

# The `xint` source code

JEAN-FRANÇOIS BURNOL

jfbu (at) free (dot) fr

Package version: 1.2f (2016/03/12); documentation date: 2016/03/12.

From source file xint.dtx. Time-stamp: <12-03-2016 at 21:52:32 CET>.

## Contents

1	Package <code>xintkernel</code> implementation . . . . .	2
2	Package <code>xinttools</code> implementation . . . . .	10
3	Package <code>xintcore</code> implementation . . . . .	38
4	Package <code>xint</code> implementation . . . . .	90
5	Package <code>xintbinhex</code> implementation . . . . .	134
6	Package <code>xintgcd</code> implementation . . . . .	148
7	Package <code>xintfrac</code> implementation . . . . .	160
8	Package <code>xintseries</code> implementation . . . . .	234
9	Package <code>xintcfrac</code> implementation . . . . .	243
10	Package <code>xintexpr</code> implementation . . . . .	265

This is 1.2f of 2016/03/12.

- Release 1.2 of 2015/10/10 has entirely rewritten the core arithmetic routines in `xintcore`. Many macros benefit indirectly from the faster core routines. The new model is yet to be extended to other portions of the code: for example the routines of `xintbinhex` could be made faster for very big inputs if they adopted some techniques from the implementation of the basic arithmetic routines.

The parser of `xintexpr` is also faster at gathering digits and does not have a limit at 5000 digits per number anymore.

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

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

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

# 1 Package *xintkernel* implementation

.1	Catcodes, $\varepsilon$ -TeX and reload detection . . .	2	.8	<code>\xintdothis, \xintorthat</code> . . . . .	7
.2	Package identification . . . . .	5	.9	<code>\xint_zapspaces</code> . . . . .	7
.3	Constants . . . . .	5	.10	<code>\odef, \oodef, \fdef</code> . . . . .	8
.4	Token management utilities . . . . .	5	.11	<code>\xintReverseOrder</code> . . . . .	8
.5	“gob til” macros and UD style fork . . .	6	.12	<code>\xintLength</code> . . . . .	8
.6	<code>\xint_afterfi</code> . . . . .	7	.13	<code>\xintMessage, \ifxintverbose</code> . . . .	9
.7	<code>\xint_bye</code> . . . . .	7			

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

First appeared as a separate package with release 1.1.

## 1.1 Catcodes, $\varepsilon$ -TeX and reload detection

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

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

Starting with version 1.06 of the package, also ``` must be catcode-protected, because we replace everywhere in the code the twice-expansion done with `\expandafter` by the systematic use of `\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 in *xintkernel.sty*) and re-used in other modules. All catcode changes have been unified and `\XINT_storecatcodes` will be used by each module to redefine `\XINT_restorecatcodes_endinput` in case catcodes have changed in-between the loading of *xint.sty* (now *xintkernel.sty*) and the module (not very probable but...).

```

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 \let\z\relax
19 \expandafter
20  \ifx\csname numexpr\endcsname\relax
21    \y{xintkernel}{\numexpr not available, aborting input}%
22  \def\z{\endgroup\endinput}%

```

# 1 Package *xintkernel* implementation

```

23  \else
24  \expandafter
25  \ifx\csname XINTsetupcatcodes\endcsname\relax
26  \else
27  \y{xintkernel}{I was already loaded, aborting input}%
28  \def\z{\endgroup\endinput}%
29  \fi
30  \fi
31  \ifx\z\relax\else\expandafter\z\fi%
32  \def\PrepareCatcodes
33  {%
34  \endgroup
35  \def\XINT_restorecatcodes
36  {% takes care of all, to allow more economical code in modules
37  \catcode0=\the\catcode0 %
38  \catcode59=\the\catcode59 % ; xintexpr
39  \catcode126=\the\catcode126 % ~ xintexpr
40  \catcode39=\the\catcode39 % ' xintexpr
41  \catcode34=\the\catcode34 % " xintbinhex, and xintexpr
42  \catcode63=\the\catcode63 % ? xintexpr
43  \catcode124=\the\catcode124 % | xintexpr
44  \catcode38=\the\catcode38 % & xintexpr
45  \catcode64=\the\catcode64 % @ xintexpr
46  \catcode33=\the\catcode33 % ! xintexpr
47  \catcode93=\the\catcode93 % ] -, xintfrac, xintseries, xintcfrac
48  \catcode91=\the\catcode91 % [ -, xintfrac, xintseries, xintcfrac
49  \catcode36=\the\catcode36 % $ xintgcd only
50  \catcode94=\the\catcode94 % ^
51  \catcode96=\the\catcode96 % `
52  \catcode47=\the\catcode47 % /
53  \catcode41=\the\catcode41 % )
54  \catcode40=\the\catcode40 % (
55  \catcode42=\the\catcode42 % *
56  \catcode43=\the\catcode43 % +
57  \catcode62=\the\catcode62 % >
58  \catcode60=\the\catcode60 % <
59  \catcode58=\the\catcode58 % :
60  \catcode46=\the\catcode46 % .
61  \catcode45=\the\catcode45 % -
62  \catcode44=\the\catcode44 % ,
63  \catcode35=\the\catcode35 % #
64  \catcode95=\the\catcode95 % _
65  \catcode125=\the\catcode125 % }
66  \catcode123=\the\catcode123 % {
67  \endlinechar=\the\endlinechar
68  \catcode13=\the\catcode13 % ^^M
69  \catcode32=\the\catcode32 %
70  \catcode61=\the\catcode61\relax % =
71  }%
72  \edef\XINT_restorecatcodes_endinput
73  {%
74  \XINT_restorecatcodes\noexpand\endinput %

```

# 1 Package *xintkernel* implementation

```

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

Other modules could possibly be loaded under a different catcode regime.
116 \def\XINTsetupcatcodes {% for use by other modules
117     \edef\XINT_restorecatcodes_endinput
118     {%
119         \XINT_restorecatcodes\noexpand\endinput %
120     }%
121     \XINT_setcatcodes
122 }%
```

## 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 HÖ for some reason escaping me (compatibility with LaTeX 2.09 or other things ??) seems to set extra precautions.

1.09c uses e-TeX `\ifdefined`.

```

123 \ifdefined\ProvidesPackage
124   \let\XINT_providespackage\relax
125 \else
126   \def\XINT_providespackage #1#2[#3]%
127       {\immediate\write-1{Package: #2 #3}%
128        \expandafter\xdef\csname ver@#2.sty\endcsname{#3}}%
129 \fi
130 \XINT_providespackage
131 \ProvidesPackage {xintkernel}%
132 [2016/03/12 1.2f Paraphernalia for the xint packages (JFB)]%
```

## 1.3 Constants

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

```

133 \chardef\xint_c_      0
134 \chardef\xint_c_i     1
135 \chardef\xint_c_ii    2
136 \chardef\xint_c_iii   3
137 \chardef\xint_c_iv    4
138 \chardef\xint_c_v     5
139 \chardef\xint_c_vi    6
140 \chardef\xint_c_vii   7
141 \chardef\xint_c_viii  8
142 \chardef\xint_c_ix    9
143 \chardef\xint_c_x     10
144 \chardef\xint_c_xiv   14
145 \chardef\xint_c_xvi   16
146 \chardef\xint_c_xviii 18
147 \chardef\xint_c_xxii  22
148 \chardef\xint_c_ii^v  32
149 \chardef\xint_c_ii^vi 64
150 \chardef\xint_c_ii^vii 128
151 \mathchardef\xint_c_ii^viii 256
152 \mathchardef\xint_c_ii^xii 4096
153 \mathchardef\xint_c_x^iv 10000
```

## 1.4 Token management utilities

```

154 \def\XINT_tmpa { }%
155 \ifx\XINT_tmpa\space\else
156   \immediate\write-1{Package xintkernel Warning: ATTENTION!}%
157   \immediate\write-1{\string\space\XINT_tmpa macro does not have its normal
158     meaning.}%
159   \immediate\write-1{\XINT_tmpa\XINT_tmpa\XINT_tmpa\XINT_tmpa
160     All kinds of catastrophes will ensue!!!!}%

```

## 1 Package *xintkernel* implementation

```
161 \fi
162 \def\XINT_tmpb {}%
163 \ifx\XINT_tmpb\empty\else
164   \immediate\write-1{Package xintkernel Warning: ATTENTION!}%
165   \immediate\write-1{\string\empty\XINT_tmpa macro does not have its normal
166     meaning.}%
167   \immediate\write-1{\XINT_tmpa\XINT_tmpa\XINT_tmpa\XINT_tmpa
168     All kinds of catastrophes will ensue!!!!}%
169 \fi
170 \let\XINT_tmpa\relax \let\XINT_tmpb\relax
171 \ifdefined\space\else\def\space { }\fi
172 \ifdefined\empty\else\def\empty {} \fi
173 \long\def\xint_gobble_   {}%
174 \long\def\xint_gobble_i   #1{}%
175 \long\def\xint_gobble_ii  #1#2{}%
176 \long\def\xint_gobble_iii #1#2#3{}%
177 \long\def\xint_gobble_iv  #1#2#3#4{}%
178 \long\def\xint_gobble_v   #1#2#3#4#5{}%
179 \long\def\xint_gobble_vi   #1#2#3#4#5#6{}%
180 \long\def\xint_gobble_vii  #1#2#3#4#5#6#7{}%
181 \long\def\xint_gobble_viii #1#2#3#4#5#6#7#8{}%
182 \long\def\xint_firstofone  #1{#1}%
183 \long\def\xint_firstoftwo  #1#2{#1}%
184 \long\def\xint_secondoftwo #1#2{#2}%
185 \long\def\xint_firstofone_thenstop #1{ #1}%
186 \long\def\xint_firstoftwo_thenstop #1#2{ #1}%
187 \long\def\xint_secondoftwo_thenstop #1#2{ #2}%
188 \def\xint_minus_thenstop { -}%
189 \def\xint_exchangetwo_keepbraces #1#2{{#2}{#1}}%
```

### 1.5 “gob til” macros and UD style fork

Some moved here from *xintcore* by release 1.2.

```
190 \long\def\xint_gob_til_R #1\R {}%
191 \long\def\xint_gob_til_W #1\W {}%
192 \long\def\xint_gob_til_Z #1\Z {}%
193 \def\xint_gob_til_zero #10{}%
194 \def\xint_gob_til_one  #11{}%
195 \def\xint_gob_til_zeros_iii  #1000{}%
196 \def\xint_gob_til_zeros_iv   #10000{}%
197 \def\xint_gob_til_eightzeroes #100000000{}%
198 \def\xint_gob_til_exclam #1!{}% catcode 12 exclam
199 \def\xint_gob_til_dot    #1.{}%
200 \def\xint_gob_til_G      #1G{}%
201 \def\xint_gob_til_minus  #1-{}%
202 \def\xint_gob_til_relax  #1\relax {}%
203 \def\xint_UDzerominusfork #10-#2#3\krof {#2}%
204 \def\xint_UDzerofork      #10#2#3\krof {#2}%
205 \def\xint_UDsignfork      #1-#2#3\krof {#2}%
206 \def\xint_UDwfork         #1\W#2#3\krof {#2}%
207 \def\xint_UDXINTWfork     #1\XINT_W#2#3\krof {#2}%
208 \def\xint_UDzerosfork     #100#2#3\krof {#2}%
209 \def\xint_UDonezerofork   #110#2#3\krof {#2}%

```

## 1 Package *xintkernel* implementation

```
210 \def\xint_UDsignsfork      #1--#2#3\krof {#2}%
211 \let\xint_relax\relax
212 \def\xint_brelax {\xint_relax}%
213 \long\def\xint_gob_til_xint_relax #1\xint_relax {}%
```

### 1.6 `\xint_afterfi`

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

### 1.7 `\xint_bye`

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

### 1.8 `\xintdothis`, `\xintorthat`

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

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

### 1.9 `\xint_zapspaces`

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

`\xint_zapspaces foo<space>\xint_gobble_i`

Will remove some brace pairs (but not spaces inside them). By the way the `\zap@spaces` of LaTeX2e handles unexpectedly things such as `\zap@spaces 1 {22} 3 4 \@empty` (spaces are not all removed). This does not happen with `\xint_zapspaces`.

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

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

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

expands to 12, and not 12\relax. Imagine also:

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

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

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

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

```

## 1.10 `\odef`, `\oodef`, `\fdef`

May be prefixed with `\global`. No parameter text.

```

222 \def\xintodef #1{\expandafter\def\expandafter#1\expandafter }%
223 \def\xintoodef #1{\expandafter\expandafter\expandafter\def
224         \expandafter\expandafter\expandafter#1%
225         \expandafter\expandafter\expandafter }%
226 \def\xintfdef #1#2{\expandafter\def\expandafter#1\expandafter
227         {\romannumeral`&&@#2}}%
228 \ifdefined\odef\else\let\odef\xintodef\fi
229 \ifdefined\oodef\else\let\oodef\xintoodef\fi
230 \ifdefined\fdef\else\let\fdef\xintfdef\fi

```

## 1.11 `\xintReverseOrder`

`\xintReverseOrder`: does NOT expand its argument. Thus one must use some `\expandafter` if argument is a macro. A faster reverse, but only applicable to (many) digit tokens has been provided with `\csh{xintReverseDigits}` from 1.2 *xintcore*.

```

231 \def\xintReverseOrder {\romannumeral0\xintreverseorder }%
232 \long\def\xintreverseorder #1%
233 {%
234     \XINT_rord_main {}#1%
235     \xint_relax
236     \xint_bye\xint_bye\xint_bye\xint_bye
237     \xint_bye\xint_bye\xint_bye\xint_bye
238     \xint_relax
239 }%
240 \long\def\XINT_rord_main #1#2#3#4#5#6#7#8#9%
241 {%
242     \xint_bye #9\XINT_rord_cleanup\xint_bye
243     \XINT_rord_main {#9#8#7#6#5#4#3#2#1}%
244 }%
245 \long\edef\XINT_rord_cleanup\xint_bye\XINT_rord_main #1#2\xint_relax
246 {%
247     \noexpand\expandafter\space\noexpand\xint_gob_til_xint_relax #1%
248 }%

```

## 1.12 `\xintLength`

`\xintLength` does NOT expand its argument.

```

249 \def\xintLength {\romannumeral0\xintlenth }%
250 \long\def\xintlenth #1%
251 {%
252     \XINT_length_loop
253     0.#1\xint_relax\xint_relax\xint_relax\xint_relax
254     \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
255 }%
256 \long\def\XINT_length_loop #1.#2#3#4#5#6#7#8#9%
257 {%
258     \xint_gob_til_xint_relax #9\XINT_length_finish_a\xint_relax

```



## 1 Package *xintkernel* implementation

```
259 \expandafter\XINT_length_loop\the\numexpr #1+\xint_c_viii.%
260 }%
261 \def\XINT_length_finish_a\xint_relax\expandafter\XINT_length_loop
262 \the\numexpr #1+\xint_c_viii.#2\xint_bye
263 {%
264 \XINT_length_finish_b #2\W\W\W\W\W\W\W\Z {#1}%
265 }%
266 \def\XINT_length_finish_b #1#2#3#4#5#6#7#8\Z
267 {%
268 \xint_gob_til_W
269 #1\XINT_length_finish_c \xint_c_
270 #2\XINT_length_finish_c \xint_c_i
271 #3\XINT_length_finish_c \xint_c_ii
272 #4\XINT_length_finish_c \xint_c_iii
273 #5\XINT_length_finish_c \xint_c_iv
274 #6\XINT_length_finish_c \xint_c_v
275 #7\XINT_length_finish_c \xint_c_vi
276 \W\XINT_length_finish_c \xint_c_vii\Z
277 }%
278 \edef\XINT_length_finish_c #1#2\Z #3%
279 {\noexpand\expandafter\space\noexpand\the\numexpr #3+#1\relax}%
```

### 1.13 *\xintMessage*, *\ifxintverbose*

1.2c added it for use by *\xintdefvar* and *\xintdeffunc* of *xintexpr*. 1.2e uses *\write128* rather than *\write16* for compatibility with future extended range of output streams, in LuaTeX in particular.

```
280 \def\xintMessage #1#2#3{%
281 \immediate\write128{Package #1 #2: (on line \the\inputlineno)}%
282 \immediate\write128{\space\space\space\space#3}%
283 }%
284 \newif\ifxintverbose
285 \XINT_restorecatcodes_endinput%
```

## 2 Package `xinttools` implementation

.1	Catcodes, $\varepsilon$ -T <sub>E</sub> X and reload detection . .	10	.18	<code>\xintSeq</code> . . . . .	24
.2	Package identification . . . . .	11	.19	<code>\xintloop</code> , <code>\xintbreakloop</code> , <code>\xintbreak-</code> <code>loopanddo</code> , <code>\xintloopskiptonext</code> . . .	27
.3	<code>\xintgodef</code> , <code>\xintgoodef</code> , <code>\xintgdef</code> .	11	.20	<code>\xintiloop</code> , <code>\xintiloopindex</code> , <code>\xintouter-</code> <code>iloopindex</code> , <code>\xintbreakiloop</code> , <code>\xint-</code> <code>breakiloopanddo</code> , <code>\xintloopskiptonext</code> , <code>\xintloopskipandredo</code> . . . . .	27
.4	<code>\xintRevWithBraces</code> . . . . .	11	.21	<code>\XINT_xflet</code> . . . . .	27
.5	<code>\xintZapFirstSpaces</code> . . . . .	12	.22	<code>\xintApplyInline</code> . . . . .	28
.6	<code>\xintZapLastSpaces</code> . . . . .	13	.23	<code>\xintFor</code> , <code>\xintFor*</code> , <code>\xintBreakFor</code> , <code>\xintBreakForAndDo</code> . . . . .	29
.7	<code>\xintZapSpaces</code> . . . . .	14	.24	<code>\XINT_forever</code> , <code>\xintintegers</code> , <code>\xintdi-</code> <code>mensions</code> , <code>\xintrationals</code> . . . . .	31
.8	<code>\xintZapSpacesB</code> . . . . .	14	.25	<code>\xintForpair</code> , <code>\xintForthree</code> , <code>\xintFor-</code> <code>four</code> . . . . .	33
.9	<code>\xintCSVtoList</code> , <code>\xintCSVtoListNon-</code> <code>Stripped</code> . . . . .	14	.26	<code>\xintAssign</code> , <code>\xintAssignArray</code> , <code>\xint-</code> <code>DigitsOf</code> . . . . .	34
.10	<code>\xintListWithSep</code> . . . . .	16			
.11	<code>\xintNthElt</code> . . . . .	16			
.12	<code>\xintKeep</code> . . . . .	17			
.13	<code>\xintKeepUnbraced</code> . . . . .	20			
.14	<code>\xintTrim</code> . . . . .	21			
.15	<code>\xintTrimUnbraced</code> . . . . .	23			
.16	<code>\xintApply</code> . . . . .	23			
.17	<code>\xintApplyUnbraced</code> . . . . .	24			

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

### 2.1 Catcodes, $\varepsilon$ -T<sub>E</sub>X 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@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 [2016/03/12 1.2f 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.) 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 &&@#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\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 \xint_gobble_viii
79     #2\XINT_revwbr_finish_c \xint_gobble_vii
80     #3\XINT_revwbr_finish_c \xint_gobble_vi
81     #4\XINT_revwbr_finish_c \xint_gobble_v
82     #5\XINT_revwbr_finish_c \xint_gobble_iv
83     #6\XINT_revwbr_finish_c \xint_gobble_iii
84     #7\XINT_revwbr_finish_c \xint_gobble_ii
85     \R\XINT_revwbr_finish_c \xint_gobble_i\Z
86 }%

```

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

```

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

```

## 2.5 \xintZapFirstSpaces

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

```

88 \def\xintZapFirstSpaces {\romannumeral0\xintzapfirstspaces }%

```

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

```

89 \long\edef\xintzapfirstspaces #1%
90   {\noexpand\XINT_zapbsp_a \space #1\xint_relax \space\space\xint_relax }%
91 \xint_firstofone {\long\edef\XINT_zapbsp_a #1 } %<- space token here
92 {%

```

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`

```
93 \noexpand\xINT_zapbsp_again? #1\noexpand\xint_bye\noexpand\xINT_zapbsp_b #1\space\space
94 }%
95 \long\def\xINT_zapbsp_again? #1{\xint_bye #1\xINT_zapbsp_again }%
96 \xint_firstofone{\def\xINT_zapbsp_again\xINT_zapbsp_b} {\xINT_zapbsp_a }%
97 \long\def\xINT_zapbsp_b #1\xint_relax #2\xint_relax {#1}%
```

## 2.6 `\xintZapLastSpaces`

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

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

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

```
99 \long\edef\xintzaplastspaces #1{\noexpand\xINT_zapbsp_a {} \noexpand\empty#1%
100 \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.

```
101 \xint_firstofone {\long\def\xINT_zapbsp_a #1#2 } %<- second space here
102 { \expandafter\xINT_zapbsp_b\expandafter{#2}{#1}}%
```

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

```
103 \long\def\xINT_zapbsp_b #1#2#3\xint_relax
104 { \XINT_zapbsp_end? #3\xINT_zapbsp_e {#2#1}\empty #3\xint_relax }%
```

When we have been over all possible `<sp><sp>` things, we reach the ending space tokens, and #3 will be a bunch of spaces (possibly none) followed by `\xint_bye`. So the #1 in `_end?` will be `\xint_bye`. In all other cases #1 can not be `\xint_bye` (assuming naturally this token does not arise in original input), hence control falls back to `\XINT_zapbsp_e` which will loop back to `\XINT_zapbsp_a`.

```
105 \long\def\xINT_zapbsp_end? #1{\xint_bye #1\xINT_zapbsp_end }%
```

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

```
106 \long\def\xINT_zapbsp_end\xINT_zapbsp_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.

```
107 \long\edef\xINT_zapbsp_e #1{\noexpand \XINT_zapbsp_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`.

```

108 \def\xintZapSpaces {\romannumeral0\xintzapspaces}%
109 \long\edef\xintzapspaces #1% like \xintZapFirstSpaces.
110      {\noexpand\XINT_zapsp_a \space #1\xint_relax \space\space\xint_relax}%
111 \xint_firstofone {\long\edef\XINT_zapsp_a #1}%
112      {\noexpand\XINT_zapsp_again? #1\noexpand\xint_bye\noexpand\XINT_zapsp_b #1\space\space}%
113 \long\def\XINT_zapsp_again? #1{\xint_bye #1\XINT_zapsp_again}%
114 \xint_firstofone{\def\XINT_zapsp_again\XINT_zapsp_b} {\XINT_zapsp_a}%
115 \xint_firstofone{\def\XINT_zapsp_b} {\XINT_zapsp_c}%
116 \long\edef\XINT_zapsp_c #1\xint_relax #2\xint_relax {\noexpand\XINT_zapsp_a
117      {} \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).

```

118 \def\xintZapSpacesB {\romannumeral0\xintzapspacesb}%
119 \long\def\xintzapspacesb #1{\XINT_zapspb_one? #1\xint_relax\xint_relax
120      \xint_bye\xintzapspaces {#1}}%
121 \long\def\XINT_zapspb_one? #1#2%
122      {\xint_gob_til_xint_relax #1\XINT_zapspb_onlyspaces\xint_relax
123      \xint_gob_til_xint_relax #2\XINT_zapspb_bracedorone\xint_relax
124      \xint_bye {#1}}%
125 \def\XINT_zapspb_onlyspaces\xint_relax
126      \xint_gob_til_xint_relax\xint_relax\XINT_zapspb_bracedorone\xint_relax
127      \xint_bye #1\xint_bye\xintzapspaces #2{ }%
128 \long\def\XINT_zapspb_bracedorone\xint_relax
129      \xint_bye #1\xint_relax\xint_bye\xintzapspaces #2{ #1}%

```

## 2.9 `\xintCSVtoList`, `\xintCSVtoListNonStripped`

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

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

```

130 \def\xintCSVtoList {\romannumeral0\xintcsvtolist}%
131 \long\def\xintcsvtolist #1{\expandafter\xintApply
132      \expandafter\xintzapspacesb
133      \expandafter{\romannumeral0\xintcsvtolistnonstripped{#1}}}%
134 \def\xintCSVtoListNoExpand {\romannumeral0\xintcsvtolistnoexpand}%
135 \long\def\xintcsvtolistnoexpand #1{\expandafter\xintApply
136      \expandafter\xintzapspacesb
137      \expandafter{\romannumeral0\xintcsvtolistnonstrippednoexpand{#1}}}%
138 \def\xintCSVtoListNonStripped {\romannumeral0\xintcsvtolistnonstripped}%
139 \def\xintCSVtoListNonStrippedNoExpand

```

## 2 Package *xinttools* implementation

```

140      {\romannumeral0\xintcsvtolistnonstrippednoexpand }%
141 \long\def\xintcsvtolistnonstripped #1%
142 {%
143   \expandafter\XINT_csvtol_loop_a\expandafter
144   {\expandafter}\romannumeral`&&@#1%
145   ,\xint_bye,\xint_bye,\xint_bye,\xint_bye
146   ,\xint_bye,\xint_bye,\xint_bye,\xint_bye,\Z
147 }%
148 \long\def\xintcsvtolistnonstrippednoexpand #1%
149 {%
150   \XINT_csvtol_loop_a
151   {#1,\xint_bye,\xint_bye,\xint_bye,\xint_bye
152   ,\xint_bye,\xint_bye,\xint_bye,\xint_bye,\Z
153 }%
154 \long\def\XINT_csvtol_loop_a #1#2,#3,#4,#5,#6,#7,#8,#9,%
155 {%
156   \xint_bye #9\XINT_csvtol_finish_a\xint_bye
157   \XINT_csvtol_loop_b {#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}}%
158 }%
159 \long\def\XINT_csvtol_loop_b #1#2{\XINT_csvtol_loop_a {#1#2}}%
160 \long\def\XINT_csvtol_finish_a\xint_bye\XINT_csvtol_loop_b #1#2#3\Z
161 {%
162   \XINT_csvtol_finish_b #3\R,\R,\R,\R,\R,\R,\R,\Z #2{#1}%
163 }%

```

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

```

164 \def\XINT_csvtol_finish_b #1,#2,#3,#4,#5,#6,#7,#8\Z
165 {%
166   \xint_gob_til_R
167   #1\expandafter\XINT_csvtol_finish_dviii\xint_gob_til_Z
168   #2\expandafter\XINT_csvtol_finish_dvii \xint_gob_til_Z
169   #3\expandafter\XINT_csvtol_finish_dvi \xint_gob_til_Z
170   #4\expandafter\XINT_csvtol_finish_dv \xint_gob_til_Z
171   #5\expandafter\XINT_csvtol_finish_div \xint_gob_til_Z
172   #6\expandafter\XINT_csvtol_finish_diii \xint_gob_til_Z
173   #7\expandafter\XINT_csvtol_finish_dii \xint_gob_til_Z
174   \R\XINT_csvtol_finish_di \Z
175 }%
176 \long\def\XINT_csvtol_finish_dviii #1#2#3#4#5#6#7#8#9{ #9}%
177 \long\def\XINT_csvtol_finish_dvii #1#2#3#4#5#6#7#8#9{ #9{#1}}%
178 \long\def\XINT_csvtol_finish_dvi #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}}%
179 \long\def\XINT_csvtol_finish_dv #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}}%
180 \long\def\XINT_csvtol_finish_div #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}{#4}}%
181 \long\def\XINT_csvtol_finish_diii #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}{#4}{#5}}%
182 \long\def\XINT_csvtol_finish_dii #1#2#3#4#5#6#7#8#9%
183   { #9{#1}{#2}{#3}{#4}{#5}{#6}}%
184 \long\def\XINT_csvtol_finish_di\Z #1#2#3#4#5#6#7#8#9%
185   { #9{#1}{#2}{#3}{#4}{#5}{#6}{#7}}%

```

## 2.10 `\xintListWithSep`

1.04. `\xintListWithSep {\sep}{a}{b}...{z}` returns a `\sep b \sep ... \sep z`. It f-expands its second argument. The 'sep' may be `\par`'s: the macro `\xintlistwithsep` etc... are all declared long. 'sep' does not have to be a single token. It is not expanded.

```

186 \def\xintListWithSep {\romannumeral0\xintlistwithsep }%
187 \def\xintListWithSepNoExpand {\romannumeral0\xintlistwithsepnoexpand }%
188 \long\def\xintlistwithsep #1#2%
189     {\expandafter\XINT_lws\expandafter {\romannumeral`&&@#2}{#1}}%
190 \long\def\XINT_lws #1#2{\XINT_lws_start {#2}#1\xint_bye }%
191 \long\def\xintlistwithsepnoexpand #1#2{\XINT_lws_start {#1}#2\xint_bye }%
192 \long\def\XINT_lws_start #1#2%
193 {%
194     \xint_bye #2\XINT_lws_dont\xint_bye
195     \XINT_lws_loop_a {#2}{#1}%
196 }%
197 \long\def\XINT_lws_dont\xint_bye\XINT_lws_loop_a #1#2{ }%
198 \long\def\XINT_lws_loop_a #1#2#3%
199 {%
200     \xint_bye #3\XINT_lws_end\xint_bye
201     \XINT_lws_loop_b {#1}{#2#3}{#2}%
202 }%
203 \long\def\XINT_lws_loop_b #1#2{\XINT_lws_loop_a {#1#2}}%
204 \long\def\XINT_lws_end\xint_bye\XINT_lws_loop_b #1#2#3{ #1}%

```

## 2.11 `\xintNthElt`

First included in release 1.06.

`\xintNthElt {i}{stuff}` f-expanding to `{a}{b}...{z}` (or 'tokens' `abcd...z`) returns the *i*th element (one pair of braces removed). The list is first f-expanded. The `\xintNthEltNoExpand` does no expansion of its second argument. Both variants expand the first argument inside `\numexpr`.

With *i* = 0, the number of items is returned. This is different from `\xintLen` which is only for numbers (particularly, it checks the sign) and different from `\xintLength` which does not f-expand its argument.

Negative values return the *|i|*th element from the end. Release 1.09m rewrote the initial bits of the code (which checked the sign of #1 and expanded or not #2), some 'improvements' made earlier in 1.09c were quite sub-efficient. Now uses `\xint_UDzerominusfork`, moved from `xint.sty`.

```

205 \def\xintNthElt          {\romannumeral0\xintnthelt }%
206 \def\xintNthEltNoExpand {\romannumeral0\xintntheltnoexpand }%
207 \def\xintnthelt #1#2%
208 {%
209     \expandafter\XINT_nthelt_a\the\numexpr #1\expandafter.%
210     \expandafter{\romannumeral`&&@#2}%
211 }%
212 \def\xintntheltnoexpand #1%
213 {%
214     \expandafter\XINT_nthelt_a\the\numexpr #1.%
215 }%
216 \def\XINT_nthelt_a #1#2.%
217 {%
218     \xint_UDzerominusfork

```



```

219      #1-{\XINT_nthelt_bzero}%
220      0#1{\XINT_nthelt_bneg {#2}}%
221      0-{\XINT_nthelt_bpos {#1#2}}%
222      \krof
223 }%
224 \long\def\XINT_nthelt_bzero #1%
225 {%
226     \XINT_length_loop 0.#1\xint_relax\xint_relax\xint_relax\xint_relax
227         \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
228 }%
229 \long\def\XINT_nthelt_bneg #1#2%
230 {%
231     \expandafter\XINT_nthelt_loop_a\expandafter {\the\numexpr #1\expandafter}%
232     \romannumeral0\xintrevwithbracesnoexpand {#2}%
233     \xint_relax\xint_relax\xint_relax\xint_relax
234     \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
235 }%
236 \long\def\XINT_nthelt_bpos #1#2%
237 {%
238     \XINT_nthelt_loop_a {#1}#2\xint_relax\xint_relax\xint_relax\xint_relax
239     \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
240 }%
241 \def\XINT_nthelt_loop_a #1%
242 {%
243     \ifnum #1>\xint_c_viii
244         \expandafter\XINT_nthelt_loop_b
245     \else
246         \XINT_nthelt_getit
247     \fi
248     {#1}%
249 }%
250 \long\def\XINT_nthelt_loop_b #1#2#3#4#5#6#7#8#9%
251 {%
252     \xint_gob_til_xint_relax #9\XINT_nthelt_silentend\xint_relax
253     \expandafter\XINT_nthelt_loop_a\expandafter{\the\numexpr #1-\xint_c_viii}%
254 }%
255 \def\XINT_nthelt_silentend #1\xint_bye { }%
256 \def\XINT_nthelt_getit\fi #1%
257 {%
258     \fi\expandafter\expandafter\expandafter\XINT_nthelt_finish
259     \csname xint_gobble_\romannumeral\numexpr#1-\xint_c_i\endcsname
260 }%
261 \long\edef\XINT_nthelt_finish #1#2\xint_bye
262     {\noexpand\xint_gob_til_xint_relax #1\noexpand\expandafter\space
263         \noexpand\xint_gobble_ii\xint_relax\space #1}%

```

## 2.12 `\xintKeep`

First included in release 1.09m.

`\xintKeep {i}{stuff f-expanding to {a}{b}...{z}}` (or ``tokens' abcd...z`, but each naked token ends up braced in the output, if  $0 < i < \text{length of token list}$ ) returns (in two expansion steps) the first  $i$  items from the list, which is first  $f$ -expanded. The  $i$  is expanded inside `\numexpr`. The

## 2 Package *xinttools* implementation

variant `\xintKeepNoExpand` does not expand the list argument.

With  $i = 0$ , the empty sequence is returned.

With  $i < 0$ , the last  $|i|$  items are returned (in the same order as in the original list) AND BRACES ARE NOT ADDED IF NOT ORIGINALLY PRESENT.

With  $|i|$  equal to or bigger than the length of the (f-expanded) list, the full list is returned.

1.2a belatedly corrects the description of what this macro does for  $i < 0$  !

I have this nagging feeling I should read this code which might be much improvable upon, but I just don't have time now (2015/10/19).

```
264 \def\xintKeep          {\romannumeral0\xintkeep }%
265 \def\xintKeepNoExpand {\romannumeral0\xintkeepnoexpand }%
266 \def\xintkeep #1#2%
267 {%
268   \expandafter\XINT_keep_a\the\numexpr #1\expandafter.%
269   \expandafter{\romannumeral`&&#2}%
270 }%
271 \def\xintkeepnoexpand #1%
272 {%
273   \expandafter\XINT_keep_a\the\numexpr #1.%
274 }%
275 \def\XINT_keep_a #1#2.%
276 {%
277   \xint_UDzerominusfork
278     #1-{\expandafter\space\xint_gobble_i }%
279     0#1{\XINT_keep_bneg_a {#2}}%
280     0-{\XINT_keep_bpos {#1#2}}%
281   \krof
282 }%
283 \long\def\XINT_keep_bneg_a #1#2%
284 {%
285   \expandafter\XINT_keep_bneg_b \the\numexpr \xintLength{#2}-#1.{#2}%
286 }%
287 \def\XINT_keep_bneg_b #1#2.%
288 {%
289   \xint_UDzerominusfork
290     #1-{\xint_firstofone_thenstop }%
291     0#1{\xint_firstofone_thenstop }%
292     0-{\XINT_trim_bpos {#1#2}}%
293   \krof
294 }%
295 \long\def\XINT_keep_bpos #1#2%
296 {%
297   \XINT_keep_loop_a {#1}{#2}\xint_relax\xint_relax\xint_relax\xint_relax
298   \xint_relax\xint_relax\xint_relax\xint_bye
299 }%
300 \def\XINT_keep_loop_a #1%
301 {%
302   \ifnum #1>\xint_c_vi
303     \expandafter\XINT_keep_loop_b
304   \else
305     \XINT_keep_finish
306   \fi
307   {#1}%

```

## 2 Package *xinttools* implementation

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

## 2.13 `\xintKeepUnbraced`

1.2a. Same as `\xintKeep` but will not maintain brace pairs around the kept items upfront.

```

359 \def\xintKeepUnbraced      {\romannumeral0\xintkeepunbraced}%
360 \def\xintKeepUnbracedNoExpand {\romannumeral0\xintkeepunbracednoexpand}%
361 \def\xintkeepunbraced #1#2%
362 {%
363   \expandafter\XINT_keepunbraced_a\the\numexpr #1\expandafter.%
364   \expandafter{\romannumeral`&&@#2}%
365}%
366 \def\xintkeepnoexpand #1%
367 {%
368   \expandafter\XINT_keepunbraced_a\the\numexpr #1.%
369}%
370 \def\XINT_keepunbraced_a #1#2.%
371 {%
372   \xint_UDzerominusfork
373     #1-{\expandafter\space\xint_gobble_i}%
374     0#1{\XINT_keep_bneg_a {#2}}%
375     0-{\XINT_keepunbraced_bpos {#1#2}}%
376   \krof
377}%
378 \long\def\XINT_keepunbraced_bpos #1#2%
379 {%
380   \XINT_keepunbraced_loop_a {#1}{#2}%
381   \xint_relax\xint_relax\xint_relax\xint_relax
382   \xint_relax\xint_relax\xint_relax\xint_bye
383}%
384 \def\XINT_keepunbraced_loop_a #1%
385 {%
386   \ifnum #1>\xint_c_vi
387     \expandafter\XINT_keepunbraced_loop_b
388   \else
389     \XINT_keepunbraced_finish
390   \fi
391   {#1}%
392}%
393 \long\def\XINT_keepunbraced_loop_b #1#2#3#4#5#6#7#8#9%
394 {%
395   \xint_gob_til_xint_relax #9\XINT_keepunbraced_enda\xint_relax
396   \expandafter\XINT_keepunbraced_loop_c\expandafter
397   {\the\numexpr #1-\xint_c_vii}{#3}{#4}{#5}{#6}{#7}{#8}{#9}.{#2}%
398}%
399 \long\def\XINT_keepunbraced_loop_c #1#2#3#4#5#6#7#8.#9%
400   {\XINT_keepunbraced_loop_a {#1}{#9#2#3#4#5#6#7#8}}%
401 \long\def\XINT_keepunbraced_enda\xint_relax
402   \expandafter\XINT_keepunbraced_loop_c\expandafter #1#2.#3#4\xint_bye
403 {%
404   \XINT_keepunbraced_endb #4\W\W\W\W\W\W\Z #2{#3}%
405}%
406 \def\XINT_keepunbraced_endb #1#2#3#4#5#6#7\Z
407 {%

```

```

408 \xint_gob_til_W
409 #1\XINT_keepunbraced_endc_
410 #2\XINT_keepunbraced_endc_i
411 #3\XINT_keepunbraced_endc_ii
412 #4\XINT_keepunbraced_endc_iii
413 #5\XINT_keepunbraced_endc_iv
414 #6\XINT_keepunbraced_endc_v
415 \W\XINT_keepunbraced_endc_vi\Z
416 }%
417 \long\def\XINT_keepunbraced_endc_ #1\Z #2#3#4#5#6#7#8#9{ #9}%
418 \long\def\XINT_keepunbraced_endc_i #1\Z #2#3#4#5#6#7#8#9{ #9#2}%
419 \long\def\XINT_keepunbraced_endc_ii #1\Z #2#3#4#5#6#7#8#9{ #9#2#3}%
420 \long\def\XINT_keepunbraced_endc_iii #1\Z #2#3#4#5#6#7#8#9{ #9#2#3#4}%
421 \long\def\XINT_keepunbraced_endc_iv #1\Z #2#3#4#5#6#7#8#9{ #9#2#3#4#5}%
422 \long\def\XINT_keepunbraced_endc_v #1\Z #2#3#4#5#6#7#8#9{ #9#2#3#4#5#6}%
423 \long\def\XINT_keepunbraced_endc_vi\Z #1#2#3#4#5#6#7#8{ #8#1#2#3#4#5#6}%
424 \long\def\XINT_keepunbraced_finish\fi #1#2#3#4#5#6#7#8#9\xint_bye
425 {%
426 \fi\XINT_keepunbraced_finish_loop_a {#1}{#3}{#4}{#5}{#6}{#7}{#8}\Z {#2}%
427 }%
428 \def\XINT_keepunbraced_finish_loop_a #1%
429 {%
430 \xint_gob_til_zero #1\XINT_keepunbraced_finish_z0%
431 \expandafter\XINT_keepunbraced_finish_loop_b\expandafter
432 {\the\numexpr #1-\xint_c_i}%
433 }%
434 \long\def\XINT_keepunbraced_finish_z0%
435 \expandafter\XINT_keepunbraced_finish_loop_b\expandafter #1#2#3\Z #4{ #4#2}%
436 \long\def\XINT_keepunbraced_finish_loop_b #1#2#3%
437 {%
438 \xint_gob_til_xint_relax #3\XINT_keepunbraced_finish_exit\xint_relax
439 \XINT_keepunbraced_finish_loop_c {#1}{#2}{#3}%
440 }%
441 \long\def\XINT_keepunbraced_finish_exit\xint_relax
442 \XINT_keepunbraced_finish_loop_c #1#2#3\Z #4{ #4#2}%
443 \long\def\XINT_keepunbraced_finish_loop_c #1#2#3%
444 {\XINT_keepunbraced_finish_loop_a {#1}{#2#3}}%

```

## 2.14 `\xintTrim`

First included in release 1.09m.

`\xintTrim {i}{stuff f-expanding to {a}{b}...{z}}` (or ``tokens' abcd...z`) 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  $|i|$  items are suppressed. In that case the kept elements (coming from the tail) will be braced on output. With  $i > 0$ , the first  $|i|$  items are suppressed: the remaining ones are left as is.

