne pas faire de `xintNum`, donc il faut un `iimax`.

```
1707 \def\xintMaxof:csv #1{\expandafter\xINT_oncsv:_a\expandafter\xintmax
1708         \expandafter\xint_firstofone\romannumeral-‘0#1,^{0/1[0]}}%
1709 \def\xintiiMaxof:csv #1{\expandafter\xINT_oncsv:_a\expandafter\xintiiimax
1710         \expandafter\xint_firstofone\romannumeral-‘0#1,^{0}%

```

10.36.6 `\xintMinof:csv`, `\xintiiMinof:csv`

1.09i. Rewritten for 1.1. For use by `\xintiexpr`.

```
1711 \def\xintMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmin
1712         \expandafter\xint_firstofone\romannumeral-‘0#1,^{0/1[0]}}}%
1713 \def\xintiiMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiimin
1714         \expandafter\xint_firstofone\romannumeral-‘0#1,^{0}%
```

10.36.7 `\xintSum:csv`, `cshxintiiSum:csv`

1.09a. Rewritten for 1.1. For use by `\xintexpr`.

```
1715 \def\xintSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintadd
1716         \expandafter\xint_firstofone\romannumeral-‘0#1,^{0/1[0]}}}%
1717 \def\xintiiSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiaadd
1718         \expandafter\xint_firstofone\romannumeral-‘0#1,^{0}%
```

10.36.8 `\xintPrd:csv`, `\xintiiPrd:csv`

1.09a. Rewritten for 1.1. For use by `\xintexpr`.

```
1719 \def\xintPrd:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmul
1720         \expandafter\xint_firstofone\romannumeral-‘0#1,^{1/1[0]}}}%
1721 \def\xintiiPrd:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiimul
1722         \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

```
1723 \def\xintGCDof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintgcd
1724         \expandafter\xint_firstofone\romannumeral-‘0#1,^{1}%
1725 \def\xintLCMof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintlcm
1726         \expandafter\xint_firstofone\romannumeral-‘0#1,^{0}%
```

10.36.10 `\XINTinFloatdigits`, `\XINTinFloatSqrtdigits`

for `\xintNewExpr` matters, mainly.

```
1727 \def\XINTinFloatdigits {\XINTinFloat [\XINTdigits]}%
1728 \def\XINTinFloatSqrtdigits {\XINTinFloatSqrt [\XINTdigits]}%
```

10.36.11 `\XINTinFloatMaxof:csv`, `\XINTinFloatMinof:csv`

1.09a. Rewritten for 1.1. For use by `\xintfloatexpr`. Name changed in 1.09h

```
1729 \def\XINTinFloatMaxof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmax
1730         \expandafter\XINTinFloatdigits\romannumeral-‘0#1,^{0[0]}}}%
1731 \def\XINTinFloatMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmin
1732         \expandafter\XINTinFloatdigits\romannumeral-‘0#1,^{0[0]}}}%
```

10.36.12 `\XINTinFloatSum:csv`, `\XINTinFloatPrd:csv`

1.09a. Rewritten for 1.1. For use by `\xintfloatexpr`.

```

1733 \def\XINTinFloatSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\XINTinfloatadd
1734 \expandafter\XINTinFloatdigits\romannumeral-'0#1,^{0[0]}}}%
1735 \def\XINTinFloatPrd:csv #1{\expandafter\XINT_oncsv:_a\expandafter\XINTinfloatmul
1736 \expandafter\XINTinFloatdigits\romannumeral-'0#1,^{1[0]}}}%

```

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

```

1737 \def\XINT_expr_twoargs #1,#2,{\{#1}{#2}}}%
1738 \def\XINT_expr_argadopt #1,#2,#3.#4#5%
1739 {%
1740 \if\relax#3\relax\expandafter\xint_firstoftwo\else
1741 \expandafter\xint_secondoftwo\fi
1742 {\#4}{\#5[\xintNum {\#2}]}{\#1}%
1743 }%
1744 \def\XINT_expr_oneortwo #1#2#3,#4,#5.%
1745 {%
1746 \if\relax#5\relax\expandafter\xint_firstoftwo\else
1747 \expandafter\xint_secondoftwo\fi
1748 {\#1{0}}{\#2{\xintNum {\#4}}}{\#3}%
1749 }%
1750 \def\XINT_iiexpr_oneortwo #1#2,#3,#4.%
1751 {%
1752 \if\relax#4\relax\expandafter\xint_firstoftwo\else
1753 \expandafter\xint_secondoftwo\fi
1754 {\#1{0}}{\#1{\#3}}{\#2}%
1755 }%
1756 \def\XINT_expr_func_num #1#2#3%
1757 {\expandafter #1\expandafter #2\csname=\xintNum {\XINT_expr_unlock #3}\endcsname}%
1758 \let\XINT_flexpr_func_num\XINT_expr_func_num
1759 \let\XINT_iiexpr_func_num\XINT_expr_func_num
1760 % [0] added Oct 25. For interaction with SPRaw::csv
1761 \def\XINT_expr_func_reduce #1#2#3%
1762 {\expandafter #1\expandafter #2\csname=\xintIrr {\XINT_expr_unlock #3}[0]\endcsname}%
1763 \let\XINT_flexpr_func_reduce\XINT_expr_func_reduce
1764 % no \XINT_iiexpr_func_reduce
1765 \def\XINT_expr_func_abs #1#2#3%
1766 {\expandafter #1\expandafter #2\csname=\xintAbs {\XINT_expr_unlock #3}\endcsname}%
1767 \let\XINT_flexpr_func_abs\XINT_expr_func_abs
1768 \def\XINT_iiexpr_func_abs #1#2#3%
1769 {\expandafter #1\expandafter #2\csname=\xintiiAbs {\XINT_expr_unlock #3}\endcsname}%
1770 \def\XINT_expr_func_sgn #1#2#3%
1771 {\expandafter #1\expandafter #2\csname=\xintSgn {\XINT_expr_unlock #3}\endcsname}%
1772 \let\XINT_flexpr_func_sgn\XINT_expr_func_sgn
1773 \def\XINT_iiexpr_func_sgn #1#2#3%
1774 {\expandafter #1\expandafter #2\csname=\xintiiSgn {\XINT_expr_unlock #3}\endcsname}%
1775 \def\XINT_expr_func_frac #1#2#3%
1776 {\expandafter #1\expandafter #2\csname=\xintTFrac {\XINT_expr_unlock #3}\endcsname}%

```