With  $|i|$  equal to or bigger than the length of the ( $f$ -expanded) list, the empty list is returned.

```

445 \def\xintTrim {\romannumeral0\xinttrim }%
446 \def\xintTrimNoExpand {\romannumeral0\xinttrimnoexpand }%
447 \def\xinttrim #1#2%

```

## 2 Package *xinttools* implementation

```
448 {%
449   \expandafter\XINT_trim_a\the\numexpr #1\expandafter.%
450   \expandafter{\romannumeral`&&@#2}%
451 }%
452 \def\XINT_trimnoexpand #1%
453 {%
454   \expandafter\XINT_trim_a\the\numexpr #1.%
455 }%
456 \def\XINT_trim_a #1#2.%
457 {%
458   \xint_UDzerominusfork
459     #1-{\xint_firstofone_thenstop }%
460     0#1{\XINT_trim_bneg_a {#2}}%
461     0-{\XINT_trim_bpos {#1#2}}%
462   \krof
463 }%
464 \long\def\XINT_trim_bneg_a #1#2%
465 {%
466   \expandafter\XINT_trim_bneg_b \the\numexpr \xintLength{#2}-#1.{#2}%
467 }%
468 \def\XINT_trim_bneg_b #1#2.%
469 {%
470   \xint_UDzerominusfork
471     #1-{\expandafter\space\xint_gobble_i }%
472     0#1{\expandafter\space\xint_gobble_i }%
473     0-{\XINT_keep_bpos {#1#2}}%
474   \krof
475 }%
476 \long\def\XINT_trim_bpos #1#2%
477 {%
478   \XINT_trim_loop_a {#1}#2\xint_relax\xint_relax\xint_relax\xint_relax
479     \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
480 }%
481 \def\XINT_trim_loop_a #1%
482 {%
483   \ifnum #1>\xint_c_vii
484     \expandafter\XINT_trim_loop_b
485   \else
486     \XINT_trim_finish
487   \fi
488   {#1}%
489 }%
490 \long\def\XINT_trim_loop_b #1#2#3#4#5#6#7#8#9%
491 {%
492   \xint_gob_til_xint_relax #9\XINT_trim_silentend\xint_relax
493   \expandafter\XINT_trim_loop_a\expandafter{\the\numexpr #1-\xint_c_viii}%
494 }%
495 \def\XINT_trim_silentend #1\xint_bye { }%
496 \def\XINT_trim_finish\fi #1%
497 {%
498   \fi\expandafter\expandafter\expandafter\XINT_trim_finish_a
499   \expandafter\expandafter\expandafter\space % avoids brace removal
```

```

500 \csname xint_gobble_\romannumeral\numexpr#1\endcsname
501 }%
502 \long\def\XINT_trim_finish_a #1\xint_relax #2\xint_bye {#1}%

```

## 2.15 \xintTrimUnbraced

### 1.2a

```

503 \def\xintTrimUnbraced          {\romannumeral0\xinttrimunbraced }%
504 \def\xintTrimUnbracedNoExpand {\romannumeral0\xinttrimunbracednoexpand }%
505 \def\xinttrimunbraced #1#2%
506 {%
507   \expandafter\XINT_trimunbraced_a\the\numexpr #1\expandafter.%
508   \expandafter{\romannumeral`&&#2}%
509 }%
510 \def\xinttrimunbracednoexpand #1%
511 {%
512   \expandafter\XINT_trimunbraced_a\the\numexpr #1.%
513 }%
514 \def\XINT_trimunbraced_a #1#2.%
515 {%
516   \xint_UDzerominusfork
517     #1-{\xint_firstofone_thenstop }%
518     0#1{\XINT_trimunbraced_bneg_a {#2}}%
519     0-{\XINT_trim_bpos {#1#2}}%
520   \krof
521 }%
522 \long\def\XINT_trimunbraced_bneg_a #1#2%
523 {%
524   \expandafter\XINT_trimunbraced_bneg_b \the\numexpr \xintLength{#2}-#1.{#2}%
525 }%
526 \def\XINT_trimunbraced_bneg_b #1#2.%
527 {%
528   \xint_UDzerominusfork
529     #1-{\expandafter\space\xint_gobble_i }%
530     0#1{\expandafter\space\xint_gobble_i }%
531     0-{\XINT_keepunbraced_bpos {#1#2}}%
532   \krof
533 }%

```

## 2.16 \xintApply

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

```

534 \def\xintApply          {\romannumeral0\xintapply }%
535 \def\xintApplyNoExpand {\romannumeral0\xintapplynoexpand }%
536 \long\def\xintapply #1#2%
537 {%
538   \expandafter\XINT_apply\expandafter {\romannumeral`&&#2}%
539   {#1}%
540 }%

```

```

541 \long\def\XINT_apply #1#2{\XINT_apply_loop_a {}{#2}#1\xint_bye }%
542 \long\def\xintapplynoexpand #1#2{\XINT_apply_loop_a {}{#1}#2\xint_bye }%
543 \long\def\XINT_apply_loop_a #1#2#3%
544 {%
545   \xint_bye #3\XINT_apply_end\xint_bye
546   \expandafter
547   \XINT_apply_loop_b
548   \expandafter {\romannumeral`&&@#2{#3}}{#1}{#2}%
549 }%
550 \long\def\XINT_apply_loop_b #1#2{\XINT_apply_loop_a {#2{#1}}}%
551 \long\def\XINT_apply_end\xint_bye\expandafter\XINT_apply_loop_b
552   \expandafter #1#2#3{ #2}%

```

## 2.17 \xintApplyUnbraced

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

```

553 \def\xintApplyUnbraced {\romannumeral0\xintapplyunbraced }%
554 \def\xintApplyUnbracedNoExpand {\romannumeral0\xintapplyunbracednoexpand }%
555 \long\def\xintapplyunbraced #1#2%
556 {%
557   \expandafter\XINT_applyunbr\expandafter {\romannumeral`&&@#2}%
558   {#1}%
559 }%
560 \long\def\XINT_applyunbr #1#2{\XINT_applyunbr_loop_a {}{#2}#1\xint_bye }%
561 \long\def\xintapplyunbracednoexpand #1#2%
562   {\XINT_applyunbr_loop_a {}{#1}#2\xint_bye }%
563 \long\def\XINT_applyunbr_loop_a #1#2#3%
564 {%
565   \xint_bye #3\XINT_applyunbr_end\xint_bye
566   \expandafter\XINT_applyunbr_loop_b
567   \expandafter {\romannumeral`&&@#2{#3}}{#1}{#2}%
568 }%
569 \long\def\XINT_applyunbr_loop_b #1#2{\XINT_applyunbr_loop_a {#2{#1}}}%
570 \long\def\XINT_applyunbr_end\xint_bye\expandafter\XINT_applyunbr_loop_b
571   \expandafter #1#2#3{ #2}%

```

## 2.18 \xintSeq

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

```

572 \def\xintSeq {\romannumeral0\xintseq }%
573 \def\xintseq #1{\XINT_seq_chkopt #1\xint_bye }%
574 \def\XINT_seq_chkopt #1%
575 {%
576   \ifx [#1\expandafter\XINT_seq_opt
577     \else\expandafter\XINT_seq_noopt
578   \fi #1%
579 }%

```



## 2 Package *xinttools* implementation

```
580 \def\XINT_seq_noopt #1\xint_bye #2%
581 {%
582   \expandafter\XINT_seq\expandafter
583     {\the\numexpr#1\expandafter}\expandafter{\the\numexpr #2}%
584 }%
585 \def\XINT_seq #1#2%
586 {%
587   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
588     \expandafter\xint_firstoftwo_thenstop
589   \or
590     \expandafter\XINT_seq_p
591   \else
592     \expandafter\XINT_seq_n
593   \fi
594   {#2}{#1}%
595 }%
596 \def\XINT_seq_p #1#2%
597 {%
598   \ifnum #1>#2
599     \expandafter\expandafter\expandafter\XINT_seq_p
600   \else
601     \expandafter\XINT_seq_e
602   \fi
603   \expandafter{\the\numexpr #1-\xint_c_i}{#2}{#1}%
604 }%
605 \def\XINT_seq_n #1#2%
606 {%
607   \ifnum #1<#2
608     \expandafter\expandafter\expandafter\XINT_seq_n
609   \else
610     \expandafter\XINT_seq_e
611   \fi
612   \expandafter{\the\numexpr #1+\xint_c_i}{#2}{#1}%
613 }%
614 \def\XINT_seq_e #1#2#3{ }%
615 \def\XINT_seq_opt [\xint_bye #1]#2#3%
616 {%
617   \expandafter\XINT_seqo\expandafter
618     {\the\numexpr #2\expandafter}\expandafter
619     {\the\numexpr #3\expandafter}\expandafter
620     {\the\numexpr #1}%
621 }%
622 \def\XINT_seqo #1#2%
623 {%
624   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
625     \expandafter\XINT_seqo_a
626   \or
627     \expandafter\XINT_seqo_pa
628   \else
629     \expandafter\XINT_seqo_na
630   \fi
631   {#1}{#2}%

```

## 2 Package *xinttools* implementation

```

632 }%
633 \def\XINT_seqo_a #1#2#3{ {#1}}%
634 \def\XINT_seqo_o #1#2#3#4{ #4}%
635 \def\XINT_seqo_pa #1#2#3%
636 {%
637   \ifcase\ifnum #3=\xint_c_ 0\else\ifnum #3>\xint_c_ 1\else -1\fi\fi\space
638     \expandafter\XINT_seqo_o
639   \or
640     \expandafter\XINT_seqo_pb
641   \else
642     \xint_afterfi{\expandafter\space\xint_gobble_iv}%
643   \fi
644   {#1}{#2}{#3}{{#1}}%
645 }%
646 \def\XINT_seqo_pb #1#2#3%
647 {%
648   \expandafter\XINT_seqo_pc\expandafter{\the\numexpr #1+#3}{#2}{#3}%
649 }%
650 \def\XINT_seqo_pc #1#2%
651 {%
652   \ifnum #1>#2
653     \expandafter\XINT_seqo_o
654   \else
655     \expandafter\XINT_seqo_pd
656   \fi
657   {#1}{#2}%
658 }%
659 \def\XINT_seqo_pd #1#2#3#4{\XINT_seqo_pb {#1}{#2}{#3}{#4{#1}}}%
660 \def\XINT_seqo_na #1#2#3%
661 {%
662   \ifcase\ifnum #3=\xint_c_ 0\else\ifnum #3>\xint_c_ 1\else -1\fi\fi\space
663     \expandafter\XINT_seqo_o
664   \or
665     \xint_afterfi{\expandafter\space\xint_gobble_iv}%
666   \else
667     \expandafter\XINT_seqo_nb
668   \fi
669   {#1}{#2}{#3}{{#1}}%
670 }%
671 \def\XINT_seqo_nb #1#2#3%
672 {%
673   \expandafter\XINT_seqo_nc\expandafter{\the\numexpr #1+#3}{#2}{#3}%
674 }%
675 \def\XINT_seqo_nc #1#2%
676 {%
677   \ifnum #1<#2
678     \expandafter\XINT_seqo_o
679   \else
680     \expandafter\XINT_seqo_nd
681   \fi
682   {#1}{#2}%
683 }%

```

```
684 \def\XINT_sequo_nd #1#2#3#4{\XINT_sequo_nb {#1}{#2}{#3}{#4{#1}}}%
```

## 2.19 \xintloop, \xintbreakloop, \xintbreakloopanddo, \xintloopskiptonext

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

```
685 \long\def\xintloop #1#2\repeat {#1#2\xintloop_again\fi\xint_gobble_i {#1#2}}%
686 \long\def\xintloop_again\fi\xint_gobble_i #1{\fi
687           #1\xintloop_again\fi\xint_gobble_i {#1}}%
688 \long\def\xintbreakloop #1\xintloop_again\fi\xint_gobble_i #2{%
689 \long\def\xintbreakloopanddo #1#2\xintloop_again\fi\xint_gobble_i #3{#1}%
690 \long\def\xintloopskiptonext #1\xintloop_again\fi\xint_gobble_i #2{%
691           #2\xintloop_again\fi\xint_gobble_i {#2}}%
```

## 2.20 \xintiloop, \xintiloopindex, \xintouteriloopindex, \xintbreakiloop, \xintbreakiloopanddo, \xintilopskiptonext, \xintilopskipandredo

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

```
692 \def\xintiloop [#1+#2]{%
693   \expandafter\xintiloop_a\the\numexpr #1\expandafter.\the\numexpr #2.}%
694 \long\def\xintiloop_a #1.#2.#3#4\repeat{%
695   #3#4\xintiloop_again\fi\xint_gobble_iii {#1}{#2}{#3#4}}%
696 \def\xintiloop_again\fi\xint_gobble_iii #1#2{%
697   \fi\expandafter\xintiloop_again_b\the\numexpr#1+#2.#2.}%
698 \long\def\xintiloop_again_b #1.#2.#3{%
699   #3\xintiloop_again\fi\xint_gobble_iii {#1}{#2}{#3}}%
700 \long\def\xintbreakiloop #1\xintiloop_again\fi\xint_gobble_iii #2#3#4{%
701 \long\def\xintbreakiloopanddo
702   #1.#2\xintiloop_again\fi\xint_gobble_iii #3#4#5{#1}%
703 \long\def\xintiloopindex #1\xintiloop_again\fi\xint_gobble_iii #2%
704   {#2#1\xintiloop_again\fi\xint_gobble_iii {#2}}%
705 \long\def\xintouteriloopindex #1\xintiloop_again
706   #2\xintiloop_again\fi\xint_gobble_iii #3%
707   {#3#1\xintiloop_again #2\xintiloop_again\fi\xint_gobble_iii {#3}}%
708 \long\def\xintilopskiptonext #1\xintiloop_again\fi\xint_gobble_iii #2#3{%
709   \expandafter\xintiloop_again_b \the\numexpr#2+#3.#3.}%
710 \long\def\xintilopskipandredo #1\xintiloop_again\fi\xint_gobble_iii #2#3#4{%
711   #4\xintiloop_again\fi\xint_gobble_iii {#2}{#3}{#4}}%
```

## 2.21 \XINT\_xflet

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

```
712 \def\XINT_xflet #1%
713 {%
714   \def\XINT_xflet_macro {#1}\XINT_xflet_zapsp
715 }%
716 \def\XINT_xflet_zapsp
717 {%
718   \expandafter\futurelet\expandafter\XINT_token
```

```

719 \expandafter\XINT_xflet_sp?\romannumeral`&&@%
720 }%
721 \def\XINT_xflet_sp?
722 {%
723 \ifx\XINT_token\XINT_sptoken
724 \expandafter\XINT_xflet_zapsp
725 \else\expandafter\XINT_xflet_zapspB
726 \fi
727 }%
728 \def\XINT_xflet_zapspB
729 {%
730 \expandafter\futurelet\expandafter\XINT_tokenB
731 \expandafter\XINT_xflet_spB?\romannumeral`&&@%
732 }%
733 \def\XINT_xflet_spB?
734 {%
735 \ifx\XINT_tokenB\XINT_sptoken
736 \expandafter\XINT_xflet_zapspB
737 \else\expandafter\XINT_xflet_eq?
738 \fi
739 }%
740 \def\XINT_xflet_eq?
741 {%
742 \ifx\XINT_token\XINT_tokenB
743 \expandafter\XINT_xflet_macro
744 \else\expandafter\XINT_xflet_zapsp
745 \fi
746 }%

```

## 2.22 \xintApplyInline

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

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

```

747 \catcode`Z 3
748 \long\def\xintApplyInline #1#2%
749 {%
750 \long\expandafter\def\expandafter\XINT_inline_macro
751 \expandafter ##\expandafter 1\expandafter {#1{##1}}%
752 \XINT_xflet\XINT_inline_b #2Z% this Z has catcode 3
753 }%
754 \def\XINT_inline_b
755 {%
756 \ifx\XINT_token Z\expandafter\xint_gobble_i
757 \else\expandafter\XINT_inline_d\fi
758 }%
759 \long\def\XINT_inline_d #1%
760 {%
761 \long\def\XINT_item{#1}\XINT_xflet\XINT_inline_e
762 }%

```

```

763 \def\XINT_inline_e
764 {%
765   \ifx\XINT_token Z\expandafter\XINT_inline_w
766   \else\expandafter\XINT_inline_f\fi
767 }%
768 \def\XINT_inline_f
769 {%
770   \expandafter\XINT_inline_g\expandafter{\XINT_inline_macro {##1}}%
771 }%
772 \long\def\XINT_inline_g #1%
773 {%
774   \expandafter\XINT_inline_macro\XINT_item
775   \long\def\XINT_inline_macro ##1{#1}\XINT_inline_d
776 }%
777 \def\XINT_inline_w #1%
778 {%
779   \expandafter\XINT_inline_macro\XINT_item
780 }%

```

## 2.23 \xintFor, \xintFor\*, \xintBreakFor, \xintBreakForAndDo

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

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

1.09e adds \XINT\_forever with \xintintegers, \xintdimensions, \xintrationals and \xintBreakFor, \xintBreakForAndDo, \xintifForFirst, \xintifForLast. On this occasion \xint\_firstoftwo and \xint\_secondoftwo are made long.

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

1.09f rewrites (2013/11/03) the code which now allows all macro parameters from #1 to #9 in \xintFor, \xintFor\*, and \XINT\_forever.

```

781 \def\XINT_tmpa #1#2{\ifnum #2<#1 \xint_afterfi {{#####2}}\fi}%
782 \def\XINT_tmpb #1#2{\ifnum #1<#2 \xint_afterfi {{#####2}}\fi}%
783 \def\XINT_tmpc #1%
784 {%
785   \expandafter\edef \csname XINT_for_left#1\endcsname
786     {\xintApplyUnbraced {\XINT_tmpa #1}{123456789}}%
787   \expandafter\edef \csname XINT_for_right#1\endcsname
788     {\xintApplyUnbraced {\XINT_tmpb #1}{123456789}}%
789 }%
790 \xintApplyInline \XINT_tmpc {123456789}%
791 \long\def\xintBreakFor      #1Z{%}%
792 \long\def\xintBreakForAndDo #1#2Z{#1}%
793 \def\xintFor {\let\xintifForFirst\xint_firstoftwo
794   \futurelet\XINT_token\XINT_for_ifstar }%
795 \def\XINT_for_ifstar {\ifx\XINT_token*\expandafter\XINT_forx
796   \else\expandafter\XINT_for \fi }%

```

## 2 Package *xinttools* implementation

```

797 \catcode`U 3 % with numexpr
798 \catcode`V 3 % with xintfrac.sty (xint.sty not enough)
799 \catcode`D 3 % with dimexpr
800 \def\XINT_flet_zapsp
801 {%
802   \futurelet\XINT_token\XINT_flet_sp?
803 }%
804 \def\XINT_flet_sp?
805 {%
806   \ifx\XINT_token\XINT_sptoken
807     \xint_afterfi{\expandafter\XINT_flet_zapsp\romannumeral0}%
808   \else\expandafter\XINT_flet_macro
809   \fi
810 }%
811 \long\def\XINT_for #1#2in#3#4#5%
812 {%
813   \expandafter\XINT_toks\expandafter
814   {\expandafter\XINT_for_d\the\numexpr #2\relax {#5}}%
815   \def\XINT_flet_macro {\expandafter\XINT_for_forever?\space}%
816   \expandafter\XINT_flet_zapsp #3Z%
817 }%
818 \def\XINT_for_forever? #1Z%
819 {%
820   \ifx\XINT_token U\XINT_to_forever\fi
821   \ifx\XINT_token V\XINT_to_forever\fi
822   \ifx\XINT_token D\XINT_to_forever\fi
823   \expandafter\the\expandafter\XINT_toks\romannumeral0\xintcsvtlist {#1}Z%
824 }%
825 \def\XINT_to_forever\fi #1\xintcsvtlist #2{\fi \XINT_forever #2}%
826 \long\def\XINT_forx *#1#2in#3#4#5%
827 {%
828   \expandafter\XINT_toks\expandafter
829   {\expandafter\XINT_forx_d\the\numexpr #2\relax {#5}}%
830   \XINT_xflet\XINT_forx_forever? #3Z%
831 }%
832 \def\XINT_forx_forever?
833 {%
834   \ifx\XINT_token U\XINT_to_forxever\fi
835   \ifx\XINT_token V\XINT_to_forxever\fi
836   \ifx\XINT_token D\XINT_to_forxever\fi
837   \XINT_forx_empty?
838 }%
839 \def\XINT_to_forxever\fi #1\XINT_forx_empty? {\fi \XINT_forever }%
840 \catcode`U 11
841 \catcode`D 11
842 \catcode`V 11
843 \def\XINT_forx_empty?
844 {%
845   \ifx\XINT_token Z\expandafter\xintBreakFor\fi
846   \the\XINT_toks
847 }%
848 \long\def\XINT_for_d #1#2#3%

```

```

849 {%
850 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
851 \XINT_toks {{#3}}%
852 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
853 \the\XINT_toks \csname XINT_for_right#1\endcsname}%
854 \XINT_toks {\XINT_x\let\xintifForFirst\xint_secondoftwo\XINT_for_d #1{#2}}%
855 \futurelet\XINT_token\XINT_for_last?
856}%
857\long\def\XINT_forx_d #1#2#3%
858{%
859 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
860 \XINT_toks {{#3}}%
861 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
862 \the\XINT_toks \csname XINT_for_right#1\endcsname}%
863 \XINT_toks {\XINT_x\let\xintifForFirst\xint_secondoftwo\XINT_forx_d #1{#2}}%
864 \XINT_xflet\XINT_for_last?
865}%
866\def\XINT_for_last?
867{%
868 \let\xintifForLast\xint_secondoftwo
869 \ifx\XINT_token Z\let\xintifForLast\xint_firstoftwo
870 \xint_afterfi{\xintBreakForAndDo{\XINT_x\xint_gobble_i Z}}\fi
871 \the\XINT_toks
872}%

```

## 2.24 \XINT\_forever, \xintintegers, \xintdimensions, \xintrationals

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

```

873 \catcode`U 3
874 \catcode`D 3
875 \catcode`V 3
876 \let\xintegers U%
877 \let\xintintegers U%
878 \let\xintdimensions D%
879 \let\xintrationals V%
880 \def\XINT_forever #1%
881 {%
882 \expandafter\XINT_forever_a
883 \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi a\expandafter\endcsname
884 \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi i\expandafter\endcsname
885 \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi \endcsname
886}%
887 \catcode`U 11
888 \catcode`D 11
889 \catcode`V 11
890 \def\XINT_?expr_Ua #1#2%
891 {\expandafter{\expandafter\numexpr\the\numexpr #1\expandafter\relax

```

## 2 Package *xinttools* implementation

```

892             \expandafter\relax\expandafter}%
893     \expandafter{\the\numexpr #2}}%
894 \def\XINT_?expr_Da #1#2%
895     {\expandafter{\expandafter\dimexpr\number\dimexpr #1\expandafter\relax
896         \expandafter s\expandafter p\expandafter\relax\expandafter}%
897     \expandafter{\number\dimexpr #2}}%
898 \catcode`Z 11
899 \def\XINT_?expr_Va #1#2%
900 {%
901     \expandafter\XINT_?expr_Vb\expandafter
902         {\romannumeral`&&\xintrowwithzeros{\xintZapSpacesB{#2}}}%
903         {\romannumeral`&&\xintrowwithzeros{\xintZapSpacesB{#1}}}%
904 }%
905 \catcode`Z 3
906 \def\XINT_?expr_Vb #1#2{\expandafter\XINT_?expr_Vc #2.#1.%}
907 \def\XINT_?expr_Vc #1/#2.#3/#4.%
908 {%
909     \xintifEq {#2}{#4}%
910         {\XINT_?expr_Vf {#3}{#1}{#2}}%
911         {\expandafter\XINT_?expr_Vd\expandafter
912             {\romannumeral0\xintiimul {#2}{#4}}%
913             {\romannumeral0\xintiimul {#1}{#4}}%
914             {\romannumeral0\xintiimul {#2}{#3}}%
915         }%
916 }%
917 \def\XINT_?expr_Vd #1#2#3{\expandafter\XINT_?expr_Ve\expandafter {#2}{#3}{#1}}%
918 \def\XINT_?expr_Ve #1#2{\expandafter\XINT_?expr_Vf\expandafter {#2}{#1}}%
919 \def\XINT_?expr_Vf #1#2#3{{#2/#3}{0}{#1}{#2}{#3}}%
920 \def\XINT_?expr_Ui {{\numexpr 1\relax}{1}}%
921 \def\XINT_?expr_Di {{\dimexpr 0pt\relax}{65536}}%
922 \def\XINT_?expr_Vi {{1/1}{0111}}%
923 \def\XINT_?expr_U #1#2%
924     {\expandafter{\expandafter\numexpr\the\numexpr #1+#2\relax\relax}{#2}}%
925 \def\XINT_?expr_D #1#2%
926     {\expandafter{\expandafter\dimexpr\the\numexpr #1+#2\relax sp\relax}{#2}}%
927 \def\XINT_?expr_V #1#2{\XINT_?expr_Vx #2}%
928 \def\XINT_?expr_Vx #1#2%
929 {%
930     \expandafter\XINT_?expr_Vy\expandafter
931         {\romannumeral0\xintiiadd {#1}{#2}}{#2}%
932 }%
933 \def\XINT_?expr_Vy #1#2#3#4%
934 {%
935     \expandafter{\romannumeral0\xintiiadd {#3}{#1}/#4}{#1}{#2}{#3}{#4}}%
936 }%
937 \def\XINT_forever_a #1#2#3#4%
938 {%
939     \ifx #4[\expandafter\XINT_forever_opt_a
940         \else\expandafter\XINT_forever_b
941         \fi #1#2#3#4%
942 }%
943 \def\XINT_forever_b #1#2#3Z{\expandafter\XINT_forever_c\the\XINT_toks #2#3}%

```



```

944 \long\def\XINT_forever_c #1#2#3#4#5%
945   {\expandafter\XINT_forever_d\expandafter #2#4#5{#3}Z}%
946 \def\XINT_forever_opt_a #1#2#3[#4+#5]#6Z%
947 {%
948   \expandafter\expandafter\expandafter
949   \XINT_forever_opt_c\expandafter\the\expandafter\XINT_toks
950   \romannumeral`&&@#1{#4}{#5}#3%
951 }%
952 \long\def\XINT_forever_opt_c #1#2#3#4#5#6{\XINT_forever_d #2{#4}{#5}#6{#3}Z}%
953 \long\def\XINT_forever_d #1#2#3#4#5%
954 {%
955   \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#5}%
956   \XINT_toks {{#2}}%
957   \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
958     \the\XINT_toks \csname XINT_for_right#1\endcsname}%
959   \XINT_x
960   \let\xintifForFirst\xint_seconddoftwo
961   \expandafter\XINT_forever_d\expandafter #1\romannumeral`&&@#4{#2}{#3}#4{#5}%
962 }%

```

## 2.25 \xintForpair, \xintForthree, \xintForfour

1.09c.

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

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

```

963 \catcode`j 3
964 \long\def\xintForpair #1#2#3in#4#5#6%
965 {%
966   \let\xintifForFirst\xint_firstoftwo
967   \XINT_toks {\XINT_forpair_d #2{#6}}%
968   \expandafter\the\expandafter\XINT_toks #4jZ%
969 }%
970 \long\def\XINT_forpair_d #1#2#3(#4)#5%
971 {%
972   \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
973   \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
974   \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
975     \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_i\endcsname}%
976   \let\xintifForLast\xint_seconddoftwo
977   \ifx #5j\expandafter\xint_firstoftwo
978     \else\expandafter\xint_seconddoftwo
979   \fi
980   {\let\xintifForLast\xint_firstoftwo
981     \xintBreakForAndDo {\XINT_x \xint_gobble_i Z}}%
982   \XINT_x
983   \let\xintifForFirst\xint_seconddoftwo\XINT_forpair_d #1{#2}%
984 }%
985 \long\def\xintForthree #1#2#3in#4#5#6%
986 {%
987   \let\xintifForFirst\xint_firstoftwo

```

```

988 \XINT_toks {\XINT_forthree_d #2{#6}}%
989 \expandafter\the\expandafter\XINT_toks #4jZ%
990 }%
991 \long\def\XINT_forthree_d #1#2#3(#4)#5%
992 {%
993 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
994 \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
995 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
996 \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_ii\endcsname}%
997 \let\xintifForLast\xint_secondoftwo
998 \ifx #5j\expandafter\xint_firstoftwo
999 \else\expandafter\xint_secondoftwo
1000 \fi
1001 {\let\xintifForLast\xint_firstoftwo
1002 \xintBreakForAndDo {\XINT_x \xint_gobble_i Z}}%
1003 \XINT_x
1004 \let\xintifForFirst\xint_secondoftwo\XINT_forthree_d #1{#2}%
1005 }%
1006 \long\def\xintForfour #1#2#3in#4#5#6%
1007 {%
1008 \let\xintifForFirst\xint_firstoftwo
1009 \XINT_toks {\XINT_forfour_d #2{#6}}%
1010 \expandafter\the\expandafter\XINT_toks #4jZ%
1011 }%
1012 \long\def\XINT_forfour_d #1#2#3(#4)#5%
1013 {%
1014 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
1015 \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
1016 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
1017 \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_iii\endcsname}%
1018 \let\xintifForLast\xint_secondoftwo
1019 \ifx #5j\expandafter\xint_firstoftwo
1020 \else\expandafter\xint_secondoftwo
1021 \fi
1022 {\let\xintifForLast\xint_firstoftwo
1023 \xintBreakForAndDo {\XINT_x \xint_gobble_i Z}}%
1024 \XINT_x
1025 \let\xintifForFirst\xint_secondoftwo\XINT_forfour_d #1{#2}%
1026 }%
1027 \catcode`Z 11
1028 \catcode`j 11

```

## 2.26 \xintAssign, \xintAssignArray, \xintDigitsOf

\xintAssign {a}{b}..{z}\to A\B...Z resp. \xintAssignArray {a}{b}..{z}\to U.  
\xintDigitsOf=\xintAssignArray.

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

```

1029 \def\xintAssign{\def\XINT_flet_macro {\XINT_assign_fork}\XINT_flet_zapsp }%
1030 \def\XINT_assign_fork
1031 {%

```

## 2 Package *xinttools* implementation

```
1032 \let\XINT_assign_def\def
1033 \ifx\XINT_token[\expandafter\XINT_assign_opt
1034 \else\expandafter\XINT_assign_a
1035 \fi
1036 }%
1037 \def\XINT_assign_opt [#1]%
1038 {%
1039 \ifcsname #1def\endcsname
1040 \expandafter\let\expandafter\XINT_assign_def \csname #1def\endcsname
1041 \else
1042 \expandafter\let\expandafter\XINT_assign_def \csname xint#1def\endcsname
1043 \fi
1044 \XINT_assign_a
1045 }%
1046 \long\def\XINT_assign_a #1\to
1047 {%
1048 \def\XINT_flet_macro{\XINT_assign_b}%
1049 \expandafter\XINT_flet_zapsp\romannumeral`&&@#1\xint_relax\to
1050 }%
1051 \long\def\XINT_assign_b
1052 {%
1053 \ifx\XINT_token\bgroup
1054 \expandafter\XINT_assign_c
1055 \else\expandafter\XINT_assign_f
1056 \fi
1057 }%
1058 \long\def\XINT_assign_f #1\xint_relax\to #2%
1059 {%
1060 \XINT_assign_def #2{#1}%
1061 }%
1062 \long\def\XINT_assign_c #1%
1063 {%
1064 \def\xint_temp {#1}%
1065 \ifx\xint_temp\xint_brelax
1066 \expandafter\XINT_assign_e
1067 \else
1068 \expandafter\XINT_assign_d
1069 \fi
1070 }%
1071 \long\def\XINT_assign_d #1\to #2%
1072 {%
1073 \expandafter\XINT_assign_def\expandafter #2\expandafter{\xint_temp}%
1074 \XINT_assign_c #1\to
1075 }%
1076 \def\XINT_assign_e #1\to {}%
1077 \def\xintRelaxArray #1%
1078 {%
1079 \edef\XINT_restoreescapechar {\escapechar\the\escapechar\relax}%
1080 \escapechar -1
1081 \expandafter\def\expandafter\xint_arrayname\expandafter {\string #1}%
1082 \XINT_restoreescapechar
1083 \xintilooop [\csname\xint_arrayname 0\endcsname+-1]
```

## 2 Package *xinttools* implementation

```

1084 \global
1085 \expandafter\let\csname\xint_arrayname\xintloopindex\endcsname\relax
1086 \ifnum \xintloopindex > \xint_c_
1087 \repeat
1088 \global\expandafter\let\csname\xint_arrayname 00\endcsname\relax
1089 \global\let #1\relax
1090 }%
1091 \def\xintAssignArray{\def\XINT_flet_macro {\XINT_assignarray_fork}%
1092 \XINT_flet_zapsp }%
1093 \def\XINT_assignarray_fork
1094 {%
1095 \let\XINT_assignarray_def\def
1096 \ifx\XINT_token[\expandafter\XINT_assignarray_opt
1097 \else\expandafter\XINT_assignarray
1098 \fi
1099 }%
1100 \def\XINT_assignarray_opt [#1]%
1101 {%
1102 \ifcsname #1def\endcsname
1103 \expandafter\let\expandafter\XINT_assignarray_def \csname #1def\endcsname
1104 \else
1105 \expandafter\let\expandafter\XINT_assignarray_def
1106 \csname xint#1def\endcsname
1107 \fi
1108 \XINT_assignarray
1109 }%
1110 \long\def\XINT_assignarray #1\to #2%
1111 {%
1112 \edef\XINT_restoreescapechar {\escapechar\the\escapechar\relax }%
1113 \escapechar -1
1114 \expandafter\def\expandafter\xint_arrayname\expandafter {\string #2}%
1115 \XINT_restoreescapechar
1116 \def\xint_itemcount {0}%
1117 \expandafter\XINT_assignarray_loop \romannumeral`&&@#1\xint_relax
1118 \csname\xint_arrayname 00\expandafter\endcsname
1119 \csname\xint_arrayname 0\expandafter\endcsname
1120 \expandafter {\xint_arrayname}#2%
1121 }%
1122 \long\def\XINT_assignarray_loop #1%
1123 {%
1124 \def\xint_temp {#1}%
1125 \ifx\xint_brelax\xint_temp
1126 \expandafter\def\csname\xint_arrayname 0\expandafter\endcsname
1127 \expandafter{\the\numexpr\xint_itemcount}%
1128 \expandafter\expandafter\expandafter\XINT_assignarray_end
1129 \else
1130 \expandafter\def\expandafter\xint_itemcount\expandafter
1131 {\the\numexpr\xint_itemcount+\xint_c_i}%
1132 \expandafter\XINT_assignarray_def
1133 \csname\xint_arrayname\xint_itemcount\expandafter\endcsname
1134 \expandafter{\xint_temp }%
1135 \expandafter\XINT_assignarray_loop

```

## 2 Package *xinttools* implementation

```
1136   \fi
1137 }%
1138 \def\XINT_assignarray_end #1#2#3#4%
1139 {%
1140   \def #4##1%
1141   {%
1142     \romannumeral0\expandafter #1\expandafter{\the\numexpr ##1}%
1143   }%
1144   \def #1##1%
1145   {%
1146     \ifnum ##1<\xint_c_
1147       \xint_afterfi {\xintError:ArrayIndexIsNegative\space }%
1148     \else
1149       \xint_afterfi {%
1150         \ifnum ##1>#2
1151           \xint_afterfi {\xintError:ArrayIndexBeyondLimit\space }%
1152         \else\xint_afterfi
1153           {\expandafter\expandafter\expandafter\space\csname #3##1\endcsname}%
1154         \fi}%
1155     \fi
1156   }%
1157 }%
1158 \let\xintDigitsOf\xintAssignArray
1159 \let\XINT_tmpa\relax \let\XINT_tmpb\relax \let\XINT_tmpc\relax
1160 \XINT_restorecatcodes_endinput%
```

### 3 Package `xintcore` implementation

.1	Catcodes, $\varepsilon$ -TeX and reload detection . . .	38	.15	<code>\xintDouble</code> . . . . .	48
.2	Package identification . . . . .	39	.16	<code>\xintHalf</code> . . . . .	48
.3	Counts for holding needed constants . . .	39	.17	<code>\xintDec</code> . . . . .	49
.4	<code>\xintNum</code> . . . . .	39	.18	<code>\xintInc</code> . . . . .	50
.5	Zeroes . . . . .	40	.19	Core arithmetic . . . . .	50
.6	Blocks of eight digits . . . . .	41	.20	<code>\xintiAdd</code> , <code>\xintiiAdd</code> . . . . .	51
.7	Blocks of eight, for needs of 1.2 <code>\xintiiDi-</code> <code>vision</code> . . . . .	43	.21	<code>\xintiSub</code> , <code>\xintiiSub</code> . . . . .	54
.8	Blocks of eight, for needs of 1.2 <code>\xintiiDi-</code> <code>vision</code> . . . . .	44	.22	<code>\xintiMul</code> , <code>\xintiiMul</code> . . . . .	59
.9	<code>\xintReverseDigits</code> . . . . .	45	.23	<code>\xintiSqr</code> , <code>\xintiiSqr</code> . . . . .	63
.10	<code>\xintSgn</code> , <code>\xintiiSgn</code> , <code>\XINT_Sgn</code> , <code>\XINT_cntSgn</code> . . . . .	46	.24	<code>\xintiPow</code> , <code>\xintiiPow</code> . . . . .	64
.11	<code>\xintiOpp</code> , <code>\xintiiOpp</code> . . . . .	46	.25	<code>\xintiFac</code> , <code>\xintiiFac</code> . . . . .	67
.12	<code>\xintiAbs</code> , <code>\xintiiAbs</code> . . . . .	47	.26	<code>\xintiDivision</code> , <code>\xintiQuo</code> , <code>\xintiRem</code> , <code>\xintiiDivision</code> , <code>\xintiiQuo</code> , <code>\xintiiRem</code>	70
.13	<code>\xintFDg</code> , <code>\xintiiFDg</code> . . . . .	47	.27	<code>\xintiDivRound</code> , <code>\xintiiDivRound</code> . . .	86
.14	<code>\xintLDg</code> , <code>\xintiiLDg</code> . . . . .	48	.28	<code>\xintiDivTrunc</code> , <code>\xintiiDivTrunc</code> . . .	87
			.29	<code>\xintiMod</code> , <code>\xintiiMod</code> . . . . .	88
			.30	“Load <code>xintfrac</code> ” macros . . . . .	88

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

The core arithmetic routines have been entirely rewritten for release 1.2.

The commenting continues (2016/03/12) to be very sparse: actually it got worse than ever with release 1.2. I will possibly add comments at a later date, but for the time being the new routines are not commented at all.

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

#### 3.1 Catcodes, $\varepsilon$ -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

```

### 3 Package *xintcore* implementation

```

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 % xintkernel 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 [2016/03/12 1.2f Expandable arithmetic on big integers (JFB)]%

```

### 3.3 Counts for holding needed constants

```

47 \ifdefined\m@ne\let\xint_c_mone\m@ne
48 \else\csname newcount\endcsname\xint_c_mone \xint_c_mone -1 \fi
49 \newcount\xint_c_x^viii \xint_c_x^viii 1000000000
50 \newcount\xint_c_x^ix \xint_c_x^ix 1000000000
51 \newcount\xint_c_x^viii_mone \xint_c_x^viii_mone 99999999
52 \newcount\xint_c_xii_e_viii \xint_c_xii_e_viii 12000000000
53 \newcount\xint_c_xi_e_viii_mone \xint_c_xi_e_viii_mone 1099999999
54 \newcount\xint_c_xii_e_viii_mone\xint_c_xii_e_viii_mone 1199999999

```

### 3.4 *\xintNum*

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

```

55 \def\xintiNum {\romannumeral0\xintinum }%
56 \def\xintinum #1%
57 {%
58 \expandafter\XINT_num_loop
59 \romannumeral &&@#1\xint_relax\xint_relax\xint_relax\xint_relax
60 \xint_relax\xint_relax\xint_relax\xint_relaxZ

```

### 3 Package *xintcore* implementation

```

61 }%
62 \let\xintNum\xintiNum \let\xintnum\xintinum
63 \def\XINT_num #1%
64 {%
65     \XINT_num_loop #1\xint_relax\xint_relax\xint_relax\xint_relax
66         \xint_relax\xint_relax\xint_relax\xint_relax\Z
67 }%
68 \def\XINT_num_loop #1#2#3#4#5#6#7#8%
69 {%
70     \xint_gob_til_xint_relax #8\XINT_num_end\xint_relax
71     \XINT_num_NumEight #1#2#3#4#5#6#7#8%
72 }%
73 \edef\XINT_num_end\xint_relax\XINT_num_NumEight #1\xint_relax #2\Z
74 {%
75     \noexpand\expandafter\space\noexpand\the\numexpr #1+\xint_c_\relax
76 }%
77 \def\XINT_num_NumEight #1#2#3#4#5#6#7#8%
78 {%
79     \ifnum \numexpr #1#2#3#4#5#6#7#8+\xint_c_= \xint_c_
80         \xint_afterfi {\expandafter\XINT_num_keepsign_a
81             \the\numexpr #1#2#3#4#5#6#7#81\relax}%
82     \else
83         \xint_afterfi {\expandafter\XINT_num_finish
84             \the\numexpr #1#2#3#4#5#6#7#8\relax}%
85     \fi
86 }%
87 \def\XINT_num_keepsign_a #1%
88 {%
89     \xint_gob_til_one#1\XINT_num_gobackto loop 1\XINT_num_keepsign_b
90 }%
91 \def\XINT_num_gobackto loop 1\XINT_num_keepsign_b {\XINT_num_loop }%
92 \def\XINT_num_keepsign_b #1{\XINT_num_loop -}%
93 \def\XINT_num_finish #1\xint_relax #2\Z { #1}%

```

### 3.5 Zeroes

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

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

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

```

94 \edef\XINT_cuz_small #1#2#3#4#5#6#7#8%
95 {%
96     \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7#8\relax
97 }%

```

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

Note 2015/11/28: with only four digits the `gob_til_fourzeroes` had proved in some old testing faster than `\ifnum` test. But with eight digits, the execution times are much closer, as I tested only now. Thus, one could as well use `\ifnum` test here. Besides the tests were not exactly for a situation like here where `\XINT_cuz_z` has two 00000000 blocks to grab.



### 3 Package *xintcore* implementation

```
98 \def\XINT_cuz #1#2#3#4#5#6#7#8#9%
99 {%
100   \xint_gob_til_R #9\XINT_cuz_e \R
101   \xint_gob_til_eightzeroes #1#2#3#4#5#6#7#8\XINT_cuz_z 00000000%
102   \XINT_cuz_done #1#2#3#4#5#6#7#8#9%
103 }%
104 \def\XINT_cuz_z 00000000\XINT_cuz_done 00000000{\XINT_cuz }%
105 \edef\XINT_cuz_done #1#2#3#4#5#6#7#8#9\R
106   {\noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7#8\relax #9}%
107 \edef\XINT_cuz_e\R #1\XINT_cuz_done #2\R
108   {\noexpand\expandafter\space\noexpand\the\numexpr #2\relax }%
```

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

```
109 \def\XINT_cuz_byviii #1#2#3#4#5#6#7#8#9%
110 {%
111   \xint_gob_til_R #9\XINT_cuz_byviii_e \R
112   \xint_gob_til_eightzeroes #1#2#3#4#5#6#7#8\XINT_cuz_byviii_z 00000000%
113   \XINT_cuz_byviii_done #1#2#3#4#5#6#7#8#9%
114 }%
115 \def\XINT_cuz_byviii_z 00000000\XINT_cuz_byviii_done 00000000{\XINT_cuz_byviii}%
116 \def\XINT_cuz_byviii_done #1\R { #1}%
117 \def\XINT_cuz_byviii_e\R #1\XINT_cuz_byviii_done #2\R{ #2}%
```

### 3.6 Blocks of eight digits

Lingua of release 1.2.

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

```
118 \def\XINT_zeroes_forviii #1#2#3#4#5#6#7#8%
119 {%
120   \xint_gob_til_R #8\XINT_zeroes_forviii_end\R\XINT_zeroes_forviii
121 }%
122 \edef\XINT_zeroes_forviii_end\R\XINT_zeroes_forviii #1#2#3#4#5#6#7#8#9\W
123 {%
124   \noexpand\expandafter\space\noexpand\xint_gob_til_one #2#3#4#5#6#7#8%
125 }%
```

This is used as

```
\the\numexpr1\XINT_rsepbyviii <8Ndigits>%
\XINT_rsepbyviii_end_A 2345678%
\XINT_rsepbyviii_end_B 2345678\relax UV
```

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

```
126 \def\XINT_rsepbyviii #1#2#3#4#5#6#7#8%
127 {%
128   \XINT_rsepbyviii_b {#1#2#3#4#5#6#7#8}%
129 }%
130 \def\XINT_rsepbyviii_b #1#2#3#4#5#6#7#8#9%
```

### 3 Package *xintcore* implementation

```

131 {%
132     #2#3#4#5#6#7#8#9\expandafter!\the\numexpr
133     1#1\expandafter.\the\numexpr 1\XINT_rsepbyviii
134 }%
135 \def\XINT_rsepbyviii_end_B #1\relax #2#3{#2.}%
136 \def\XINT_rsepbyviii_end_A #1#2\expandafter #3\relax #4#5{#2.1#5.}%

```

This is used typically as

```

\romannumeral0\expandafter\XINT_sepandrev <8Ndigits>%
\XINT_rsepbyviii_end_A 2345678%
\XINT_rsepbyviii_end_B 2345678\relax UV\R.\R.\R.\R.\R.\R.\R.\W

```

and will produce 1<8digits>!1<8digits>!1<8digits>!... where the blocks have been globally reversed. The UV here are only place holders to share same syntax as \XINT\_sepandrev\_andcount, they are gobbled (#2 in \XINT\_sepandrev\_done).