```

1777 \def\XINT_flexpr_func_frac #1#2#3{\expandafter #1\expandafter #2\csname
1778   .=\XINTinFloatFrac {\XINT_expr_unlock #3}\endcsname }%
1779 % no \XINT_iiexpr_func_frac
1780 \def\XINT_expr_func_floor #1#2#3%
1781   {\expandafter #1\expandafter #2\csname .=\xintFloor {\XINT_expr_unlock #3}\endcsname }%
1782 \let\XINT_flexpr_func_floor\XINT_expr_func_floor
1783 \def\XINT_iiexpr_func_floor #1#2#3%
1784 {% mais absurde si on ne peut pas avoir quotient comme input
1785   \expandafter #1\expandafter #2\csname .=\xintiFloor {\XINT_expr_unlock #3}\endcsname }%
1786 \def\XINT_expr_func_ceil #1#2#3%
1787   {\expandafter #1\expandafter #2\csname .=\xintCeil {\XINT_expr_unlock #3}\endcsname }%
1788 \let\XINT_flexpr_func_ceil\XINT_expr_func_ceil
1789 \def\XINT_iiexpr_func_ceil #1#2#3%
1790 {% mais absurde si on ne peut pas avoir quotient comme input
1791   \expandafter #1\expandafter #2\csname .=\xintiCeil {\XINT_expr_unlock #3}\endcsname }%
1792 \def\XINT_expr_func_sqr #1#2#3%
1793   {\expandafter #1\expandafter #2\csname .=\xintSqr {\XINT_expr_unlock #3}\endcsname }%
1794 \def\XINT_flexpr_func_sqr #1#2#3%
1795 {%
1796   \expandafter #1\expandafter #2\csname
1797   .=\XINTinFloatMul % [\XINTdigits]% pour simplifier mes affaires avec \xintNewExpr
1798     {\XINT_expr_unlock #3}{\XINT_expr_unlock #3}\endcsname
1799 }%
1800 \def\XINT_iiexpr_func_sqr #1#2#3%
1801   {\expandafter #1\expandafter #2\csname .=\xintiiSqr {\XINT_expr_unlock #3}\endcsname }%
1802 \def\XINT_expr_func_sqrt #1#2#3%
1803 {%
1804   \expandafter #1\expandafter #2\csname .=%
1805   \expandafter\XINT_expr_argandopt
1806   \romannumeral-'0\XINT_expr_unlock#3,,.\XINTinFloatSqrtdigits\XINTinFloatSqrt
1807   \endcsname
1808 }%
1809 \let\XINT_flexpr_func_sqrt\XINT_expr_func_sqrt
1810 \def\XINT_iiexpr_func_sqrt #1#2#3%
1811   {\expandafter #1\expandafter #2\csname .=\xintiiSqrt {\XINT_expr_unlock #3}\endcsname }%
1812 \def\XINT_expr_func_round #1#2#3%
1813 {%
1814   \expandafter #1\expandafter #2\csname .=%
1815   \expandafter\XINT_expr_oneortwo
1816   \expandafter\xintiRound\expandafter\xintRound
1817   \romannumeral-'0\XINT_expr_unlock #3,,.\endcsname
1818 }%
1819 \let\XINT_flexpr_func_round\XINT_expr_func_round
1820 \def\XINT_iiexpr_func_round #1#2#3%
1821 {%
1822   \expandafter #1\expandafter #2\csname .=%
1823   \expandafter\XINT_iiexpr_oneortwo\expandafter\xintiRound
1824   \romannumeral-'0\XINT_expr_unlock #3,,.\endcsname
1825 }%
1826 \def\XINT_expr_func_trunc #1#2#3%
1827 {%
1828   \expandafter #1\expandafter #2\csname .=%

```



```

1829 \expandafter\XINT_expr_oneortwo
1830 \expandafter\xintiTrunc\expandafter\xintTrunc
1831 \romannumeral-‘0\XINT_expr_unlock #3,,.\endcsname
1832 }%
1833 \let\XINT_flexpr_func_trunc\XINT_expr_func_trunc
1834 \def\XINT_iiexpr_func_trunc #1#2#3%
1835 {%
1836 \expandafter #1\expandafter #2\csname .=%
1837 \expandafter\XINT_iiexpr_oneortwo\expandafter\xintiTrunc
1838 \romannumeral-‘0\XINT_expr_unlock #3,,.\endcsname
1839 }%
1840 \def\XINT_expr_func_float #1#2#3%
1841 {%
1842 \expandafter #1\expandafter #2\csname .=%
1843 \expandafter\XINT_expr_argandopt
1844 \romannumeral-‘0\XINT_expr_unlock #3,,.\XINTinFloatdigits\XINTinFloat
1845 \endcsname
1846 }%
1847 \let\XINT_flexpr_func_float\XINT_expr_func_float
1848 % \XINT_iiexpr_func_float not defined
1849 \def\XINT_expr_func_mod #1#2#3%
1850 {%
1851 \expandafter #1\expandafter #2\csname .=%
1852 \expandafter\expandafter\expandafter\xintMod
1853 \expandafter\XINT_expr_twoargs
1854 \romannumeral-‘0\XINT_expr_unlock #3,\endcsname
1855 }%
1856 \def\XINT_flexpr_func_mod #1#2#3%
1857 {%
1858 \expandafter #1\expandafter #2\csname .=%
1859 \expandafter\XINTinFloatMod
1860 \romannumeral-‘0\expandafter\XINT_expr_twoargs
1861 \romannumeral-‘0\XINT_expr_unlock #3,\endcsname
1862 }%
1863 \def\XINT_iiexpr_func_mod #1#2#3%
1864 {%
1865 \expandafter #1\expandafter #2\csname .=%
1866 \expandafter\expandafter\expandafter\xintiMod
1867 \expandafter\XINT_expr_twoargs
1868 \romannumeral-‘0\XINT_expr_unlock #3,\endcsname
1869 }%
1870 \def\XINT_expr_func_quo #1#2#3%
1871 {%
1872 \expandafter #1\expandafter #2\csname .=%
1873 \expandafter\expandafter\expandafter\xintiQuo
1874 \expandafter\XINT_expr_twoargs
1875 \romannumeral-‘0\XINT_expr_unlock #3,\endcsname
1876 }%
1877 \let\XINT_flexpr_func_quo\XINT_expr_func_quo
1878 \def\XINT_iiexpr_func_quo #1#2#3%
1879 {%
1880 \expandafter #1\expandafter #2\csname .=%

```



```

1881 \expandafter\expandafter\expandafter\xintiiQuo
1882 \expandafter\XINT_expr_twoargs
1883 \romannumeral-'0\XINT_expr_unlock #3,\endcsname
1884 }%
1885 \def\XINT_expr_func_rem #1#2#3%
1886 {%
1887 \expandafter #1\expandafter #2\csname .=%
1888 \expandafter\expandafter\expandafter\xintiRem
1889 \expandafter\XINT_expr_twoargs
1890 \romannumeral-'0\XINT_expr_unlock #3,\endcsname
1891 }%
1892 \let\XINT_flexpr_func_rem\XINT_expr_func_rem
1893 \def\XINT_iiexpr_func_rem #1#2#3%
1894 {%
1895 \expandafter #1\expandafter #2\csname .=%
1896 \expandafter\expandafter\expandafter\xintiiRem
1897 \expandafter\XINT_expr_twoargs
1898 \romannumeral-'0\XINT_expr_unlock #3,\endcsname
1899 }%
1900 \def\XINT_expr_func_gcd #1#2#3%
1901 {\expandafter #1\expandafter #2\csname
1902 .=\xintGCDoF:csv{\XINT_expr_unlock #3}\endcsname }%
1903 \let\XINT_flexpr_func_gcd\XINT_expr_func_gcd
1904 \let\XINT_iiexpr_func_gcd\XINT_expr_func_gcd
1905 \def\XINT_expr_func_lcm #1#2#3%
1906 {\expandafter #1\expandafter #2\csname
1907 .=\xintLCMoF:csv{\XINT_expr_unlock #3}\endcsname }%
1908 \let\XINT_flexpr_func_lcm\XINT_expr_func_lcm
1909 \let\XINT_iiexpr_func_lcm\XINT_expr_func_lcm
1910 \def\XINT_expr_func_max #1#2#3%
1911 {\expandafter #1\expandafter #2\csname
1912 .=\xintMaxoF:csv{\XINT_expr_unlock #3}\endcsname }%
1913 \def\XINT_iiexpr_func_max #1#2#3%
1914 {\expandafter #1\expandafter #2\csname
1915 .=\xintiiMaxoF:csv{\XINT_expr_unlock #3}\endcsname }%
1916 \def\XINT_flexpr_func_max #1#2#3%
1917 {\expandafter #1\expandafter #2\csname
1918 .=\XINTinFloatMaxoF:csv{\XINT_expr_unlock #3}\endcsname }%
1919 \def\XINT_expr_func_min #1#2#3%
1920 {\expandafter #1\expandafter #2\csname
1921 .=\xintMinof:csv{\XINT_expr_unlock #3}\endcsname }%
1922 \def\XINT_iiexpr_func_min #1#2#3%
1923 {\expandafter #1\expandafter #2\csname
1924 .=\xintiiMinof:csv{\XINT_expr_unlock #3}\endcsname }%
1925 \def\XINT_flexpr_func_min #1#2#3%
1926 {\expandafter #1\expandafter #2\csname
1927 .=\XINTinFloatMinof:csv{\XINT_expr_unlock #3}\endcsname }%
1928 \expandafter\def\csname XINT_expr_func_+\endcsname #1#2#3%
1929 {\expandafter #1\expandafter #2\csname
1930 .=\xintSum:csv{\XINT_expr_unlock #3}\endcsname }%
1931 \expandafter\def\csname XINT_flexpr_func_+\endcsname #1#2#3%
1932 {\expandafter #1\expandafter #2\csname

```