```

137 \def\XINT_sepandrev
138 {%
139     \expandafter\XINT_sepandrev_a\the\numexpr 1\XINT_rsepbyviii
140 }%
141 \def\XINT_sepandrev_a {\XINT_sepandrev_b {}}%
142 \def\XINT_sepandrev_b #1#2.#3.#4.#5.#6.#7.#8.#9.%
143 {%
144     \xint_gob_til_R #9\XINT_sepandrev_end\R
145     \XINT_sepandrev_b {#9!#8!#7!#6!#5!#4!#3!#2!#1}%
146 }%
147 \def\XINT_sepandrev_end\R\XINT_sepandrev_b #1#2\W {\XINT_sepandrev_done #1}%
148 \def\XINT_sepandrev_done #1#2!{ }%

```

This is used typically as

```

\romannumeral0\expandafter\XINT_sepandrev_andcount
\the\numexpr1\XINT_rsepbyviii <8Ndigits>%
\XINT_rsepbyviii_end_A 2345678%
\XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_iii
\R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
\R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W

```

and will produce <length>.1<8digits>!1<8digits>!1<8digits>!... where the blocks have been globally reversed and <length> is the number of blocks.

```

149 \def\XINT_sepandrev_andcount
150 {%
151     \expandafter\XINT_sepandrev_andcount_a\the\numexpr 1\XINT_rsepbyviii
152 }%
153 \def\XINT_sepandrev_andcount_a {\XINT_sepandrev_andcount_b 0.{}%
154 \def\XINT_sepandrev_andcount_b #1.#2#3.#4.#5.#6.#7.#8.#9.%
155 {%
156     \xint_gob_til_R #9\XINT_sepandrev_andcount_end\R
157     \expandafter\XINT_sepandrev_andcount_b \the\numexpr #1+\xint_c_xiv.%
158     {#9!#8!#7!#6!#5!#4!#3!#2}%
159 }%
160 \def\XINT_sepandrev_andcount_end\R
161     \expandafter\XINT_sepandrev_andcount_b\the\numexpr #1+\xint_c_xiv.#2#3#4\W
162 {\expandafter\XINT_sepandrev_andcount_done\the\numexpr \xint_c_ii*#3+#1.#2}%
163 \edef\XINT_sepandrev_andcount_done #1.#21#3!%
164     {\noexpand\expandafter\space\noexpand\the\numexpr #1-#3.}%

```

### 3 Package *xintcore* implementation

Used as `\romannumeral0\XINT_unrevbyviii 1<8d>!. . . 1<8d>!` terminated by `1\Z!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\W`.

The `\romannumeral` in `unrevbyviii_a` is for special effects (expand some token which was put as `1<token>!` at the end of the original blocks). Used by subtraction during `\XINT_sub_out`, in particular.

```
165 \def\XINT_unrevbyviii #1#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
166 {%
167   \xint_gob_til_R #9\XINT_unrevbyviii_a\R
168   \XINT_unrevbyviii {#9#8#7#6#5#4#3#2#1}%
169 }%
170 \edef\XINT_unrevbyviii_a\R\XINT_unrevbyviii #1#2\W
171   {\noexpand\expandafter\space
172   \noexpand\romannumeral`&&\noexpand\xint_gob_til_Z #1}%
```

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

```
173 \def\XINT_smallunrevbyviii #1!1#2!1#3!1#4!1#5!1#6!1#7!1#8!#9\W%
174 {%
175   \expandafter\XINT_cuz_small\xint_gob_til_Z #8#7#6#5#4#3#2#1%
176 }%
```

### 3.7 Blocks of eight, for needs of 1.2 `\xintiiDivision`.

This is used as

```
\the\numexpr\XINT_sepbyviii_andcount <8Ndigits>%
\XINT_sepbyviii_end 2345678\relax
\xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!
\xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_\W
```

It will produce `1<8d>!1<8d>!. . . 1<8d>!. 1.<count of blocks>`. Used by `\XINT_div_prepare_g` for `\XINT_div_prepare_h`.

```
177 \def\XINT_sepbyviii_andcount
178 {%
179   \expandafter\XINT_sepbyviii_andcount_a\the\numexpr\XINT_sepbyviii
180 }%
181 \def\XINT_sepbyviii #1#2#3#4#5#6#7#8%
182 {%
183   1#1#2#3#4#5#6#7#8\expandafter!\the\numexpr\XINT_sepbyviii
184 }%
185 \def\XINT_sepbyviii_end #1\relax {\relax\XINT_sepbyviii_andcount_end!}%
186 \def\XINT_sepbyviii_andcount_a {\XINT_sepbyviii_andcount_b \xint_c_.}%
187 \def\XINT_sepbyviii_andcount_b #1.#2!#3!#4!#5!#6!#7!#8!#9!%
188 {%
189   #2\expandafter!\the\numexpr#3\expandafter!\the\numexpr#4\expandafter
190   !\the\numexpr#5\expandafter!\the\numexpr#6\expandafter!\the\numexpr
191   #7\expandafter!\the\numexpr#8\expandafter!\the\numexpr#9\expandafter!\the\numexpr
192   \expandafter\XINT_sepbyviii_andcount_b\the\numexpr #1+\xint_c_viii.%
193 }%
194 \def\XINT_sepbyviii_andcount_end #1\XINT_sepbyviii_andcount_b\the\numexpr
195   #2+\xint_c_viii.#3#4\W {\expandafter.\the\numexpr #2+#3.}%
```

### 3.8 Blocks of eight, for needs of 1.2 \xintiiDivision.

This is used as

\romannumeral0\XINT\_rev\_nounsep {}<blocks 1<8d>!>\R!\R!\R!\R!\R!\R!\R!\R!\W

It reverses the blocks, keeping the 1's and ! separators. Used multiple times in division algorithm. The inserted {} here is *\*not\** optional. Attention does not make disappear a 1!.

```
196 \def\XINT_rev_nounsep #1#2!#3!#4!#5!#6!#7!#8!#9!%
197 {%
198   \xint_gob_til_R #9\XINT_rev_nounsep_end\R
199   \XINT_rev_nounsep {#9!#8!#7!#6!#5!#4!#3!#2!#1}%
200 }%
201 \def\XINT_rev_nounsep_end\R\XINT_rev_nounsep #1#2\W {\XINT_rev_nounsep_done #1}%
202 \def\XINT_rev_nounsep_done #11{ 1}%
```

This is used as

\the\numexpr\XINT\_sepbyviii\_Z <8Ndigits>\XINT\_sepbyviii\_Z\_end 2345678\relax

It produces 1<8d>!\...1<8d>!1\Z!

```
203 \def\XINT_sepbyviii_Z #1#2#3#4#5#6#7#8%
204 {%
205   1#1#2#3#4#5#6#7#8\expandafter!\the\numexpr\XINT_sepbyviii_Z
206 }%
207 \def\XINT_sepbyviii_Z_end #1\relax {\relax\Z!}%
```

This is used as

\romannumeral0\XINT\_unsep\_cuzsmall {}<blocks of 1<8d>!>1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W

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

```
208 \def\XINT_unsep_cuzsmall #11#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
209 {%
210   \xint_gob_til_R #9\XINT_unsep_cuzsmall_end\R
211   \XINT_unsep_cuzsmall {#1#2#3#4#5#6#7#8#9}%
212 }%
213 \def\XINT_unsep_cuzsmall_end\R
214   \XINT_unsep_cuzsmall #1{\XINT_unsep_cuzsmall_done #1}%
215 \def\XINT_unsep_cuzsmall_done #1\R #2\W{\XINT_cuz_small #1}%
216 \def\XINT_unsep_delim {1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W}%
```

This is used by division to remove separators from the produced quotient. The quotient is produced in the correct order. The routine will also remove leading zeroes. An extra initial block of 8 zeroes is possible and thus if present must be removed. Then the next eight digits must be cleaned of leading zeroes.

```
217 \def\XINT_div_unsepQ #11#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
218 {%
219   \xint_gob_til_R #9\XINT_div_unsepQ_end\R
220   \XINT_div_unsepQ {#1#2#3#4#5#6#7#8#9}%
221 }%
222 \def\XINT_div_unsepQ_end\R\XINT_div_unsepQ #1{\XINT_div_unsepQ_x #1}%
223 \def\XINT_div_unsepQ_x #1#2#3#4#5#6#7#8#9%
```

```

224 {%
225   \xint_gob_til_R #9\XINT_div_unsepQ_e \R
226   \xint_gob_til_eightzeroes #1#2#3#4#5#6#7#8\XINT_div_unsepQ_y 00000000%
227   \expandafter\XINT_div_unsepQ_done \the\numexpr #1#2#3#4#5#6#7#8.#9%
228 }%
229 \def\XINT_div_unsepQ_e\R\xint_gob_til_eightzeroes #1\XINT_div_unsepQ_y #2\W
230   {\the\numexpr #1\relax \Z}%
231 \def\XINT_div_unsepQ_y #1.#2\R #3\W{\XINT_cuz_small #2\Z}%
232 \def\XINT_div_unsepQ_done #1.#2\R #3\W { #1#2\Z}%

```

This is used by division to remove separators from the produced remainder. The remainder is here in correct order. It must be cleaned of leading zeroes, possibly all the way. Terminator was  $1\backslash R!1\backslash R!1\backslash R!1\backslash R!1\backslash R!1\backslash R!1\backslash R!1\backslash W$

```

233 \def\XINT_div_unsepR #11#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
234 {%
235   \xint_gob_til_R #9\XINT_div_unsepR_end\R
236   \XINT_div_unsepR {#1#2#3#4#5#6#7#8#9}%
237 }%
238 \def\XINT_div_unsepR_end\R\XINT_div_unsepR #1{\XINT_div_unsepR_done #1}%
239 \def\XINT_div_unsepR_done #1\R #2\W {\XINT_cuz #1\R}%

```

### 3.9 \xintReverseDigits

#### 1.2. Needed now by \xintLDg.

```

240 \def\XINT_microrevsep #1#2#3#4#5#6#7#8%
241 {%
242   1#8#7#6#5#4#3#2#1\expandafter!\the\numexpr\XINT_microrevsep
243 }%
244 \def\XINT_microrevsep_end #1\W #2\expandafter #3\Z{#2!}%
245 \def\xintReverseDigits {\romannumeral0\xintreversedigits}%
246 \def\xintreversedigits #1{\expandafter\XINT_reversedigits\romannumeral`&&@#1\Z}%
247 \def\XINT_reversedigits #1%
248 {%
249   \xint_UDsignfork
250   #1{\expandafter\xint_minus_thenstop\romannumeral0\XINT_reversedigits_a}%
251   -{\XINT_reversedigits_a #1}%
252   \krof
253 }%
254 \def\XINT_reversedigits_a #1\Z
255 {%
256   \expandafter\XINT_revdigits_a\the\numexpr\expandafter\XINT_microrevsep
257   \romannumeral`&&@#1{\XINT_microrevsep_end\W}\XINT_microrevsep_end
258   \XINT_microrevsep_end\XINT_microrevsep_end
259   \XINT_microrevsep_end\XINT_microrevsep_end
260   \XINT_microrevsep_end\XINT_microrevsep_end\Z
261   1\Z!1\backslash R!1\backslash R!1\backslash R!1\backslash R!1\backslash R!1\backslash R!1\backslash W
262 }%
263 \def\XINT_revdigits_a {\XINT_revdigits_b {}}%
264 \def\XINT_revdigits_b #11#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
265 {%
266   \xint_gob_til_R #9\XINT_revdigits_end\R

```

```

267 \XINT_revdigits_b {#9#8#7#6#5#4#3#2#1}%
268 }%
269 \edef\XINT_revdigits_end\R\XINT_revdigits_b #1#2\W
270 {\noexpand\expandafter\space\noexpand\xint_gob_til_Z #1}%

```

### 3.10 `\xintSgn`, `\xintiiSgn`, `\XINT_Sgn`, `\XINT_cntSgn`

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

```

271 \def\xintiiSgn {\romannumeral0\xintiisgn }%
272 \def\xintiisgn #1%
273 {%
274   \expandafter\XINT_sgn \romannumeral`&&@#1\Z%
275 }%
276 \def\xintSgn {\romannumeral0\xintsgn }%
277 \def\xintsgn #1%
278 {%
279   \expandafter\XINT_sgn \romannumeral0\xintnum{#1}\Z%
280 }%
281 \def\XINT_sgn #1#2\Z
282 {%
283   \xint_UDzerominusfork
284   #1-{ 0}%
285   0#1{ -1}%
286   0-{ 1}%
287   \krof
288 }%
289 \def\XINT_Sgn #1#2\Z
290 {%
291   \xint_UDzerominusfork
292   #1-{0}%
293   0#1{-1}%
294   0-{1}%
295   \krof
296 }%
297 \def\XINT_cntSgn #1#2\Z
298 {%
299   \xint_UDzerominusfork
300   #1-\xint_c_
301   0#1\xint_c_mone
302   0-\xint_c_i
303   \krof
304 }%

```

### 3.11 `\xintiOpp`, `\xintiiOpp`

```

305 \def\xintiiOpp {\romannumeral0\xintiioff }%
306 \def\xintiioff #1%
307 {%
308   \expandafter\XINT_opp \romannumeral`&&@#1%
309 }%
310 \def\xintiOpp {\romannumeral0\xintiopp }%
311 \def\xintiopp #1%

```

```

312 {%
313   \expandafter\XINT_opp \romannumeral0\xintnum{#1}%
314 }%
315 \def\XINT_Opp #1{\romannumeral0\XINT_opp #1}%
316 \def\XINT_opp #1%
317 {%
318   \xint_UDzerominusfork
319   #1-{ 0}%      zero
320   0#1{ }%      negative
321   0-{ -#1}%    positive
322   \krof
323 }%

```

### 3.12 \xintiAbs, \xintiiAbs

```

324 \def\xintiiAbs {\romannumeral0\xintiiabs }%
325 \def\xintiiabs #1%
326 {%
327   \expandafter\XINT_abs \romannumeral`&&@#1%
328 }%
329 \def\xintiAbs {\romannumeral0\xintiabs }%
330 \def\xintiabs #1%
331 {%
332   \expandafter\XINT_abs \romannumeral0\xintnum{#1}%
333 }%
334 \def\XINT_Abs #1{\romannumeral0\XINT_abs #1}%
335 \def\XINT_abs #1%
336 {%
337   \xint_UDsignfork
338   #1{ }%
339   -{ #1}%
340   \krof
341 }%

```

### 3.13 \xintFDg, \xintiiFDg

```

342 \def\xintiiFDg {\romannumeral0\xintiifdg }%
343 \def\xintiifdg #1%
344 {%
345   \expandafter\XINT_fdg \romannumeral`&&@#1\W\Z
346 }%
347 \def\xintFDg {\romannumeral0\xintfdg }%
348 \def\xintfdg #1%
349 {%
350   \expandafter\XINT_fdg \romannumeral0\xintnum{#1}\W\Z
351 }%
352 \def\XINT_FDg #1{\romannumeral0\XINT_fdg #1\W\Z }%
353 \def\XINT_fdg #1#2#3\Z
354 {%
355   \xint_UDzerominusfork
356   #1-{ 0}%      zero
357   0#1{ #2}%    negative
358   0-{ #1}%    positive
359   \krof
360 }%

```

### 3.14 `\xintLDg`, `\xintiLDg`

```

361 \def\xintLDg {\romannumeral0\xintldg }%
362 \def\xintldg #1{\xintiildg {\xintNum{#1}}}%
363 \def\xintiLDg {\romannumeral0\xintiildg }%
364 \def\xintiildg #1%
365 {%
366   \expandafter\XINT_ldg_done\romannumeral0%
367   \expandafter\XINT_revdigits_a\the\numexpr\expandafter\XINT_microrevsep
368   \romannumeral0\expandafter\XINT_abs
369   \romannumeral`&&@#1{\XINT_microrevsep_end\W}\XINT_microrevsep_end
370   \XINT_microrevsep_end\XINT_microrevsep_end
371   \XINT_microrevsep_end\XINT_microrevsep_end
372   \XINT_microrevsep_end\XINT_microrevsep_end\Z
373   1\Z!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
374   \Z
375 }%
376 \def\XINT_ldg_done #1#2\Z { #1}%

```

### 3.15 `\xintDouble`

```

377 \def\xintDouble {\romannumeral0\xintdouble }%
378 \def\xintdouble #1%
379 {%
380   \expandafter\XINT_dbl\romannumeral`&&@#1\Z
381 }%
382 \def\XINT_dbl #1%
383 {%
384   \xint_UDzerominusfork
385   #1-\XINT_dbl_zero
386   0#1\XINT_dbl_neg
387   0-{\XINT_dbl_pos #1}%
388   \krof
389 }%
390 \def\XINT_dbl_zero #1\Z { 0}%
391 \def\XINT_dbl_neg
392   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_dbl_pos }%
393 \def\XINT_dbl_pos #1\Z
394 {%
395   \expandafter\XINT_dbl_pos_aa
396   \romannumeral0\expandafter\XINT_sepandrev
397   \romannumeral0\XINT_zeroes_forviii #1\R\R\R\R\R\R\R\{10}0000001\W
398   #1\XINT_rsepbyviii_end_A 2345678%
399   \XINT_rsepbyviii_end_B 2345678\relax XX%
400   \R.\R.\R.\R.\R.\R.\R.\R.\W 1\Z!%
401   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
402 }%
403 \def\XINT_dbl_pos_aa
404 {%
405   \expandafter\XINT_mul_out\the\numexpr\XINT_verysmallmul 0.2!%
406 }%

```

### 3.16 `\xintHalf`

```

407 \def\xintHalf {\romannumeral0\xinthalf }%

```



```

408 \def\xinthalf #1%
409 {%
410     \expandafter\XINT_half\romannumeral`&&@#1\Z
411 }%
412 \def\XINT_half #1%
413 {%
414     \xint_UDzerominusfork
415     #1-\XINT_half_zero
416     0#1\XINT_half_neg
417     0-{\XINT_half_pos #1}%
418     \krof
419 }%
420 \def\XINT_half_zero #1\Z { 0}%
421 \def\XINT_half_neg {\expandafter\XINT_opp\romannumeral0\XINT_half_pos }%
422 \def\XINT_half_pos #1\Z
423 {%
424     \expandafter\XINT_half_pos_a
425     \romannumeral0\expandafter\XINT_sepandrev
426     \romannumeral0\XINT_zeroes_forviii #1\R\R\R\R\R\R\R\{10}0000001\W
427     #1\XINT_rsepbyviii_end_A 2345678%
428     \XINT_rsepbyviii_end_B 2345678\relax XX%
429     \R.\R.\R.\R.\R.\R.\R.\R.\W
430     1\Z!%
431     1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
432 }%
433 \def\XINT_half_pos_a
434     {\expandafter\XINT_half_pos_b\the\numexpr\XINT_verysmallmul 0.5!}%
435 \def\XINT_half_pos_b 1#1#2#3#4#5#6#7#8!1#9%
436 {%
437     \xint_gob_til_Z #9\XINT_half_small \Z
438     \XINT_mul_out 1#1#2#3#4#5#6#7!1#9%
439 }%
440 \edef\XINT_half_small \Z\XINT_mul_out 1#1!#2\W
441 {%
442     \noexpand\expandafter\space\noexpand\the\numexpr #1\relax
443 }%

```

### 3.17 *\xintDec*

```

444 \def\xintDec {\romannumeral0\xintdec }%
445 \def\xintdec #1%
446 {%
447     \expandafter\XINT_dec\romannumeral`&&@#1\Z
448 }%
449 \def\XINT_dec #1%
450 {%
451     \xint_UDzerominusfork
452     #1-\XINT_dec_zero
453     0#1\XINT_dec_neg
454     0-{\XINT_dec_pos #1}%
455     \krof
456 }%
457 \def\XINT_dec_zero #1\Z { -1}%
458 \def\XINT_dec_neg

```

```

459 {\expandafter\xint_minus_thenstop\romannumeral0\xint_inc_pos }%
460 \def\xint_dec_pos #1\Z
461 {%
462   \expandafter\xint_dec_pos_aa
463   \romannumeral0\expandafter\xint_sepandrev
464   \romannumeral0\xint_zeroes_forviii #1\R\R\R\R\R\R\R\R{10}0000001\W
465   #1\xint_rsepbyviii_end_A 2345678%
466   \xint_rsepbyviii_end_B 2345678\relax XX%
467   \R.\R.\R.\R.\R.\R.\R.\R.\W
468   \Z!\Z!\Z!\Z!\W
469 }%
470 \def\xint_dec_pos_aa {\xint_sub_aa 100000001!\Z!\Z!\Z!\Z!\W }%

```

### 3.18 \xintInc

```

471 \def\xintInc {\romannumeral0\xintinc }%
472 \def\xintinc #1%
473 {%
474   \expandafter\xint_inc\romannumeral`&&@#1\Z
475 }%
476 \def\xint_inc #1%
477 {%
478   \xint_UDzerominusfork
479   #1-\xint_inc_zero
480   0#1\xint_inc_neg
481   0-{\xint_inc_pos #1}%
482   \krof
483 }%
484 \def\xint_inc_zero #1\Z { 1}%
485 \def\xint_inc_neg {\expandafter\xint_opp\romannumeral0\xint_dec_pos }%
486 \def\xint_inc_pos #1\Z
487 {%
488   \expandafter\xint_inc_pos_aa
489   \romannumeral0\expandafter\xint_sepandrev
490   \romannumeral0\xint_zeroes_forviii #1\R\R\R\R\R\R\R\R{10}0000001\W
491   #1\xint_rsepbyviii_end_A 2345678%
492   \xint_rsepbyviii_end_B 2345678\relax XX%
493   \R.\R.\R.\R.\R.\R.\R.\R.\W
494   1\Z!1\Z!1\Z!1\Z!\W
495   1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
496 }%
497 \def\xint_inc_pos_aa {\xint_add_aa 100000001!1\Z!1\Z!1\Z!1\Z!\W }%

```

### 3.19 Core arithmetic

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

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

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

3.20 `\xintiAdd`, `\xintiiAdd`

```

498 \def\xintiAdd      {\romannumeral0\xintiadd }%
499 \def\xintiadd      #1{\expandafter\XINT_iadd\romannumeral0\xintnum{#1}\Z }%
500 \def\xintiiAdd      {\romannumeral0\xintiiadd }%
501 \def\xintiiadd      #1{\expandafter\XINT_iiadd\romannumeral`&&@#1\Z }%
502 \def\XINT_iiadd     #1#2\Z #3%
503 {%
504   \expandafter\XINT_add_nfork\expandafter #1\romannumeral`&&@#3\Z #2\Z
505 }%
506 \def\XINT_iadd      #1#2\Z #3%
507 {%
508   \expandafter\XINT_add_nfork\expandafter #1\romannumeral0\xintnum{#3}\Z #2\Z
509 }%
510 \def\XINT_add_fork   #1#2\Z #3\Z {\XINT_add_nfork #1#3\Z #2\Z}%
511 \def\XINT_add_nfork  #1#2%
512 {%
513   \xint_UDzerofork
514     #1\XINT_add_firstiszero
515     #2\XINT_add_secondiszero
516     0}%
517   \krof
518   \xint_UDsignsfork
519     #1#2\XINT_add_minusminus
520     #1-\XINT_add_minusplus
521     #2-\XINT_add_plusminus
522     --\XINT_add_plusplus
523   \krof #1#2%
524 }%
525 \def\XINT_add_firstiszero #1\krof 0#2#3\Z #4\Z { #2#3}%
526 \def\XINT_add_secondiszero #1\krof #20#3\Z #4\Z { #2#4}%
527 \def\XINT_add_minusminus #1#2%
528   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_add_pp_a {}{}}%
529 \def\XINT_add_minusplus #1#2{\XINT_sub_mm_a {}#2}%
530 \def\XINT_add_plusminus #1#2%
531   {\expandafter\XINT_opp\romannumeral0\XINT_sub_mm_a #1{}}%
532 \def\XINT_add_pp_a #1#2#3\Z
533 {%
534   \expandafter\XINT_add_pp_b
535     \romannumeral0\expandafter\XINT_sepandrev_andcount
536     \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
537     #2#3\XINT_rsepybviii_end_A 2345678%
538     \XINT_rsepybviii_end_B 2345678\relax\xint_c_ii\xint_c_iii
539     \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
540     \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
541   \X #1%
542 }%
543 \let\XINT_add_plusplus \XINT_add_pp_a
544 \def\XINT_add_pp_b #1.#2\X #3\Z
545 {%
546   \expandafter\XINT_add_checklengths
547   \the\numexpr #1\expandafter.%
548   \romannumeral0\expandafter\XINT_sepandrev_andcount

```

### 3 Package *xintcore* implementation

```

549 \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\{10}0000001\W
550 #3\XINT_rsepbyviii_end_A 2345678%
551 \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_iii
552 \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
553 \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
554 1\Z!1\Z!1\Z!1\Z!\W #21\Z!1\Z!1\Z!1\Z!\W
555 1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
556 }%

```

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

```

557 \def\XINT_add_checklengths #1.#2.%
558 {%
559 \ifnum #2>#1
560 \expandafter\XINT_add_exchange
561 \else
562 \expandafter\XINT_add_A
563 \fi
564 #1.#2.%
565 }%
566 \def\XINT_add_exchange #1.#2.#3\W #4\W
567 {%
568 \XINT_add_A #2.#1.#4\W #3\W
569 }%
570 \def\XINT_add_A #1.#2.%
571 {%
572 \ifnum #1>\xint_c_vi
573 \expandafter\XINT_add_aa
574 \else \expandafter\XINT_add_aa_small
575 \fi
576 }%
577 \def\XINT_add_aa {\expandafter\XINT_add_out\the\numexpr\XINT_add_a \xint_c_ii}%
578 \def\XINT_add_out{\expandafter\XINT_cuz_small\romannumeral0\XINT_unrevbyviii {}}%
579 \def\XINT_add_aa_small
580 {\expandafter\XINT_smallunrevbyviii\the\numexpr\XINT_add_a \xint_c_ii}%

```

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

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

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

```

581 \def\XINT_add_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
582 {%
583 \XINT_add_b #1!#6!#2!#7!#3!#8!#4!#9!#5\W
584 }%
585 \def\XINT_add_b #11#2#3!#4!%
586 {%
587 \xint_gob_til_Z #2\XINT_add_bi \Z
588 \expandafter\XINT_add_c\the\numexpr#1+1#2#3+#4-\xint_c_ii.%

```

### 3 Package *xintcore* implementation

```
589 }%
590 \def\XINT_add_bi\Z\expandafter\XINT_add_c
591   \the\numexpr#1+#2+#3-\xint_c_ii.#4!#5!#6!#7!#8!#9!\W
592 {%
593   \XINT_add_k #1#3!#5!#7!#9!%
594 }%
595 \def\XINT_add_c #1#2.%
596 {%
597   1#2\expandafter!\the\numexpr\XINT_add_d #1%
598 }%
599 \def\XINT_add_d #11#2#3!#4!%
600 {%
601   \xint_gob_til_Z #2\XINT_add_di \Z
602   \expandafter\XINT_add_e\the\numexpr#1+1#2#3+#4-\xint_c_ii.%
603 }%
604 \def\XINT_add_di\Z\expandafter\XINT_add_e
605   \the\numexpr#1+#2+#3-\xint_c_ii.#4!#5!#6!#7!#8\W
606 {%
607   \XINT_add_k #1#3!#5!#7!%
608 }%
609 \def\XINT_add_e #1#2.%
610 {%
611   1#2\expandafter!\the\numexpr\XINT_add_f #1%
612 }%
613 \def\XINT_add_f #11#2#3!#4!%
614 {%
615   \xint_gob_til_Z #2\XINT_add_fi \Z
616   \expandafter\XINT_add_g\the\numexpr#1+1#2#3+#4-\xint_c_ii.%
617 }%
618 \def\XINT_add_fi\Z\expandafter\XINT_add_g
619   \the\numexpr#1+#2+#3-\xint_c_ii.#4!#5!#6\W
620 {%
621   \XINT_add_k #1#3!#5!%
622 }%
623 \def\XINT_add_g #1#2.%
624 {%
625   1#2\expandafter!\the\numexpr\XINT_add_h #1%
626 }%
627 \def\XINT_add_h #11#2#3!#4!%
628 {%
629   \xint_gob_til_Z #2\XINT_add_hi \Z
630   \expandafter\XINT_add_i\the\numexpr#1+1#2#3+#4-\xint_c_ii.%
631 }%
632 \def\XINT_add_hi\Z
633   \expandafter\XINT_add_i\the\numexpr#1+#2+#3-\xint_c_ii.#4\W
634 {%
635   \XINT_add_k #1#3!%
636 }%
637 \def\XINT_add_i #1#2.%
638 {%
639   1#2\expandafter!\the\numexpr\XINT_add_a #1%
640 }%
```

### 3 Package *xintcore* implementation

```
641 \def\XINT_add_k #1{\if #12\expandafter\XINT_add_ke\else\expandafter\XINT_add_l \fi}%
642 \def\XINT_add_ke #11\Z #2\W {\XINT_add_kf #11\Z!}%
643 \def\XINT_add_kf 1{1\relax}%
644 \def\XINT_add_l 1#1#2{\xint_gob_til_Z #1\XINT_add_lf \Z \XINT_add_m 1#1#2}%
645 \def\XINT_add_lf #1\W {1\relax 00000001!1\Z!}%
646 \def\XINT_add_m #1!\{\expandafter\XINT_add_n\the\numexpr\xint_c_i+#1.%
647 \def\XINT_add_n #1#2.{1#2\expandafter!\the\numexpr\XINT_add_o #1}%
```

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

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

#### 3.21 *\xintiSub*, *\xintiiSub*

Entirely rewritten for 1.2.

```
649 \def\xintiiSub {\romannumeral0\xintiisub}%
650 \def\xintiisub #1{\expandafter\XINT_iisub\romannumeral`&&@#1\Z}%
651 \def\XINT_iisub #1#2\Z #3%
652 {%
653   \expandafter\XINT_sub_nfork\expandafter #1\romannumeral`&&@#3\Z #2\Z
654}%
655 \def\xintiSub {\romannumeral0\xintisub}%
656 \def\xintisub #1{\expandafter\XINT_isub\romannumeral0\xintnum{#1}\Z}%
657 \def\XINT_isub #1#2\Z #3%
658 {%
659   \expandafter\XINT_sub_nfork\expandafter #1\romannumeral0\xintnum{#3}\Z #2\Z
660}%
661 \def\XINT_sub_nfork #1#2%
662 {%
663   \xint_UDzerofork
664   #1\XINT_sub_firstiszero
665   #2\XINT_sub_secondiszero
666   0{}}%
667 \krof
668 \xint_UDsignsfork
669   #1#2\XINT_sub_minusminus
670   #1-\XINT_sub_minusplus
671   #2-\XINT_sub_plusminus
672   --\XINT_sub_plusplus
673 \krof #1#2%
674}%
675 \def\XINT_sub_firstiszero #1\krof 0#2#3\Z #4\Z {\XINT_opp #2#3}%
676 \def\XINT_sub_secondiszero #1\krof #20#3\Z #4\Z { #2#4}%
677 \def\XINT_sub_plusminus #1#2{\XINT_add_pp_a #1{}}%
678 \def\XINT_sub_plusplus #1#2%
679   {\expandafter\XINT_opp\romannumeral0\XINT_sub_mm_a #1#2}%
680 \def\XINT_sub_minusplus #1#2%
681   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_add_pp_a {#2}%
682 \def\XINT_sub_minusminus #1#2{\XINT_sub_mm_a {}}}%
683 \def\XINT_sub_mm_a #1#2#3\Z
684 {%
685   \expandafter\XINT_sub_mm_b
686   \romannumeral0\expandafter\XINT_sepandrev_andcount
```

### 3 Package *xintcore* implementation

```

687 \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R{10}0000001\W
688 #2#3\XINT_rsepbyviii_end_A 2345678%
689 \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_iii
690 \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
691 \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
692 \X #1%
693 }%
694 \def\XINT_sub_mm_b #1.#2\X #3\Z
695 {%
696 \expandafter\XINT_sub_checklengths
697 \the\numexpr #1\expandafter.%
698 \romannumeral0\expandafter\XINT_sepandrev_andcount
699 \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R{10}0000001\W
700 #3\XINT_rsepbyviii_end_A 2345678%
701 \XINT_rsepbyviii_end_B 2345678\relax \xint_c_ii\xint_c_iii
702 \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
703 \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
704 \Z!\Z!\Z!\Z!\W #2\Z!\Z!\Z!\Z!\W
705 }%
706 \def\XINT_sub_checklengths #1.#2.%
707 {%
708 \ifnum #2>#1
709 \expandafter\XINT_sub_exchange
710 \else
711 \expandafter\XINT_sub_aa
712 \fi
713 }%
714 \def\XINT_sub_exchange #1\W #2\W
715 {%
716 \expandafter\XINT_opp\romannumeral0\XINT_sub_aa #2\W #1\W
717 }%
718 \def\XINT_sub_aa {\expandafter\XINT_sub_out\the\numexpr\XINT_sub_a \xint_c_i }%

The {} after \XINT_unrevbyviii could be removed, but attention then at \XINT_sub_startrescue
which must be modified (no need for #1).

719 \def\XINT_sub_out #1\Z #2#3\W
720 {%
721 \if-#2\expandafter\XINT_sub_startrescue\fi
722 \expandafter\XINT_cuz_small
723 \romannumeral0\XINT_unrevbyviii { }#11\Z!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
724 }%

```

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

Call: `\the\numexpr \XINT_sub_a 1#1\Z!\Z!\Z!\Z!\W #2\Z!\Z!\Z!\Z!\W` where #1 and #2 are blocks of 1<8d>!, and #1 \*must\* be at most as long as #2.

The routine wants to compute #2-#1.

Notice that currently the terminators on input differ from those for addition. Also, currently (1.2f) the routine can not be called without final reversal and clean up of the result.

`\numexpr` governed expansion stops with various possibilities:

1. #1 was shorter (in number of 8 digits blocks) than #2.

\*1a There may be no carry in which case we end up with

1<8d>!...1<8d>!\Z!\Z!\Z!\Z!\W

If there is a carry things are more complicated.

### 3 Package *xintcore* implementation

\*1b If the first hit block of #2 is > 1 no problem we are like in the no-carry case.

\*1c If it is exactly 1 then we will have leading zeros; but there may be also before that arbitrarily many produced zeros, all these leading zeros will have to be cleaned up. This is done via ending the expansion with the shape

1<8d>!...1<8d>!1\XINT\_cuz\_byviii!\Z 0\W\R

\*1d If the block value is zero, subtraction produces 99999999 and goes on. This is the only situation where the carry can propagate. This case can never produce extra blocks of leading zeros but may well end up with an ending zero block. In this subcase, the \numexpr is then made to stop with a 1!. This 1! will disappear during final reverse.

2a. #1 was of same length as #2, but <= #2. Then we end up expansion with

1<8d>!...1<8d>!1\XINT\_cuz\_byviii!\Z 0\W\R

and the blocks will have to be cleaned up of leading zeroes after reversal.

2b. #1 was of same length as #2, but > #2. Then we end up with blocks 1<8d>!...1<8d>! followed by -1\Z-\W

Thus \XINT\_sub\_out examines the token after the first \Z, which may be ! or 0 or -. If ! or 0, \XINT\_unrevbyviii will be executed (gobbling a possible final 1!), and followed in case 2a or 1c by \XINT\_cuz\_byviii (note the extra \R which terminates it), and then in both 1\* and 2a by \XINT\_cuz\_small.

If we were in 2b we proceed to \XINT\_sub\_startrescue which I will comment another day (the extra -1 at the end from -1\Z-\W will become a -1! and the - will serve in \XINT\_sub\_rescue\_d as loop terminator).

Currently (1.2f) we can not easily use these low level routines in a binary split approach due to the fact that first input must be at most as long as second but also because the final reversal is not in a common second stage, due to the separate treatment for case 2b.

For the record: subtraction was correct (I think) in xint releases up to 1.2, but 1.2 had a broken treatment of the 1d case. For example \xintiiSub {10000000112345678}{12345679} produced 99999999. This got fixed in 1.2c, but that fix broke the 1c case :((, for example \xintiiSub {10000000000000000}{9999999999999997} was now returning 00000003. Alas.

This was only realized later on 2016/02/29 (in fact it impacted \xintiiSqrt). Hopefully 1.2f got it right at last.

```
725 \def\XINT_sub_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
726 {%
727     \XINT_sub_b #1!#6!#2!#7!#3!#8!#4!#9!#5\W
728 }%
729 \def\XINT_sub_b #1#2#3!#4!%
730 {%
731     \xint_gob_til_Z #2\XINT_sub_bi \Z
732     \expandafter\XINT_sub_c\the\numexpr#1+1#4-#3-\xint_c_i.%
733 }%
734 \def\XINT_sub_c 1#1#2.%
735 {%
736     1#2\expandafter!\the\numexpr\XINT_sub_d #1%
737 }%
738 \def\XINT_sub_d #1#2#3!#4!%
739 {%
740     \xint_gob_til_Z #2\XINT_sub_di \Z
741     \expandafter\XINT_sub_e\the\numexpr#1+1#4-#3-\xint_c_i.%
742 }%
743 \def\XINT_sub_e 1#1#2.%
744 {%
745     1#2\expandafter!\the\numexpr\XINT_sub_f #1%
746 }%
```



### 3 Package *xintcore* implementation

```

747 \def\XINT_sub_f #1#2#3!#4!%
748 {%
749     \xint_gob_til_Z #2\XINT_sub_fi \Z
750     \expandafter\XINT_sub_g\the\numexpr#1+1#4-#3-\xint_c_i.%
751 }%
752 \def\XINT_sub_g 1#1#2.%
753 {%
754     1#2\expandafter!\the\numexpr\XINT_sub_h #1%
755 }%
756 \def\XINT_sub_h #1#2#3!#4!%
757 {%
758     \xint_gob_til_Z #2\XINT_sub_hi \Z
759     \expandafter\XINT_sub_i\the\numexpr#1+1#4-#3-\xint_c_i.%
760 }%
761 \def\XINT_sub_i 1#1#2.%
762 {%
763     1#2\expandafter!\the\numexpr\XINT_sub_a #1%
764 }%
765 \def\XINT_sub_bi\Z
766     \expandafter\XINT_sub_c\the\numexpr#1+1#2-#3.#4!#5!#6!#7!#8!#9!\W
767 {%
768     \XINT_sub_k #1#2!#5!#7!#9!%
769 }%
770 \def\XINT_sub_di\Z
771     \expandafter\XINT_sub_e\the\numexpr#1+1#2-#3.#4!#5!#6!#7!#8\W
772 {%
773     \XINT_sub_k #1#2!#5!#7!%
774 }%
775 \def\XINT_sub_fi\Z
776     \expandafter\XINT_sub_g\the\numexpr#1+1#2-#3.#4!#5!#6\W
777 {%
778     \XINT_sub_k #1#2!#5!%
779 }%
780 \def\XINT_sub_hi\Z
781     \expandafter\XINT_sub_i\the\numexpr#1+1#2-#3.#4\W
782 {%
783     \XINT_sub_k #1#2!%
784 }%

```

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

If not, then we are certain that even if there is carry it will not propagate beyond the end of A. But it may propagate far transforming chains of 00000000 into 99999999, and if it does go to the final block which is just 1<00000001>!, we will have those eight zeros to clean up. (but we have to be careful that if we encounter 1<00000001>! and this is not the final block, we should not make something silly either).

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

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

```

785 \def\XINT_sub_k #1#2%

```

### 3 Package *xintcore* implementation

```

786 {%
787   \xint_gob_til_Z #2\XINT_sub_p\Z \XINT_sub_l #1#2%
788 }%
789 \def\XINT_sub_l #1{\xint_UDzerofork #1\XINT_sub_l_carry 0\XINT_sub_l_nocarry\krof}%
790 \def\XINT_sub_l_nocarry 1{1\relax }%
791 \def\XINT_sub_l_carry 1#1!\ifcase #1
792   \expandafter \XINT_sub_l_zeroa\or\expandafter\XINT_sub_l_one\else
793   \expandafter \XINT_sub_l_done\fi 1#1!}%
794 \def\XINT_sub_l_done {-\xint_c_i+}%
795 \def\XINT_sub_l_one 1#1!#2%
796 {%
797   \xint_gob_til_Z #2\XINT_sub_l_oneone\Z 1\relax 00000000!#2%
798 }%
799 \def\XINT_sub_l_oneone #1\W {1\relax \XINT_cuz_byviii!\Z 0\W\R }%
800 \def\XINT_sub_l_zeroa 1#1!\ifcase #1
801 \def\XINT_sub_l_zeroa 1#1!\ifcase #1
802   \expandafter \XINT_sub_l_zeroa\or\expandafter\XINT_sub_l_zone\else
803   \expandafter \XINT_sub_l_done\fi 1#1!}%
804 \def\XINT_sub_l_zone 1#1!#2%
805 {%
806   \xint_gob_til_Z #2\XINT_sub_l_zoneone\Z 1\relax 00000000!#2%
807 }%
808 \def\XINT_sub_l_zoneone\Z 1\relax 00000000{1}%

```

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

```

809 \def\XINT_sub_p\Z\XINT_sub_l #1#2\W
810 {%
811   \xint_UDzerofork
812   #1{-1\relax\Z -\W}%
813   0{1\relax \XINT_cuz_byviii!\Z 0\W\R }%
814   \krof
815 }%

```

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

```

816 \def\XINT_sub_startrescue\expandafter\XINT_cuz_small
817   \romannumeral0\XINT_unrevbyviii #1#2\Z!#3\W
818 {%
819   \expandafter\XINT_sub_rescue_finish
820   \the\numexpr\XINT_sub_rescue_a #2!%
821   1\Z!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W \R
822 }%
823 \def\XINT_sub_rescue_finish
824   {\expandafter-\romannumeral0\expandafter\XINT_cuz\romannumeral0\XINT_unrevbyviii {}}%
825 \def\XINT_sub_rescue_a #1!%
826 {%
827   \expandafter\XINT_sub_rescue_c\the\numexpr \xint_c_xii_e_viii-#1.%
828 }%

```

```

829 \def\XINT_sub_rescue_c #1#2.%
830 {%
831     1#2\expandafter!\the\numexpr\XINT_sub_rescue_d #1%
832 }%
833 \def\XINT_sub_rescue_d #1#2#3!%
834 {%
835     \xint_gob_til_minus #2\XINT_sub_rescue_z -%
836     \expandafter\XINT_sub_rescue_c\the\numexpr \xint_c_xii_e_viii_mone-#2#3+#1.%
837 }%
838 \def\XINT_sub_rescue_z #1.{1!}%

```

### 3.22 \xintiMul, \xintiiMul

Completely rewritten for 1.2.