```

1933      .=\XINTinFloatSum:csv{\XINT_expr_unlock #3}\endcsname }%
1934 \expandafter\def\csname XINT_iiexpr_func_+\endcsname #1#2#3%
1935     {\expandafter #1\expandafter #2\csname
1936      .=\xintiiSum:csv{\XINT_expr_unlock #3}\endcsname }%
1937 \expandafter\def\csname XINT_expr_func_*\endcsname #1#2#3%
1938     {\expandafter #1\expandafter #2\csname
1939      .=\xintPrd:csv{\XINT_expr_unlock #3}\endcsname }%
1940 \expandafter\def\csname XINT_flexpr_func_*\endcsname #1#2#3%
1941     {\expandafter #1\expandafter #2\csname
1942      .=\XINTinFloatPrd:csv{\XINT_expr_unlock #3}\endcsname }%
1943 \expandafter\def\csname XINT_iiexpr_func_*\endcsname #1#2#3%
1944     {\expandafter #1\expandafter #2\csname
1945      .=\xintiiPrd:csv{\XINT_expr_unlock #3}\endcsname }%
1946 \def\XINT_expr_func_? #1#2#3%
1947     {\expandafter #1\expandafter #2\csname
1948      .=\xintiiIsNotZero {\XINT_expr_unlock #3}\endcsname }%
1949 \let\XINT_flexpr_func_? \XINT_expr_func_?
1950 \let\XINT_iiexpr_func_? \XINT_expr_func_?
1951 \def\XINT_expr_func_! #1#2#3%
1952 {\expandafter #1\expandafter #2\csname.=\xintiiIsZero {\XINT_expr_unlock #3}\endcsname }%
1953 \let\XINT_flexpr_func_! \XINT_expr_func_!
1954 \let\XINT_iiexpr_func_! \XINT_expr_func_!
1955 \def\XINT_expr_func_not #1#2#3%
1956 {\expandafter #1\expandafter #2\csname.=\xintiiIsZero {\XINT_expr_unlock #3}\endcsname }%
1957 \let\XINT_flexpr_func_not \XINT_expr_func_not
1958 \let\XINT_iiexpr_func_not \XINT_expr_func_not
1959 \def\XINT_expr_func_all #1#2#3%
1960     {\expandafter #1\expandafter #2\csname
1961      .=\xintANDof:csv{\XINT_expr_unlock #3}\endcsname }%
1962 \let\XINT_flexpr_func_all\XINT_expr_func_all
1963 \let\XINT_iiexpr_func_all\XINT_expr_func_all
1964 \def\XINT_expr_func_any #1#2#3%
1965     {\expandafter #1\expandafter #2\csname
1966      .=\xintORof:csv{\XINT_expr_unlock #3}\endcsname }%
1967 \let\XINT_flexpr_func_any\XINT_expr_func_any
1968 \let\XINT_iiexpr_func_any\XINT_expr_func_any
1969 \def\XINT_expr_func_xor #1#2#3%
1970     {\expandafter #1\expandafter #2\csname
1971      .=\xintXORof:csv{\XINT_expr_unlock #3}\endcsname }%
1972 \let\XINT_flexpr_func_xor\XINT_expr_func_xor
1973 \let\XINT_iiexpr_func_xor\XINT_expr_func_xor
1974 \def\xintifNotZero: #1,#2,#3,{\xintiiifNotZero{#1}{#2}{#3}}%
1975 \def\XINT_expr_func_if #1#2#3%
1976     {\expandafter #1\expandafter #2\csname
1977      .=\expandafter\xintifNotZero:\romannumeral-‘0\XINT_expr_unlock #3,\endcsname }%
1978 \let\XINT_flexpr_func_if\XINT_expr_func_if
1979 \let\XINT_iiexpr_func_if\XINT_expr_func_if
1980 \def\xintifSgn: #1,#2,#3,#4,{\xintiiifSgn{#1}{#2}{#3}{#4}}%
1981 \def\XINT_expr_func_ifsgn #1#2#3%
1982 {%
1983     \expandafter #1\expandafter #2\csname
1984     .=\expandafter\xintifSgn:\romannumeral-‘0\XINT_expr_unlock #3,\endcsname

```

```

1985 }%
1986 \let\XINT_flexpr_func_ifsgn\XINT_expr_func_ifsgn
1987 \let\XINT_iiexpr_func_ifsgn\XINT_expr_func_ifsgn
1988 \def\XINT_expr_func_first #1#2#3%
1989     {\expandafter #1\expandafter #2\csname.=\expandafter\XINT_expr_func_firsta
1990         \romannumeral-‘0\XINT_expr_unlock #3,^\endcsname }%
1991 \def\XINT_expr_func_firsta #1,#2^{#1}%
1992 \let\XINT_flexpr_func_first\XINT_expr_func_first
1993 \let\XINT_iiexpr_func_first\XINT_expr_func_first
1994 \def\XINT_expr_func_last #1#2#3% will not work in \xintNewExpr if macro param involved
1995     {\expandafter #1\expandafter #2\csname.=\expandafter\XINT_expr_func_lasta
1996         \romannumeral-‘0\XINT_expr_unlock #3,^\endcsname }%
1997 \def\XINT_expr_func_lasta #1,#2%
1998     {\if ^#2 #1\expandafter\xint_gobble_ii\fi \XINT_expr_func_lasta #2}%
1999 \let\XINT_flexpr_func_last\XINT_expr_func_last
2000 \let\XINT_iiexpr_func_last\XINT_expr_func_last
2001 \def\XINT_expr_func_odd #1#2#3%
2002     {\expandafter #1\expandafter #2\csname.=\xintOdd{\XINT_expr_unlock #3}\endcsname}%
2003 \let\XINT_flexpr_func_odd\XINT_expr_func_odd
2004 \def\XINT_iiexpr_func_odd #1#2#3%
2005     {\expandafter #1\expandafter #2\csname.=\xintiiOdd{\XINT_expr_unlock #3}\endcsname}%
2006 \def\XINT_expr_func_even #1#2#3%
2007     {\expandafter #1\expandafter #2\csname.=\xintEven{\XINT_expr_unlock #3}\endcsname}%
2008 \let\XINT_flexpr_func_even\XINT_expr_func_even
2009 \def\XINT_iiexpr_func_even #1#2#3%
2010     {\expandafter #1\expandafter #2\csname.=\xintiiEven{\XINT_expr_unlock #3}\endcsname}%
2011 \def\XINT_expr_func_nuple #1#2#3%
2012     {\expandafter #1\expandafter #2\csname .=\XINT_expr_unlock #3\endcsname }%
2013 \let\XINT_flexpr_func_nuple\XINT_expr_func_nuple
2014 \let\XINT_iiexpr_func_nuple\XINT_expr_func_nuple
2015 \def\XINT_expr_func_reversed #1#2#3%
2016     {\expandafter #1\expandafter #2\csname .=\xintReversed::csv
2017         {\XINT_expr_unlock #3}\endcsname }%
2018 \let\XINT_flexpr_func_reversed\XINT_expr_func_reversed
2019 \let\XINT_iiexpr_func_reversed\XINT_expr_func_reversed
2020 \def\xintReversed::csv #1% should be done directly, of course
2021     {\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.

```

2022 \def\xintSeqB:f:csv #1#2%
2023     {\expandafter\XINT_seqb:f:csv \expandafter{\romannumeral0\xintra{#2}}{#1}}%
2024 \def\XINT_seqb:f:csv #1#2{\expandafter\XINT_seqb:f:csv_a\romannumeral-‘0#2#1!}%
2025 \def\XINT_seqb:f:csv_a #1#2;#3;#4!{%
2026     \expandafter\xint_gobble_i\romannumeral-‘0%
2027     \xintifCmp {#3}{#4}\XINT_seqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_seqb:f:csv_bg
2028     #1{#3}{#4}{#2}}%
2029 \def\XINT_seqb:f:csv_be #1#2#3#4#5{,#2}%
2030 \def\XINT_seqb:f:csv_bl #1{\if #1p\expandafter\XINT_seqb:f:csv_pa\else

```

```

2031 \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2032 \def\XINT_seqb:f:csv_pa #1#2#3#4{\expandafter\XINT_seqb:f:csv_p\expandafter
2033 \romannumeral0\xintadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2034 \def\XINT_seqb:f:csv_p #1#2%
2035 {%
2036 \xintifCmp {#1}{#2}\XINT_seqb:f:csv_pa\XINT_seqb:f:csv_pb\XINT_seqb:f:csv_pc
2037 {#1}{#2}%
2038 }%
2039 \def\XINT_seqb:f:csv_pb #1#2#3#4{#3,#1}%
2040 \def\XINT_seqb:f:csv_pc #1#2#3#4{#3}%
2041 \def\XINT_seqb:f:csv_bg #1{\if #1n\expandafter\XINT_seqb:f:csv_na\else
2042 \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2043 \def\XINT_seqb:f:csv_na #1#2#3#4{\expandafter\XINT_seqb:f:csv_n\expandafter
2044 \romannumeral0\xintadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2045 \def\XINT_seqb:f:csv_n #1#2%
2046 {%
2047 \xintifCmp {#1}{#2}\XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_seqb:f:csv_na
2048 {#1}{#2}%
2049 }%
2050 \def\XINT_seqb:f:csv_nb #1#2#3#4{#3,#1}%
2051 \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.

```

2052 \def\xintiiSeqb:f:csv #1#2%
2053 {\expandafter\XINT_iiseqb:f:csv \expandafter\romannumeral-‘0#2}{#1}}%
2054 \def\XINT_iiseqb:f:csv #1#2{\expandafter\XINT_iiseqb:f:csv_a\romannumeral-‘0#2#1!}%
2055 \def\XINT_iiseqb:f:csv_a #1#2;#3;#4!{%
2056 \expandafter\xint_gobble_i\romannumeral-‘0%
2057 \xintSgnFork{\XINT_Cmp {#3}{#4}}%
2058 \XINT_iiseqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_iiseqb:f:csv_bg
2059 #1{#3}{#4}}{#2}}%
2060 \def\XINT_iiseqb:f:csv_bl #1{\if #1p\expandafter\XINT_iiseqb:f:csv_pa\else
2061 \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2062 \def\XINT_iiseqb:f:csv_pa #1#2#3#4{\expandafter\XINT_iiseqb:f:csv_p\expandafter
2063 \romannumeral0\xintiiadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2064 \def\XINT_iiseqb:f:csv_p #1#2%
2065 {%
2066 \xintSgnFork{\XINT_Cmp {#1}{#2}}%
2067 \XINT_iiseqb:f:csv_pa\XINT_iiseqb:f:csv_pb\XINT_iiseqb:f:csv_pc {#1}{#2}%
2068 }%
2069 \def\XINT_iiseqb:f:csv_pb #1#2#3#4{#3,#1}%
2070 \def\XINT_iiseqb:f:csv_pc #1#2#3#4{#3}%
2071 \def\XINT_iiseqb:f:csv_bg #1{\if #1n\expandafter\XINT_iiseqb:f:csv_na\else
2072 \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2073 \def\XINT_iiseqb:f:csv_na #1#2#3#4{\expandafter\XINT_iiseqb:f:csv_n\expandafter
2074 \romannumeral0\xintiiadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2075 \def\XINT_iiseqb:f:csv_n #1#2%
2076 {%
2077 \xintSgnFork{\XINT_Cmp {#1}{#2}}%

```

```
2078 \XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_iiseqb:f:csv_na {#1}{#2}%
2079 }%
```

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.

```
2080 \def\XINTinFloatSeqB:f:csv #1#2{\expandafter\XINT_flseqb:f:csv \expandafter
2081   {\romannumeral0\XINTinfloat [\XINTdigits]{#2}}{#1}}%
2082 \def\XINT_flseqb:f:csv #1#2{\expandafter\XINT_flseqb:f:csv_a\romannumeral-‘0#2#1!}%
2083 \def\XINT_flseqb:f:csv_a #1#2;#3;#4!{%
2084   \expandafter\xint_gobble_i\romannumeral-‘0%
2085   \xintifCmp {#3}{#4}\XINT_flseqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_flseqb:f:csv_bg
2086   #1{#3}{#4}}{#2}}%
2087 \def\XINT_flseqb:f:csv_bl #1{\if #1p\expandafter\XINT_flseqb:f:csv_pa\else
2088   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2089 \def\XINT_flseqb:f:csv_pa #1#2#3#4{\expandafter\XINT_flseqb:f:csv_p\expandafter
2090   {\romannumeral0\XINTinfloatadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2091 \def\XINT_flseqb:f:csv_p #1#2%
2092 {%
2093   \xintifCmp {#1}{#2}%
2094   \XINT_flseqb:f:csv_pa\XINT_flseqb:f:csv_pb\XINT_flseqb:f:csv_pc {#1}{#2}%
2095 }%
2096 \def\XINT_flseqb:f:csv_pb #1#2#3#4{#3,#1}%
2097 \def\XINT_flseqb:f:csv_pc #1#2#3#4{#3}%
2098 \def\XINT_flseqb:f:csv_bg #1{\if #1n\expandafter\XINT_flseqb:f:csv_na\else
2099   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2100 \def\XINT_flseqb:f:csv_na #1#2#3#4{\expandafter\XINT_flseqb:f:csv_n\expandafter
2101   {\romannumeral0\XINTinfloatadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2102 \def\XINT_flseqb:f:csv_n #1#2%
2103 {%
2104   \xintifCmp {#1}{#2}%
2105   \XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_flseqb:f:csv_na {#1}{#2}%
2106 }%
```

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

10.39.1 \xintApply::csv

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

```
2107 \def\xintApply::csv #1#2%
2108   {\expandafter\XINT_applyon::_a\expandafter {\romannumeral-‘0#2}{#1}}%
2109 \def\XINT_applyon::_a #1#2{\XINT_applyon::_b {#2}}{#1,,}%
2110 \def\XINT_applyon::_b #1#2#3,{\expandafter\XINT_applyon::_c \romannumeral-‘0#3,{#1}{#2}}%
2111 \def\XINT_applyon::_c #1{\if #1,\expandafter\XINT_applyon::_end
2112   \else\expandafter\XINT_applyon::_d\fi #1}%
2113 \def\XINT_applyon::_d #1,#2{\expandafter\XINT_applyon::_e\romannumeral-‘0#2{#1},{#2}}%
2114 \def\XINT_applyon::_e #1,#2#3{\XINT_applyon::_b {#2}{#3, #1}}%
2115 \def\XINT_applyon::_end #1,#2#3{\xint_secondoftwo #3}%

```

10.39.2 \xintApply:::csv