```

839 \def\xintiMul {\romannumeral0\xintimul }%
840 \def\xintimul #1%
841 {%
842     \expandafter\XINT_imul\romannumeral0\xintnum{#1}\Z
843 }%
844 \def\XINT_imul #1#2\Z #3%
845 {%
846     \expandafter\XINT_mul_nfork\expandafter #1\romannumeral0\xintnum{#3}\Z #2\Z
847 }%
848 \def\xintiiMul {\romannumeral0\xintiimul }%
849 \def\xintiimul #1%
850 {%
851     \expandafter\XINT_iimul\romannumeral`&&@#1\Z
852 }%
853 \def\XINT_iimul #1#2\Z #3%
854 {%
855     \expandafter\XINT_mul_nfork\expandafter #1\romannumeral`&&@#3\Z #2\Z
856 }%

```

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

```

857 \def\XINT_mul_fork #1#2\Z #3\Z{\XINT_mul_nfork #1#3\Z #2\Z}%
858 \def\XINT_mul_nfork #1#2%
859 {%
860     \xint_UDzerofork
861     #1\XINT_mul_zero
862     #2\XINT_mul_zero
863     0}%
864 \krof
865 \xint_UDsignsfork
866     #1#2\XINT_mul_minusminus
867     #1-\XINT_mul_minusplus
868     #2-\XINT_mul_plusminus
869     --\XINT_mul_plusplus
870 \krof #1#2%
871 }%
872 \def\XINT_mul_zero #1\krof #2#3\Z #4\Z { 0}%
873 \def\XINT_mul_minusminus #1#2{\XINT_mul_plusplus {}{}}%

```

### 3 Package *xintcore* implementation

```

874 \def\XINT_mul_minusplus #1#2%
875   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_mul_plusplus {}#2}%
876 \def\XINT_mul_plusminus #1#2%
877   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_mul_plusplus #1{}}%
878 \def\XINT_mul_plusplus #1#2#3\Z
879 {%
880   \expandafter\XINT_mul_pre_b
881     \romannumeral0\expandafter\XINT_sepandrev_andcount
882     \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
883     #2#3\XINT_rsepbyviii_end_A 2345678%
884     \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_iii
885     \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
886     \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
887   \W #1%
888 }%
889 \def\XINT_mul_pre_b #1.#2\W #3\Z
890 {%
891   \expandafter\XINT_mul_checklengths
892   \the\numexpr #1\expandafter.%
893   \romannumeral0\expandafter\XINT_sepandrev_andcount
894   \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
895   #3\XINT_rsepbyviii_end_A 2345678%
896   \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_iii
897   \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
898   \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
899   1\Z!\W #21\Z!%
900   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
901 }%

```

Cooking recipe, 2015/10/05.

```

902 \def\XINT_mul_checklengths #1.#2.%
903 {%
904   \ifnum #2=\xint_c_i\expandafter\XINT_mul_smallbyfirst\fi
905   \ifnum #1=\xint_c_i\expandafter\XINT_mul_smallbysecond\fi
906   \ifnum #2<#1
907     \ifnum \numexpr (#2-\xint_c_i)*(#1-#2)<383
908       \XINT_mul_exchange
909     \fi
910   \else
911     \ifnum \numexpr (#1-\xint_c_i)*(#2-#1)>383
912       \XINT_mul_exchange
913     \fi
914   \fi
915   \XINT_mul_start
916 }%
917 \def\XINT_mul_smallbyfirst #1\XINT_mul_start 1#2!1\Z!\W
918 {%
919   \ifnum#2=\xint_c_i\expandafter\XINT_mul_oneisone\fi
920   \ifnum#2<\xint_c_xxii\expandafter\XINT_mul_verysmall\fi
921   \expandafter\XINT_mul_out\the\numexpr\XINT_smallmul 1#2!%
922 }%
923 \def\XINT_mul_smallbysecond #1\XINT_mul_start #2\W 1#3!1\Z!%

```

### 3 Package *xintcore* implementation

```

924 {%
925   \ifnum#3=\xint_c_i\expandafter\XINT_mul_oneisone\fi
926   \ifnum#3<\xint_c_xxii\expandafter\XINT_mul_verysmall\fi
927   \expandafter\XINT_mul_out\the\numexpr\XINT_smallmul 1#3!#2%
928 }%
929 \def\XINT_mul_oneisone #1!\XINT_mul_out }%
930 \def\XINT_mul_verysmall\expandafter\XINT_mul_out
931   \the\numexpr\XINT_smallmul 1#1!%
932   {\expandafter\XINT_mul_out\the\numexpr\XINT_verysmallmul 0.#1!}%
933 \def\XINT_mul_exchange #1\XINT_mul_start #2\W #3!#2\Z!%
934   {\fi\fi\XINT_mul_start #3!#2\Z!\W #2}%

```

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

```

935 \def\XINT_mul_start
936   {\expandafter\XINT_mul_out\the\numexpr\XINT_mul_loop 100000000!1\Z!\W}%
937 \def\XINT_mul_out
938   {\expandafter\XINT_cuz_small\romannumeral0\XINT_unrevbyviii {}}%

```

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

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

The delimiting patterns for addition was changed to use `1\Z!` to fit what is used on output (by necessity).

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

Ends after having produced: `1<8d>!...1<8d>!1\Z!`. The most significant digit block is the last one. It can not be `100000000!` except if naturally the loop was called with a zero operand.

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

```

939 \def\XINT_mul_loop #1\W #2\W 1#3!%
940 {%
941   \xint_gob_til_Z #3\XINT_mul_e \Z
942   \expandafter\XINT_mul_a\the\numexpr \XINT_smallmul 1#3!#2\W
943   #1\W #2\W
944 }%

```

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

### 3 Package *xintcore* implementation

```

945 \def\XINT_mul_a #1\W #2\W
946 {%
947   \expandafter\XINT_mul_b\the\numexpr
948   \XINT_add_a \xint_c_ii #21\Z!1\Z!1\Z!\W #11\Z!1\Z!1\Z!\W\W
949 }%
950 \def\XINT_mul_b 1#1!{1#1\expandafter!\the\numexpr\XINT_mul_loop }%
951 \def\XINT_mul_e\Z #1\W 1#2\W #3\W {1\relax #2}%

```

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

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

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

Multiplication by zero will produce blocks of zeros.

```

952 \def\XINT_minimulwc_a 1#1.#2.#3!#4#5#6#7#8.%
953 {%
954   \expandafter\XINT_minimulwc_b
955   \the\numexpr \xint_c_x^ix+#1+#3*#8.#3*#4#5#6#7+#2*#8.#2*#4#5#6#7.%
956 }%
957 \def\XINT_minimulwc_b 1#1#2#3#4#5#6.#7.%
958 {%
959   \expandafter\XINT_minimulwc_c
960   \the\numexpr \xint_c_x^ix+#1#2#3#4#5+#7.#6.%
961 }%
962 \def\XINT_minimulwc_c 1#1#2#3#4#5#6.#7.#8.%
963 {%
964   1#6#7\expandafter!%
965   \the\numexpr\expandafter\XINT_smallmul_a
966   \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8.%
967 }%
968 \def\XINT_smallmul 1#1#2#3#4#5!{\XINT_smallmul_a 100000000.#1#2#3#4.#5!}%
969 \def\XINT_smallmul_a #1.#2.#3!1#4!%
970 {%
971   \xint_gob_til_Z #4\XINT_smallmul_e\Z
972   \XINT_minimulwc_a #1.#2.#3!#4.#2.#3!%
973 }%
974 \def\XINT_smallmul_e\Z\XINT_minimulwc_a 1#1.#2\Z #3!%
975   {\xint_gob_til_eightzeroes #1\XINT_smallmul_f 000000001\relax #1!1\Z!}%
976 \def\XINT_smallmul_f 000000001\relax 00000000!1{1\relax}%

```

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

```

977 \def\XINT_verysmallmul #1.#2!1#3!%
978 {%
979   \xint_gob_til_Z #3\XINT_verysmallmul_e\Z
980   \expandafter\XINT_verysmallmul_a
981   \the\numexpr #2*#3+#1.#2!%
982 }%

```

### 3 Package *xintcore* implementation

```

983 \def\XINT_verysmallmul_e\Z\expandafter\XINT_verysmallmul_a\the\numexpr
984   #1+#2#3.#4!%
985 {\xint_gob_til_zero #2\XINT_verysmallmul_f 0\xint_c_x^viii+#2#3!1\Z!}%
986 \def\XINT_verysmallmul_f #1!1{1\relax}%
987 \def\XINT_verysmallmul_a #1#2.%
988 {%
989   \unless\ifnum #1#2<\xint_c_x^ix
990     \expandafter\XINT_verysmallmul_bi\else
991     \expandafter\XINT_verysmallmul_bj\fi
992   \the\numexpr \xint_c_x^ix+#1#2.%
993 }%
994 \def\XINT_verysmallmul_bj{\expandafter\XINT_verysmallmul_cj }%
995 \def\XINT_verysmallmul_cj 1#1#2.%
996   {1#2\expandafter!\the\numexpr\XINT_verysmallmul #1.}%
997 \def\XINT_verysmallmul_bi\the\numexpr\xint_c_x^ix+#1#2#3.%
998   {1#3\expandafter!\the\numexpr\XINT_verysmallmul #1#2.}%

```

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

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

```

999 \def\XINT_minimul_a #1.#2!#3#4#5#6#7!%
1000 {%
1001   \expandafter\XINT_minimul_b
1002   \the\numexpr \xint_c_x^viii+#2*#7.#2*#3#4#5#6+#1*#7.#1*#3#4#5#6.%
1003 }%
1004 \def\XINT_minimul_b 1#1#2#3#4#5.#6.%
1005 {%
1006   \expandafter\XINT_minimul_c
1007   \the\numexpr \xint_c_x^ix+#1#2#3#4+#6.#5.%
1008 }%
1009 \def\XINT_minimul_c 1#1#2#3#4#5#6.#7.#8.%
1010 {%
1011   1#6#7\expandafter!\the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8!%
1012 }%

```

### 3.23 \xintiSqr, \xintiiSqr

Rewritten for 1.2.

```

1013 \def\xintiiSqr {\romannumeral0\xintiisqr }%
1014 \def\xintiisqr #1%
1015 {%
1016   \expandafter\XINT_sqr\romannumeral0\xintiiabs{#1}\Z
1017 }%
1018 \def\xintiSqr {\romannumeral0\xintisqr }%
1019 \def\xintisqr #1%
1020 {%
1021   \expandafter\XINT_sqr\romannumeral0\xintiabs{#1}\Z
1022 }%
1023 \def\XINT_sqr #1\Z
1024 {%
1025   \expandafter\XINT_sqr_a

```

### 3 Package *xintcore* implementation

```

1026 \romannumeral0\expandafter\XINT_sepandrev_andcount
1027 \romannumeral0\XINT_zeroes_forviii #1\R\R\R\R\R\R\R\R{10}0000001\W
1028 #1\XINT_rsepbyviii_end_A 2345678%
1029 \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_iii
1030 \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
1031 \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
1032 \Z
1033 }%

```

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

```

1034 \def\XINT_sqr_a #1.%
1035 {%
1036 \ifnum #1=\xint_c_i \expandafter\XINT_sqr_small
1037 \else\expandafter\XINT_sqr_start\fi
1038 }%
1039 \def\XINT_sqr_small 1#1#2#3#4#5!\Z
1040 {%
1041 \ifnum #1#2#3#4#5<46341 \expandafter\XINT_sqr_verysmall\fi
1042 \expandafter\XINT_sqr_small_out
1043 \the\numexpr\XINT_minimul_a #1#2#3#4.#5!#1#2#3#4#5!%
1044 }%
1045 \edef\XINT_sqr_verysmall
1046 \expandafter\XINT_sqr_small_out\the\numexpr\XINT_minimul_a #1!#2!%
1047 {\noexpand\expandafter\space\noexpand\the\numexpr #2*#2\relax}%
1048 \def\XINT_sqr_small_out 1#1!1#2!%
1049 {%
1050 \XINT_cuz #2#1\R
1051 }%

```

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

```

1052 \def\XINT_sqr_start #1\Z
1053 {%
1054 \expandafter\XINT_mul_out
1055 \the\numexpr\XINT_mul_loop 100000000!1\Z!\W #11\Z!\W #11\Z!%
1056 1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1057 }%

```

#### 3.24 `\xintiPow`, `\xintiiPow`

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

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

```

1058 \def\xintiiPow {\romannumeral0\xintiipow }%
1059 \def\xintiipow #1#2%
1060 {%
1061 \expandafter\xint_pow\the\numexpr #2\expandafter.\romannumeral`&&@#1\Z%

```



### 3 Package *xintcore* implementation

```

1062 }%
1063 \def\xintiPow {\romannumeral0\xintipow }%
1064 \def\xintipow #1#2%
1065 {%
1066   \expandafter\xint_pow\the\numexpr #2\expandafter.\romannumeral0\xintnum{#1}\Z%
1067 }%
1068 \def\xint_pow #1.#2#3\Z
1069 {%
1070   \xint_UDzerominusfork
1071   #2-\XINT_pow_AisZero
1072   0#2\XINT_pow_Aneg
1073   0-{\XINT_pow_Apos #2}%
1074   \krof {#1}%
1075 }%
1076 \def\XINT_pow_AisZero #1#2\Z
1077 {%
1078   \ifcase\XINT_cntSgn #1\Z
1079     \xint_afterfi { 1}%
1080   \or
1081     \xint_afterfi { 0}%
1082   \else
1083     \xint_afterfi {\xintError:DivisionByZero\space 0}%
1084   \fi
1085 }%
1086 \def\XINT_pow_Aneg #1%
1087 {%
1088   \ifodd #1
1089     \expandafter\XINT_opp\romannumeral0%
1090   \fi
1091   \XINT_pow_Apos {}{#1}%
1092 }%
1093 \def\XINT_pow_Apos #1#2{\XINT_pow_Apos_a {#2}#1}%
1094 \def\XINT_pow_Apos_a #1#2#3%
1095 {%
1096   \xint_gob_til_Z #3\XINT_pow_Apos_short\Z
1097   \XINT_pow_AatleastTwo {#1}#2#3%
1098 }%
1099 \def\XINT_pow_Apos_short\Z\XINT_pow_AatleastTwo #1#2\Z
1100 {%
1101   \ifcase #2
1102     \xintError:thiscannothappen!
1103   \or \expandafter\XINT_pow_AisOne
1104   \else\expandafter\XINT_pow_AatleastTwo
1105   \fi {#1}#2\Z
1106 }%
1107 \def\XINT_pow_AisOne #1\Z{ 1}%
1108 \def\XINT_pow_AatleastTwo #1%
1109 {%
1110   \ifcase\XINT_cntSgn #1\Z
1111     \expandafter\XINT_pow_BisZero
1112   \or
1113     \expandafter\XINT_pow_I_in

```

### 3 Package *xintcore* implementation

```

1114 \else
1115 \expandafter\XINT_pow_BisNegative
1116 \fi
1117 {#1}%
1118 }%
1119 \edef\XINT_pow_BisNegative #1\Z
1120 {\noexpand\xintError:FractionRoundedToZero\space 0}%
1121 \def\XINT_pow_BisZero #1\Z{ 1}%

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

```

```

1122 \def\XINT_pow_I_in #1#2\Z
1123 {%
1124 \expandafter\XINT_pow_I_loop
1125 \the\numexpr #1\expandafter.%
1126 \romannumeral0\expandafter\XINT_sepandrev
1127 \romannumeral0\XINT_zeroes_forviii #2\R\R\R\R\R\R\R\R{10}0000001\W
1128 #2\XINT_rsepbyviii_end_A 2345678%
1129 \XINT_rsepbyviii_end_B 2345678\relax XX%
1130 \R.\R.\R.\R.\R.\R.\R.\R.\W 1\Z!\W
1131 1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1132 }%
1133 \def\XINT_pow_I_loop #1.%
1134 {%
1135 \ifnum #1 = \xint_c_i\expandafter\XINT_pow_I_exit\fi
1136 \ifodd #1
1137 \expandafter\XINT_pow_II_in
1138 \else
1139 \expandafter\XINT_pow_I_squareit
1140 \fi #1.%
1141 }%
1142 \def\XINT_pow_I_exit \ifodd #1\fi #2.#3\W {\XINT_mul_out #3}%

```

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

```

1143 \def\XINT_pow_I_squareit #1.#2\W%
1144 {%
1145 \expandafter\XINT_pow_I_loop
1146 \the\numexpr #1/\xint_c_ii\expandafter.%
1147 \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W
1148 }%
1149 \def\XINT_pow_mulbutcheckifsmall #1!1#2%
1150 {%
1151 \xint_gob_til_Z #2\XINT_pow_mul_small\Z
1152 \XINT_mul_loop 100000000!1\Z!\W #1!1#2%
1153 }%
1154 \def\XINT_pow_mul_small\Z \XINT_mul_loop 100000000!1\Z!\W 1#1!1\Z!\W
1155 {%
1156 \XINT_smallmul 1#1!%
1157 }%
1158 \def\XINT_pow_II_in #1.#2\W

```

```

1159 {%
1160   \expandafter\XINT_pow_II_loop
1161   \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter.%
1162   \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W #2\W
1163 }%
1164 \def\XINT_pow_II_loop #1.%
1165 {%
1166   \ifnum #1 = \xint_c_i\expandafter\XINT_pow_II_exit\fi
1167   \ifodd #1
1168     \expandafter\XINT_pow_II_odda
1169   \else
1170     \expandafter\XINT_pow_II_even
1171   \fi #1.%
1172 }%
1173 \def\XINT_pow_II_exit\ifodd #1\fi #2.#3\W #4\W
1174 {%
1175   \expandafter\XINT_mul_out
1176   \the\numexpr\XINT_pow_mulbutcheckifsmall #4\W #3%
1177 }%
1178 \def\XINT_pow_II_even #1.#2\W
1179 {%
1180   \expandafter\XINT_pow_II_loop
1181   \the\numexpr #1/\xint_c_ii\expandafter.%
1182   \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W
1183 }%
1184 \def\XINT_pow_II_odda #1.#2\W #3\W
1185 {%
1186   \expandafter\XINT_pow_II_oddb
1187   \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter.%
1188   \the\numexpr\XINT_pow_mulbutcheckifsmall #3\W #2\W #2\W
1189 }%
1190 \def\XINT_pow_II_oddb #1.#2\W #3\W
1191 {%
1192   \expandafter\XINT_pow_II_loop
1193   \the\numexpr #1\expandafter.%
1194   \the\numexpr\XINT_pow_mulbutcheckifsmall #3\W #3\W #2\W
1195 }%

```

### 3.25 *\xintiFac*, *\xintiiFac*

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

The routine has been partially rewritten with release 1.2 to exploit the new inner structure of multiplication. I impose an intrinsic limit of the argument at maximal value 9999 (1.2f sets it at 10000, there was no reason for 9999 and not 10000). Anyhow with current default settings of the *etex* memory and the current 1.2 routine (last commit: *eadalb1*), the maximal possible computation is 5971! (which has 19956 digits). Also, I add *\xintiiFac* which does only *\romannumeral-`0* and not *\numexpr* on its argument. This is for a silly slight optimization of the *\xintiexpr* (and *\bnumexpr*) parsers. If the argument is  $\geq 2^{31}$  an arithmetic overflow will occur in the *\ifnum*. This is not as good as in the *\numexpr*, but well.

2015/11/14 added note on the implementation: we can roughly estimate for big  $n$  that we do  $n/2$  multiplications  $\alpha \cdot X$  where  $\alpha = (k+1)(k+2) < 10^8$  and  $X = k!$  has size of order  $k \log(k)$ , with  $k$  along a step 2 arithmetic sequence up to  $n$ . Each small multiplication should have a linear cost

### 3 Package *xintcore* implementation

hence  $O(k \log(k))$  (as we maintain the reversed representation) hence a total cost of  $O(n^2 \log(n))$ ; on computing  $n!$  for  $n=100, 200, \dots, 2000$  I obtained a good fit (only roughly 20% variation) of the computation time with the square of the length of  $n!$  -- to the extent that the big variability of `\pdfelapsedtime` allows to draw any conclusion -- I did not repeat the computations as many times as I should have. I currently do not quite understand why in this range it seems computation times are better fitted by  $O(n^2 \log^2 n)$  than by  $O(n^2 \log n)$ . True, final reverse is  $O(N^2)$  with  $N$  of order  $n \log n$ , but for this range of  $n$ 's this is marginal (and I tested also with this final reverse skipped).

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

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

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

```
1196 \def\xintiiFac {\romannumeral0\xintiifac }%
1197 \def\xintiifac #1{\expandafter\XINT_fac_fork\the\numexpr#1.}%
1198 \def\xintiFac {\romannumeral0\xintifac }%
1199 \let\xintifac\xintiifac
```

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

```
1200 \def\XINT_fac_fork #1#2.%
1201 {%
1202   \xint_UDzerominusfork
1203   #1-\XINT_fac_zero
1204   0#1\XINT_fac_neg
1205   0-\XINT_fac_checksize
1206   \krof #1#2.%
1207 }%
1208 \def\XINT_fac_zero #1.{ 1}%
1209 \edef\XINT_fac_neg #1.{\noexpand\xintError:FactorialOfNegative\space 1}%
1210 \def\XINT_fac_checksize #1.%
1211 {%
1212   \ifnum #1>\xint_c_x^iv
1213     \xint_dothis{\expandafter\xintError:TooBigFactorial
1214               \expandafter\space\expandafter 1\xint_gob_til_W }\fi
1215   \ifnum #1>465 \xint_dothis{\XINT_fac_bigloop_a #1.}\fi
1216   \ifnum #1>101 \xint_dothis{\XINT_fac_medloop_a #1.\XINT_mul_out}\fi
1217     \xint_orthat{\XINT_fac_smallloop_a #1.\XINT_mul_out}%
1218   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1219 }%
1220 \def\XINT_fac_bigloop_a #1.%
1221 {%
1222   \expandafter\XINT_fac_bigloop_b \the\numexpr
1223   #1+\xint_c_i-\xint_c_ii*((#1-464)/\xint_c_ii).#1.%
```

### 3 Package *xintcore* implementation

```

1224 }%
1225 \def\XINT_fac_bigloop_b #1.#2.%
1226 {%
1227     \expandafter\XINT_fac_medloop_a
1228     \the\numexpr #1-\xint_c_i.\{\XINT_fac_bigloop_loop #1.#2.\}%
1229 }%
1230 \def\XINT_fac_bigloop_loop #1.#2.%
1231 {%
1232     \ifnum #1>#2 \expandafter\XINT_fac_bigloop_exit\fi
1233     \expandafter\XINT_fac_bigloop_loop
1234     \the\numexpr #1+\xint_c_ii\expandafter.%
1235     \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_bigloop_mul #1!%
1236 }%
1237 \def\XINT_fac_bigloop_exit #1!\{\XINT_mul_out}%
1238 \def\XINT_fac_bigloop_mul #1!%
1239 {%
1240     \expandafter\XINT_smallmul
1241     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
1242 }%
1243 \def\XINT_fac_medloop_a #1.%
1244 {%
1245     \expandafter\XINT_fac_medloop_b
1246     \the\numexpr #1+\xint_c_i-\xint_c_iii*((#1-100)/\xint_c_iii).#1.%
1247 }%
1248 \def\XINT_fac_medloop_b #1.#2.%
1249 {%
1250     \expandafter\XINT_fac_smallloop_a
1251     \the\numexpr #1-\xint_c_i.\{\XINT_fac_medloop_loop #1.#2.\}%
1252 }%
1253 \def\XINT_fac_medloop_loop #1.#2.%
1254 {%
1255     \ifnum #1>#2 \expandafter\XINT_fac_loop_exit\fi
1256     \expandafter\XINT_fac_medloop_loop
1257     \the\numexpr #1+\xint_c_iii\expandafter.%
1258     \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_medloop_mul #1!%
1259 }%
1260 \def\XINT_fac_medloop_mul #1!%
1261 {%
1262     \expandafter\XINT_smallmul
1263     \the\numexpr
1264     \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1265 }%
1266 \def\XINT_fac_smallloop_a #1.%
1267 {%
1268     \csname
1269     XINT_fac_smallloop_\the\numexpr #1-\xint_c_iv*(#1/\xint_c_iv)\relax
1270     \endcsname #1.%
1271 }%
1272 \expandafter\def\csname XINT_fac_smallloop_1\endcsname #1.%
1273 {%
1274     \XINT_fac_smallloop_loop 2.#1.100000001!1\Z!%
1275 }%

```

### 3 Package *xintcore* implementation

```

1276 \expandafter\def\csname XINT_fac_smallloop_-2\endcsname #1.%
1277 {%
1278   \XINT_fac_smallloop_loop 3.#1.100000002!1\Z!%
1279 }%
1280 \expandafter\def\csname XINT_fac_smallloop_-1\endcsname #1.%
1281 {%
1282   \XINT_fac_smallloop_loop 4.#1.100000006!1\Z!%
1283 }%
1284 \expandafter\def\csname XINT_fac_smallloop_0\endcsname #1.%
1285 {%
1286   \XINT_fac_smallloop_loop 5.#1.1000000024!1\Z!%
1287 }%
1288 \def\XINT_fac_smallloop_loop #1.#2.%
1289 {%
1290   \ifnum #1>#2 \expandafter\XINT_fac_loop_exit\fi
1291   \expandafter\XINT_fac_smallloop_loop
1292   \the\numexpr #1+\xint_c_iv\expandafter.%
1293   \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_smallloop_mul #1!%
1294 }%
1295 \def\XINT_fac_smallloop_mul #1!%
1296 {%
1297   \expandafter\XINT_smallmul
1298   \the\numexpr
1299     \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1300 }%
1301 \def\XINT_fac_loop_exit #1!#2\Z!#3{#3#2\Z!}%

```

#### 3.26 *\xintiDivision*, *\xintiQuo*, *\xintiRem*, *\xintiiDivision*, *\xintiiQuo*, *\xintiiRem*

Completely rewritten for 1.2.

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

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

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

```

1302 \def\xintiiQuo {\romannumeral0\xintiiquo }%
1303 \def\xintiiRem {\romannumeral0\xintiirem }%
1304 \def\xintiiquo {\expandafter\xint_firstoftwo_thenstop\romannumeral0\xintiidivision }%
1305 \def\xintiirem {\expandafter\xint_secondoftwo_thenstop\romannumeral0\xintiidivision }%
1306 \def\xintiQuo {\romannumeral0\xintiquo }%
1307 \def\xintiRem {\romannumeral0\xintirem }%

```

### 3 Package *xintcore* implementation

```

1308 \def\xintiquo {\expandafter\xint_firstoftwo_thenstop\romannumeral0\xintidivision}%
1309 \def\xintirem {\expandafter\xint_secondoftwo_thenstop\romannumeral0\xintidivision}%
1310 \let\xintQuo\xintiQuo\let\xintquo\xintiquo % deprecated
1311 \let\xintRem\xintiRem\let\xintrem\xintirem % deprecated

```

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

```

1312 \def\xintiDivision {\romannumeral0\xintidivision}%
1313 \def\xintidivision #1{\expandafter\XINT_idivision\romannumeral0\xintnum{#1}\Z}%
1314 \def\XINT_idivision #1#2\Z #3{\expandafter\XINT_iidivision_a\expandafter #1%
1315 \romannumeral0\xintnum{#3}\Z #2\Z}%
1316 \def\xintiiDivision {\romannumeral0\xintiidivision}%
1317 \def\xintiidivision #1{\expandafter\XINT_iidivision \romannumeral`&&@#1\Z}%
1318 \def\XINT_iidivision #1#2\Z #3{\expandafter\XINT_iidivision_a\expandafter #1%
1319 \romannumeral`&&@#3\Z #2\Z}%

```

On regarde les signes de A et de B.

```

1320 \def\XINT_iidivision_a #1#2% #1 de A, #2 de B.
1321 {%
1322   \if0#2\xint_dothis\XINT_iidivision_divbyzero\fi
1323   \if0#1\xint_dothis\XINT_iidivision_aiszero\fi
1324   \if-#2\xint_dothis{\expandafter\XINT_iidivision_bneg
1325 \romannumeral0\XINT_iidivision_bpos #1}\fi
1326   \xint_orthat{\XINT_iidivision_bpos #1#2}%
1327}%
1328 \def\XINT_iidivision_divbyzero #1\Z #2\Z {\xintError:DivisionByZero{0}{0}}%
1329 \def\XINT_iidivision_aiszero #1\Z #2\Z {{0}{0}}%
1330 \def\XINT_iidivision_bneg #1% q->-q, r unchanged
1331 {\expandafter{\romannumeral0\XINT_opp #1}}%
1332 \def\XINT_iidivision_bpos #1%
1333 {%
1334   \xint_UDsignfork
1335   #1\XINT_iidivision_aneg
1336   -{\XINT_iidivision_apos #1}%
1337   \krof
1338}%

```

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

```

1339 \def\XINT_iidivision_apos #1#2\Z #3\Z{\XINT_div_prepare {#2}{#1#3}}%
1340 \def\XINT_iidivision_aneg #1\Z #2\Z
1341 {\expandafter
1342 \XINT_iidivision_aneg_b\romannumeral0\XINT_div_prepare {#1}{#2}{#1}}%
1343 \def\XINT_iidivision_aneg_b #1#2{\if0\XINT_Sgn #2\Z
1344 \expandafter\XINT_iidivision_aneg_rzero
1345 \else
1346 \expandafter\XINT_iidivision_aneg_rpos
1347 \fi {#1}{#2}}%

```

### 3 Package *xintcore* implementation

```

1348 \def\XINT_iidivision_aneg_rzero #1#2#3{{-#1}{0}}% necessarily q was >0
1349 \def\XINT_iidivision_aneg_rpos #1%
1350 {%
1351     \expandafter\XINT_iidivision_aneg_end\expandafter
1352         {\expandafter-\romannumeral0\xintinc {#1}}% q-> -(1+q)
1353 }%
1354 \def\XINT_iidivision_aneg_end #1#2#3%
1355 {%
1356     \expandafter\xint_exchangetwo_keepbraces
1357     \expandafter{\romannumeral0\XINT_sub_mm_a {}{}#3\Z #2\Z}{#1}% r-> b-r
1358 }%

```

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

```

1359 \def\XINT_div_prepare #1%
1360 {%
1361     \XINT_div_prepare_a #1\R\R\R\R\R\R\R {10}0000001\W !{#1}%
1362 }%
1363 \def\XINT_div_prepare_a #1#2#3#4#5#6#7#8#9%
1364 {%
1365     \xint_gob_til_R #9\XINT_div_prepare_small\R
1366     \XINT_div_prepare_b #9%
1367 }%

```

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

```

1368 \def\XINT_div_prepare_small\R #1!#2%
1369 {%
1370     \ifcase #2
1371     \or\expandafter\XINT_div_BisOne
1372     \or\expandafter\XINT_div_BisTwo
1373     \else\expandafter\XINT_div_small_a
1374     \fi {#2}%
1375 }%
1376 \def\XINT_div_BisOne #1#2{{#2}{0}}%
1377 \def\XINT_div_BisTwo #1#2%
1378 {%
1379     \expandafter\expandafter\expandafter\XINT_div_BisTwo_a
1380     \ifodd\xintLDg{#2} \expandafter1\else \expandafter0\fi {#2}%
1381 }%
1382 \def\XINT_div_BisTwo_a #1#2%
1383 {%
1384     \expandafter{\romannumeral0\xinthalft {#2}}{#1}%
1385 }%

```

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

```

1386 \def\XINT_div_small_a #1#2%
1387 {%
1388     \expandafter\XINT_div_small_b
1389     \the\numexpr #1/\xint_c_ii\expandafter

```



### 3 Package *xintcore* implementation

```

1390 .\the\numexpr \xint_c_x^viii+#1\expandafter!%
1391 \romannumeral0%
1392 \XINT_div_small_ba #2\R\R\R\R\R\R\R\R{10}0000001\W
1393 #2\XINT_sepbyviii_Z_end 2345678\relax
1394 }%

```

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

```

1395 \def\XINT_div_small_b #1!#2{#2#1!}%

```

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

```

1396 \def\XINT_div_small_ba #1#2#3#4#5#6#7#8#9%
1397 {%
1398 \xint_gob_til_R #9\XINT_div_smallsmall\R
1399 \expandafter\XINT_div_dosmallldiv
1400 \the\numexpr\expandafter\XINT_sepbyviii_Z
1401 \romannumeral0\XINT_zeroes_forviii
1402 #1#2#3#4#5#6#7#8#9%
1403 }%

```

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

```

1404 \def\XINT_div_smallsmall\R
1405 \expandafter\XINT_div_dosmallldiv
1406 \the\numexpr\expandafter\XINT_sepbyviii_Z
1407 \romannumeral0\XINT_zeroes_forviii #1\R #2\relax
1408 {\XINT_div_dosmallsmall}{#1}}%
1409 \def\XINT_div_dosmallsmall #1.1#2!#3%
1410 {%
1411 \expandafter\XINT_div_smallsmallend
1412 \the\numexpr (#3+#1)/#2-\xint_c_i.#2.#3.%
1413 }%
1414 \def\XINT_div_smallsmallend #1.#2.#3.{\expandafter
1415 {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #3-#1*#2}}%

```

Si  $A \geq 10^8$ , il est maintenant sous la forme  $1<8d>!\dots 1<8d>!1\Z!$  avec plus significatifs en premier. Donc on poursuit par `\expandafter\XINT_sdiv_out\the\numexpr\XINT_smallldivx_a x.1B!1<8d>!\dots 1<8d>!1\Z!` avec  $x = \text{round}(B/2)$ ,  $1B = 10^8 + B$ .

```

1416 \def\XINT_div_dosmallldiv
1417 {\XINT_sdiv_out\the\numexpr\XINT_smallldivx_a}%

```

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

```

1418 \def\XINT_div_prepare_b
1419 {\expandafter\XINT_div_prepare_c\romannumeral0\XINT_zeroes_forviii }%
1420 \def\XINT_div_prepare_c #1!%
1421 {%
1422 \XINT_div_prepare_d #1.00000000!{#1}%
1423 }%

```

### 3 Package *xintcore* implementation

```

1424 \def\XINT_div_prepare_d #1#2#3#4#5#6#7#8#9%
1425 {%
1426   \expandafter\XINT_div_prepare_e\xint_gob_til_dot #1#2#3#4#5#6#7#8#9!%
1427 }%
1428 \def\XINT_div_prepare_e #1!#2!#3#4%
1429 {%
1430   \XINT_div_prepare_f #4#3\X {#1}{#3}%
1431 }%

```

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

```

1432 \def\XINT_div_prepare_f #1#2#3#4#5#6#7#8#9\X
1433 {%
1434   \expandafter\XINT_div_prepare_g
1435   \the\numexpr #1#2#3#4#5#6#7#8+\xint_c_i\expandafter
1436   .\the\numexpr (#1#2#3#4#5#6#7#8+\xint_c_i)/\xint_c_ii\expandafter
1437   .\the\numexpr #1#2#3#4#5#6#7#8\expandafter
1438   .\romannumeral0\XINT_sepandrev_andcount
1439   #1#2#3#4#5#6#7#8#9\XINT_rsepbyviii_end_A 2345678%
1440   \XINT_rsepbyviii_end_B 2345678%
1441   \relax\xint_c_ii\xint_c_iii
1442   \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
1443   \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
1444   \X
1445 }%
1446 \def\XINT_div_prepare_g #1.#2.#3.#4.#5\X #6#7#8%
1447 {%
1448   \expandafter\XINT_div_prepare_h
1449   \the\numexpr\expandafter\XINT_sepbyviii_andcount
1450   \romannumeral0\XINT_zeroes_forviii #8#7\R\R\R\R\R\R\R\{10}0000001\W
1451   #8#7\XINT_sepbyviii_end 2345678\relax
1452   \xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!%
1453   \xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_\W
1454   {#1}{#2}{#3}{#4}{#5}{#6}%
1455 }%
1456 \def\XINT_div_prepare_h #11.#2.#3#4#5#6#7#8%
1457 {%
1458   \XINT_div_start_a {#2}{#6}{#1}{#3}{#4}{#5}{#7}{#8}%
1459 }%

```

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

```

1460 \def\XINT_div_start_a #1#2%
1461 {%
1462   \ifnum #1 < #2
1463     \expandafter\XINT_div_zeroQ
1464   \else
1465     \expandafter\XINT_div_start_b

```

### 3 Package *xintcore* implementation

```

1466 \fi
1467 {#1}{#2}%
1468 }%
1469 \def\XINT_div_zeroQ #1#2#3#4#5#6#7%
1470 {%
1471 \expandafter\XINT_div_zeroQ_end
1472 \romannumeral0\XINT_unsep_cuzsmall
1473 #31\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W .%
1474 }%
1475 \def\XINT_div_zeroQ_end #1.#2%
1476 {\expandafter{\expandafter0\expandafter}\XINT_div_cleanR #1#2.}%

```

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

```

1477 \def\XINT_div_start_b #1#2#3#4#5#6%
1478 {%
1479 \expandafter\XINT_div_finish\the\numexpr
1480 \XINT_div_start_c {#2}.#3.{#6}{#1}{#2}{#4}{#5}{#6}}%
1481 }%
1482 \def\XINT_div_finish
1483 {%
1484 \expandafter\XINT_div_finish_a \romannumeral`&&\XINT_div_unsepQ
1485 }%
1486 \def\XINT_div_finish_a #1\Z #2.{\XINT_div_finish_b #2.{#1}}%

```

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

```

1487 \def\XINT_div_finish_b #1%
1488 {%
1489 \if0#1%
1490 \expandafter\XINT_div_finish_bRzero
1491 \else
1492 \expandafter\XINT_div_finish_bRpos
1493 \fi
1494 #1%
1495 }%
1496 \def\XINT_div_finish_bRzero 0.#1#2{{#1}{0}}%
1497 \def\XINT_div_finish_bRpos #1.#2#3%
1498 {%
1499 \expandafter\xint_exchangetwo_keepbraces\XINT_div_cleanR #1#3.{#2}%
1500 }%
1501 \def\XINT_div_cleanR #1000000000.{#1}}%

```

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

```

1502 \def\XINT_div_start_c #1%
1503 {%
1504 \ifnum #1>\xint_c_vi
1505 \expandafter\XINT_div_start_ca
1506 \else
1507 \expandafter\XINT_div_start_cb
1508 \fi {#1}%
1509 }%

```

### 3 Package *xintcore* implementation

```

1510 \def\XINT_div_start_ca #1#2.#3!#4!#5!#6!#7!#8!#9!%
1511 {%
1512     \expandafter\XINT_div_start_c\expandafter
1513     {\the\numexpr #1-\xint_c_vii}\#2#3!#4!#5!#6!#7!#8!#9!.%
1514 }%
1515 \def\XINT_div_start_cb #1%
1516     {\csname XINT_div_start_c_\romannumeral\numexpr#1\endcsname}%
1517 \def\XINT_div_start_c_i #1.#2!%
1518     {\XINT_div_start_c_ #1#2!.%}%
1519 \def\XINT_div_start_c_ii #1.#2!#3!%
1520     {\XINT_div_start_c_ #1#2!#3!.%}%
1521 \def\XINT_div_start_c_iii #1.#2!#3!#4!%
1522     {\XINT_div_start_c_ #1#2!#3!#4!.%}%
1523 \def\XINT_div_start_c_iv #1.#2!#3!#4!#5!%
1524     {\XINT_div_start_c_ #1#2!#3!#4!#5!.%}%
1525 \def\XINT_div_start_c_v #1.#2!#3!#4!#5!#6!%
1526     {\XINT_div_start_c_ #1#2!#3!#4!#5!#6!.%}%
1527 \def\XINT_div_start_c_vi #1.#2!#3!#4!#5!#6!#7!%
1528     {\XINT_div_start_c_ #1#2!#3!#4!#5!#6!#7!.%}%

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

1529 \def\XINT_div_start_c_ 1#1!#2.#3.#4#5#6%
1530 {%
1531     \XINT_div_I_a {#1}{#4}{1#1!#2}{#6}{00000000}#5{#3}{#6}%
1532 }%

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

1533 \def\XINT_div_I_a #1#2%
1534 {%
1535     \expandafter\XINT_div_I_b\the\numexpr #1/#2.{#1}{#2}%
1536 }%
1537 \def\XINT_div_I_b #1%
1538 {%
1539     \xint_gob_til_zero #1\XINT_div_I_czero 0\XINT_div_I_c #1%
1540 }%

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

1541 \def\XINT_div_I_czero 0\XINT_div_I_c 0.#1#2#3#4#5{1#5\XINT_div_I_g {#3}}%
1542 \def\XINT_div_I_c #1.#2#3%
1543 {%
1544     \expandafter\XINT_div_I_da\the\numexpr #2-#1*#3.#1.{#2}{#3}%
1545 }%

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

1546 \def\XINT_div_I_da #1.%
1547 {%
1548     \ifnum #1>\xint_c_ix
1549         \expandafter\XINT_div_I_dP

```

### 3 Package *xintcore* implementation

```

1550 \else
1551 \ifnum #1<\xint_c_
1552 \expandafter\expandafter\expandafter\XINT_div_I_dN
1553 \else
1554 \expandafter\expandafter\expandafter\XINT_div_I_db
1555 \fi
1556 \fi
1557 }%

```

attention très mauvaises notations avec *\_b* et *\_db*.