```

2116 \def\xintApply:::csv #1#2#3%
2117   {\expandafter\xint_applyon:::_a\expandafter{\romannumeral-‘0#2}{#1}{#3}}%
2118 \def\xint_applyon:::_a #1#2#3{\xint_applyon:::_b {#2}{#3}{#1},,%}
2119 \def\xint_applyon:::_b #1#2#3#4,%
2120   {\expandafter\xint_applyon:::_c \romannumeral-‘0#4,{#1}{#2}{#3}}%
2121 \def\xint_applyon:::_c #1{\if #1,\expandafter\xint_applyon:::_end
2122   \else\expandafter\xint_applyon:::_d\fi #1}%
2123 \def\xint_applyon:::_d #1,#2#3%
2124   {\expandafter\xint_applyon:::_e\expandafter
2125   {\romannumeral-‘0\xintApply:::csv {#2}{#1}{#3}},{#2}{#3}}%
2126 \def\xint_applyon:::_e #1,#2#3#4{\xint_applyon:::_b {#2}{#3}{#4}, #1}%
2127 \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

```

2128 \def\XINT_expr_RApply:::csv #1#2#3#4%
2129   {\xintApply:::csv{\expandafter#1\xint_exchangetwo_keepbraces{#4}}{#3}}%
2130 \def\XINT_expr_LApply:::csv #1#2#3#4{\xintApply:::csv{#1}{#3}}{#4}}%
2131 \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. Lists are a pain.

```

2132 \catcode‘~ 12 % by the way, catcode is set to 3 in \XINTsetupcatcodes
2133 \catcode‘$ 12 % $
2134 \def\XINT_xptwo_getab_b #1#2!#3%
2135   {\expandafter\xint_xptwo_getab_c\romannumeral-‘0#3!#1{#1#2}}%
2136 \def\XINT_xptwo_getab_c #1#2!#3#4#5#6{#1#3{#5}{#6}{#1#2}{#4}}%
2137 \def\xint_ddfork #1$#2#3\krof {#2}%
2138 \def\XINT_NEfork #1#2{\xint_ddfork
2139   #1#2\XINT_expr_RLApply:::csv
2140   #1$\XINT_expr_RApply:::csv% $
2141   $#2\XINT_expr_LApply:::csv% $
2142   $$\XINT_NEfork_nn #1#2}%
2143   \krof }%
2144 \def\XINT_NEfork_nn #1#2#3#4{%
2145   \if #1##\xint_dothis{#3}\fi
2146   \if #1~\xint_dothis{#3}\fi
2147   \if #2##\xint_dothis{#3}\fi
2148   \if #2~\xint_dothis{#3}\fi
2149   \xint_orthat {\csname #4NE\endcsname }%
2150   }%
2151 \def\XINT_NEfork_one #1#2!#3#4#5#6{%
2152   \if ###1\xint_dothis {#3}\fi
2153   \if ~#1\xint_dothis {#3}\fi
2154   \if $#1\xint_dothis {\xintApply:::csv{#3#5}}\fi %$
2155   \xint_orthat {\csname #4NE\endcsname #6}{#1#2}%
2156 }%
2157 \toks0 {}%
2158 \xintFor #1 in {DivTrunc,iiDivTrunc,iiDivRound,Mod,iiMod,iRound,Round,iTrunc,Trunc,%

```

```

2159      Lt,Gt,Eq,LtorEq,GtorEq,Neq,AND,OR,XOR,iQuo,iRem,Add,Sub,Mul,Div,Pow,E,%
2160      iiAdd,iiSub,iiMul,iiPow,iiQuo,iiRem,iiE,SeqA::csv,iiSeqA::csv}\do
2161 \{\toks0
2162 \expandafter{\the\toks0
2163 \expandafter\let\csname xint#1NE\expandafter\endcsname\csname xint#1\endcsname
2164 \expandafter\def\csname xint#1\endcsname ####1####2{%
2165     \expandafter\XINT_NEfork
2166     \romannumeral-‘0\expandafter\XINT_xptwo_getab_b
2167     \romannumeral-‘0####2!{####1}{~xint#1}{xint#1}}%
2168 }%
2169 }%
2170 \xintFor #1 in {Num,Irr,Abs,iiAbs,Sgn,iiSgn,TFrac,Floor,iFloor,Ceil,iCeil,%
2171     Sqr,iiSqr,iSqrt,iiIsZero,iiIsNotZero,iiifNotZero,iiifSgn,Odd,Even,iiOdd,iiEven,%
2172     Opp,iiOpp,iiifZero,Fac,iFac,Bool,Toggle}\do
2173 \{\toks0
2174 \expandafter{\the\toks0
2175 \expandafter\let\csname xint#1NE\expandafter\endcsname\csname xint#1\endcsname
2176 \expandafter\def\csname xint#1\endcsname ####1{%
2177     \expandafter\XINT_NEfork_one\romannumeral-‘0####1!{~xint#1}{xint#1}{}}}%
2178 }%
2179 }%
2180 \xintFor #1 in {Add,Sub,Mul,Div,Power,E,Mod,SeqA::csv}\do
2181 \{\toks0
2182 \expandafter{\the\toks0
2183 \expandafter\let\csname XINTinFloat#1NE\expandafter\endcsname
2184     \csname XINTinFloat#1\endcsname
2185 \expandafter\def\csname XINTinFloat#1\endcsname ####1####2{%
2186     \expandafter\XINT_NEfork
2187     \romannumeral-‘0\expandafter\XINT_xptwo_getab_b
2188     \romannumeral-‘0####2!{####1}{~XINTinFloat#1}{XINTinFloat#1}}%
2189 }%
2190 }%
2191 \toks0
2192 \expandafter{\the\toks0
2193 \let\xintSeqB::csvNE\xintSeqB::csv
2194 \def\xintSeqB::csv ##1##2{%
2195     \expandafter\XINT_NEfork
2196     \romannumeral-‘0\expandafter\XINT_xptwo_getab_b
2197     \romannumeral-‘0##2!{##1}{$noexpand\xintSeqB:f:csv}{xintSeqB::csv}}%
2198 \let\xintiiSeqB::csvNE\xintiiSeqB::csv
2199 \def\xintiiSeqB::csv ##1##2{%
2200     \expandafter\XINT_NEfork
2201     \romannumeral-‘0\expandafter\XINT_xptwo_getab_b
2202     \romannumeral-‘0##2!{##1}{$noexpand\xintiiSeqB:f:csv}{xintiiSeqB::csv}}%
2203 \let\XINTinFloatSeqB::csvNE\XINTinFloatSeqB::csv
2204 \def\XINTinFloatSeqB::csv ##1##2{%
2205     \expandafter\XINT_NEfork
2206     \romannumeral-‘0\expandafter\XINT_xptwo_getab_b
2207     \romannumeral-‘0##2!{##1}{$noexpand$XINTinFloatSeqB:f:csv}{XINTinFloatSeqB::csv}}%
2208 \let\xintSeq::csvNE\xintSeq::csv
2209 \def\xintSeq::csv ##1##2{%
2210     \expandafter\XINT_NEfork

```