```

1558 \def\XINT_div_I_dN #1.%
1559 {%
1560 \expandafter\XINT_div_I_b\the\numexpr #1-\xint_c_i.%
1561 }%
1562 \def\XINT_div_I_db #1.#2#3#4#5%
1563 {%
1564 \expandafter\XINT_div_I_dc\expandafter #1%
1565 \romannumeral0\expandafter\XINT_div_sub\expandafter
1566 {\romannumeral0\XINT_rev_nounsep {}#4\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1567 {\the\numexpr\XINT_div_verysmallmul #1!#51\Z!}%
1568 \Z {}#4}{}#5}%
1569 }%

```

La soustraction spéciale renvoie simplement - si le chiffre *q* est trop grand. On invoque dans ce cas *I\_dP*.

```

1570 \def\XINT_div_I_dc #1#2%
1571 {%
1572 \if-#2\expandafter\XINT_div_I_dd\else\expandafter\XINT_div_I_de\fi
1573 #1#2%
1574 }%
1575 \def\XINT_div_I_dd #1-\Z
1576 {%
1577 \if #11\expandafter\XINT_div_I_dz\fi
1578 \expandafter\XINT_div_I_dP\the\numexpr #1-\xint_c_i.XX%
1579 }%
1580 \def\XINT_div_I_dz #1XX#2#3#4%
1581 {%
1582 1#4\XINT_div_I_g {}#2}%
1583 }%
1584 \def\XINT_div_I_de #1#2\Z #3#4#5{1#5+#1\XINT_div_I_g {}#2}}%

```

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

```

1585 \def\XINT_div_I_dP #1.#2#3#4#5#6%
1586 {%
1587 1#6+#1\expandafter\XINT_div_I_g\expandafter
1588 {\romannumeral0\expandafter\XINT_div_sub\expandafter
1589 {\romannumeral0\XINT_rev_nounsep {}#4\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1590 {\the\numexpr\XINT_div_verysmallmul #1!#51\Z!}%
1591 }%
1592 }%

```

### 3 Package *xintcore* implementation

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

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

```
1593 \def\XINT_div_I_g #1#2#3#4#5#6#7%
1594 {%
1595     \expandafter !\the\numexpr
1596     \ifnum#2=#3
1597         \expandafter\XINT_div_exittofinish
1598     \else
1599         \expandafter\XINT_div_I_h
1600     \fi
1601     {#4}#1.#6.{{#4}{#5}{#3}{#2}}{#7}%
1602 }%
```

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

```
1603 \def\XINT_div_exittofinish #1#2.#3.#4#5%
1604 {%
1605     1\expandafter\expandafter\expandafter!\expandafter\XINT_unsep_delim
1606     \romannumeral0\XINT_div_unsepR #2#31\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W.%
1607 }%
```

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

```
1608 \def\XINT_div_I_h #1.#2!#3.#4#5%
1609 {%
1610     \XINT_div_II_b #1#2!.{{#5}{#4}{#3}}{#5}%
1611 }%
```

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

```
1612 \def\XINT_div_II_b #11#2!#3!%
1613 {%
1614     \xint_gob_til_eightzeroes #2\XINT_div_II_skipc 00000000%
1615     \XINT_div_II_c #1{1#2}{#3}%
1616 }%
```

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

```
1617 \def\XINT_div_II_skipc 00000000\XINT_div_II_c #1#2#3#4#5.#6#7%
1618 {%
1619     \XINT_div_II_k #7{#4!#5}{#6}{00000000}%
1620 }%
```

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

ATTENTION!

### 3 Package *xintcore* implementation

2015/10/29 :j'avais introduit un bug ici dans 1.2 2015/10/15, car \XINT\_div\_mini veut un diviseur de huit chiffres, or si le dénominateur B débute par x=99999999, on aura x'=100000000, d'où évidemment un bug. Bon il faut intercepter x'=100000000.

I need to recognize x'=100000000 in some not too penalizing way. Anyway, will try to optimize some other day.

```
1621 \def\XINT_div_II_c #1#2#3#4%
1622 {%
1623   \expandafter\XINT_div_II_d\the\numexpr\XINT_div_xmini
1624   #1.#2!#3!#4!{#1}{#2}{#3!#4!}%
1625 }%
1626 \def\XINT_div_xmini #1%
1627 {%
1628   \xint_gob_til_one #1\XINT_div_xmini_a 1\XINT_div_mini #1%
1629 }%
1630 \def\XINT_div_xmini_a 1\XINT_div_mini 1#1%
1631 {%
1632   \xint_gob_til_zero #1\XINT_div_xmini_b 0\XINT_div_mini 1#1%
1633 }%
1634 \def\XINT_div_xmini_b 0\XINT_div_mini 10#1#2#3#4#5#6#7%
1635 {%
1636   \xint_gob_til_zero #7\XINT_div_xmini_c 0\XINT_div_mini 10#1#2#3#4#5#6#7%
1637 }%
```

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

```
1638 \def\XINT_div_xmini_c 0\XINT_div_mini 100000000.50000000!#1!#2!{#1!}%
```

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

```
1639 \def\XINT_div_II_d 1#1#2#3#4#5!#6#7#8.#9%
1640 {%
1641   \expandafter\XINT_div_II_e
1642   \romannumeral0\expandafter\XINT_div_sub\expandafter
1643   {\romannumeral0\XINT_rev_nounsep {#8\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1644   {\the\numexpr\XINT_div_smallmul_a 100000000.#1#2#3#4.#5!#9!#Z!}%
1645   .{#6}{#7}{#9}{#1#2#3#4#5}%
1646 }%
```

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

```
1647 \def\XINT_div_II_e 1#1!%
1648 {%
1649   \xint_gob_til_eightzeroes #1\XINT_div_II_skipf 00000000%
1650   \XINT_div_II_f 1#1!%
1651 }%
```

10000000!alpha sur K chiffres.#2=x',#3=y,#4=B,#5=q1, #6={{x'y},x,K,L}, #7=alpha',B«c» -> {x'y}x,K,L (à diminuer de 1), {alpha sur K}B{q1}{alpha'}B«c»

```
1652 \def\XINT_div_II_skipf 00000000\XINT_div_II_f 100000000!#1.#2#3#4#5#6%
1653 {%
1654   \XINT_div_II_k #6{#1}{#4}{#5}%
1655 }%
```

### 3 Package *xintcore* implementation

1<a1>!1<a2>!, alpha (sur K+1 blocs de 8). x', y, B, q1, {{x'y},x,K,L}, alpha', B,«c».

Here also we are dividing with x' which could be  $10^8$  in the exceptional case  $x=99999999$ . Must intercept it before sending to \XINT\_div\_mini.

```
1656 \def\XINT_div_II_f #1!#2!#3.%
1657 {%
1658   \XINT_div_II_fa {#1!#2!}{#1!#2!#3}%
1659 }%
1660 \def\XINT_div_II_fa #1#2#3#4%
1661 {%
1662   \expandafter\XINT_div_II_g \the\numexpr\XINT_div_xmini #3.#4!#1{#2}%
1663 }%
```

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

```
1664 \def\XINT_div_II_g 1#1#2#3#4#5!#6#7#8%
1665 {%
1666   \expandafter \XINT_div_II_h
1667   \the\numexpr 1#1#2#3#4#5+#8\expandafter\expandafter\expandafter
1668   .\expandafter\expandafter\expandafter
1669   {\expandafter\xint_gob_til_exclam
1670    \romannumeral0\expandafter\XINT_div_sub\expandafter
1671    {\romannumeral0\XINT_rev_nounsep }#6\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1672    {\the\numexpr\XINT_div_smallmul_a 100000000.#1#2#3#4.#5!#71\Z!}%
1673    {#7}%
1674 }%
```

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

```
1675 \def\XINT_div_II_h 1#1.#2#3#4%
1676 {%
1677   \XINT_div_II_k #4{#2}{#3}{#1}%
1678 }%
```

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

```
1679 \def\XINT_div_II_k #1#2#3#4#5%
1680 {%
1681   \expandafter\XINT_div_II_l \the\numexpr #4-\xint_c_i.{#3}#1{#2}#5.%
1682 }%
1683 \def\XINT_div_II_l 1#1.#2#3#4#51#6!%
1684 {%
1685   \XINT_div_II_m {{#1}{#2}}{{#3}{#4}}{{#5}}{{#5}}{{#6}1#6!%
1686 }%
```

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

```
1687 \def\XINT_div_II_m #1#2#3#4.#5#6%
1688 {%
1689   \XINT_div_I_a {#3}{#2}{#4}{#5}{#6}#1%
1690 }%
```



### 3 Package *xintcore* implementation

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

```

1691 \def\XINT_div_minimulwc_a 1#1.#2.#3!#4#5#6#7#8.%
1692 {%
1693   \expandafter\XINT_div_minimulwc_b
1694   \the\numexpr \xint_c_x^ix+#1+#3*#8.#3*#4#5#6#7+#2*#8.#2*#4#5#6#7.%
1695 }%
1696 \def\XINT_div_minimulwc_b 1#1#2#3#4#5#6.#7.%
1697 {%
1698   \expandafter\XINT_div_minimulwc_c
1699   \the\numexpr \xint_c_x^ix+#1#2#3#4#5+#7.#6.%
1700 }%
1701 \def\XINT_div_minimulwc_c 1#1#2#3#4#5#6.#7.#8.%
1702 {%
1703   1#6#7\expandafter!%
1704   \the\numexpr\expandafter\XINT_div_smallmul_a
1705   \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8.%
1706 }%
1707 \def\XINT_div_smallmul_a #1.#2.#3!1#4!%
1708 {%
1709   \xint_gob_til_Z #4\XINT_div_smallmul_e\Z
1710   \XINT_div_minimulwc_a #1.#2.#3!#4.#2.#3!%
1711 }%
1712 \def\XINT_div_smallmul_e\Z\XINT_div_minimulwc_a 1#1.#2\Z #3!{1\relax #1!}%

```

Special very small multiplication for division. We only need to cater for multiplicands from 1 to 9. The ending is different from standard `verysmallmul`, a zero carry is not suppressed. And no final `1\Z!` is added. If multiplicand is just 1 let's not forget to add the zero carry 10000000! at the end.

```

1713 \def\XINT_div_verysmallmul #1%
1714   {\xint_gob_til_one #1\XINT_div_verysmallisone 1\XINT_div_verysmallmul_a 0.#1}%
1715 \def\XINT_div_verysmallisone 1\XINT_div_verysmallmul_a 0.1!1#11\Z!%
1716   {1\relax #11000000000!}%
1717 \def\XINT_div_verysmallmul_a #1.#2!1#3!%
1718 {%
1719   \xint_gob_til_Z #3\XINT_div_verysmallmul_e\Z
1720   \expandafter\XINT_div_verysmallmul_b
1721   \the\numexpr \xint_c_x^ix+#2*#3+#1.#2!%
1722 }%
1723 \def\XINT_div_verysmallmul_b 1#1#2.%
1724   {1#2\expandafter!\the\numexpr\XINT_div_verysmallmul_a #1.}%
1725 \def\XINT_div_verysmallmul_e\Z #1\Z +#2#3!{1\relax 0000000#2!}%

```

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

```

1726 \def\XINT_div_sub #1#2%
1727 {%
1728   \expandafter\XINT_div_sub_clean
1729   \the\numexpr\expandafter\XINT_div_sub_a\expandafter
1730   1#2\Z!\Z!\Z!\Z!\Z!\W #1\Z!\Z!\Z!\Z!\Z!\W
1731 }%

```

### 3 Package *xintcore* implementation

```
1732 \def\XINT_div_sub_clean #1-#2#3\W
1733 {%
1734   \if1#2\expandafter\XINT_rev_nounsep\else\expandafter\XINT_div_sub_neg\fi
1735   {}#1\R!\R!\R!\R!\R!\R!\R!\R!\W
1736 }%
1737 \def\XINT_div_sub_neg #1\W { -}%
1738 \def\XINT_div_sub_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
1739 {%
1740   \XINT_div_sub_b #1!#6!#2!#7!#3!#8!#4!#9!#5\W
1741 }%
1742 \def\XINT_div_sub_b #1#2#3!#4!%
1743 {%
1744   \xint_gob_til_Z #4\XINT_div_sub_bi \Z
1745   \expandafter\XINT_div_sub_c\the\numexpr#1-#3+1#4-\xint_c_i.%
1746 }%
1747 \def\XINT_div_sub_c 1#1#2.%
1748 {%
1749   1#2\expandafter!\the\numexpr\XINT_div_sub_d #1%
1750 }%
1751 \def\XINT_div_sub_d #1#2#3!#4!%
1752 {%
1753   \xint_gob_til_Z #4\XINT_div_sub_di \Z
1754   \expandafter\XINT_div_sub_e\the\numexpr#1-#3+1#4-\xint_c_i.%
1755 }%
1756 \def\XINT_div_sub_e 1#1#2.%
1757 {%
1758   1#2\expandafter!\the\numexpr\XINT_div_sub_f #1%
1759 }%
1760 \def\XINT_div_sub_f #1#2#3!#4!%
1761 {%
1762   \xint_gob_til_Z #4\XINT_div_sub_fi \Z
1763   \expandafter\XINT_div_sub_g\the\numexpr#1-#3+1#4-\xint_c_i.%
1764 }%
1765 \def\XINT_div_sub_g 1#1#2.%
1766 {%
1767   1#2\expandafter!\the\numexpr\XINT_div_sub_h #1%
1768 }%
1769 \def\XINT_div_sub_h #1#2#3!#4!%
1770 {%
1771   \xint_gob_til_Z #4\XINT_div_sub_hi \Z
1772   \expandafter\XINT_div_sub_i\the\numexpr#1-#3+1#4-\xint_c_i.%
1773 }%
1774 \def\XINT_div_sub_i 1#1#2.%
1775 {%
1776   1#2\expandafter!\the\numexpr\XINT_div_sub_a #1%
1777 }%
1778 \def\XINT_div_sub_bi\Z
1779   \expandafter\XINT_div_sub_c\the\numexpr#1-#2+#3.#4!#5!#6!#7!#8!#9!\Z !\W
1780 {%
1781   \XINT_div_sub_l #1#2!#5!#7!#9!%
1782 }%
1783 \def\XINT_div_sub_di\Z
```

### 3 Package *xintcore* implementation

```

1784 \expandafter\XINT_div_sub_e\the\numexpr#1-#2+#3.#4!#5!#6!#7!#8\W
1785 {%
1786 \XINT_div_sub_l #1#2!#5!#7!%
1787 }%
1788 \def\XINT_div_sub_fi\Z
1789 \expandafter\XINT_div_sub_g\the\numexpr#1-#2+#3.#4!#5!#6\W
1790 {%
1791 \XINT_div_sub_l #1#2!#5!%
1792 }%
1793 \def\XINT_div_sub_hi\Z
1794 \expandafter\XINT_div_sub_i\the\numexpr#1-#2+#3.#4\W
1795 {%
1796 \XINT_div_sub_l #1#2!%
1797 }%
1798 \def\XINT_div_sub_l #1%
1799 {%
1800 \xint_UDzerofork
1801 #1{-2\relax}%
1802 0\XINT_div_sub_r
1803 \krof
1804 }%
1805 \def\XINT_div_sub_r #1!%
1806 {%
1807 -\ifnum 0#1=\xint_c_ 1\else2\fi\relax
1808 }%

```

Ici  $B < 10^8$  (et est  $> 2$ ). On exécute

$\expandafter\XINT_sdiv\_out\the\numexpr\XINT\_smallldivx\_a\ x.1B!1<8d>!\dots1<8d>!1\Z!$   
avec  $x = \text{round}(B/2)$ ,  $1B = 10^8 + B$ , et A déjà en blocs  $1<8d>!$  (non renversés). Le  $\the\numexpr\XINT\_smallldivx\_a$   
va produire  $Q\Z\ R\W$  avec un  $R < 10^8$ , et un Q sous forme de blocs  $1<8d>!$  terminé par  $1!$  et nécessi-  
tant le nettoyage du premier bloc. Dans cette branche le B n'a pas été multiplié par une puissance  
de 10, il peut avoir moins de huit chiffres.

```

1809 \def\XINT_sdiv_out #1\Z!#2!%
1810 {\expandafter
1811 {\romannumeral0\XINT_unsep_cuzsmall#11\R!1\R!1\R!1\R!1\R!1\R!1\R!1\W}%
1812 {#2}}%

```

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

```

1813 \def\XINT_smallldivx_a #1.1#2!1#3!%
1814 {%
1815 \expandafter\XINT_smallldivx_b
1816 \the\numexpr (#3+#1)/#2-\xint_c_i!#1.#2!#3!%
1817 }%
1818 \def\XINT_smallldivx_b #1#2!%
1819 {%
1820 \if0#1\else
1821 \xint_c_x^viii+#1#2\xint_afterfi{\expandafter!\the\numexpr}\fi
1822 \XINT_smallldiv_c #1#2!%
1823 }%
1824 \def\XINT_smallldiv_c #1!#2.#3!#4!%
1825 {%

```

### 3 Package *xintcore* implementation

```
1826 \expandafter\XINT_smallldiv_d\the\numexpr #4-#1*#3!#2.#3!%
1827 }%
```

On va boucler ici: #1 est un reste, #2 est x.B (avec B sans le 1 mais sur huit chiffres). #3#4 est le premier bloc qui reste de A. Si on a terminé avec A, alors #1 est le reste final. Le quotient lui est terminé par un 1!: ce 1! disparaîtra dans le nettoyage par \XINT\_unsep\_cuzsmall. Ce dernier, malgré le fait qu'on soit dans le bon ordre déjà fait une macro dans le style  $O(N^2)$  car sinon le nombre maximal de chiffres serait moitié moins à cause des nettoyages nécessaires après \numexpr. Je suis obligé de faire un nettoyage final car comme l'expansion est engendrée par \numexpr, elle me boufferait des leading zeros si je ne mettais pas un 1 devant chaque bloc en sortie de Q.

```
1828 \def\XINT_smallldiv_d #1!#2!1#3#4!%
1829 {%
1830 \xint_gob_til_Z #3\XINT_smallldiv_end \Z
1831 \XINT_smallldiv_e #1!#2!1#3#4!%
1832 }%
1833 \def\XINT_smallldiv_end\Z\XINT_smallldiv_e #1!#2!1\Z!\{1!\Z!#1!}%

```

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

```
1834 \def\XINT_smallldiv_e #1!#2.#3!%
1835 {%
1836 \expandafter\XINT_smallldiv_f\the\numexpr
1837 \xint_c_xi_e_viii_mone+#1*\xint_c_x^viii/#3!#2.#3!#1!%
1838 }%
1839 \def\XINT_smallldiv_f 1#1#2#3#4#5#6!#7.#8!%
1840 {%
1841 \xint_gob_til_zero #1\XINT_smallldiv_fz 0%
1842 \expandafter\XINT_smallldiv_g
1843 \the\numexpr\XINT_minimul_a #2#3#4#5.#6!#8!#2#3#4#5#6!#7.#8!%
1844 }%
1845 \def\XINT_smallldiv_fz 0%
1846 \expandafter\XINT_smallldiv_g\the\numexpr\XINT_minimul_a
1847 9999.9999!#1!99999999!#2!0!1#3!%
1848 {%
1849 \XINT_smallldiv_i .#3!\xint_c_!#2!%
1850 }%
1851 \def\XINT_smallldiv_g 1#1!1#2!#3!#4!#5!#6!%
1852 {%
1853 \expandafter\XINT_smallldiv_h\the\numexpr 1#6-#1.#2!#5!#3!#4!%
1854 }%
1855 \def\XINT_smallldiv_h 1#1#2.#3!#4!%
1856 {%
1857 \expandafter\XINT_smallldiv_i\the\numexpr #4-#3+#1-\xint_c_i.#2!%
1858 }%
1859 \def\XINT_smallldiv_i #1.#2!#3!#4.#5!%
1860 {%
1861 \expandafter\XINT_smallldiv_j\the\numexpr (#1#2+#4)/#5-\xint_c_i!#3!#1#2!#4.#5!%
1862 }%
1863 \def\XINT_smallldiv_j #1!#2!%
1864 {%
1865 \xint_c_x^viii+#1+#2\expandafter!\the\numexpr\XINT_smallldiv_k

```

### 3 Package *xintcore* implementation

```
1866 #1!%
1867 }%
```

On boucle vers `\XINT_smallldiv_d`.

```
1868 \def\XINT_smallldiv_k #1!#2!#3.#4!%
1869 {%
1870   \expandafter\XINT_smallldiv_d\the\numexpr #2-#1*#4!#3.#4!%
1871 }%
```

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

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

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

```
1872 \def\XINT_div_mini #1.#2!1#3!%
1873 {%
1874   \expandafter\XINT_div_mini_a\the\numexpr
1875   \xint_c_xi_e_viii_mone+#3*\xint_c_x^viii/#1!#1.#2!#3!%
1876 }%
```

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

```
1877 \def\XINT_div_mini_a 1#1#2#3#4#5#6!#7.#8!%
1878 {%
1879   \xint_gob_til_zero #1\XINT_div_mini_w 0%
1880   \expandafter\XINT_div_mini_b
1881   \the\numexpr\XINT_minimul_a #2#3#4#5.#6!#7!#2#3#4#5#6!#7.#8!%
1882 }%
1883 \def\XINT_div_mini_w 0%
1884   \expandafter\XINT_div_mini_b\the\numexpr\XINT_minimul_a
1885   9999.9999!#1!99999999!#2.#3!00000000!#4!%
1886 {%
1887   \xint_c_x^viii_mone+(#4+#3)/#2!%
1888 }%
1889 \def\XINT_div_mini_b 1#1!1#2!#3!#4!#5!#6!%
1890 {%
1891   \expandafter\XINT_div_mini_c
1892   \the\numexpr 1#6-#1.#2!#5!#3!#4!%
1893 }%
1894 \def\XINT_div_mini_c 1#1#2.#3!#4!%
1895 {%
1896   \expandafter\XINT_div_mini_d
1897   \the\numexpr #4-#3+#1-\xint_c_i.#2!%
1898 }%
1899 \def\XINT_div_mini_d #1.#2!#3!#4.#5!%
1900 {%
```

```
1901 \xint_c_x^viii_mone+#3+(#1#2+#5)/#4!%
1902 }%
```

### 3.27 \xintiDivRound, \xintiiDivRound

1.1, transferred from first release of *bnumexpr*. Rewritten for 1.2.

```
1903 \def\xintiDivRound {\romannumeral0\xintidivround }%
1904 \def\xintidivround #1%
1905 {\expandafter\XINT_idivround\romannumeral0\xintnum{#1}\Z }%
1906 \def\xintiiDivRound {\romannumeral0\xintiidivround }%
1907 \def\xintiidivround #1{\expandafter\XINT_iidivround \romannumeral`&&@#1\Z }%
1908 \def\XINT_idivround #1#2\Z #3%
1909 {\expandafter\XINT_iidivround_a\expandafter #1%
1910 \romannumeral0\xintnum{#3}\Z #2\Z }%
1911 \def\XINT_iidivround #1#2\Z #3%
1912 {\expandafter\XINT_iidivround_a\expandafter #1\romannumeral`&&@#3\Z #2\Z }%
1913 \def\XINT_iidivround_a #1#2% #1 de A, #2 de B.
1914 {%
1915 \if0#2\xint_dothis\XINT_iidivround_divbyzero\fi
1916 \if0#1\xint_dothis\XINT_iidivround_aiszero\fi
1917 \if-#2\xint_dothis{\XINT_iidivround_bneg #1}\fi
1918 \xint_orthat{\XINT_iidivround_bpos #1#2}%
1919 }%
1920 \def\XINT_iidivround_divbyzero #1\Z #2\Z {\xintError:DivisionByZero\space 0}%
1921 \def\XINT_iidivround_aiszero #1\Z #2\Z { 0}%
1922 \def\XINT_iidivround_bpos #1%
1923 {%
1924 \xint_UDsignfork
1925 #1{\xintiiopp\XINT_iidivround_pos {}}%
1926 -{\XINT_iidivround_pos #1}%
1927 \krof
1928 }%
1929 \def\XINT_iidivround_bneg #1%
1930 {%
1931 \xint_UDsignfork
1932 #1{\XINT_iidivround_pos {}}%
1933 -{\xintiiopp\XINT_iidivround_pos #1}%
1934 \krof
1935 }%
1936 \def\XINT_iidivround_pos #1#2\Z #3\Z
1937 {%
1938 \expandafter\XINT_iidivround_pos_a
1939 \romannumeral0\XINT_div_prepare {#2}{#1#30}%
1940 }%
```

The 1.2c interface to addition changed, the \Z's are now 1\Z's here. Will have to come back here for improvements. The 1\Z!1\Z!1\Z!1\Z!\W are for the addition which is done if rounding up.

```
1941 \def\XINT_iidivround_pos_a #1#2%
1942 {%
1943 \expandafter\XINT_iidivround_pos_b
1944 \romannumeral0\expandafter\XINT_sepandrev
```

### 3 Package *xintcore* implementation

```
1945 \romannumeral0\XINT_zeroes_forviii #1\R\R\R\R\R\R\R\{10}0000001\W
1946 #1\XINT_rsepybviii_end_A 2345678\XINT_rsepybviii_end_B 2345678\relax XX%
1947 \R.\R.\R.\R.\R.\R.\R.\R.\W
1948 1\Z!1\Z!1\Z!1\Z!\W\R
1949 }%
1950 \def\XINT_iidivround_pos_b #1#2#3#4#5#6#7#8!1#9%
1951 {%
1952 \xint_gob_til_Z #9\XINT_iidivround_small\Z
1953 \ifnum #8>\xint_c_iv
1954 \expandafter\XINT_iidivround_pos_up
1955 \else \expandafter\XINT_iidivround_pos_finish
1956 \fi
1957 1#1#2#3#4#5#6#70!1#9%
1958 }%
1959 \def\XINT_iidivround_pos_up
1960 {%
1961 \expandafter\XINT_iidivround_pos_finish
1962 \the\numexpr\XINT_add_a\xint_c_ii 100000010!1\Z!1\Z!1\Z!1\Z!\W
1963 }%
1964 \def\XINT_iidivround_pos_finish #10!#21\Z!#3\R
1965 {%
1966 \expandafter\XINT_cuz_small\romannumeral0\XINT_unrevbviii }%
1967 #1!#21\Z!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1968 }%
1969 \def\XINT_iidivround_small\Z\ifnum #1>#2\fi 1#30!#4\W\R
1970 {%
1971 \ifnum #1>\xint_c_iv
1972 \expandafter\XINT_iidivround_small_up
1973 \else \expandafter\XINT_iidivround_small_trunc
1974 \fi {#3}%
1975 }%
1976 \edef\XINT_iidivround_small_up #1%
1977 {\noexpand\expandafter\space\noexpand\the\numexpr #1+\xint_c_i\relax }%
1978 \edef\XINT_iidivround_small_trunc #1%
1979 {\noexpand\expandafter\space\noexpand\the\numexpr #1\relax }%
```

#### 3.28 \xintiDivTrunc, \xintiiDivTrunc

```
1980 \def\xintiDivTrunc {\romannumeral0\xintidivtrunc }%
1981 \def\xintidivtrunc #1{\expandafter\XINT_iidivtrunc\romannumeral0\xintnum{#1}\Z }%
1982 \def\xintiiDivTrunc {\romannumeral0\xintiidivtrunc }%
1983 \def\xintiidivtrunc #1{\expandafter\XINT_iidivtrunc \romannumeral`&&@#1\Z }%
1984 \def\XINT_iidivtrunc #1#2\Z #3{\expandafter\XINT_iidivtrunc_a\expandafter #1%
1985 \romannumeral`&&@#3\Z #2\Z }%
1986 \def\XINT_iidivtrunc_a #1#2% #1 de A, #2 de B.
1987 {%
1988 \if0#2\xint_dothis\XINT_iidivround_divbyzero\fi
1989 \if0#1\xint_dothis\XINT_iidivround_aiszero\fi
1990 \if-#2\xint_dothis{\XINT_iidivtrunc_bneg #1}\fi
1991 \xint_orthat{\XINT_iidivtrunc_bpos #1#2}%
1992 }%
1993 \def\XINT_iidivtrunc_bpos #1%
1994 {%
```

```

1995 \xint_UDsignfork
1996 #1{\xintiiopp\XINT_iidivtrunc_pos {}}%
1997 -{\XINT_iidivtrunc_pos #1}%
1998 \krof
1999 }%
2000 \def\XINT_iidivtrunc_bneg #1%
2001 {%
2002 \xint_UDsignfork
2003 #1{\XINT_iidivtrunc_pos {}}%
2004 -{\xintiiopp\XINT_iidivtrunc_pos #1}%
2005 \krof
2006 }%
2007 \def\XINT_iidivtrunc_pos #1#2\Z #3\Z%
2008 {\expandafter\xint_firstoftwo_thenstop
2009 \romannumeral0\XINT_div_prepare {#2}{#1#3}}%

```

### 3.29 *\xintiMod*, *\xintiiMod*

```

2010 \def\xintiMod {\romannumeral0\xintimod }%
2011 \def\xintimod #1{\expandafter\XINT_iimod\romannumeral0\xintnum{#1}\Z }%
2012 \def\xintiiMod {\romannumeral0\xintiiimod }%
2013 \def\xintiiimod #1{\expandafter\XINT_iimod \romannumeral`&&@#1\Z }%
2014 \def\XINT_iimod #1#2\Z #3{\expandafter\XINT_iimod_a\expandafter #1%
2015 \romannumeral`&&@#3\Z #2\Z }%
2016 \def\XINT_iimod_a #1#2# #1 de A, #2 de B.
2017 {%
2018 \if0#2\xint_dothis\XINT_iidivround_divbyzero\fi
2019 \if0#1\xint_dothis\XINT_iidivround_aiszero\fi
2020 \if-#2\xint_dothis{\XINT_iimod_bneg #1}\fi
2021 \xint_orthat{\XINT_iimod_bpos #1#2}%
2022 }%
2023 \def\XINT_iimod_bpos #1%
2024 {%
2025 \xint_UDsignfork
2026 #1{\xintiiopp\XINT_iimod_pos {}}%
2027 -{\XINT_iimod_pos #1}%
2028 \krof
2029 }%
2030 \def\XINT_iimod_bneg #1%
2031 {%
2032 \xint_UDsignfork
2033 #1{\xintiiopp\XINT_iimod_pos {}}%
2034 -{\XINT_iimod_pos #1}%
2035 \krof
2036 }%
2037 \def\XINT_iimod_pos #1#2\Z #3\Z%
2038 {\expandafter\xint_secondoftwo_thenstop\romannumeral0\XINT_div_prepare
2039 {#2}{#1#3}}%

```

### 3.30 “Load *xintfrac*” macros

Originally was used in *\xintiexpr*. Transferred from *xintfrac* for 1.1.

```

2040 \catcode`! 11

```



### 3 Package *xintcore* implementation

```
2041 \def\xintAbs {\Did_you_mean_iiAbs?or_load_xintfrac!}%
2042 \def\xintOpp {\Did_you_mean_iiOpp?or_load_xintfrac!}%
2043 \def\xintAdd {\Did_you_mean_iiAdd?or_load_xintfrac!}%
2044 \def\xintSub {\Did_you_mean_iiSub?or_load_xintfrac!}%
2045 \def\xintMul {\Did_you_mean_iiMul?or_load_xintfrac!}%
2046 \def\xintPow {\Did_you_mean_iiPow?or_load_xintfrac!}%
2047 \def\xintSqr {\Did_you_mean_iiSqr?or_load_xintfrac!}%
2048 \XINT_restorecatcodes_endinput%
```

## 4 Package `xint` implementation

.1	Package identification . . . . .	91	.25	<code>\xintANDof</code> . . . . .	101
.2	More token management . . . . .	91	.26	<code>\xintORof</code> . . . . .	102
.3	<code>\xintSgnFork</code> . . . . .	91	.27	<code>\xintXORof</code> . . . . .	102
.4	<code>\xintIsOne</code> , <code>\xintiiIsOne</code> . . . . .	92	.28	<code>\xintGeq</code> , <code>\xintiiGeq</code> . . . . .	102
.5	<code>\xintRev</code> . . . . .	92	.29	<code>\xintiMax</code> , <code>\xintiiMax</code> . . . . .	105
.6	<code>\xintLen</code> . . . . .	92	.30	<code>\xintiMaxof</code> , <code>\xintiiMaxof</code> . . . . .	106
.7	<code>\xintBool</code> , <code>\xintToggle</code> . . . . .	93	.31	<code>\xintiMin</code> , <code>\xintiiMin</code> . . . . .	107
.8	<code>\xintifSgn</code> , <code>\xintiiifSgn</code> . . . . .	93	.32	<code>\xintiMinof</code> , <code>\xintiiMinof</code> . . . . .	108
.9	<code>\xintifZero</code> , <code>\xintifNotZero</code> , <code>\xintii-</code> <code>ifZero</code> , <code>\xintiiifNotZero</code> . . . . .	93	.33	<code>\xintiiSum</code> . . . . .	109
.10	<code>\xintifOne</code> , <code>\xintiiifOne</code> . . . . .	94	.34	<code>\xintiiPrd</code> . . . . .	109
.11	<code>\xintifTrueAelseB</code> , <code>\xintifFalseAelseB</code> . . . . .	95	.35	<code>\xintMON</code> , <code>\xintMMON</code> , <code>\xintiiMON</code> , <code>\xinti-</code> <code>imMON</code> . . . . .	110
.12	<code>\xintifCmp</code> , <code>\xintiiifCmp</code> . . . . .	95	.36	<code>\xintOdd</code> , <code>\xintiiOdd</code> , <code>\xintEven</code> , <code>\xinti-</code> <code>iEven</code> . . . . .	110
.13	<code>\xintifEq</code> , <code>\xintiiifEq</code> . . . . .	95	.37	<code>\xintDSL</code> . . . . .	111
.14	<code>\xintifGt</code> , <code>\xintiiifGt</code> . . . . .	96	.38	<code>\xintDSR</code> . . . . .	111
.15	<code>\xintifLt</code> , <code>\xintiiifLt</code> . . . . .	96	.39	<code>\xintDSH</code> , <code>\xintDSHr</code> . . . . .	112
.16	<code>\xintifOdd</code> , <code>\xintiiifOdd</code> . . . . .	96	.40	<code>\xintDSx</code> . . . . .	113
.17	<code>\xintCmp</code> , <code>\xintiiCmp</code> . . . . .	97	.41	<code>\xintDecSplit</code> , <code>\xintDecSplitL</code> , <code>\xint-</code> <code>DecSplitR</code> . . . . .	115
.18	<code>\xintEq</code> , <code>\xintGt</code> , <code>\xintLt</code> . . . . .	100	.42	<code>\xintiiSqrt</code> , <code>\xintiiSqrtR</code> , <code>\xinti-</code> <code>iSquareRoot</code> . . . . .	118
.19	<code>\xintNeq</code> , <code>\xintGtorEq</code> , <code>\xintLtorEq</code> . . . . .	100	.43	<code>\xintiiBinomial</code> , <code>\xintiBinomial</code> . . . . .	124
.20	<code>\xintiiEq</code> , <code>\xintiiGt</code> , <code>\xintiiLt</code> . . . . .	100	.44	<code>\xintiiPFactorial</code> , <code>\xintiPFactorial</code> . . . . .	130
.21	<code>\xintiiNeq</code> , <code>\xintiiGtorEq</code> , <code>\xintiiLtorEq</code> . . . . .	101	.45	<code>\xintiiE</code> . . . . .	132
.22	<code>\xintIsZero</code> , <code>\xintIsNotZero</code> , <code>\xintii-</code> <code>IsZero</code> , <code>\xintiiIsNotZero</code> . . . . .	101	.46	“Load <code>xintfrac</code> ” macros . . . . .	132
.23	<code>\xintIsTrue</code> , <code>\xintNot</code> , <code>\xintIsFalse</code> . . . . .	101			
.24	<code>\xintAND</code> , <code>\xintOR</code> , <code>\xintXOR</code> . . . . .	101			

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

```

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@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 [2016/03/12 1.2f 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}%
53 \edef\xint_cleanupzeros_andstop #1#2#3#4%
54 {%
55   \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4\relax
56 }%

```

## 4.3 *\xintSgnFork*

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

```

57 \def\xintSgnFork {\romannumeral0\xintsgnfork }%
58 \def\xintsgnfork #1%
59 {%
60   \ifcase #1 \expandafter\xint_secondofthree_thenstop
61             \or\expandafter\xint_thirdofthree_thenstop
62             \else\expandafter\xint_firstofthree_thenstop
63   \fi
64 }%

```

#### 4.4 `\xintIsOne`, `\xintiiIsOne`

Added in 1.03. 1.09a defines `\xintIsOne`. 1.1a adds `\xintiiIsOne`.

```

65 \def\xintiiIsOne {\romannumeral0\xintiiisone }%
66 \def\xintiiisone #1{\expandafter\XINT_isone\romannumeral`&&@#1\W\Z }%
67 \def\xintIsOne {\romannumeral0\xintisone }%
68 \def\xintisone #1{\expandafter\XINT_isone\romannumeral0\xintnum{#1}\W\Z }%
69 \def\XINT_isOne #1{\romannumeral0\XINT_isone #1\W\Z }%
70 \def\XINT_isone #1#2%
71 {%
72   \xint_gob_til_one #1\XINT_isone_b 1%
73   \expandafter\space\expandafter 0\xint_gob_til_Z #2%
74 }%
75 \def\XINT_isone_b #1\xint_gob_til_Z #2%
76 {%
77   \xint_gob_til_W #2\XINT_isone_yes \W
78   \expandafter\space\expandafter 0\xint_gob_til_Z
79 }%
80 \def\XINT_isone_yes #1\Z { 1}%

```

#### 4.5 `\xintRev`

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

1.2 has `\xintReverseDigits` and I thus make `\xintRev` an alias. Remarks above not addressed.

```

81 \let\xintRev\xintReverseDigits

```

#### 4.6 `\xintLen`

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

```

82 \def\xintLen {\romannumeral0\xintlen }%
83 \def\xintlen #1%
84 {%
85   \expandafter\XINT_len_fork
86   \romannumeral0\xintnum{#1}\xint_relax\xint_relax\xint_relax\xint_relax
87   \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
88 }%
89 \def\XINT_Len #1% variant which does not expand via \xintnum.
90 {%
91   \romannumeral0\XINT_len_fork
92   #1\xint_relax\xint_relax\xint_relax\xint_relax
93   \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
94 }%
95 \def\XINT_len_fork #1%
96 {%
97   \expandafter\XINT_length_loop
98   \xint_UDsignfork
99   #1{0.}%

```

```

100     -{0.#1}%
101     \krof
102 }%

```

## 4.7 `\xintBool`, `\xintToggle`

1.09c

```

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

```

## 4.8 `\xintifSgn`, `\xintiiifSgn`

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

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

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

```

106 \def\xintifSgn {\romannumeral0\xintifsgn }%
107 \def\xintifsgn #1%
108 {%
109     \ifcase \xintSgn{#1}
110         \expandafter\xint_secondofthree_thenstop
111         \or\expandafter\xint_thirdofthree_thenstop
112         \else\expandafter\xint_firstofthree_thenstop
113     \fi
114 }%
115 \def\xintiiifSgn {\romannumeral0\xintiiifsgn }%
116 \def\xintiiifsgn #1%
117 {%
118     \ifcase \xintiiSgn{#1}
119         \expandafter\xint_secondofthree_thenstop
120         \or\expandafter\xint_thirdofthree_thenstop
121         \else\expandafter\xint_firstofthree_thenstop
122     \fi
123 }%

```

## 4.9 `\xintifZero`, `\xintifNotZero`, `\xintiiifZero`, `\xintiiifNotZero`

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

```

124 \def\xintifZero {\romannumeral0\xintifzero }%
125 \def\xintifzero #1%
126 {%
127     \if0\xintSgn{#1}%
128         \expandafter\xint_firstoftwo_thenstop
129     \else
130         \expandafter\xint_secondoftwo_thenstop
131     \fi

```

```

132 }%
133 \def\xintifNotZero {\romannumeral0\xintifnotzero }%
134 \def\xintifnotzero #1%
135 {%
136   \if0\xintSgn{#1}%
137     \expandafter\xint_secondoftwo_thenstop
138   \else
139     \expandafter\xint_firstoftwo_thenstop
140   \fi
141 }%
142 \def\xintiiifZero {\romannumeral0\xintiiifzero }%
143 \def\xintiiifzero #1%
144 {%
145   \if0\xintiiSgn{#1}%
146     \expandafter\xint_firstoftwo_thenstop
147   \else
148     \expandafter\xint_secondoftwo_thenstop
149   \fi
150 }%
151 \def\xintiiifNotZero {\romannumeral0\xintiiifnotzero }%
152 \def\xintiiifnotzero #1%
153 {%
154   \if0\xintiiSgn{#1}%
155     \expandafter\xint_secondoftwo_thenstop
156   \else
157     \expandafter\xint_firstoftwo_thenstop
158   \fi
159 }%

```

#### 4.10 \xintifOne,\xintiiifOne

added in 1.09i. 1.1a adds \xintiiifOne.

```

160 \def\xintiiifOne {\romannumeral0\xintiiifone }%
161 \def\xintiiifone #1%
162 {%
163   \if1\xintiiIsOne{#1}%
164     \expandafter\xint_firstoftwo_thenstop
165   \else
166     \expandafter\xint_secondoftwo_thenstop
167   \fi
168 }%
169 \def\xintifOne {\romannumeral0\xintifone }%
170 \def\xintifone #1%
171 {%
172   \if1\xintIsOne{#1}%
173     \expandafter\xint_firstoftwo_thenstop
174   \else
175     \expandafter\xint_secondoftwo_thenstop
176   \fi
177 }%

```

#### 4.11 `\xintifTrueAelseB`, `\xintifFalseAelseB`

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

```
178 \let\xintifTrueAelseB\xintifNotZero
179 \let\xintifFalseAelseB\xintifZero
180 \let\xintifTrue\xintifNotZero
181 \let\xintifTrueFalse\xintifNotZero
```

#### 4.12 `\xintifCmp`, `\xintiiifCmp`

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

```
182 \def\xintifCmp {\romannumeral0\xintifcmp}%
183 \def\xintifcmp #1#2%
184 {%
185     \ifcase\xintCmp {#1}{#2}
186         \expandafter\xint_secondofthree_thenstop
187     \or\expandafter\xint_thirdofthree_thenstop
188     \else\expandafter\xint_firstofthree_thenstop
189     \fi
190}%
191 \def\xintiiifCmp {\romannumeral0\xintiiifcmp}%
192 \def\xintiiifcmp #1#2%
193 {%
194     \ifcase\xintiiCmp {#1}{#2}
195         \expandafter\xint_secondofthree_thenstop
196     \or\expandafter\xint_thirdofthree_thenstop
197     \else\expandafter\xint_firstofthree_thenstop
198     \fi
199}%

```

#### 4.13 `\xintifEq`, `\xintiiifEq`

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

```
200 \def\xintifEq {\romannumeral0\xintifeq}%
201 \def\xintifeq #1#2%
202 {%
203     \if0\xintCmp{#1}{#2}%
204         \expandafter\xint_firstoftwo_thenstop
205     \else\expandafter\xint_secondoftwo_thenstop
206     \fi
207}%
208 \def\xintiiifEq {\romannumeral0\xintiiifeq}%
209 \def\xintiiifeq #1#2%
210 {%
211     \if0\xintiiCmp{#1}{#2}%
212         \expandafter\xint_firstoftwo_thenstop
213     \else\expandafter\xint_secondoftwo_thenstop
214     \fi
215}%

```

**4.14 `\xintifGt`, `\xintiiifGt`**1.09a `\xintifGt {n}{m}{YES if n>m}{NO if n<=m}`. 1.1a adds ii variant

```

216 \def\xintifGt {\romannumeral0\xintifgt }%
217 \def\xintifgt #1#2%
218 {%
219     \if1\xintCmp{#1}{#2}%
220         \expandafter\xint_firstoftwo_thenstop
221     \else\expandafter\xint_secondoftwo_thenstop
222     \fi
223 }%
224 \def\xintiiifGt {\romannumeral0\xintiiifgt }%
225 \def\xintiiifgt #1#2%
226 {%
227     \if1\xintiiCmp{#1}{#2}%
228         \expandafter\xint_firstoftwo_thenstop
229     \else\expandafter\xint_secondoftwo_thenstop
230     \fi
231 }%

```

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

```

232 \def\xintifLt {\romannumeral0\xintiflt }%
233 \def\xintiflt #1#2%
234 {%
235     \ifnum\xintCmp{#1}{#2}<\xint_c_
236         \expandafter\xint_firstoftwo_thenstop
237     \else \expandafter\xint_secondoftwo_thenstop
238     \fi
239 }%
240 \def\xintiiifLt {\romannumeral0\xintiiiflt }%
241 \def\xintiiiflt #1#2%
242 {%
243     \ifnum\xintiiCmp{#1}{#2}<\xint_c_
244         \expandafter\xint_firstoftwo_thenstop
245     \else \expandafter\xint_secondoftwo_thenstop
246     \fi
247 }%

```

**4.16 `\xintifOdd`, `\xintiiifOdd`**1.09e. Restyled in 1.09i. 1.1a adds `\xintiiifOdd`.

```

248 \def\xintiiifOdd {\romannumeral0\xintiiifodd }%
249 \def\xintiiifodd #1%
250 {%
251     \if\xintiiOdd{#1}1%
252         \expandafter\xint_firstoftwo_thenstop
253     \else
254         \expandafter\xint_secondoftwo_thenstop

```



```

255   \fi
256 }%
257 \def\xintifOdd {\romannumeral0\xintifodd }%
258 \def\xintifodd #1%
259 {%
260   \if\xintOdd{#1}1%
261   \expandafter\xint_firstoftwo_thenstop
262   \else
263   \expandafter\xint_secondoftwo_thenstop
264   \fi
265 }%

```

#### 4.17 *\xintCmp*, *\xintiiCmp*

Faster than doing the full subtraction.

```

266 \def\xintCmp    {\romannumeral0\xintcmp }%
267 \def\xintcmp    #1{\expandafter\XINT_icmp\romannumeral0\xintnum{#1}\Z }%
268 \def\xintiiCmp  {\romannumeral0\xintiicmp }%
269 \def\xintiicmp  #1{\expandafter\XINT_iicmp\romannumeral`&&@#1\Z }%
270 \def\XINT_iicmp #1#2\Z #3%
271 {%
272   \expandafter\XINT_cmp_nfork\expandafter #1\romannumeral`&&@#3\Z #2\Z
273 }%
274 \let\XINT_Cmp   \xintiiCmp
275 \def\XINT_icmp  #1#2\Z #3%
276 {%
277   \expandafter\XINT_cmp_nfork\expandafter #1\romannumeral0\xintnum{#3}\Z #2\Z
278 }%
279 \def\XINT_cmp_nfork #1#2%
280 {%
281   \xint_UDzerofork
282   #1\XINT_cmp_firstiszero
283   #2\XINT_cmp_secondiszero
284   0{ }%
285   \krof
286   \xint_UDsignsfork
287   #1#2\XINT_cmp_minusminus
288   #1-\XINT_cmp_minusplus
289   #2-\XINT_cmp_plusminus
290   --\XINT_cmp_plusplus
291   \krof #1#2%
292 }%
293 \def\XINT_cmp_firstiszero #1\krof 0#2#3\Z #4\Z
294 {%
295   \xint_UDzerominusfork
296   #2-{ 0}%
297   0#2{ 1}%
298   0-{ -1}%
299   \krof
300 }%
301 \def\XINT_cmp_secondiszero #1\krof #20#3\Z #4\Z
302 {%

```

#### 4 Package `xint` implementation

```

303 \xint_UDzerominusfork
304 #2-{ 0}%
305 0#2{ -1}%
306 0-{ 1}%
307 \krof
308 }%
309 \def\xint_cmp_plusminus #1\Z #2\Z{ 1}%
310 \def\xint_cmp_minusplus #1\Z #2\Z{ -1}%
311 \def\xint_cmp_minusminus
312 --{\expandafter\xint_opp\romannumeral0\xint_cmp_plusplus {}{}}%
313 \def\xint_cmp_plusplus #1#2#3\Z
314 {%
315 \expandafter\xint_cmp_pp
316 \romannumeral0\expandafter\xint_sepandrev_andcount
317 \romannumeral0\xint_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
318 #2#3\xint_rsepbyviii_end_A 2345678%
319 \xint_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_iii
320 \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
321 \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
322 \X #1%
323 }%
324 \def\xint_cmp_pp #1.#2\X #3\Z
325 {%
326 \expandafter\xint_cmp_checklengths
327 \the\numexpr #1\expandafter.%
328 \romannumeral0\expandafter\xint_sepandrev_andcount
329 \romannumeral0\xint_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
330 #3\xint_rsepbyviii_end_A 2345678%
331 \xint_rsepbyviii_end_B 2345678\relax \xint_c_ii\xint_c_iii
332 \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
333 \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
334 \Z!\Z!\Z!\Z!\Z!\W #2\Z!\Z!\Z!\Z!\Z!\W
335 }%
336 \def\xint_cmp_checklengths #1.#2.%
337 {%
338 \ifnum #1=#2
339 \expandafter\xint_firstoftwo
340 \else
341 \expandafter\xint_secondoftwo
342 \fi
343 \xint_cmp_aa {\xint_cmp_distinctlengths {#1}{#2}}%
344 }%
345 \def\xint_cmp_distinctlengths #1#2#3\W #4\W
346 {%
347 \ifnum #1>#2
348 \expandafter\xint_firstoftwo
349 \else
350 \expandafter\xint_secondoftwo
351 \fi
352 { -1}{ 1}%
353 }%
354 \def\xint_cmp_aa {\expandafter\xint_cmp_w\the\numexpr\xint_cmp_a \xint_c_i }%

```

#### 4 Package *xint* implementation

```

355 \def\XINT_cmp_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
356 {%
357     \XINT_cmp_b #1!#6!#2!#7!#3!#8!#4!#9!#5\W
358 }%
359 \def\XINT_cmp_b #1#2#3!#4!%
360 {%
361     \xint_gob_til_Z #2\XINT_cmp_bi \Z
362     \expandafter\XINT_cmp_c\the\numexpr#1+1#4-#3-\xint_c_i.%
363 }%
364 \def\XINT_cmp_c 1#1#2.%
365 {%
366     1#2\expandafter!\the\numexpr\XINT_cmp_d #1%
367 }%
368 \def\XINT_cmp_d #1#2#3!#4!%
369 {%
370     \xint_gob_til_Z #2\XINT_cmp_di \Z
371     \expandafter\XINT_cmp_e\the\numexpr#1+1#4-#3-\xint_c_i.%
372 }%
373 \def\XINT_cmp_e 1#1#2.%
374 {%
375     1#2\expandafter!\the\numexpr\XINT_cmp_f #1%
376 }%
377 \def\XINT_cmp_f #1#2#3!#4!%
378 {%
379     \xint_gob_til_Z #2\XINT_cmp_fi \Z
380     \expandafter\XINT_cmp_g\the\numexpr#1+1#4-#3-\xint_c_i.%
381 }%
382 \def\XINT_cmp_g 1#1#2.%
383 {%
384     1#2\expandafter!\the\numexpr\XINT_cmp_h #1%
385 }%
386 \def\XINT_cmp_h #1#2#3!#4!%
387 {%
388     \xint_gob_til_Z #2\XINT_cmp_hi \Z
389     \expandafter\XINT_cmp_i\the\numexpr#1+1#4-#3-\xint_c_i.%
390 }%
391 \def\XINT_cmp_i 1#1#2.%
392 {%
393     1#2\expandafter!\the\numexpr\XINT_cmp_a #1%
394 }%
395 \def\XINT_cmp_bi\Z
396     \expandafter\XINT_cmp_c\the\numexpr#1+1#2-#3.#4!#5!#6!#7!#8!#9!\Z !\W
397 {%
398     \XINT_cmp_k #1#2!#5!#7!#9!%
399 }%
400 \def\XINT_cmp_di\Z
401     \expandafter\XINT_cmp_e\the\numexpr#1+1#2-#3.#4!#5!#6!#7!#8\W
402 {%
403     \XINT_cmp_k #1#2!#5!#7!%
404 }%
405 \def\XINT_cmp_fi\Z
406     \expandafter\XINT_cmp_g\the\numexpr#1+1#2-#3.#4!#5!#6\W

```

```

407 {%
408   \XINT_cmp_k #1#2!#5!%
409 }%
410 \def\XINT_cmp_hi\Z
411   \expandafter\XINT_cmp_i\the\numexpr#1+1#2-#3.#4\W
412 {%
413   \XINT_cmp_k #1#2!%
414 }%
415 \def\XINT_cmp_k #1#2\W
416 {%
417   \xint_UDzerofork
418     #1{-1\relax \XINT_cmp_greater}%
419     0{-1\relax \XINT_cmp_lessequal}%
420   \krof
421 }%
422 \def\XINT_cmp_w #1-1#2{#2#11\Z!\W}%
423 \def\XINT_cmp_greater #1\Z!\W{ 1}%
424 \def\XINT_cmp_lessequal #1!%
425   {\xint_gob_til_Z #1\XINT_cmp_equal\Z
426     \xint_gob_til_eightzeroes #1\XINT_cmp_continue 00000000%
427     \XINT_cmp_less }%
428 \def\XINT_cmp_less #1\W { -1}%
429 \def\XINT_cmp_continue 00000000\XINT_cmp_less {\XINT_cmp_lessequal }%
430 \def\XINT_cmp_equal\Z\xint_gob_til_eightzeroes\Z\XINT_cmp_continue
431   00000000\XINT_cmp_less\W { 0}%

```

#### 4.18 `\xintEq`, `\xintGt`, `\xintLt`

1.09a.

```

432 \def\xintEq {\romannumeral0\xinteq }\def\xinteq #1#2{\xintifeq{#1}{#2}{1}{0}}%
433 \def\xintGt {\romannumeral0\xintgt }\def\xintgt #1#2{\xintifgt{#1}{#2}{1}{0}}%
434 \def\xintLt {\romannumeral0\xintl } \def\xintl #1#2{\xintiflt{#1}{#2}{1}{0}}%

```

#### 4.19 `\xintNeq`, `\xintGtorEq`, `\xintLtorEq`

1.1. Pour `xintexpr`. No lowercase macros

```

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

```

#### 4.20 `\xintiiEq`, `\xintiiGt`, `\xintiiLt`

1.1a Pour `\xintiiexpr`. No lowercase macros.

```

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

```

## 4.21 `\xintiiNeq`, `\xintiiGtorEq`, `\xintiiLtorEq`

1.1a. Pour `\xintiiexpr`. No lowercase macros.

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

## 4.22 `\xintIsZero`, `\xintIsNotZero`, `\xintiiIsZero`, `\xintiiIsNotZero`

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

```
444 \def\xintIsZero {\romannumeral0\xintiszero }%
445 \def\xintiszero #1{\if0\xintSgn{#1}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%
446 \def\xintIsNotZero {\romannumeral0\xintisnotzero }%
447 \def\xintisnotzero
448     #1{\if0\xintSgn{#1}\xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi}%
449 \def\xintiiIsZero {\romannumeral0\xintiiiszero }%
450 \def\xintiiiszero #1{\if0\xintiiSgn{#1}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%
451 \def\xintiiIsNotZero {\romannumeral0\xintiiisnotzero }%
452 \def\xintiiisnotzero
453     #1{\if0\xintiiSgn{#1}\xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi}%
```

## 4.23 `\xintIsTrue`, `\xintNot`, `\xintIsFalse`

1.09c

```
454 \let\xintIsTrue\xintIsNotZero
455 \let\xintNot\xintIsZero
456 \let\xintIsFalse\xintIsZero
```

## 4.24 `\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

```
457 \def\xintAND {\romannumeral0\xintand }%
458 \def\xintand #1#2{\if0\xintSgn{#1}\expandafter\xint_firstoftwo
459     \else\expandafter\xint_secondoftwo\fi
460     { 0}{\xintisnotzero{#2}}}%
461 \def\xintOR {\romannumeral0\xintor }%
462 \def\xintor #1#2{\if0\xintSgn{#1}\expandafter\xint_firstoftwo
463     \else\expandafter\xint_secondoftwo\fi
464     {\xintisnotzero{#2}}{ 1}}%
465 \def\xintXOR {\romannumeral0\xintxor }%
466 \def\xintxor #1#2{\if\xintIsZero{#1}\xintIsZero{#2}%
467     \xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi }%
```

## 4.25 `\xintANDof`

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

```
468 \def\xintANDof {\romannumeral0\xintandof }%
```

```

469 \def\xintandof    #1{\expandafter\XINT_andof_a\romannumeral`&&@#1\relax }%
470 \def\XINT_andof_a #1{\expandafter\XINT_andof_b\romannumeral`&&@#1\Z }%
471 \def\XINT_andof_b #1%
472     {\xint_gob_til_relax #1\XINT_andof_e\relax\XINT_andof_c #1}%
473 \def\XINT_andof_c #1\Z
474     {\xintifTrueAelseB {#1}{\XINT_andof_a}{\XINT_andof_no}}%
475 \def\XINT_andof_no #1\relax { 0}%
476 \def\XINT_andof_e #1\Z { 1}%

```

## 4.26 `\xintORof`

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

```

477 \def\xintORof      {\romannumeral0\xintorof }%
478 \def\xintorof      #1{\expandafter\XINT_orof_a\romannumeral`&&@#1\relax }%
479 \def\XINT_orof_a #1{\expandafter\XINT_orof_b\romannumeral`&&@#1\Z }%
480 \def\XINT_orof_b #1%
481     {\xint_gob_til_relax #1\XINT_orof_e\relax\XINT_orof_c #1}%
482 \def\XINT_orof_c #1\Z
483     {\xintifTrueAelseB {#1}{\XINT_orof_yes}{\XINT_orof_a}}%
484 \def\XINT_orof_yes #1\relax { 1}%
485 \def\XINT_orof_e #1\Z { 0}%

```

## 4.27 `\xintXORof`

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

```

486 \def\xintXORof     {\romannumeral0\xintxorof }%
487 \def\xintxorof     #1{\expandafter\XINT_xorof_a\expandafter
488     0\romannumeral`&&@#1\relax }%
489 \def\XINT_xorof_a #1#2{\expandafter\XINT_xorof_b\romannumeral`&&@#2\Z #1}%
490 \def\XINT_xorof_b #1%
491     {\xint_gob_til_relax #1\XINT_xorof_e\relax\XINT_xorof_c #1}%
492 \def\XINT_xorof_c #1\Z #2%
493     {\xintifTrueAelseB {#1}{\if #20\xint_afterfi{\XINT_xorof_a 1}%
494         \else\xint_afterfi{\XINT_xorof_a 0}\fi}%
495         {\XINT_xorof_a #2}%
496     }%
497 \def\XINT_xorof_e #1\Z #2{ #2}%

```

## 4.28 `\xintGeq`, `\xintiGeq`

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

```

498 \def\xintGeq       {\romannumeral0\xintgeq }%
499 \def\xintgeq       #1{\expandafter\XINT_geq\romannumeral0\xintnum{#1}\Z }%
500 \def\xintiGeq      {\romannumeral0\xintiigeq }%
501 \def\xintiigeq     #1{\expandafter\XINT_iigeq\romannumeral`&&@#1\Z }%
502 \def\XINT_iigeq #1#2\Z #3%
503 {%
504     \expandafter\XINT_geq_fork\expandafter #1\romannumeral`&&@#3\Z #2\Z
505 }%
506 \let\XINT_geq_pre \xintiigeq % TEMPORAIRE (oui, mais depuis quand ?)

```

#### 4 Package *xint* implementation

```

507 \let\XINT_Geq \xintGeq      % TEMPORAIRE ATTENTION FAIT xintNum (et alors?)
508 \def\XINT_geq #1#2\Z #3%
509 {%
510   \expandafter\XINT_geq_fork\expandafter #1\romannumeral0\xintnum{#3}\Z #2\Z
511 }%
512 \def\XINT_geq_fork #1#2%
513 {%
514   \xint_UDzerofork
515   #1\XINT_geq_firstiszero
516   #2\XINT_geq_secondiszero
517   0{}}%
518   \krof
519   \xint_UDsignsfork
520   #1#2\XINT_geq_minusminus
521   #1-\XINT_geq_minusplus
522   #2-\XINT_geq_plusminus
523   --\XINT_geq_plusplus
524   \krof #1#2%
525 }%
526 \def\XINT_geq_firstiszero #1\krof 0#2#3\Z #4\Z
527   {\xint_UDzerofork #2{ 1}0{ 0}\krof }%
528 \def\XINT_geq_secondiszero #1\krof #20#3\Z #4\Z { 1}%
529 \def\XINT_geq_plusminus #1-{\XINT_geq_plusplus #1{}}%
530 \def\XINT_geq_minusplus -#1{\XINT_geq_plusplus {}}#1}%
531 \def\XINT_geq_minusminus --{\XINT_geq_plusplus {}}{}%
532 \def\XINT_geq_plusplus #1#2#3\Z #4\Z {\XINT_geq_pp #1#4\Z #2#3\Z }%
533 \def\XINT_geq_pp #1\Z
534 {%
535   \expandafter\XINT_geq_pp_a
536   \romannumeral0\expandafter\XINT_sepandrev_andcount
537   \romannumeral0\XINT_zeroes_forviii #1\R\R\R\R\R\R\R\R{10}0000001\W
538   #1\XINT_rsepybviii_end_A 2345678%
539   \XINT_rsepybviii_end_B 2345678\relax\xint_c_ii\xint_c_iii
540   \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
541   \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
542   \X
543 }%
544 \def\XINT_geq_pp_a #1.#2\X #3\Z
545 {%
546   \expandafter\XINT_geq_checklengths
547   \the\numexpr #1\expandafter.%
548   \romannumeral0\expandafter\XINT_sepandrev_andcount
549   \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
550   #3\XINT_rsepybviii_end_A 2345678%
551   \XINT_rsepybviii_end_B 2345678\relax \xint_c_ii\xint_c_iii
552   \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
553   \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
554   \Z!\Z!\Z!\Z!\Z!\W #2\Z!\Z!\Z!\Z!\Z!\W
555 }%
556 \def\XINT_geq_checklengths #1.#2.%
557 {%
558   \ifnum #1=#2

```

#### 4 Package *xint* implementation

```

559     \expandafter\xint_firstoftwo
560   \else
561     \expandafter\xint_secondoftwo
562   \fi
563   \XINT_geq_aa {\XINT_geq_distinctlengths {#1}{#2}}%
564 }%
565 \def\XINT_geq_distinctlengths #1#2#3\W #4\W
566 {%
567   \ifnum #1>#2
568     \expandafter\xint_firstoftwo
569   \else
570     \expandafter\xint_secondoftwo
571   \fi
572   { 1}{ 0}%
573 }%
574 \def\XINT_geq_aa {\expandafter\XINT_geq_w\the\numexpr\XINT_geq_a \xint_c_i }%
575 \def\XINT_geq_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
576 {%
577   \XINT_geq_b #1!#6!#2!#7!#3!#8!#4!#9!#5\W
578 }%
579 \def\XINT_geq_b #1#2#3!#4!%
580 {%
581   \xint_gob_til_Z #2\XINT_geq_bi \Z
582   \expandafter\XINT_geq_c\the\numexpr#1+1#4-#3-\xint_c_i.%
583 }%
584 \def\XINT_geq_c 1#1#2.%
585 {%
586   1#2\expandafter!\the\numexpr\XINT_geq_d #1%
587 }%
588 \def\XINT_geq_d #1#2#3!#4!%
589 {%
590   \xint_gob_til_Z #2\XINT_geq_di \Z
591   \expandafter\XINT_geq_e\the\numexpr#1+1#4-#3-\xint_c_i.%
592 }%
593 \def\XINT_geq_e 1#1#2.%
594 {%
595   1#2\expandafter!\the\numexpr\XINT_geq_f #1%
596 }%
597 \def\XINT_geq_f #1#2#3!#4!%
598 {%
599   \xint_gob_til_Z #2\XINT_geq_fi \Z
600   \expandafter\XINT_geq_g\the\numexpr#1+1#4-#3-\xint_c_i.%
601 }%
602 \def\XINT_geq_g 1#1#2.%
603 {%
604   1#2\expandafter!\the\numexpr\XINT_geq_h #1%
605 }%
606 \def\XINT_geq_h #1#2#3!#4!%
607 {%
608   \xint_gob_til_Z #2\XINT_geq_hi \Z
609   \expandafter\XINT_geq_i\the\numexpr#1+1#4-#3-\xint_c_i.%
610 }%

```



#### 4 Package `xint` implementation

```
611 \def\XINT_geq_i #1#2.%
612 {%
613     1#2\expandafter!\the\numexpr\XINT_geq_a #1%
614 }%
615 \def\XINT_geq_bi\Z
616     \expandafter\XINT_geq_c\the\numexpr#1+1#2-#3.#4!#5!#6!#7!#8!#9!\Z !\W
617 {%
618     \XINT_geq_k #1#2!#5!#7!#9!%
619 }%
620 \def\XINT_geq_di\Z
621     \expandafter\XINT_geq_e\the\numexpr#1+1#2-#3.#4!#5!#6!#7!#8\W
622 {%
623     \XINT_geq_k #1#2!#5!#7!%
624 }%
625 \def\XINT_geq_fi\Z
626     \expandafter\XINT_geq_g\the\numexpr#1+1#2-#3.#4!#5!#6\W
627 {%
628     \XINT_geq_k #1#2!#5!%
629 }%
630 \def\XINT_geq_hi\Z
631     \expandafter\XINT_geq_i\the\numexpr#1+1#2-#3.#4\W
632 {%
633     \XINT_geq_k #1#2!%
634 }%
635 \def\XINT_geq_k #1#2\W
636 {%
637     \xint_UDzerofork
638     #1{-1\relax { 0}}%
639     0{-1\relax { 1}}%
640     \krof
641 }%
642 \def\XINT_geq_w #1-1#2{#2}%
```

#### 4.29 `\xintiMax`, `\xintiiMax`

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

```
643 \def\xintiMax {\romannumeral0\xintimax }%
644 \def\xintimax #1%
645 {%
646     \expandafter\xint_max\expandafter {\romannumeral0\xintnum{#1}}%
647 }%
648 \def\xint_max #1#2%
649 {%
650     \expandafter\XINT_max_pre\expandafter {\romannumeral0\xintnum{#2}}{#1}%
651 }%
652 \def\xintiiMax {\romannumeral0\xintiimax }%
653 \def\xintiimax #1%
654 {%
655     \expandafter\xint_iimax\expandafter {\romannumeral`&&@#1}%
656 }%
657 \def\xint_iimax #1#2%
658 {%
```

#### 4 Package `xint` implementation

```

659 \expandafter\XINT_max_pre\expandafter {\romannumeral`&&@#2}{#1}%
660 }%
661 \def\XINT_max_pre #1#2{\XINT_max_fork #1\Z #2\Z {#2}{#1}}%
662 \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*

663 \def\XINT_max_fork #1#2\Z #3#4\Z
664 {%
665 \xint_UDsignsfork
666 #1#3\XINT_max_minusminus % A < 0, B < 0
667 #1-\XINT_max_minusplus % B < 0, A >= 0
668 #3-\XINT_max_plusminus % A < 0, B >= 0
669 --{\xint_UDzerosfork
670 #1#3\XINT_max_zerozero % A = B = 0
671 #10\XINT_max_zeropplus % B = 0, A > 0
672 #30\XINT_max_pluszero % A = 0, B > 0
673 00\XINT_max_plusplus % A, B > 0
674 \krof }%
675 \krof
676 {#2}{#4}#1#3%
677 }%

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

678 \def\XINT_max_zerozero #1#2#3#4{\xint_firstoftwo_thenstop }%
679 \def\XINT_max_zeropplus #1#2#3#4{\xint_firstoftwo_thenstop }%
680 \def\XINT_max_pluszero #1#2#3#4{\xint_secondoftwo_thenstop }%
681 \def\XINT_max_minusplus #1#2#3#4{\xint_firstoftwo_thenstop }%
682 \def\XINT_max_plusminus #1#2#3#4{\xint_secondoftwo_thenstop }%
683 \def\XINT_max_plusplus #1#2#3#4%
684 {%
685 \ifodd\XINT_Geq {#4#2}{#3#1}
686 \expandafter\xint_firstoftwo_thenstop
687 \else
688 \expandafter\xint_secondoftwo_thenstop
689 \fi
690 }%

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

691 \def\XINT_max_minusminus #1#2#3#4%
692 {%
693 \ifodd\XINT_Geq {#1}{#2}
694 \expandafter\xint_firstoftwo_thenstop
695 \else
696 \expandafter\xint_secondoftwo_thenstop
697 \fi
698 }%

```

#### 4.30 `\xintiMaxof`, `\xintiiMaxof`

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

```

699 \def\xintiMaxof      {\romannumeral0\xintimaxof }%
700 \def\xintimaxof      #1{\expandafter\XINT_imaxof_a\romannumeral`&&#1\relax }%
701 \def\XINT_imaxof_a    #1{\expandafter\XINT_imaxof_b\romannumeral0\xintnum{#1}\Z }%
702 \def\XINT_imaxof_b    #1\Z #2%
703       {\expandafter\XINT_imaxof_c\romannumeral`&&#2\Z {#1}\Z}%
704 \def\XINT_imaxof_c    #1%
705       {\xint_gob_til_relax #1\XINT_imaxof_e\relax\XINT_imaxof_d #1}%
706 \def\XINT_imaxof_d    #1\Z
707       {\expandafter\XINT_imaxof_b\romannumeral0\xintimax {#1}}%
708 \def\XINT_imaxof_e    #1\Z #2\Z { #2}%
709 \def\xintiMaxof      {\romannumeral0\xintiimaxof }%
710 \def\xintiimaxof      #1{\expandafter\XINT_iimaxof_a\romannumeral`&&#1\relax }%
711 \def\XINT_iimaxof_a    #1{\expandafter\XINT_iimaxof_b\romannumeral`&&#1\Z }%
712 \def\XINT_iimaxof_b    #1\Z #2%
713       {\expandafter\XINT_iimaxof_c\romannumeral`&&#2\Z {#1}\Z}%
714 \def\XINT_iimaxof_c    #1%
715       {\xint_gob_til_relax #1\XINT_iimaxof_e\relax\XINT_iimaxof_d #1}%
716 \def\XINT_iimaxof_d    #1\Z
717       {\expandafter\XINT_iimaxof_b\romannumeral0\xintiimax {#1}}%
718 \def\XINT_iimaxof_e    #1\Z #2\Z { #2}%

```

#### 4.31 \xintiMin, \xintiMin

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

```

719 \def\xintiMin {\romannumeral0\xintimin }%
720 \def\xintimin #1%
721 {%
722   \expandafter\xint_min\expandafter {\romannumeral0\xintnum{#1}}%
723 }%
724 \def\xint_min #1#2%
725 {%
726   \expandafter\XINT_min_pre\expandafter {\romannumeral0\xintnum{#2}}{#1}%
727 }%
728 \def\xintiMin {\romannumeral0\xintiimin }%
729 \def\xintiimin #1%
730 {%
731   \expandafter\xint_iimin\expandafter {\romannumeral`&&#1}%
732 }%
733 \def\xint_iimin #1#2%
734 {%
735   \expandafter\XINT_min_pre\expandafter {\romannumeral`&&#2}{#1}%
736 }%
737 \def\XINT_min_pre #1#2{\XINT_min_fork #1\Z #2\Z {#2}{#1}}%
738 \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\*

```

739 \def\XINT_min_fork #1#2\Z #3#4\Z
740 {%
741   \xint_UDsignsfork

```

#### 4 Package `xint` implementation

```

742      #1#3\XINT_min_minusminus % A < 0, B < 0
743      #1-\XINT_min_minusplus % B < 0, A >= 0
744      #3-\XINT_min_plusminus % A < 0, B >= 0
745      --{\xint_UDzerosfork
746          #1#3\XINT_min_zerozero % A = B = 0
747          #10\XINT_min_zeropplus % B = 0, A > 0
748          #30\XINT_min_pluszero % A = 0, B > 0
749          00\XINT_min_plusplus % A, B > 0
750      \krof }%
751  \krof
752  {#2}{#4}#1#3%
753 }%

```

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

```

754 \def\XINT_min_zerozero #1#2#3#4{\xint_firstoftwo_thenstop }%
755 \def\XINT_min_zeropplus #1#2#3#4{\xint_secondoftwo_thenstop }%
756 \def\XINT_min_pluszero #1#2#3#4{\xint_firstoftwo_thenstop }%
757 \def\XINT_min_minusplus #1#2#3#4{\xint_secondoftwo_thenstop }%
758 \def\XINT_min_plusminus #1#2#3#4{\xint_firstoftwo_thenstop }%
759 \def\XINT_min_plusplus #1#2#3#4%
760 {%
761   \ifodd\XINT_Geq {#4#2}{#3#1}
762   \expandafter\xint_secondoftwo_thenstop
763   \else
764   \expandafter\xint_firstoftwo_thenstop
765   \fi
766 }%

```

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

```

767 \def\XINT_min_minusminus #1#2#3#4%
768 {%
769   \ifodd\XINT_Geq {#1}{#2}
770   \expandafter\xint_secondoftwo_thenstop
771   \else
772   \expandafter\xint_firstoftwo_thenstop
773   \fi
774 }%

```

#### 4.32 `\xintiMinof`, `\xintiiMinof`

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

```

775 \def\xintiMinof {\romannumeral0\xintiminof }%
776 \def\xintiminof #1{\expandafter\XINT_iminof_a\romannumeral`&&@#1\relax }%
777 \def\XINT_iminof_a #1{\expandafter\XINT_iminof_b\romannumeral0\xintnum{#1}\Z }%
778 \def\XINT_iminof_b #1\Z #2%
779   {\expandafter\XINT_iminof_c\romannumeral`&&@#2\Z {#1}\Z}%
780 \def\XINT_iminof_c #1%
781   {\xint_gob_til_relax #1\XINT_iminof_e\relax\XINT_iminof_d #1}%
782 \def\XINT_iminof_d #1\Z
783   {\expandafter\XINT_iminof_b\romannumeral0\xintimin {#1}}%
784 \def\XINT_iminof_e #1\Z #2\Z { #2}%

```

```

785 \def\xintiMinof      {\romannumeral0\xintiiminof }%
786 \def\xintiiminof    #1{\expandafter\XINT_iiminof_a\romannumeral`&&@#1\relax }%
787 \def\XINT_iiminof_a #1{\expandafter\XINT_iiminof_b\romannumeral`&&@#1\Z }%
788 \def\XINT_iiminof_b #1\Z #2%
789     {\expandafter\XINT_iiminof_c\romannumeral`&&@#2\Z {#1}\Z}%
790 \def\XINT_iiminof_c #1%
791     {\xint_gob_til_relax #1\XINT_iiminof_e\relax\XINT_iiminof_d #1}%
792 \def\XINT_iiminof_d #1\Z
793     {\expandafter\XINT_iiminof_b\romannumeral0\xintiimin {#1}}%
794 \def\XINT_iiminof_e #1\Z #2\Z { #2}%

```

### 4.33 \xintiiSum

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

```

795 \def\xintiiSum {\romannumeral0\xintiisum }%
796 \def\xintiisum #1{\xintiisumexpr #1\relax }%
797 \def\xintiiSumExpr {\romannumeral0\xintiisumexpr }%
798 \def\xintiisumexpr {\expandafter\XINT_sumexpr\romannumeral`&&@}%
799 \def\XINT_sumexpr {\XINT_sum_loop_a 0\Z }%
800 \def\XINT_sum_loop_a #1\Z #2%
801     {\expandafter\XINT_sum_loop_b \romannumeral`&&@#2\Z #1\Z \Z}%
802 \def\XINT_sum_loop_b #1%
803     {\xint_gob_til_relax #1\XINT_sum_finished\relax\XINT_sum_loop_c #1}%
804 \def\XINT_sum_loop_c
805     {\expandafter\XINT_sum_loop_a\romannumeral0\XINT_add_fork }%
806 \def\XINT_sum_finished #1\Z #2\Z \Z { #2}%

```

### 4.34 \xintiiPrd

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

```

807 \def\xintiiPrd {\romannumeral0\xintiiprd }%
808 \def\xintiiprd #1{\xintiiprdexpr #1\relax }%
809 \def\xintiiPrdExpr {\romannumeral0\xintiiprdexpr }%
810 \def\xintiiprdexpr {\expandafter\XINT_prdexpr\romannumeral`&&@}%
811 \def\XINT_prdexpr {\XINT_prod_loop_a 1\Z }%
812 \def\XINT_prod_loop_a #1\Z #2%
813     {\expandafter\XINT_prod_loop_b \romannumeral`&&@#2\Z #1\Z \Z}%
814 \def\XINT_prod_loop_b #1%
815     {\xint_gob_til_relax #1\XINT_prod_finished\relax\XINT_prod_loop_c #1}%
816 \def\XINT_prod_loop_c
817     {\expandafter\XINT_prod_loop_a\romannumeral0\XINT_mul_fork }%
818 \def\XINT_prod_finished\relax\XINT_prod_loop_c #1\Z #2\Z \Z { #2}%

```

---

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

**4.35  $\backslash$ xintMON,  $\backslash$ xintMMON,  $\backslash$ xintiiMON,  $\backslash$ xintiiMMON**MINUS ONE TO THE POWER N and  $(-1)^{N-1}$ 

```

819 \def\xintiiMON {\romannumeral0\xintiimon }%
820 \def\xintiimon #1%
821 {%
822   \ifodd\xintiilDg {#1}
823     \xint_afterfi{ -1}%
824   \else
825     \xint_afterfi{ 1}%
826   \fi
827 }%
828 \def\xintiiMMON {\romannumeral0\xintiimmon }%
829 \def\xintiimmon #1%
830 {%
831   \ifodd\xintiilDg {#1}
832     \xint_afterfi{ 1}%
833   \else
834     \xint_afterfi{ -1}%
835   \fi
836 }%
837 \def\xintMON {\romannumeral0\xintmon }%
838 \def\xintmon #1%
839 {%
840   \ifodd\xintLDg {#1}
841     \xint_afterfi{ -1}%
842   \else
843     \xint_afterfi{ 1}%
844   \fi
845 }%
846 \def\xintMMON {\romannumeral0\xintmmon }%
847 \def\xintmmon #1%
848 {%
849   \ifodd\xintLDg {#1}
850     \xint_afterfi{ 1}%
851   \else
852     \xint_afterfi{ -1}%
853   \fi
854 }%

```

**4.36  $\backslash$ xintOdd,  $\backslash$ xintiiOdd,  $\backslash$ xintEven,  $\backslash$ xintiiEven**

```

855 \def\xintiiOdd {\romannumeral0\xintiiodd }%
856 \def\xintiiodd #1%
857 {%
858   \ifodd\xintiilDg{#1}
859     \xint_afterfi{ 1}%
860   \else
861     \xint_afterfi{ 0}%
862   \fi
863 }%
864 \def\xintiiEven {\romannumeral0\xintiieven }%

```

```

865 \def\xintiieven #1%
866 {%
867   \ifodd\xintiilDg{#1}
868     \xint_afterfi{ 0}%
869   \else
870     \xint_afterfi{ 1}%
871   \fi
872 }%
873 \def\xintOdd {\romannumeral0\xintodd }%
874 \def\xintodd #1%
875 {%
876   \ifodd\xintLDg{#1}
877     \xint_afterfi{ 1}%
878   \else
879     \xint_afterfi{ 0}%
880   \fi
881 }%
882 \def\xintEven {\romannumeral0\xinteven }%
883 \def\xinteven #1%
884 {%
885   \ifodd\xintLDg{#1}
886     \xint_afterfi{ 0}%
887   \else
888     \xint_afterfi{ 1}%
889   \fi
890 }%

```

#### 4.37 `\xintDSL`

DECIMAL SHIFT LEFT (=MULTIPLICATION PAR 10)

```

891 \def\xintDSL {\romannumeral0\xintdsl }%
892 \def\xintdsl #1%
893 {%
894   \expandafter\XINT_dsl \romannumeral`&&@#1\Z
895 }%
896 \def\XINT_DSL #1{\romannumeral0\XINT_dsl #1\Z }%
897 \def\XINT_dsl #1%
898 {%
899   \xint_gob_til_zero #1\xint_dsl_zero 0\XINT_dsl_ #1%
900 }%
901 \def\xint_dsl_zero 0\XINT_dsl_ 0#1\Z { 0}%
902 \def\XINT_dsl_ #1\Z { #10}%

```

#### 4.38 `\xintDSR`

DECIMAL SHIFT RIGHT (=DIVISION PAR 10).

```

903 \def\xintDSR {\romannumeral0\xintdsr }%
904 \def\xintdsr #1%
905 {%
906   \expandafter\XINT_dsr_a\expandafter {\romannumeral`&&@#1}\W\Z
907 }%

```

```

908 \def\XINT_DSR #1{\romannumeral0\XINT_dsr_a {#1}\W\Z }%
909 \def\XINT_dsr_a
910 {%
911   \expandafter\XINT_dsr_b\romannumeral0\xintreverseorder
912 }%
913 \def\XINT_dsr_b #1#2#3\Z
914 {%
915   \xint_gob_til_W #2\xint_dsr_onedigit\W
916   \xint_gob_til_minus #2\xint_dsr_onedigit-%
917   \expandafter\XINT_dsr_removev
918   \romannumeral0\xintreverseorder {#2#3}%
919 }%
920 \def\xint_dsr_onedigit #1\xintreverseorder #2{ 0}%
921 \def\XINT_dsr_removev #1\W { }%

```

#### 4.39 \xintDSH, \xintDSHr

DECIMAL SHIFTS \xintDSH {x}{A}

si  $x \leq 0$ , fait  $A \rightarrow A \cdot 10^{|x|}$ . 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).

```

922 \def\xintDSHr {\romannumeral0\xintdshr }%
923 \def\xintdshr #1%
924 {%
925   \expandafter\XINT_dshr_checkxpositive \the\numexpr #1\relax\Z
926 }%
927 \def\XINT_dshr_checkxpositive #1%
928 {%
929   \xint_UDzerominusfork
930   0#1\XINT_dshr_xzeroorneg
931   #1-\XINT_dshr_xzeroorneg
932   0-\XINT_dshr_xpositive
933   \krof #1%
934 }%
935 \def\XINT_dshr_xzeroorneg #1\Z #2{ 0}%
936 \def\XINT_dshr_xpositive #1\Z
937 {%
938   \expandafter\xint_secondoftwo_thenstop\romannumeral0\xintdsx {#1}%
939 }%
940 \def\xintDSH {\romannumeral0\xintdsh }%
941 \def\xintdsh #1#2%
942 {%
943   \expandafter\xint_dsh\expandafter {\romannumeral`&&@#2}{#1}%
944 }%
945 \def\xint_dsh #1#2%
946 {%
947   \expandafter\XINT_dsh_checksignx \the\numexpr #2\relax\Z {#1}%
948 }%
949 \def\XINT_dsh_checksignx #1%
950 {%
951   \xint_UDzerominusfork

```



```

952      #1-\XINT_dsh_xiszero
953      0#1\XINT_dsx_xisNeg_checkA      % on passe direct dans DSx
954      0-\XINT_dsh_xisPos #1}%
955      \krof
956 }%
957 \def\XINT_dsh_xiszero #1\Z #2{ #2}%
958 \def\XINT_dsh_xisPos #1\Z #2%
959 {%
960      \expandafter\xint_firstoftwo_thenstop
961      \romannumeral0\XINT_dsx_checksiga #2\Z {#1}% via DSx
962 }%

```

#### 4.40 \xintDSx

--> 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  $10^x Q \pmod R$  où il faut prendre le signe plus si  $Q$  est positif ou nul et le signe moins si  $Q$  est strictement négatif.