```

2211 \romannumeral-‘0\expandafter\XINT_xptwo_getab_b
2212 \romannumeral-‘0##2!{##1}{$noexpand$xintSeq::csv}{xintSeq::csv}}%
2213 \let\xintiiSeq::csvNE\xintiiSeq::csv
2214 \def\xintiiSeq::csv ##1##2{%
2215 \expandafter\XINT_NEfork
2216 \romannumeral-‘0\expandafter\XINT_xptwo_getab_b
2217 \romannumeral-‘0##2!{##1}{$noexpand$xintiiSeq::csv}{xintiiSeq::csv}}%
2218 \let\XINTinFloatSeq::csvNE\XINTinFloatSeq::csv
2219 \def\XINTinFloatSeq::csv ##1##2{%
2220 \expandafter\XINT_NEfork
2221 \romannumeral-‘0\expandafter\XINT_xptwo_getab_b
2222 \romannumeral-‘0##2!{##1}{$noexpand$XINTinFloatSeq::csv}{XINTinFloatSeq::csv}}%
2223 \let\XINTinFloatFracNE\XINTinFloatFrac
2224 \def\XINTinFloatFrac ##1{\expandafter\XINT_NEfork_one\romannumeral-‘0##1!%
2225 {~XINTinFloatFrac}{XINTinFloatFrac}}{}}}%
2226 \let\XINTinFloatdigitsNE\XINTinFloatdigits
2227 \def\XINTinFloatdigits ##1{\expandafter\XINT_NEfork_one\romannumeral-‘0##1!%
2228 {~XINTinFloatdigits}{XINTinFloatdigits}}{}}}%
2229 \let\XINTinFloatSqrtdigitsNE\XINTinFloatSqrtdigits
2230 \def\XINTinFloatSqrtdigits ##1{\expandafter\XINT_NEfork_one\romannumeral-‘0##1!%
2231 {~XINTinFloatSqrtdigits}{XINTinFloatSqrtdigits}}{}}}%
2232 \let\XINTinFloatNE\XINTinFloat
2233 \def\XINTinFloat [##1]##2{% not ultimately general, but got tired
2234 \expandafter\XINT_NEfork_one
2235 \romannumeral-‘0##2!{~XINTinFloat[##1]}{XINTinFloat}}{[##1]}}%
2236 \let\XINTinFloatSqrtNE\XINTinFloatSqrt
2237 \def\XINTinFloatSqrt [##1]##2{%
2238 \expandafter\XINT_NEfork_one
2239 \romannumeral-‘0##2!{~XINTinFloatSqrt[##1]}{XINTinFloatSqrt}}{[##1]}}%
2240 }%
2241 \xintFor #1 in {ANDof,ORof,XORof,iiMaxof,iiMinof,iiSum,iiPrd,
2242 GCDof,LCMof,Sum,Prd,Maxof,Minof}\do
2243 {\toks0
2244 \expandafter{\the\toks0 \expandafter\def\csname xint#1:csv\endcsname {~xint#1:csv}}%
2245 }%
2246 \xintFor #1 in {Maxof,Minof,Sum,Prd}\do
2247 {\toks0
2248 \expandafter{\the\toks0
2249 \expandafter\def\csname XINTinFloat#1:csv\endcsname {~XINTinFloat#1:csv}}%
2250 }%
2251 \expandafter\def\expandafter\XINT_expr_redefinmacros\expandafter
2252 {\the\toks0
2253 \def\XINT_flexpr_noopt {\expandafter\XINT_flexpr_withopt_b\expandafter-%
2254 \romannumeral0\xintbarefloateval }%
2255 \def\XINT_flexpr_withopt_b ##1##2%
2256 {\expandafter\XINT_flexpr_wrap\csname .;##1.=\XINT_expr_unlock ##2\endcsname }%
2257 \def\XINT_expr_unlock_sp ##1.;##2##3.==##4!{\if -##2\expandafter\xint_firstoftwo
2258 \else\expandafter\xint_secondoftwo\fi \XINTdigits{{##2##3}}{{##4}}}%
2259 \def\XINT_expr_print ##1{\expandafter\xintSPRaw::csv\expandafter
2260 {\romannumeral-‘0\XINT_expr_unlock ##1}}%
2261 \def\XINT_iiexpr_print ##1{\expandafter\xintCSV::csv\expandafter
2262 {\romannumeral-‘0\XINT_expr_unlock ##1}}%

```