December 4, 2015: As the new techniques of the 1.2 release limit the basic arithmetic to less than about 20000 digits, I should again consider a faster `dsx_loop` possibly impacting the input save stack impacting style; preserving a special routine if really needed for `\xintXTrunc` or the other concerned routines (if `\xintXTrunc` really needs it, which I have to check.)

2016/03/12: this is old code. 1.2f has removed the part of it which checked the size of `x` parameter.

```

963 \def\xintDSx {\romannumeral0\xintdsx }%
964 \def\xintdsx #1#2%
965 {%
966      \expandafter\xint_dsx\expandafter {\romannumeral`&&@#2}{#1}%
967 }%
968 \def\xint_dsx #1#2%
969 {%
970      \expandafter\XINT_dsx_checksiga \the\numexpr #2\relax\Z {#1}%
971 }%
972 \def\XINT_DSx #1#2{\romannumeral0\XINT_dsx_checksiga #1\Z {#2}}%
973 \def\XINT_dsx #1#2{\XINT_dsx_checksiga #1\Z {#2}}%
974 \def\XINT_dsx_checksiga #1%
975 {%
976      \xint_UDzerominusfork
977      #1-\XINT_dsx_xisZero
978      0#1\XINT_dsx_xisNeg_checkA
979      0-\XINT_dsx_xisPos #1}%
980      \krof
981 }%
982 \def\XINT_dsx_xisZero #1\Z #2{{#2}{0}}% attention comme  $x > 0$ 
983 \def\XINT_dsx_xisNeg_checkA #1\Z #2%
984 {%
985      \XINT_dsx_xisNeg_checkA_ #2\Z {#1}%
986 }%

```

#### 4 Package *xint* implementation

```

987 \def\XINT_dsx_xisNeg_checkA_ #1#2\Z #3%
988 {%
989     \xint_gob_til_zero #1\XINT_dsx_xisNeg_Azero 0%
990     \XINT_dsx_zeroloop #3.{}\Z {#1#2}%
991 }%
992 \def\XINT_dsx_xisNeg_Azero #1\Z #2{ 0}%
993 \def\XINT_dsx_addzerosnofuss #1{\XINT_dsx_zeroloop #1.{}\Z }%
994 \def\XINT_dsx_zeroloop #1.#2%
995 {%
996     \ifnum #1<\xint_c_ix \expandafter\XINT_dsx_exita\fi
997     \expandafter\XINT_dsx_zeroloop\the\numexpr #1-\xint_c_viii.{#2000000000}%
998 }%
999 \def\XINT_dsx_exita
1000     \expandafter\XINT_dsx_zeroloop\the\numexpr #1-\xint_c_viii.#2%
1001 {%
1002     \expandafter\expandafter\expandafter
1003     \XINT_dsx_addzeros\csname xint_gobble_\expandafter
1004         \romannumeral\numexpr \xint_c_viii-(#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
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_checkiffirstempty
1023     \romannumeral0\XINT_split_xfork #2.#1\W\W\W\W\W\W\W\W\Z
1024 }%
1025 \def\XINT_dsx_AisNeg_checkiffirstempty #1%
1026 {%
1027     \xint_gob_til_dot #1\XINT_dsx_AisNeg_finish_zero.%
1028     \XINT_dsx_AisNeg_finish_notzero #1%
1029 }%
1030 \def\XINT_dsx_AisNeg_finish_zero.\XINT_dsx_AisNeg_finish_notzero.#1.%
1031 {%
1032     \expandafter\XINT_dsx_end
1033     \expandafter {\romannumeral0\XINT_num {-#1}}{0}%
1034 }%
1035 \def\XINT_dsx_AisNeg_finish_notzero #1.#2.%
1036 {%
1037     \expandafter\XINT_dsx_end
1038     \expandafter {\romannumeral0\XINT_num {#2}}{-#1}%

```

```

1039 }%
1040 \def\XINT_dsx_AisPos #1\Z #2%
1041 {%
1042   \expandafter\XINT_dsx_AisPos_finish
1043   \romannumeral0\XINT_split_xfork #2.#1\W\W\W\W\W\W\W\W\Z
1044 }%
1045 \def\XINT_dsx_AisPos_finish #1.#2.%
1046 {%
1047   \expandafter\XINT_dsx_end
1048   \expandafter {\romannumeral0\XINT_num {#2}}%
1049   {\romannumeral0\XINT_num {#1}}%
1050 }%
1051 \def\XINT_dsx_end #1#2{\expandafter{#2}{#1}}%

```

#### 4.41 \xintDecSplit, \xintDecSplitL, \xintDecSplitR

##### DECIMAL SPLIT

The macro `\xintDecSplit {x}{A}` first replaces `A` with `|A|` (\*) This macro cuts the number into two pieces `L` and `R`. The concatenation `LR` always reproduces `|A|`, and `R` may be empty or have leading zeros. The position of the cut is specified by the first argument `x`. If `x` is zero or positive the cut location is `x` slots to the left of the right end of the number. If `x` becomes equal to or larger than the length of the number then `L` becomes empty. If `x` is negative the location of the cut is `|x|` slots to the right of the left end of the number.

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

2016/03/12: this is old code. 1.2f has removed the part of it which checked the size of `x` parameter.

```

1052 \def\xintDecSplit {\romannumeral0\xintdecsplit }%
1053 \def\xintdecsplit #1#2%
1054 {%
1055   \expandafter\XINT_split_finish
1056   \romannumeral0\expandafter\XINT_split_xfork
1057   \the\numexpr #1\expandafter.\romannumeral0\xinttiabs {#2}\W\W\W\W\W\W\W\W\Z
1058 }%
1059 \def\xintDecSplitL {\romannumeral0\xintdecsplitl }%
1060 \def\xintdecsplitl #1#2%
1061 {%
1062   \expandafter\XINT_splitl_finish
1063   \romannumeral0\expandafter\XINT_split_xfork
1064   \the\numexpr #1\expandafter.\romannumeral0\xinttiabs {#2}\W\W\W\W\W\W\W\W\Z
1065 }%
1066 \def\xintDecSplitR {\romannumeral0\xintdecsplitr }%
1067 \def\xintdecsplitr #1#2%
1068 {%
1069   \expandafter\XINT_splitr_finish
1070   \romannumeral0\expandafter\XINT_split_xfork
1071   \the\numexpr #1\expandafter.\romannumeral0\xinttiabs {#2}\W\W\W\W\W\W\W\W\Z
1072 }%
1073 \def\XINT_split_finish #1.#2.{{#1}{#2}}%
1074 \def\XINT_splitl_finish #1.#2.{{#1}}%
1075 \def\XINT_splitr_finish #1.#2.{{#2}}%
1076 \def\XINT_split_xfork #1%

```

#### 4 Package `xint` implementation

```

1077 {%
1078   \xint_UDzerominusfork
1079   #1-\XINT_split_zerospit
1080   0#1\XINT_split_fromleft
1081   0-\XINT_split_fromright #1}%
1082   \krof
1083 }%
1084 \def\XINT_split_zerospit #1.#2\W\W\W\W\W\W\W\W\Z{ #2..}%
1085 \def\XINT_split_fromleft #1.%#2\W\W\W\W\W\W\W\W\Z
1086 {%
1087   \XINT_split_fromleft_loop #1.{}%#2\W\W\W\W\W\W\W\W\Z
1088 }%
1089 \def\XINT_split_fromleft_loop #1.%
1090 {%
1091   \ifnum #1<\xint_c_viii\expandafter\XINT_split_fromleft_exita\fi
1092   \expandafter\XINT_split_fromleft_loop_perhaps
1093   \the\numexpr #1-\xint_c_viii\expandafter.\XINT_split_fromleft_eight
1094 }%
1095 \def\XINT_split_fromleft_eight #1#2#3#4#5#6#7#8#9{#9{#1#2#3#4#5#6#7#8#9}}%
1096 \def\XINT_split_fromleft_loop_perhaps #1.#2%
1097 {%
1098   \xint_gob_til_W #2\XINT_split_fromleft_toofar\W
1099   \XINT_split_fromleft_loop #1.%
1100 }%
1101 \def\XINT_split_fromleft_toofar\W\XINT_split_fromleft_loop #1.#2#3\Z
1102 {%
1103   \XINT_split_fromleft_toofar_b #2\Z
1104 }%
1105 \def\XINT_split_fromleft_toofar_b #1\W #2\Z { #1..}%
1106 \def\XINT_split_fromleft_exita
1107   \expandafter\XINT_split_fromleft_loop_perhaps
1108   \the\numexpr #1-\xint_c_viii\expandafter.\XINT_split_fromleft_eight
1109 {%
1110   \csname XINT_split_fromleft_endsplit_\romannumeral #1\endcsname
1111 }%
1112 \def\XINT_split_fromleft_endsplit_ #1#2\W #3\Z { #1.#2.%}
1113 \def\XINT_split_fromleft_endsplit_i #1#2%
1114   {\XINT_split_fromleft_checkiftoofar #2{#1#2}}%
1115 \def\XINT_split_fromleft_endsplit_ii #1#2#3%
1116   {\XINT_split_fromleft_checkiftoofar #3{#1#2#3}}%
1117 \def\XINT_split_fromleft_endsplit_iii #1#2#3#4%
1118   {\XINT_split_fromleft_checkiftoofar #4{#1#2#3#4}}%
1119 \def\XINT_split_fromleft_endsplit_iv #1#2#3#4#5%
1120   {\XINT_split_fromleft_checkiftoofar #5{#1#2#3#4#5}}%
1121 \def\XINT_split_fromleft_endsplit_v #1#2#3#4#5#6%
1122   {\XINT_split_fromleft_checkiftoofar #6{#1#2#3#4#5#6}}%
1123 \def\XINT_split_fromleft_endsplit_vi #1#2#3#4#5#6#7%
1124   {\XINT_split_fromleft_checkiftoofar #7{#1#2#3#4#5#6#7}}%
1125 \def\XINT_split_fromleft_endsplit_vii #1#2#3#4#5#6#7#8%
1126   {\XINT_split_fromleft_checkiftoofar #8{#1#2#3#4#5#6#7#8}}%
1127 \def\XINT_split_fromleft_checkiftoofar #1#2#3\W #4\Z
1128 {%

```

#### 4 Package *xint* implementation

```

1129 \xint_gob_til_W #1\XINT_split_fromleft_wenttoofar\W \space#2.#3.%
1130 }%
1131 \def\XINT_split_fromleft_wenttoofar\W\space #1.%
1132 {%
1133 \XINT_split_fromleft_wenttoofar_b #1\Z
1134 }%
1135 \def\XINT_split_fromleft_wenttoofar_b #1\W #2\Z { #1.}%
1136 \def\XINT_split_fromright #1.#2\W\W\W\W\W\W\W\W\Z
1137 {%
1138 \expandafter\XINT_split_fromright_loop
1139 \the\numexpr #1\expandafter.\expandafter{\expandafter}%
1140 \romannumeral0\xintreverseorder{#2}\W\W\W\W\W\W\W\W\Z #2.%
1141 }%
1142 \def\XINT_split_fromright_loop #1.%
1143 {%
1144 \ifnum #1<\xint_c_viii\expandafter\XINT_split_fromright_exita\fi
1145 \expandafter\XINT_split_fromright_loop_perhaps
1146 \the\numexpr #1-\xint_c_viii\expandafter.\XINT_split_fromright_eight
1147 }%
1148 \def\XINT_split_fromright_eight #1#2#3#4#5#6#7#8#9{#9{#9#8#7#6#5#4#3#2#1}}%
1149 \def\XINT_split_fromright_loop_perhaps #1.#2%
1150 {%
1151 \xint_gob_til_W #2\XINT_split_fromright_toofar\W
1152 \XINT_split_fromright_loop #1.%
1153 }%
1154 \def\XINT_split_fromright_toofar\W\XINT_split_fromright_loop #1\Z {.%}%
1155 \def\XINT_split_fromright_exita
1156 \expandafter\XINT_split_fromright_loop_perhaps
1157 \the\numexpr #1-\xint_c_viii\expandafter.\XINT_split_fromright_eight
1158 {%
1159 \csname XINT_split_fromright_endsplit_\romannumeral #1\endcsname
1160 }%
1161 \def\XINT_split_fromright_endsplit_ #1#2\W #3\Z #4.%
1162 {%
1163 \xintreverseorder {#2}.#1.%
1164 }%
1165 \def\XINT_split_fromright_endsplit_i #1#2%
1166 {\XINT_split_fromright_checkiftoofar #2{#2#1}}%
1167 \def\XINT_split_fromright_endsplit_ii #1#2#3%
1168 {\XINT_split_fromright_checkiftoofar #3{#3#2#1}}%
1169 \def\XINT_split_fromright_endsplit_iii #1#2#3#4%
1170 {\XINT_split_fromright_checkiftoofar #4{#4#3#2#1}}%
1171 \def\XINT_split_fromright_endsplit_iv #1#2#3#4#5%
1172 {\XINT_split_fromright_checkiftoofar #5{#5#4#3#2#1}}%
1173 \def\XINT_split_fromright_endsplit_v #1#2#3#4#5#6%
1174 {\XINT_split_fromright_checkiftoofar #6{#6#5#4#3#2#1}}%
1175 \def\XINT_split_fromright_endsplit_vi #1#2#3#4#5#6#7%
1176 {\XINT_split_fromright_checkiftoofar #7{#7#6#5#4#3#2#1}}%
1177 \def\XINT_split_fromright_endsplit_vii #1#2#3#4#5#6#7#8%
1178 {\XINT_split_fromright_checkiftoofar #8{#8#7#6#5#4#3#2#1}}%
1179 \def\XINT_split_fromright_checkiftoofar #1%
1180 {%

```

```

1181 \xint_gob_til_W #1\XINT_split_fromright_wenttoofar\W
1182 \XINT_split_fromright_endsplit_
1183 }%
1184 \def\XINT_split_fromright_wenttoofar\W
1185 \XINT_split_fromright_endsplit_ #1\Z {.%}

```

#### 4.42 \xintiiSqrt, \xintiiSqrtR, \xintiiSquareRoot

First done with 1.08.

1.1 added \xintiiSquareRoot.

1.1a added \xintiiSqrtR.

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

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

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

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

```

1186 \def\xintiiSqrt {\romannumeral0\xintiisqrt }%
1187 \def\xintiiSqrtR {\romannumeral0\xintiisqrtr }%
1188 \def\xintiiSquareRoot {\romannumeral0\xintiisquareroot }%
1189 \def\xintiSqrt {\romannumeral0\xintisqrt }%
1190 \def\xintiSquareRoot {\romannumeral0\xintisquareroot }%
1191 \def\xintisqrt {\expandafter\XINT_sqrt_post\romannumeral0\xintisquareroot }%
1192 \def\xintiisqrt {\expandafter\XINT_sqrt_post\romannumeral0\xintiisquareroot }%
1193 \def\xintiisqrtr {\expandafter\XINT_sqrtr_post\romannumeral0\xintiisquareroot }%
1194 \def\XINT_sqrt_post #1#2{\XINT_dec_pos #1\Z }%

```

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

```

1195 \def\XINT_sqrtr_post #1#2{\xintiiiflt {#2}{#1}{ #1}{\XINT_dec_pos #1\Z}}%
1196 \def\xintisquareroot #1%
1197 {\expandafter\XINT_sqrt_checkin\romannumeral0\xintnum{#1}\Z }%
1198 \def\xintiisquareroot #1{\expandafter\XINT_sqrt_checkin\romannumeral`&&@#1\Z }%
1199 \def\XINT_sqrt_checkin #1%
1200 {%
1201 \xint_UDzerominusfork
1202 #1-\XINT_sqrt_iszero
1203 0#1\XINT_sqrt_isneg
1204 0-\XINT_sqrt #1}%
1205 \krof
1206 }%

```

#### 4 Package *xint* implementation

```

1207 \def\XINT_sqrt_iszero #1\Z { 11}%
1208 \edef\XINT_sqrt_isneg #1\Z {\noexpand\xintError:RootOfNegative\space 11}%
1209 \def\XINT_sqrt #1\Z
1210 {%
1211   \expandafter\XINT_sqrt_start\romannumeral0\xintlenght {#1}.#1.%
1212 }%
1213 \def\XINT_sqrt_start #1.%
1214 {%
1215   \ifnum #1<\xint_c_x\xint_dothis\XINT_sqrt_small_a\fi
1216   \xint_orthat\XINT_sqrt_big_a #1.%
1217 }%
1218 \def\XINT_sqrt_small_a #1.{\XINT_sqrt_a #1.\XINT_sqrt_small_d }%
1219 \def\XINT_sqrt_big_a #1.{\XINT_sqrt_a #1.\XINT_sqrt_big_d }%
1220 \def\XINT_sqrt_a #1.%
1221 {%
1222   \ifodd #1
1223     \expandafter\XINT_sqrt_b0
1224   \else
1225     \expandafter\XINT_sqrt_bE
1226   \fi
1227   #1.%
1228 }%
1229 \def\XINT_sqrt_bE #1.#2#3#4%
1230 {%
1231   \XINT_sqrt_c {#3#4}#2{#1}#3#4%
1232 }%
1233 \def\XINT_sqrt_b0 #1.#2#3%
1234 {%
1235   \XINT_sqrt_c #3#2{#1}#3%
1236 }%
1237 \def\XINT_sqrt_c #1#2%
1238 {%
1239   \expandafter #2%
1240   \the\numexpr \ifnum #1>\xint_c_ii
1241     \ifnum #1>\xint_c_vi
1242       \ifnum #1>12 \ifnum #1>20 \ifnum #1>30
1243         \ifnum #1>42 \ifnum #1>56 \ifnum #1>72
1244           \ifnum #1>90
1245             10\else 9\fi \else 8\fi \else 7\fi \else 6\fi \else 5\fi
1246             \else 4\fi \else 3\fi \else 2\fi \else 1\fi .%
1247 }%
1248 \def\XINT_sqrt_small_d #1.#2%
1249 {%
1250   \expandafter\XINT_sqrt_small_e
1251   \the\numexpr #1\ifcase \numexpr #2/\xint_c_ii-\xint_c_i\relax
1252     \or 0\or 00\or 000\or 0000\fi .%
1253 }%
1254 \def\XINT_sqrt_small_e #1.#2.%
1255 {%
1256   \expandafter\XINT_sqrt_small_ea\the\numexpr #1*#1-#2.#1.%
1257 }%
1258 \def\XINT_sqrt_small_ea #1%

```

#### 4 Package *xint* implementation

```

1259 {%
1260   \if0#1\xint_dothis\XINT_sqrt_small_ez\fi
1261   \if-#1\xint_dothis\XINT_sqrt_small_eb\fi
1262   \xint_orthat\XINT_sqrt_small_f #1%
1263 }%
1264 \def\XINT_sqrt_small_ez 0.#1.{\expandafter{\the\numexpr#1+\xint_c_i
1265   \expandafter}\expandafter{\the\numexpr #1*\xint_c_ii+\xint_c_i}}%
1266 \def\XINT_sqrt_small_eb -#1.#2.%
1267 {%
1268   \expandafter\XINT_sqrt_small_ec \the\numexpr
1269   (#1-\xint_c_i+#2)/(\xint_c_ii*#2).#1.#2.%
1270 }%
1271 \def\XINT_sqrt_small_ec #1.#2.#3.%
1272 {%
1273   \expandafter\XINT_sqrt_small_f \the\numexpr
1274   -#2+\xint_c_ii*#3*#1+#1*#1\expandafter.\the\numexpr #3+#1.%
1275 }%
1276 \def\XINT_sqrt_small_f #1.#2.%
1277 {%
1278   \expandafter\XINT_sqrt_small_g
1279   \the\numexpr (#1+#2)/(\xint_c_ii*#2)-\xint_c_i.#1.#2.%
1280 }%
1281 \def\XINT_sqrt_small_g #1#2.%
1282 {%
1283   \if 0#1%
1284     \expandafter\XINT_sqrt_small_end
1285   \else
1286     \expandafter\XINT_sqrt_small_h
1287   \fi
1288   #1#2.%
1289 }%
1290 \def\XINT_sqrt_small_h #1.#2.#3.%
1291 {%
1292   \expandafter\XINT_sqrt_small_f
1293   \the\numexpr #2-\xint_c_ii*#1*#3+#1*#1\expandafter.%
1294   \the\numexpr #3-#1.%
1295 }%
1296 \def\XINT_sqrt_small_end #1.#2.#3.{{#3}{#2}}%
1297 \def\XINT_sqrt_big_d #1.#2%
1298 {%
1299   \ifodd #2 \xint_dothis{\expandafter\XINT_sqrt_big_e0}\fi
1300   \xint_orthat{\expandafter\XINT_sqrt_big_eE}%
1301   \the\numexpr (#2-\xint_c_i)/\xint_c_ii.#1;%
1302 }%
1303 \def\XINT_sqrt_big_eE #1;#2#3#4#5#6#7#8#9%
1304 {%
1305   \XINT_sqrt_big_eE_a #1;{#2#3#4#5#6#7#8#9}%
1306 }%
1307 \def\XINT_sqrt_big_eE_a #1.#2;#3%
1308 {%
1309   \expandafter\XINT_sqrt_bigormed_f
1310   \romannumeral0\XINT_sqrt_small_e #2000.#3.#1;%

```



#### 4 Package *xint* implementation

```

1311 }%
1312 \def\XINT_sqrt_big_e0 #1;#2#3#4#5#6#7#8#9%
1313 {%
1314   \XINT_sqrt_big_e0_a #1;{#2#3#4#5#6#7#8#9}%
1315 }%
1316 \def\XINT_sqrt_big_e0_a #1.#2;#3#4%
1317 {%
1318   \expandafter\XINT_sqrt_bigormed_f
1319   \romannumeral0\XINT_sqrt_small_e #20000.#3#4.#1;%
1320 }%
1321 \def\XINT_sqrt_bigormed_f #1#2#3;%
1322 {%
1323   \ifnum#3<\xint_c_ix
1324     \xint_dothis {\csname XINT_sqrt_med_f\romannumeral#3\endcsname}%
1325   \fi
1326   \xint_orthat\XINT_sqrt_big_f #1.#2.#3;%
1327 }%
1328 \def\XINT_sqrt_med_fv {\XINT_sqrt_med_fa .}%
1329 \def\XINT_sqrt_med_fvi {\XINT_sqrt_med_fa 0.}%
1330 \def\XINT_sqrt_med_fvii {\XINT_sqrt_med_fa 00.}%
1331 \def\XINT_sqrt_med_fviii{\XINT_sqrt_med_fa 000.}%
1332 \def\XINT_sqrt_med_fa #1.#2.#3.#4;%
1333 {%
1334   \expandafter\XINT_sqrt_med_fb
1335   \the\numexpr (#30#1-5#1)/(\xint_c_ii*#2).#1.#2.#3.%
1336 }%
1337 \def\XINT_sqrt_med_fb #1.#2.#3.#4.#5.%
1338 {%
1339   \expandafter\XINT_sqrt_small_ea
1340   \the\numexpr (#40#2-\xint_c_ii*#3*#1)*10#2+(#1*#1-#5)\expandafter.%
1341   \the\numexpr #30#2-#1.%
1342 }%
1343 \def\XINT_sqrt_big_f #1;#2#3#4#5#6#7#8#9%
1344 {%
1345   \XINT_sqrt_big_fa #1;{#2#3#4#5#6#7#8#9}%
1346 }%
1347 \def\XINT_sqrt_big_fa #1.#2.#3;#4%
1348 {%
1349   \expandafter\XINT_sqrt_big_ga
1350   \the\numexpr #3-\xint_c_viii\expandafter.%
1351   \romannumeral0\XINT_sqrt_med_fa 000.#1.#2.;#4.%
1352 }%
1353 %
1354 \def\XINT_sqrt_big_ga #1.#2#3%
1355 {%
1356   \ifnum #1>\xint_c_viii
1357     \expandafter\XINT_sqrt_big_gb\else
1358     \expandafter\XINT_sqrt_big_ka
1359   \fi #1.#3.#2.%
1360 }%
1361 \def\XINT_sqrt_big_gb #1.#2.#3.%
1362 {%

```

#### 4 Package *xint* implementation

```

1363 \expandafter\XINT_sqrt_big_gc
1364 \the\numexpr (\xint_c_ii*#2-\xint_c_i)*\xint_c_x^viii/(\xint_c_iv*#3).%
1365 #3.#2.#1;%
1366 }%
1367 \def\XINT_sqrt_big_gc #1.#2.#3.%
1368 {%
1369 \expandafter\XINT_sqrt_big_gd
1370 \romannumeral0\xintiadd
1371 {\xintiiSub {#3000000000}\xintDouble{\xintiiMul{#2}{#1}}00000000}%
1372 {\xintiiSqr {#1}}.%
1373 \romannumeral0\xintiisub{#2000000000}{#1}.%
1374 }%
1375 \def\XINT_sqrt_big_gd #1.#2.%
1376 {%
1377 \expandafter\XINT_sqrt_big_ge #2.#1.%
1378 }%
1379 \def\XINT_sqrt_big_ge #1;#2#3#4#5#6#7#8#9%
1380 {\XINT_sqrt_big_gf #1.#2#3#4#5#6#7#8#9;}%
1381 \def\XINT_sqrt_big_gf #1;#2#3#4#5#6#7#8#9%
1382 {\XINT_sqrt_big_gg #1#2#3#4#5#6#7#8#9.}%
1383 \def\XINT_sqrt_big_gg #1.#2.#3.#4.%
1384 {%
1385 \expandafter\XINT_sqrt_big_gloop
1386 \expandafter\xint_c_xvi\expandafter.%
1387 \the\numexpr #3-\xint_c_viii\expandafter.%
1388 \romannumeral0\xintiisub {#2}{\xintiNum{#4}}.#1.%
1389 }%
1390 \def\XINT_sqrt_big_gloop #1.#2.%
1391 {%
1392 \unless\ifnum #1<#2 \xint_dothis\XINT_sqrt_big_ka \fi
1393 \xint_orthat{\XINT_sqrt_big_gi #1.}#2.%
1394 }%
1395 \def\XINT_sqrt_big_gi #1.%
1396 {%
1397 \expandafter\XINT_sqrt_big_gj
1398 \romannumeral0\XINT_dsx_addzerosnofuss{#1}{}.#1.%
1399 }%
1400 \def\XINT_sqrt_big_gj #1.#2.#3.#4.#5.%
1401 {%
1402 \expandafter\XINT_sqrt_big_gk
1403 \romannumeral0\xintiidivision {#4#1}{\XINT_dbl_pos #5\Z}.%
1404 #1.#5.#2.#3.%
1405 }%
1406 \def\XINT_sqrt_big_gk #1#2.#3.#4.%
1407 {%
1408 \expandafter\XINT_sqrt_big_gl
1409 \romannumeral0\xintiadd {#2#3}{\xintiiSqr{#1}}.%
1410 \romannumeral0\xintiisub {#4#3}{#1}.%
1411 }%
1412 \def\XINT_sqrt_big_gl #1.#2.%
1413 {%
1414 \expandafter\XINT_sqrt_big_gm #2.#1.%

```

#### 4 Package *xint* implementation

```

1415 }%
1416 \def\XINT_sqrt_big_gm #1.#2.#3.#4.#5.%
1417 {%
1418   \expandafter\XINT_sqrt_big_gn
1419   \romannumeral0\expandafter\XINT_split_fromleft_loop
1420   \the\numexpr\xint_c_ii*#3.{\#5\W\W\W\W\W\W\W\W\Z
1421   #1.#2.#3.#4.%
1422 }%
1423 \def\XINT_sqrt_big_gn #1.#2.#3.#4.#5.#6.%
1424 {%
1425   \expandafter\XINT_sqrt_big_gloop
1426   \the\numexpr \xint_c_ii*#5\expandafter.%
1427   \the\numexpr #6-#5\expandafter.%
1428   \romannumeral0\xintiisub{#4}{\xintiNum{#1}}.#3.#2.%
1429 }%
1430 \def\XINT_sqrt_big_ka #1.#2.#3.#4.%
1431 {%
1432   \expandafter\XINT_sqrt_big_kb
1433   \romannumeral0\XINT_dsx_addzerosnofuss {#1}{#3}.%
1434   \romannumeral0\xintiisub
1435   {\XINT_dsx_addzerosnofuss {\numexpr\xint_c_ii*#1}{#2}}%
1436   {\xintiNum{#4}}.%
1437 }%
1438 \def\XINT_sqrt_big_kb #1.#2.%
1439 {%
1440   \expandafter\XINT_sqrt_big_kc #2.#1.%
1441 }%
1442 \def\XINT_sqrt_big_kc #1%
1443 {%
1444   \if0#1\xint_dothis\XINT_sqrt_big_kz\fi
1445   \xint_orthat\XINT_sqrt_big_kloop #1%
1446 }%
1447 \def\XINT_sqrt_big_kz 0.#1.%
1448 {%
1449   \expandafter\XINT_sqrt_big_kend
1450   \romannumeral0\xintinc{\XINT_dbl_pos #1\Z}.#1.%
1451 }%
1452 \def\XINT_sqrt_big_kend #1.#2.%
1453 {%
1454   \expandafter{\romannumeral0\xintinc{#2}}{#1}%
1455 }%
1456 \def\XINT_sqrt_big_kloop #1.#2.%
1457 {%
1458   \expandafter\XINT_sqrt_big_ke
1459   \romannumeral0\xintiidivision{#1}{\romannumeral0\XINT_dbl_pos #2\Z}{#2}%
1460 }%
1461 \def\XINT_sqrt_big_ke #1%
1462 {%
1463   \if0\XINT_Sgn #1\Z
1464     \expandafter \XINT_sqrt_big_end
1465   \else \expandafter \XINT_sqrt_big_kf
1466   \fi {#1}%

```

#### 4 Package `xint` implementation

```

1467 }%
1468 \def\XINT_sqrt_big_kf #1#2#3%
1469 {%
1470   \expandafter\XINT_sqrt_big_kg
1471   \romannumeral0\xintiisub {#3}{#1}.%
1472   \romannumeral0\xintiiaadd {#2}{\xintiiSqr {#1}}.%
1473 }%
1474 \def\XINT_sqrt_big_kg #1.#2.%
1475 {%
1476   \expandafter\XINT_sqrt_big_kloop #2.#1.%
1477 }%
1478 \def\XINT_sqrt_big_end #1#2#3{{#3}{#2}}%

```

#### 4.43 `\xintiiBinomial`, `\xintiBinomial`

2015/11/28-29 for 1.2f.

```

1479 \def\xintiiBinomial {\romannumeral0\xintiibinomial }%
1480 \def\xintiibinomial #1#2%
1481 {%
1482   \expandafter\XINT_binom_pre\the\numexpr #1\expandafter.\the\numexpr #2.%
1483 }%
1484 \def\XINT_binom_pre #1.#2.%
1485 {%
1486   \expandafter\XINT_binom_fork \the\numexpr#1-#2.#2.#1.%
1487 }%
1488 \def\xintiBinomial{\romannumeral0\xintibinomial}%
1489 \let\xintibinomial\xintiibinomial

```

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

```

1490 \def\XINT_binom_fork #1#2.#3#4.#5#6.%
1491 {%
1492   \if-#1\xint_dothis {\xintError:OutOfRangeBinomial\space 0}\fi
1493   \if-#3\xint_dothis {\xintError:OutOfRangeBinomial\space 0}\fi
1494   \if0#1\xint_dothis{ 1}\fi
1495   \if0#3\xint_dothis{ 1}\fi
1496   \ifnum #5#6>\xint_c_x^viii_mone\xint_dothis{\xintError:OutOfRangeBinomial\space 0}\fi
1497   \ifnum #1#2>#3#4 \xint_dothis{\XINT_binom_a #1#2.#3#4.}\fi
1498   \xint_orthat{\XINT_binom_a #3#4.#1#2.}%
1499 }%

```

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

Le `1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W` est le terminateur pour le `\XINT_unsep_cuzsmall` final.

```

1500 \def\XINT_binom_a #1.#2.%
1501 {%
1502   \expandafter\XINT_binom_b\the\numexpr \xint_c_i+#1.1.#2.100000001!1!\Z!0!%
1503 }%

```

#### 4 Package *xint* implementation

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

```
1504 \def\xINT_binom_b #1.%
1505 {%
1506   \ifnum #1>9999 \xint_dothis\xINT_binom_vbigloop \fi
1507   \ifnum #1>463 \xint_dothis\xINT_binom_bigloop \fi
1508   \ifnum #1>98 \xint_dothis\xINT_binom_medloop \fi
1509   \ifnum #1>29 \xint_dothis\xINT_binom_smallloop \fi
1510   \xint_orthat\xINT_binom_vsmallloop #1.%
1511 }%
```

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

```
1512 \def\xINT_binom_smallloop #1.#2.#3.%
1513 {%
1514   \ifcase\numexpr #3-#2\relax
1515     \expandafter\xINT_binom_end_
1516   \or \expandafter\xINT_binom_end_i
1517   \or \expandafter\xINT_binom_end_ii
1518   \or \expandafter\xINT_binom_end_iii
1519   \else\expandafter\xINT_binom_smallloop_a
1520   \fi #1.#2.#3.%
1521 }%
```

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

```
1522 \def\xINT_binom_smallloop_a #1.#2.#3.%
1523 {%
1524   \expandafter\xINT_binom_smallloop_b
1525   \the\numexpr #1+\xint_c_iv\expandafter.%
1526   \the\numexpr #2+\xint_c_iv\expandafter.%
1527   \the\numexpr #3\expandafter.%
1528   \the\numexpr\expandafter\xINT_binom_div
1529   \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1530   !\romannumeral0\expandafter\xINT_binom_mul
1531   \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1532 }%
1533 \def\xINT_binom_smallloop_b #1.%
1534 {%
1535   \ifnum #1>98 \expandafter\xINT_binom_medloop \else
1536     \expandafter\xINT_binom_smallloop \fi #1.%
1537 }%
```

Ici on prend trois par trois.

```
1538 \def\xINT_binom_medloop #1.#2.#3.%
```

#### 4 Package *xint* implementation

```
1539 {%
1540   \ifcase\numexpr #3-#2\relax
1541     \expandafter\XINT_binom_end_
1542   \or \expandafter\XINT_binom_end_i
1543   \or \expandafter\XINT_binom_end_ii
1544   \else\expandafter\XINT_binom_medloop_a
1545   \fi #1.#2.#3.%
1546}%
1547\def\XINT_binom_medloop_a #1.#2.#3.%
1548{%
1549   \expandafter\XINT_binom_medloop_b
1550   \the\numexpr #1+\xint_c_iii\expandafter.%
1551   \the\numexpr #2+\xint_c_iii\expandafter.%
1552   \the\numexpr #3\expandafter.%
1553   \the\numexpr\expandafter\XINT_binom_div
1554     \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
1555   !\romannumeral0\expandafter\XINT_binom_mul
1556     \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1557}%
1558\def\XINT_binom_medloop_b #1.%
1559{%
1560   \ifnum #1>463 \expandafter\XINT_binom_bigloop   \else
1561               \expandafter\XINT_binom_medloop   \fi #1.%
1562}%

```

Ici on prend deux par deux.

```
1563\def\XINT_binom_bigloop #1.#2.#3.%
1564{%
1565   \ifcase\numexpr #3-#2\relax
1566     \expandafter\XINT_binom_end_
1567   \or \expandafter\XINT_binom_end_i
1568   \else\expandafter\XINT_binom_bigloop_a
1569   \fi #1.#2.#3.%
1570}%
1571\def\XINT_binom_bigloop_a #1.#2.#3.%
1572{%
1573   \expandafter\XINT_binom_bigloop_b
1574   \the\numexpr #1+\xint_c_ii\expandafter.%
1575   \the\numexpr #2+\xint_c_ii\expandafter.%
1576   \the\numexpr #3\expandafter.%
1577   \the\numexpr\expandafter\XINT_binom_div
1578     \the\numexpr #2*(#2+\xint_c_i)\expandafter
1579   !\romannumeral0\expandafter\XINT_binom_mul
1580     \the\numexpr #1*(#1+\xint_c_i)!%
1581}%
1582\def\XINT_binom_bigloop_b #1.%
1583{%
1584   \ifnum #1>9999 \expandafter\XINT_binom_vbigloop \else
1585               \expandafter\XINT_binom_bigloop   \fi #1.%
1586}%

```

Et finalement un par un.