```

2263 \def\XINT_boolexpr_print #1{\expandafter\xintIsTrue::csv\expandafter
2264      {\romannumeral-'0\XINT_expr_unlock #1}}%
2265 \def\xintCSV::csv {\xintCSV::csv}% spaces to separate from possible catcode 11
2266 \def\xintSPRaw::csv {\xintSPRaw::csv}% stuff after
2267 \def\xintPFloat::csv {\xintPFloat::csv}%
2268 \def\xintIsTrue::csv {\xintIsTrue::csv}%
2269 \def\xintRound::csv {\xintRound::csv}%
2270 % \def\XINTinFloat::csv {\XINTinFloat::csv}% should not be needed.
2271 \def\xintReversed::csv {\xintReversed::csv}%
2272 \def\xintListSel:csv {\xintListSel:csv}%
2273}%
2274\toks0 {}%
2275\def\xintNewExpr {\xint_NewExpr\xinttheexpr}%
2276\def\xintNewFloatExpr {\xint_NewExpr\xintthefloatexpr}%
2277\def\xintNewIExpr {\xint_NewExpr\xinttheiexpr}%
2278% \let\xintNewNumExpr\xintNewIExpr % made obsolete for 1.1 release
2279\def\xintNewIIExpr {\xint_NewExpr\xinttheiiexpr}%
2280\def\xintNewBoolExpr {\xint_NewExpr\xinttheboolexpr}%
2281\def\XINT_newexpr_finish #1>{\noexpand\romannumeral-'0}%
2282\def\xint_NewExpr #1#2[#3]%
2283{%
2284 \begingroup
2285 \ifcase #3\relax
2286 \toks0 {\xdef #2}%
2287 \or \toks0 {\xdef #2##1}%
2288 \or \toks0 {\xdef #2##1##2}%
2289 \or \toks0 {\xdef #2##1##2##3}%
2290 \or \toks0 {\xdef #2##1##2##3##4}%
2291 \or \toks0 {\xdef #2##1##2##3##4##5}%
2292 \or \toks0 {\xdef #2##1##2##3##4##5##6}%
2293 \or \toks0 {\xdef #2##1##2##3##4##5##6##7}%
2294 \or \toks0 {\xdef #2##1##2##3##4##5##6##7##8}%
2295 \or \toks0 {\xdef #2##1##2##3##4##5##6##7##8##9}%
2296 \fi
2297 \xintexprSafeCatcodes
2298 \XINT_NewExpr #1%
2299}%
2300\catcode'\~ 13 \catcode'\@ 14 \catcode'\% 6 \catcode'\# 12 \catcode'\$ 11 @ $
2301\def\XINT_NewExpr %1%2@
2302{@
2303 \def\XINT_tmpa %%1%%2%%3%%4%%5%%6%%7%%8%%9{%2}@
2304 \XINT_expr_redefinemasros
2305 \def~{$noexpand$}@ vvv vérifier si vraiment besoin pour ^ et !
2306 \catcode': 11 \catcode'_ 11 @ \catcode'^ 11 \catcode'! 11
2307 \catcode'# 12 \catcode'~ 13 \escapechar 126
2308 \endlinechar -1 \everyeof {\noexpand }@
2309 \edef\XINT_tmph
2310 {\scantokens\expandafter
2311 {\romannumeral-'0\expandafter%1\XINT_tmpa {#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}\relax}@
2312 }@
2313 \escapechar 92 \catcode'# 6 \catcode'$ 0 @ $
2314 \the\toks0

```

```

2315     {\scantokens\expandafter{\expandafter\XINT_newexpr_finish\meaning\XINT_tmpb}}@
2316 \endgroup
2317 }@
2318 \catcode'% 14
2319 \let\xintexprRestoreCatcodes\empty
2320 \def\xintexprSafeCatcodes
2321 {%
2322     \edef\xintexprRestoreCatcodes {%
2323         \catcode59=\the\catcode59 % ;
2324         \catcode34=\the\catcode34 % "
2325         \catcode63=\the\catcode63 % ?
2326         \catcode124=\the\catcode124 % |
2327         \catcode38=\the\catcode38 % &
2328         \catcode33=\the\catcode33 % !
2329         \catcode93=\the\catcode93 % ]
2330         \catcode91=\the\catcode91 % [
2331         \catcode94=\the\catcode94 % ^
2332         \catcode95=\the\catcode95 % _
2333         \catcode47=\the\catcode47 % /
2334         \catcode41=\the\catcode41 % )
2335         \catcode40=\the\catcode40 % (
2336         \catcode42=\the\catcode42 % *
2337         \catcode43=\the\catcode43 % +
2338         \catcode62=\the\catcode62 % >
2339         \catcode60=\the\catcode60 % <
2340         \catcode58=\the\catcode58 % :
2341         \catcode46=\the\catcode46 % .
2342         \catcode45=\the\catcode45 % -
2343         \catcode44=\the\catcode44 % ,
2344         \catcode61=\the\catcode61 % =
2345         \catcode32=\the\catcode32\relax % space
2346     }%
2347     \catcode59=12 % ;
2348     \catcode34=12 % "
2349     \catcode63=12 % ?
2350     \catcode124=12 % |
2351     \catcode38=4 % &
2352     \catcode33=12 % !
2353     \catcode93=12 % ]
2354     \catcode91=12 % [
2355     \catcode94=7 % ^
2356     \catcode95=8 % _
2357     \catcode47=12 % /
2358     \catcode41=12 % )
2359     \catcode40=12 % (
2360     \catcode42=12 % *
2361     \catcode43=12 % +
2362     \catcode62=12 % >
2363     \catcode60=12 % <
2364     \catcode58=12 % :
2365     \catcode46=12 % .
2366     \catcode45=12 % -

```

Package `xintexpr` implementation

```
2367      \catcode44=12 % ,
2368      \catcode61=12 % =
2369      \catcode32=10 % space
2370 }%
2371 \let\XINT_tmpa\relax \let\XINT_tmpb\relax \let\XINT_tmppc\relax
2372 \XINT_restorecatcodes_endinput%

xintkernel: 224. Total number of code lines: 12264. Each package starts with
xinttools:1049. circa 50 lines dealing with catcodes, package identification
xintcore:2074. and reloading management, also for Plain TeX. Version 1.1 of
xint:1493. 2014/10/28.
xintbinhex: 609.
xintgcd: 437.
xintfrac:2599.
xintseries: 386.
xintcfac:1021.
xintexpr:2372.
```