#### 4 Package *xint* implementation

```

1587 \def\XINT_binom_vbigloop #1.#2.#3.%
1588 {%
1589     \ifnum #3=#2
1590         \expandafter\XINT_binom_end_
1591     \else\expandafter\XINT_binom_vbigloop_a
1592     \fi #1.#2.#3.%
1593 }%
1594 \def\XINT_binom_vbigloop_a #1.#2.#3.%
1595 {%
1596     \expandafter\XINT_binom_vbigloop
1597     \the\numexpr #1+\xint_c_i\expandafter.%
1598     \the\numexpr #2+\xint_c_i\expandafter.%
1599     \the\numexpr #3\expandafter.%
1600     \the\numexpr\expandafter\XINT_binom_div\the\numexpr #2\expandafter
1601     !\romannumeral0\XINT_binom_mul #1!%
1602 }%

```

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

```

1603 \def\XINT_binom_vsmallloop #1.#2.#3.%
1604 {%
1605     \ifcase\numexpr #3-#2\relax
1606         \expandafter\XINT_binom_vsmallend_
1607     \or \expandafter\XINT_binom_vsmallend_i
1608     \or \expandafter\XINT_binom_vsmallend_ii
1609     \or \expandafter\XINT_binom_vsmallend_iii
1610     \else\expandafter\XINT_binom_vsmallloop_a
1611     \fi #1.#2.#3.%
1612 }%
1613 \def\XINT_binom_vsmallloop_a #1.%
1614 {%
1615     \ifnum #1>26 \expandafter\XINT_binom_smallloop_a \else
1616         \expandafter\XINT_binom_vsmallloop_b \fi #1.%
1617 }%
1618 \def\XINT_binom_vsmallloop_b #1.#2.#3.%
1619 {%
1620     \expandafter\XINT_binom_vsmallloop
1621     \the\numexpr #1+\xint_c_iv\expandafter.%
1622     \the\numexpr #2+\xint_c_iv\expandafter.%
1623     \the\numexpr #3\expandafter.%
1624     \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1625     \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1626     !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1627 }%
1628 \def\XINT_binom_mul #1!#21!\Z!0!%
1629 {%
1630     \expandafter\XINT_rev_nounsep\expandafter{\expandafter}%
1631     \the\numexpr\expandafter\XINT_smallmul
1632     \the\numexpr\xint_c_x^viii+#1\expandafter

```

#### 4 Package *xint* implementation

```

1633    !\romannumeral0\XINT_rev_nounsep {}1\Z!#2%
1634    \R!\R!\R!\R!\R!\R!\R!\R!\W
1635    \R!\R!\R!\R!\R!\R!\R!\R!\W
1636    1\Z!%
1637 }%
1638 \def\XINT_binom_div #1!1\Z!%
1639 {%
1640    \expandafter\XINT_smallldivx_a
1641    \the\numexpr #1/\xint_c_ii\expandafter.%
1642    \the\numexpr \xint_c_x^viii+#1!%
1643 }%

```

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

```

1644 \def\XINT_binom_vsmallmuldiv #1!#2!1#3!\{\xint_c_x^viii+#2*#3/#1!}%

```

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

```

1645 \def\XINT_binom_end_iii #1.#2.#3.%
1646 {%
1647    \expandafter\XINT_binom_finish
1648    \the\numexpr\expandafter\XINT_binom_div
1649    \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1650    !\romannumeral0\expandafter\XINT_binom_mul
1651    \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1652 }%
1653 \def\XINT_binom_end_ii #1.#2.#3.%
1654 {%
1655    \expandafter\XINT_binom_finish
1656    \the\numexpr\expandafter\XINT_binom_div
1657    \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
1658    !\romannumeral0\expandafter\XINT_binom_mul
1659    \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1660 }%
1661 \def\XINT_binom_end_i #1.#2.#3.%
1662 {%
1663    \expandafter\XINT_binom_finish
1664    \the\numexpr\expandafter\XINT_binom_div
1665    \the\numexpr #2*(#2+\xint_c_i)\expandafter
1666    !\romannumeral0\expandafter\XINT_binom_mul
1667    \the\numexpr #1*(#1+\xint_c_i)!%
1668 }%
1669 \def\XINT_binom_end_ #1.#2.#3.%
1670 {%
1671    \expandafter\XINT_binom_finish
1672    \the\numexpr\expandafter\XINT_binom_div\the\numexpr #2\expandafter
1673    !\romannumeral0\XINT_binom_mul #1!%
1674 }%
1675 \def\XINT_binom_finish #1\Z!0!%
1676 {\XINT_unsep_cuzsmall #11\R!1\R!1\R!1\R!1\R!1\R!1\R!1\W}%

```

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



#### 4 Package *xint* implementation

```

1677 \def\XINT_binom_vsmallend_iii #1.%
1678 {%
1679     \ifnum #1>26 \expandafter\XINT_binom_end_iii \else
1680         \expandafter\XINT_binom_vsmallend_iiib \fi #1.%
1681 }%
1682 \def\XINT_binom_vsmallend_iiib #1.#2.#3.%
1683 {%
1684     \expandafter\XINT_binom_vsmallfinish
1685     \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1686     \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1687     !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1688 }%
1689 \def\XINT_binom_vsmallend_ii #1.%
1690 {%
1691     \ifnum #1>27 \expandafter\XINT_binom_end_ii \else
1692         \expandafter\XINT_binom_vsmallend_iib \fi #1.%
1693 }%
1694 \def\XINT_binom_vsmallend_iib #1.#2.#3.%
1695 {%
1696     \expandafter\XINT_binom_vsmallfinish
1697     \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1698     \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
1699     !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1700 }%
1701 \def\XINT_binom_vsmallend_i #1.%
1702 {%
1703     \ifnum #1>28 \expandafter\XINT_binom_end_i \else
1704         \expandafter\XINT_binom_vsmallend_ib \fi #1.%
1705 }%
1706 \def\XINT_binom_vsmallend_ib #1.#2.#3.%
1707 {%
1708     \expandafter\XINT_binom_vsmallfinish
1709     \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1710     \the\numexpr #2*(#2+\xint_c_i)\expandafter
1711     !\the\numexpr #1*(#1+\xint_c_i)!%
1712 }%
1713 \def\XINT_binom_vsmallend_ #1.%
1714 {%
1715     \ifnum #1>29 \expandafter\XINT_binom_end_ \else
1716         \expandafter\XINT_binom_vsmallend_b \fi #1.%
1717 }%
1718 \def\XINT_binom_vsmallend_b #1.#2.#3.%
1719 {%
1720     \expandafter\XINT_binom_vsmallfinish
1721     \the\numexpr\XINT_binom_vsmallmuldiv #2!#1!%
1722 }%
1723 \edef\XINT_binom_vsmallfinish 1#1!1!\Z!0!%
1724 {\noexpand\expandafter\space\noexpand\the\numexpr #1\relax}%

```

4.44 `\xintiiPFactorial`, `\xintiPFactorial`

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

```

1725 \def\xintiiPFactorial {\romannumeral0\xintiipfactorial }%
1726 \def\xintiipfactorial #1#2%
1727 {%
1728   \expandafter\XINT_pfac_fork\the\numexpr#1\expandafter.\the\numexpr #2.%
1729 }%
1730 \def\xintiPFactorial{\romannumeral0\xintipfactorial}%
1731 \let\xintipfactorial\xintiipfactorial

Code is a simplified version of the one for \xintiiBinomial, with no attempt is made at doing a
"very small" portion if applicable.

1732 \def\XINT_pfac_fork #1#2.#3.%
1733 {%
1734   \if-#1\xint_dothis {\xintError:OutOfRangePFac\space 0}\fi
1735   \ifnum #1#2=#3 \xint_dothis{ 1}\fi
1736   \ifnum #1#2>#3 \xint_dothis{\xintError:OutOfRangePFac\space 0}\fi
1737   \ifnum #3>\xint_c_x^viii_mone\xint_dothis{\xintError:OutOfRangePFac\space 0}\fi
1738   \xint_orthat {\XINT_pfac_a #1#2.#3.}%
1739 }%
1740 \def\XINT_pfac_a #1.#2.%
1741 {%
1742   \expandafter\XINT_pfac_b\the\numexpr \xint_c_i+#1.#2.100000001!1Z!%
1743   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\W
1744 }%
1745 \def\XINT_pfac_b #1.%
1746 {%
1747   \ifnum #1>9999 \xint_dothis\XINT_pfac_vbigloop \fi
1748   \ifnum #1>463 \xint_dothis\XINT_pfac_bigloop \fi
1749   \ifnum #1>98 \xint_dothis\XINT_pfac_medloop \fi
1750   \xint_orthat\XINT_pfac_smallloop #1.%
1751 }%
1752 \def\XINT_pfac_smallloop #1.#2.%
1753 {%
1754   \ifcase\numexpr #2-#1\relax
1755     \expandafter\XINT_pfac_end_
1756   \or \expandafter\XINT_pfac_end_i
1757   \or \expandafter\XINT_pfac_end_ii
1758   \or \expandafter\XINT_pfac_end_iii
1759   \else\expandafter\XINT_pfac_smallloop_a
1760   \fi #1.#2.%
1761 }%
1762 \def\XINT_pfac_smallloop_a #1.#2.%
1763 {%
1764   \expandafter\XINT_pfac_smallloop_b
1765   \the\numexpr #1+\xint_c_iv\expandafter.%
1766   \the\numexpr #2\expandafter.%
1767   \the\numexpr\expandafter\XINT_smallmul
1768   \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1769 }%
```

#### 4 Package *xint* implementation

```

1770 \def\XINT_pfac_smallloop_b #1.%
1771 {%
1772     \ifnum #1>98 \expandafter\XINT_pfac_medloop \else
1773         \expandafter\XINT_pfac_smallloop \fi #1.%
1774 }%
1775 \def\XINT_pfac_medloop #1.#2.%
1776 {%
1777     \ifcase\numexpr #2-#1\relax
1778         \expandafter\XINT_pfac_end_
1779     \or \expandafter\XINT_pfac_end_i
1780     \or \expandafter\XINT_pfac_end_ii
1781     \else\expandafter\XINT_pfac_medloop_a
1782     \fi #1.#2.%
1783 }%
1784 \def\XINT_pfac_medloop_a #1.#2.%
1785 {%
1786     \expandafter\XINT_pfac_medloop_b
1787     \the\numexpr #1+\xint_c_iii\expandafter.%
1788     \the\numexpr #2\expandafter.%
1789     \the\numexpr\expandafter\XINT_smallmul
1790     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1791 }%
1792 \def\XINT_pfac_medloop_b #1.%
1793 {%
1794     \ifnum #1>463 \expandafter\XINT_pfac_bigloop \else
1795         \expandafter\XINT_pfac_medloop \fi #1.%
1796 }%
1797 \def\XINT_pfac_bigloop #1.#2.%
1798 {%
1799     \ifcase\numexpr #2-#1\relax
1800         \expandafter\XINT_pfac_end_
1801     \or \expandafter\XINT_pfac_end_i
1802     \else\expandafter\XINT_pfac_bigloop_a
1803     \fi #1.#2.%
1804 }%
1805 \def\XINT_pfac_bigloop_a #1.#2.%
1806 {%
1807     \expandafter\XINT_pfac_bigloop_b
1808     \the\numexpr #1+\xint_c_ii\expandafter.%
1809     \the\numexpr #2\expandafter.%
1810     \the\numexpr\expandafter
1811     \XINT_smallmul\the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
1812 }%
1813 \def\XINT_pfac_bigloop_b #1.%
1814 {%
1815     \ifnum #1>9999 \expandafter\XINT_pfac_vbigloop \else
1816         \expandafter\XINT_pfac_bigloop \fi #1.%
1817 }%
1818 \def\XINT_pfac_vbigloop #1.#2.%
1819 {%
1820     \ifnum #2=#1
1821         \expandafter\XINT_pfac_end_

```

```

1822 \else\expandafter\XINT_pfac_vbigloop_a
1823 \fi #1.#2.%
1824 }%
1825 \def\XINT_pfac_vbigloop_a #1.#2.%
1826 {%
1827 \expandafter\XINT_pfac_vbigloop
1828 \the\numexpr #1+\xint_c_i\expandafter.%
1829 \the\numexpr #2\expandafter.%
1830 \the\numexpr\expandafter\XINT_smallmul\the\numexpr\xint_c_x^viii+#1!%
1831 }%
1832 \def\XINT_pfac_end_iii #1.#2.%
1833 {%
1834 \expandafter\XINT_mul_out
1835 \the\numexpr\expandafter\XINT_smallmul
1836 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1837 }%
1838 \def\XINT_pfac_end_ii #1.#2.%
1839 {%
1840 \expandafter\XINT_mul_out
1841 \the\numexpr\expandafter\XINT_smallmul
1842 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1843 }%
1844 \def\XINT_pfac_end_i #1.#2.%
1845 {%
1846 \expandafter\XINT_mul_out
1847 \the\numexpr\expandafter\XINT_smallmul
1848 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
1849 }%
1850 \def\XINT_pfac_end_ #1.#2.%
1851 {%
1852 \expandafter\XINT_mul_out
1853 \the\numexpr\expandafter\XINT_smallmul\the\numexpr \xint_c_x^viii+#1!%
1854 }%

```

#### 4.45 *\xintiiE*

Originally was used in *\xintiexpr*. Transferred from *xintfrac* for 1.1.

```

1855 \def\xintiiE {\romannumeral0\xintiiE}% used in \xintMod.
1856 \def\xintiiE #1#2%
1857 {\expandafter\XINT_iiE\the\numexpr #2\expandafter.\expandafter{\romannumeral`&&@#1}}%
1858 \def\XINT_iiE #1.#2{\ifnum#1>\xint_c_ \xint_dothis{\xint_dsh {#2}{-#1}}\fi
1859 \xint_orthat{ #2}}%

```

#### 4.46 “Load *xintfrac*” macros

Originally was used in *\xintiexpr*. Transferred from *xintfrac* for 1.1.

```

1860 \catcode`! 11
1861 \def\xintMax {\Did_you_mean_iiMax?or_load_xintfrac!}%
1862 \def\xintMin {\Did_you_mean_iiMin?or_load_xintfrac!}%
1863 \def\xintMaxof {\Did_you_mean_iMaxof?or_load_xintfrac!}%
1864 \def\xintMinof {\Did_you_mean_iMinof?or_load_xintfrac!}%

```

#### 4 Package *xint* implementation

```
1865 \def\xintSum {\Did_you_mean_iiSum?or_load_xintfrac!}%  
1866 \def\xintPrd {\Did_you_mean_iiPrd?or_load_xintfrac!}%  
1867 \def\xintPrdExpr {\Did_you_mean_iiPrdExpr?or_load_xintfrac!}%  
1868 \def\xintSumExpr {\Did_you_mean_iiSumExpr?or_load_xintfrac!}%  
1869 \XINT_restorecatcodes_endinput%
```

## 5 Package *xintbinhex* implementation

.1	Catcodes, $\varepsilon$ -TeX and reload detection . .	134	.6	<code>\xintHexToDec</code> . . . . .	140
.2	Package identification . . . . .	135	.7	<code>\xintBinToDec</code> . . . . .	142
.3	Constants, etc... . . . . .	135	.8	<code>\xintBinToHex</code> . . . . .	144
.4	<code>\XINT_OQ</code> . . . . .	137	.9	<code>\xintHexToBin</code> . . . . .	145
.5	<code>\xintDecToHex</code> , <code>\xintDecToBin</code> . . . .	137	.10	<code>\xintCHexToBin</code> . . . . .	146

The commenting is currently (2016/03/12) very sparse.

### 5.1 Catcodes, $\varepsilon$ -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 [2016/03/12 1.2f Expandable binary and hexadecimal conversions (JFB)]%

```

## 5.3 Constants, etc...

### 1.08

```

47 \newcount\xint_c_ii^xv \xint_c_ii^xv 32768
48 \newcount\xint_c_ii^xvi \xint_c_ii^xvi 65536
49 \newcount\xint_c_x^v \xint_c_x^v 100000
50 \def\xINT_tmpa #1{\ifx\relax#1\else
51   \expandafter\edef\csname XINT_sdth_#1\endcsname
52   {\ifcase #1 0\or 1\or 2\or 3\or 4\or 5\or 6\or 7\or
53     8\or 9\or A\or B\or C\or D\or E\or F\fi}%
54   \expandafter\xINT_tmpa\fi }%
55 \XINT_tmpa {0}{1}{2}{3}{4}{5}{6}{7}{8}{9}{10}{11}{12}{13}{14}{15}\relax
56 \def\xINT_tmpa #1{\ifx\relax#1\else
57   \expandafter\edef\csname XINT_sdtb_#1\endcsname
58   {\ifcase #1
59     0000\or 0001\or 0010\or 0011\or 0100\or 0101\or 0110\or 0111\or
60     1000\or 1001\or 1010\or 1011\or 1100\or 1101\or 1110\or 1111\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 \let\xINT_tmpa\relax
64 \expandafter\def\csname XINT_sbtd_0000\endcsname {0}%
65 \expandafter\def\csname XINT_sbtd_0001\endcsname {1}%
66 \expandafter\def\csname XINT_sbtd_0010\endcsname {2}%
67 \expandafter\def\csname XINT_sbtd_0011\endcsname {3}%
68 \expandafter\def\csname XINT_sbtd_0100\endcsname {4}%
69 \expandafter\def\csname XINT_sbtd_0101\endcsname {5}%
70 \expandafter\def\csname XINT_sbtd_0110\endcsname {6}%
71 \expandafter\def\csname XINT_sbtd_0111\endcsname {7}%
72 \expandafter\def\csname XINT_sbtd_1000\endcsname {8}%
73 \expandafter\def\csname XINT_sbtd_1001\endcsname {9}%
74 \expandafter\def\csname XINT_sbtd_1010\endcsname {10}%
75 \expandafter\def\csname XINT_sbtd_1011\endcsname {11}%
76 \expandafter\def\csname XINT_sbtd_1100\endcsname {12}%
77 \expandafter\def\csname XINT_sbtd_1101\endcsname {13}%
78 \expandafter\def\csname XINT_sbtd_1110\endcsname {14}%
79 \expandafter\def\csname XINT_sbtd_1111\endcsname {15}%
80 \expandafter\let\csname XINT_sbth_0000\expandafter\endcsname
81   \csname XINT_sbtd_0000\endcsname
82 \expandafter\let\csname XINT_sbth_0001\expandafter\endcsname
83   \csname XINT_sbtd_0001\endcsname
84 \expandafter\let\csname XINT_sbth_0010\expandafter\endcsname

```

## 5 Package *xintbinhex* implementation

```

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

```



## 5 Package *xintbinhex* implementation

```
137 }%
138 \def\XINT_smallbin_a #1#2%
139 {%
140     \csname XINT_sdtb_#1\expandafter\expandafter\expandafter\endcsname
141     \csname XINT_sdtb_\the\numexpr #2-\xint_c_xvi*#1\endcsname
142 }%
```

### 5.4 \XINT\_OQ

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

```
143 \def\XINT_OQ #1#2#3#4#5#6#7#8#9%
144 {%
145     \xint_gob_til_R #9\XINT_OQ_end_a\R\XINT_OQ {#9#8#7#6#5#4#3#2#1}%
146 }%
147 \def\XINT_OQ_end_a\R\XINT_OQ #1#2\Z
148 {%
149     \XINT_OQ_end_b #1\Z
150 }%
151 \def\XINT_OQ_end_b #1#2#3#4#5#6#7#8%
152 {%
153     \xint_gob_til_R
154         #8\XINT_OQ_end_viii
155         #7\XINT_OQ_end_vii
156         #6\XINT_OQ_end_vi
157         #5\XINT_OQ_end_v
158         #4\XINT_OQ_end_iv
159         #3\XINT_OQ_end_iii
160         #2\XINT_OQ_end_ii
161         \R\XINT_OQ_end_i
162         \Z #2#3#4#5#6#7#8%
163 }%
164 \def\XINT_OQ_end_viii #1\Z #2#3#4#5#6#7#8#9\Z { #9}%
165 \def\XINT_OQ_end_vii #1\Z #2#3#4#5#6#7#8#9\Z { #8#90000000}%
166 \def\XINT_OQ_end_vi #1\Z #2#3#4#5#6#7#8#9\Z { #7#8#9000000}%
167 \def\XINT_OQ_end_v #1\Z #2#3#4#5#6#7#8#9\Z { #6#7#8#900000}%
168 \def\XINT_OQ_end_iv #1\Z #2#3#4#5#6#7#8#9\Z { #5#6#7#8#90000}%
169 \def\XINT_OQ_end_iii #1\Z #2#3#4#5#6#7#8#9\Z { #4#5#6#7#8#9000}%
170 \def\XINT_OQ_end_ii #1\Z #2#3#4#5#6#7#8#9\Z { #3#4#5#6#7#8#900}%
171 \def\XINT_OQ_end_i #1\Z #2#3#4#5#6#7#8\Z { #1#2#3#4#5#6#7#80}%

```

### 5.5 \xintDecToHex, \xintDecToBin

#### 1.08

```
172 \def\xintDecToHex {\romannumeral0\xintdectohex }%
173 \def\xintdectohex #1%
174     {\expandafter\XINT_dth_checkin\romannumeral`&&@#1\W\W\W\W \T}%
175 \def\XINT_dth_checkin #1%
176 {%
177     \xint_UDsignfork
178     #1\XINT_dth_N

```

## 5 Package *xintbinhex* implementation

```

179     -{\XINT_dth_P #1}%
180     \krof
181 }%
182 \def\XINT_dth_N {\expandafter\xint_minus_thenstop\romannumeral0\XINT_dth_P }%
183 \def\XINT_dth_P {\expandafter\XINT_dth_III\romannumeral`&&\XINT_dtbh_I {0.}}%
184 \def\xintDecToBin {\romannumeral0\xintdectobin }%
185 \def\xintdectobin #1%
186     {\expandafter\XINT_dtb_checkin\romannumeral`&&@#1\W\W\W\W \T }%
187 \def\XINT_dtb_checkin #1%
188 {%
189     \xint_UDsignfork
190     #1\XINT_dtb_N
191     -{\XINT_dtb_P #1}%
192     \krof
193 }%
194 \def\XINT_dtb_N {\expandafter\xint_minus_thenstop\romannumeral0\XINT_dtb_P }%
195 \def\XINT_dtb_P {\expandafter\XINT_dtb_III\romannumeral`&&\XINT_dtbh_I {0.}}%
196 \def\XINT_dtbh_I #1#2#3#4#5%
197 {%
198     \xint_gob_til_W #5\XINT_dtbh_II_a\W\XINT_dtbh_I_a  {{{#2#3#4#5}#1\Z.%
199 }%
200 \def\XINT_dtbh_II_a\W\XINT_dtbh_I_a #1#2{\XINT_dtbh_II_b #2}%
201 \def\XINT_dtbh_II_b #1#2#3#4%
202 {%
203     \xint_gob_til_W
204     #1\XINT_dtbh_II_c
205     #2\XINT_dtbh_II_ci
206     #3\XINT_dtbh_II_cii
207     \W\XINT_dtbh_II_ciii #1#2#3#4%
208 }%
209 \def\XINT_dtbh_II_c \W\XINT_dtbh_II_ci
210     \W\XINT_dtbh_II_cii
211     \W\XINT_dtbh_II_ciii \W\W\W\W {}{}%
212 \def\XINT_dtbh_II_ci #1\XINT_dtbh_II_ciii #2\W\W\W
213     {\XINT_dtbh_II_d {}{#2}{0}}%
214 \def\XINT_dtbh_II_cii\W\XINT_dtbh_II_ciii #1#2\W\W
215     {\XINT_dtbh_II_d {}{#1#2}{00}}%
216 \def\XINT_dtbh_II_ciii #1#2#3\W
217     {\XINT_dtbh_II_d {}{#1#2#3}{000}}%
218 \def\XINT_dtbh_I_a #1#2#3.%
219 {%
220     \xint_gob_til_Z #3\XINT_dtbh_I_z\Z
221     \expandafter\XINT_dtbh_I_b\the\numexpr #2+#30000.{#1}%
222 }%
223 \def\XINT_dtbh_I_b #1.%
224 {%
225     \expandafter\XINT_dtbh_I_c\the\numexpr
226     (#1+\xint_c_ii^xv)/\xint_c_ii^xvi-\xint_c_i.#1.%
227 }%
228 \def\XINT_dtbh_I_c #1.#2.%
229 {%
230     \expandafter\XINT_dtbh_I_d\expandafter

```

## 5 Package *xintbinhex* implementation

```

231     {\the\numexpr #2-\xint_c_ii^xvi*#1}{#1}%
232 }%
233 \def\xINT_dtbh_I_d #1#2#3{\xINT_dtbh_I_a {#3#1.}{#2}}%
234 \def\xINT_dtbh_I_z\Z\expandafter\xINT_dtbh_I_b\the\numexpr #1+#2.%
235 {%
236     \ifnum #1=\xint_c_ \expandafter\xINT_dtbh_I_end_zb\fi
237     \xINT_dtbh_I_end_za {#1}%
238 }%
239 \def\xINT_dtbh_I_end_za #1#2{\xINT_dtbh_I {#2#1.}}%
240 \def\xINT_dtbh_I_end_zb\xINT_dtbh_I_end_za #1#2{\xINT_dtbh_I {#2}}%
241 \def\xINT_dtbh_II_d #1#2#3#4.%
242 {%
243     \xint_gob_til_Z #4\xINT_dtbh_II_z\Z
244     \expandafter\xINT_dtbh_II_e\the\numexpr #2+#4#3.{#1}{#3}%
245 }%
246 \def\xINT_dtbh_II_e #1.%
247 {%
248     \expandafter\xINT_dtbh_II_f\the\numexpr
249     (#1+\xint_c_ii^xv)/\xint_c_ii^xvi-\xint_c_i.#1.%
250 }%
251 \def\xINT_dtbh_II_f #1.#2.%
252 {%
253     \expandafter\xINT_dtbh_II_g\expandafter
254     {\the\numexpr #2-\xint_c_ii^xvi*#1}{#1}%
255 }%
256 \def\xINT_dtbh_II_g #1#2#3{\xINT_dtbh_II_d {#3#1.}{#2}}%
257 \def\xINT_dtbh_II_z\Z\expandafter\xINT_dtbh_II_e\the\numexpr #1+#2.%
258 {%
259     \ifnum #1=\xint_c_ \expandafter\xINT_dtbh_II_end_zb\fi
260     \xINT_dtbh_II_end_za {#1}%
261 }%
262 \def\xINT_dtbh_II_end_za #1#2#3{{#2#1.\Z.}}%
263 \def\xINT_dtbh_II_end_zb\xINT_dtbh_II_end_za #1#2#3{{#2\Z.}}%
264 \def\xINT_dth_III #1#2.%
265 {%
266     \xint_gob_til_Z #2\xINT_dth_end\Z
267     \expandafter\xINT_dth_III\expandafter
268     {\romannumeral`&&\xINT_dth_small #2.#1}%
269 }%
270 \def\xINT_dth_small #1.%
271 {%
272     \expandafter\xINT_smallhex\expandafter
273     {\the\numexpr (#1+\xint_c_ii^vii)/\xint_c_ii^viii-\xint_c_i\expandafter}%
274     \romannumeral`&&\expandafter\xINT_smallhex\expandafter
275     {\the\numexpr
276     #1-((#1+\xint_c_ii^vii)/\xint_c_ii^viii-\xint_c_i)*\xint_c_ii^viii}%
277 }%
278 \def\xINT_dth_end\Z\expandafter\xINT_dth_III\expandafter #1#2\T
279 {%
280     \xINT_dth_end_b #1%
281 }%
282 \def\xINT_dth_end_b #1.{\xINT_dth_end_c }%

```

## 5 Package *xintbinhex* implementation

```
283 \def\XINT_dth_end_c #1{\xint_gob_til_zero #1\XINT_dth_end_d 0\space #1}%
284 \def\XINT_dth_end_d 0\space 0#1%
285 {%
286   \xint_gob_til_zero #1\XINT_dth_end_e 0\space #1%
287 }%
288 \def\XINT_dth_end_e 0\space 0#1%
289 {%
290   \xint_gob_til_zero #1\XINT_dth_end_f 0\space #1%
291 }%
292 \def\XINT_dth_end_f 0\space 0{ }%
293 \def\XINT_dtb_III #1#2.%
294 {%
295   \xint_gob_til_Z #2\XINT_dtb_end_Z
296   \expandafter\XINT_dtb_III\expandafter
297   {\romannumeral`&&\XINT_dtb_small #2.#1}%
298 }%
299 \def\XINT_dtb_small #1.%
300 {%
301   \expandafter\XINT_smallbin\expandafter
302   {\the\numexpr (#1+\xint_c_ii^vii)/\xint_c_ii^viii-\xint_c_i\expandafter}%
303   \romannumeral`&&\expandafter\XINT_smallbin\expandafter
304   {\the\numexpr
305     #1-((#1+\xint_c_ii^vii)/\xint_c_ii^viii-\xint_c_i)*\xint_c_ii^viii}%
306 }%
307 \def\XINT_dtb_end_Z\expandafter\XINT_dtb_III\expandafter #1#2\T
308 {%
309   \XINT_dtb_end_b #1%
310 }%
311 \def\XINT_dtb_end_b #1.{\XINT_dtb_end_c }%
312 \def\XINT_dtb_end_c #1#2#3#4#5#6#7#8%
313 {%
314   \expandafter\XINT_dtb_end_d\the\numexpr #1#2#3#4#5#6#7#8\relax
315 }%
316 \edef\XINT_dtb_end_d #1#2#3#4#5#6#7#8#9%
317 {%
318   \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7#8#9\relax
319 }%
```

### 5.6 *\xintHexToDec*

#### 1.08

```
320 \def\xintHexToDec {\romannumeral0\xinthextodec }%
321 \def\xinthextodec #1%
322   {\expandafter\XINT_htd_checkin\romannumeral`&&@#1\W\W\W\W \T }%
323 \def\XINT_htd_checkin #1%
324 {%
325   \xint_UDsignfork
326   #1\XINT_htd_neg
327   -{\XINT_htd_I {0000}#1}%
328   \krof
329 }%
330 \def\XINT_htd_neg {\expandafter\xint_minus_thenstop
```

## 5 Package *xintbinhex* implementation

```

331          \romannumeral0\XINT_htd_I {0000}}}%
332 \def\XINT_htd_I #1#2#3#4#5%
333 {%
334     \xint_gob_til_W #5\XINT_htd_II_a\W
335     \XINT_htd_I_a {}{"#2#3#4#5}#1\Z\Z\Z\Z
336 }%
337 \def\XINT_htd_II_a \W\XINT_htd_I_a #1#2{\XINT_htd_II_b #2}%
338 \def\XINT_htd_II_b "#1#2#3#4%
339 {%
340     \xint_gob_til_W
341     #1\XINT_htd_II_c
342     #2\XINT_htd_II_ci
343     #3\XINT_htd_II_cii
344     \W\XINT_htd_II_ciii #1#2#3#4%
345 }%
346 \def\XINT_htd_II_c \W\XINT_htd_II_ci
347     \W\XINT_htd_II_cii
348     \W\XINT_htd_II_ciii \W\W\W\W #1\Z\Z\Z\Z\T
349 {%
350     \expandafter\xint_cleanupzeros_andstop
351     \romannumeral0\XINT_rord_main {}#1%
352     \xint_relax
353     \xint_bye\xint_bye\xint_bye\xint_bye
354     \xint_bye\xint_bye\xint_bye\xint_bye
355     \xint_relax
356 }%
357 \def\XINT_htd_II_ci #1\XINT_htd_II_ciii
358     #2\W\W\W {\XINT_htd_II_d {}{"#2}{\xint_c_xvi}}}%
359 \def\XINT_htd_II_cii\W\XINT_htd_II_ciii
360     #1#2\W\W {\XINT_htd_II_d {}{"#1#2}{\xint_c_ii^viii}}}%
361 \def\XINT_htd_II_ciii #1#2#3\W {\XINT_htd_II_d {}{"#1#2#3}{\xint_c_ii^xii}}}%
362 \def\XINT_htd_I_a #1#2#3#4#5#6%
363 {%
364     \xint_gob_til_Z #3\XINT_htd_I_end_a\Z
365     \expandafter\XINT_htd_I_b\the\numexpr
366     #2+\xint_c_ii^xvi*#6#5#4#3+\xint_c_x^ix\relax {#1}%
367 }%
368 \def\XINT_htd_I_b 1#1#2#3#4#5#6#7#8#9{\XINT_htd_I_c {#1#2#3#4#5}{#9#8#7#6}}}%
369 \def\XINT_htd_I_c #1#2#3{\XINT_htd_I_a {#3#2}{#1}}}%
370 \def\XINT_htd_I_end_a\Z\expandafter\XINT_htd_I_b\the\numexpr #1+#2\relax
371 {%
372     \expandafter\XINT_htd_I_end_b\the\numexpr \xint_c_x^v+#1\relax
373 }%
374 \def\XINT_htd_I_end_b 1#1#2#3#4#5%
375 {%
376     \xint_gob_til_zero #1\XINT_htd_I_end_bz0%
377     \XINT_htd_I_end_c #1#2#3#4#5%
378 }%
379 \def\XINT_htd_I_end_c #1#2#3#4#5#6{\XINT_htd_I {#6#5#4#3#2#1000}}}%
380 \def\XINT_htd_I_end_bz0\XINT_htd_I_end_c 0#1#2#3#4%
381 {%
382     \xint_gob_til_zeros_iv #1#2#3#4\XINT_htd_I_end_bzz 0000%

```

```

383 \XINT_htd_I_end_D {#4#3#2#1}%
384 }%
385 \def\XINT_htd_I_end_D #1#2{\XINT_htd_I {#2#1}}%
386 \def\XINT_htd_I_end_bzz 0000\XINT_htd_I_end_D #1{\XINT_htd_I }%
387 \def\XINT_htd_II_d #1#2#3#4#5#6#7%
388 {%
389 \xint_gob_til_Z #4\XINT_htd_II_end_a\Z
390 \expandafter\XINT_htd_II_e\the\numexpr
391 #2+#3*#7#6#5#4+\xint_c_x^viii\relax {#1}{#3}%
392 }%
393 \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}}%
394 \def\XINT_htd_II_f #1#2#3{\XINT_htd_II_d {#2#3}{#1}}%
395 \def\XINT_htd_II_end_a\Z\expandafter\XINT_htd_II_e
396 \the\numexpr #1+#2\relax #3#4\T
397 {%
398 \XINT_htd_II_end_b #1#3%
399 }%
400 \edef\XINT_htd_II_end_b #1#2#3#4#5#6#7#8%
401 {%
402 \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7#8\relax
403 }%

```

## 5.7 \xintBinToDec

### 1.08

```

404 \def\xintBinToDec {\romannumeral0\xintbintodec }%
405 \def\xintbintodec #1{\expandafter\XINT_btd_checkin
406 \romannumeral`&&@#1\W\W\W\W\W\W\W\W \T }%
407 \def\XINT_btd_checkin #1%
408 {%
409 \xint_UDsignfork
410 #1\XINT_btd_neg
411 -{\XINT_btd_I {000000}{#1}}%
412 \krof
413 }%
414 \def\XINT_btd_neg {\expandafter\xint_minus_thenstop
415 \romannumeral0\XINT_btd_I {000000}}%
416 \def\XINT_btd_I #1#2#3#4#5#6#7#8#9%
417 {%
418 \xint_gob_til_W #9\XINT_btd_II_a {#2#3#4#5#6#7#8#9}\W
419 \XINT_btd_I_a {}{\csname XINT_sbtd_#2#3#4#5\endcsname*\xint_c_xvi+%
420 \csname XINT_sbtd_#6#7#8#9\endcsname}%
421 #1\Z\Z\Z\Z\Z\Z
422 }%
423 \def\XINT_btd_II_a #1\W\XINT_btd_I_a #2#3{\XINT_btd_II_b #1}%
424 \def\XINT_btd_II_b #1#2#3#4#5#6#7#8%
425 {%
426 \xint_gob_til_W
427 #1\XINT_btd_II_c
428 #2\XINT_btd_II_ci
429 #3\XINT_btd_II_cii
430 #4\XINT_btd_II_ciii

```

## 5 Package *xintbinhex* implementation

```

431      #5\XINT_btd_II_civ
432      #6\XINT_btd_II_cv
433      #7\XINT_btd_II_cvi
434      \W\XINT_btd_II_cvii #1#2#3#4#5#6#7#8%
435 }%
436 \def\XINT_btd_II_c #1\XINT_btd_II_cvii \W\W\W\W\W\W\W\W #2\Z\Z\Z\Z\Z\Z\T
437 {%
438     \expandafter\XINT_btd_II_c_end
439     \romannumeral0\XINT_rord_main {}#2%
440     \xint_relax
441     \xint_bye\xint_bye\xint_bye\xint_bye
442     \xint_bye\xint_bye\xint_bye\xint_bye
443     \xint_relax
444 }%
445 \edef\XINT_btd_II_c_end #1#2#3#4#5#6%
446 {%
447     \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6\relax
448 }%
449 \def\XINT_btd_II_ci #1\XINT_btd_II_cvii #2\W\W\W\W\W\W\W
450     {\XINT_btd_II_d }{{#2}{\xint_c_ii }}%
451 \def\XINT_btd_II_cii #1\XINT_btd_II_cvii #2\W\W\W\W\W\W
452     {\XINT_btd_II_d }{{\csname XINT_sbtd_00#2\endcsname }{\xint_c_iv }}%
453 \def\XINT_btd_II_ciii #1\XINT_btd_II_cvii #2\W\W\W\W\W\W
454     {\XINT_btd_II_d }{{\csname XINT_sbtd_0#2\endcsname }{\xint_c_viii }}%
455 \def\XINT_btd_II_civ #1\XINT_btd_II_cvii #2\W\W\W\W\W
456     {\XINT_btd_II_d }{{\csname XINT_sbtd_#2\endcsname }{\xint_c_xvi }}%
457 \def\XINT_btd_II_cv #1\XINT_btd_II_cvii #2#3#4#5#6\W\W\W
458 {%
459     \XINT_btd_II_d }{{\csname XINT_sbtd_#2#3#4#5\endcsname*\xint_c_ii+%
460         #6}{\xint_c_ii^v }}%
461 }%
462 \def\XINT_btd_II_cvi #1\XINT_btd_II_cvii #2#3#4#5#6#7\W\W
463 {%
464     \XINT_btd_II_d }{{\csname XINT_sbtd_#2#3#4#5\endcsname*\xint_c_iv+%
465         \csname XINT_sbtd_00#6#7\endcsname }{\xint_c_ii^vi }}%
466 }%
467 \def\XINT_btd_II_cvii #1#2#3#4#5#6#7\W
468 {%
469     \XINT_btd_II_d }{{\csname XINT_sbtd_#1#2#3#4\endcsname*\xint_c_viii+%
470         \csname XINT_sbtd_0#5#6#7\endcsname }{\xint_c_ii^vii }}%
471 }%
472 \def\XINT_btd_II_d #1#2#3#4#5#6#7#8#9%
473 {%
474     \xint_gob_til_Z #4\XINT_btd_II_end_a\Z
475     \expandafter\XINT_btd_II_e\the\numexpr
476     #2+(\xint_c_x^ix+#3*#9#8#7#6#5#4)\relax {#1}{#3}%
477 }%
478 \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}}%
479 \def\XINT_btd_II_f #1#2#3{\XINT_btd_II_d {#2#3}{#1}}%
480 \def\XINT_btd_II_end_a\Z\expandafter\XINT_btd_II_e
481     \the\numexpr #1+(\Z\relax #3#4\T
482 {%

```

```

483 \XINT_btd_II_end_b #1#3%
484 }%
485 \edef\XINT_btd_II_end_b #1#2#3#4#5#6#7#8#9%
486 {%
487 \noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5#6#7#8#9\relax
488 }%
489 \def\XINT_btd_I_a #1#2#3#4#5#6#7#8%
490 {%
491 \xint_gob_til_Z #3\XINT_btd_I_end_a\Z
492 \expandafter\XINT_btd_I_b\the\numexpr
493 #2+\xint_c_ii^viii*#8#7#6#5#4#3+\xint_c_x^ix\relax {#1}%
494 }%
495 \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}}%
496 \def\XINT_btd_I_c #1#2#3{\XINT_btd_I_a {#3#2}{#1}}%
497 \def\XINT_btd_I_end_a\Z\expandafter\XINT_btd_I_b
498 \the\numexpr #1+\xint_c_ii^viii #2\relax
499 {%
500 \expandafter\XINT_btd_I_end_b\the\numexpr 1000+#1\relax
501 }%
502 \def\XINT_btd_I_end_b 1#1#2#3%
503 {%
504 \xint_gob_til_zeros_iii #1#2#3\XINT_btd_I_end_bz 000%
505 \XINT_btd_I_end_c #1#2#3%
506 }%
507 \def\XINT_btd_I_end_c #1#2#3#4{\XINT_btd_I {#4#3#2#1000}}%
508 \def\XINT_btd_I_end_bz 000\XINT_btd_I_end_c 000{\XINT_btd_I }%

```

## 5.8 \xintBinToHex

### 1.08

```

509 \def\xintBinToHex {\romannumeral0\xintbinto hex}%
510 \def\xintbinto hex #1%
511 {%
512 \expandafter\XINT_bth_checkin
513 \romannumeral0\expandafter\XINT_num_loop
514 \romannumeral`&&@#1\xint_relax\xint_relax
515 \xint_relax\xint_relax
516 \xint_relax\xint_relax\xint_relax\xint_relax\Z
517 \R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W\W
518 }%
519 \def\XINT_bth_checkin #1%
520 {%
521 \xint_UDsignfork
522 #1\XINT_bth_N
523 -{\XINT_bth_P #1}%
524 \krof
525 }%
526 \def\XINT_bth_N {\expandafter\xint_minus_thenstop\romannumeral0\XINT_bth_P}%
527 \def\XINT_bth_P {\expandafter\XINT_bth_I\expandafter{\expandafter}%
528 \romannumeral0\XINT_OQ {}}%
529 \def\XINT_bth_I #1#2#3#4#5#6#7#8#9%
530 {%

```



```

531 \xint_gob_til_W #9\XINT_bth_end_a\W
532 \expandafter\expandafter\expandafter
533 \XINT_bth_I
534 \expandafter\expandafter\expandafter
535 {\csname XINT_sbth_#9#8#7#6\expandafter\expandafter\expandafter\endcsname
536 \csname XINT_sbth_#5#4#3#2\endcsname #1}%
537 }%
538 \def\XINT_bth_end_a\W \expandafter\expandafter\expandafter
539 \XINT_bth_I \expandafter\expandafter\expandafter #1%
540 {%
541 \XINT_bth_end_b #1%
542 }%
543 \def\XINT_bth_end_b #1\endcsname #2\endcsname #3%
544 {%
545 \xint_gob_til_zero #3\XINT_bth_end_z 0\space #3%
546 }%
547 \def\XINT_bth_end_z0\space 0{ }%

```

## 5.9 \xintHexToBin

### 1.08

```

548 \def\xintHexToBin {\romannumeral0\xinthextobin }%
549 \def\xinthextobin #1%
550 {%
551 \expandafter\XINT_htb_checkin\romannumeral`&&@#1GGGGGGGG\T
552 }%
553 \def\XINT_htb_checkin #1%
554 {%
555 \xint_UDsignfork
556 #1\XINT_htb_N
557 -{\XINT_htb_P #1}%
558 \krof
559 }%
560 \def\XINT_htb_N {\expandafter\xint_minus_thenstop\romannumeral0\XINT_htb_P }%
561 \def\XINT_htb_P {\XINT_htb_I_a {}}%
562 \def\XINT_htb_I_a #1#2#3#4#5#6#7#8#9%
563 {%
564 \xint_gob_til_G #9\XINT_htb_II_a G%
565 \expandafter\expandafter\expandafter
566 \XINT_htb_I_b
567 \expandafter\expandafter\expandafter
568 {\csname XINT_shtb_#2\expandafter\expandafter\expandafter\endcsname
569 \csname XINT_shtb_#3\expandafter\expandafter\expandafter\endcsname
570 \csname XINT_shtb_#4\expandafter\expandafter\expandafter\endcsname
571 \csname XINT_shtb_#5\expandafter\expandafter\expandafter\endcsname
572 \csname XINT_shtb_#6\expandafter\expandafter\expandafter\endcsname
573 \csname XINT_shtb_#7\expandafter\expandafter\expandafter\endcsname
574 \csname XINT_shtb_#8\expandafter\expandafter\expandafter\endcsname
575 \csname XINT_shtb_#9\endcsname }{#1}%
576 }%
577 \def\XINT_htb_I_b #1#2{\XINT_htb_I_a {#2#1}}%
578 \def\XINT_htb_II_a G\expandafter\expandafter\expandafter\XINT_htb_I_b

```

```

579 {%
580   \expandafter\expandafter\expandafter \XINT_htb_II_b
581 }%
582 \def\XINT_htb_II_b #1#2#3\T
583 {%
584   \XINT_num_loop #2#1%
585   \xint_relax\xint_relax\xint_relax\xint_relax
586   \xint_relax\xint_relax\xint_relax\xint_relax\Z
587 }%

```

## 5.10 \xintCHexToBin

1.08

```

588 \def\xintCHexToBin {\romannumeral0\xintchextobin }%
589 \def\xintchextobin #1%
590 {%
591   \expandafter\XINT_chtb_checkin\romannumeral`&&@#1%
592   \R\R\R\R\R\R\R\R\Z \W\W\W\W\W\W\W\W
593 }%
594 \def\XINT_chtb_checkin #1%
595 {%
596   \xint_UDsignfork
597   #1\XINT_chtb_N
598   -{\XINT_chtb_P #1}%
599   \krof
600 }%
601 \def\XINT_chtb_N {\expandafter\xint_minus_thenstop\romannumeral0\XINT_chtb_P }%
602 \def\XINT_chtb_P {\expandafter\XINT_chtb_I\expandafter{\expandafter}%
603   \romannumeral0\XINT_OQ {}}%
604 \def\XINT_chtb_I #1#2#3#4#5#6#7#8#9%
605 {%
606   \xint_gob_til_W #9\XINT_chtb_end_a\W
607   \expandafter\expandafter\expandafter
608   \XINT_chtb_I
609   \expandafter\expandafter\expandafter
610   {\csname XINT_shtb_#9\expandafter\expandafter\expandafter\endcsname
611   \csname XINT_shtb_#8\expandafter\expandafter\expandafter\endcsname
612   \csname XINT_shtb_#7\expandafter\expandafter\expandafter\endcsname
613   \csname XINT_shtb_#6\expandafter\expandafter\expandafter\endcsname
614   \csname XINT_shtb_#5\expandafter\expandafter\expandafter\endcsname
615   \csname XINT_shtb_#4\expandafter\expandafter\expandafter\endcsname
616   \csname XINT_shtb_#3\expandafter\expandafter\expandafter\endcsname
617   \csname XINT_shtb_#2\endcsname
618   #1}%
619 }%
620 \def\XINT_chtb_end_a\W\expandafter\expandafter\expandafter
621   \XINT_chtb_I\expandafter\expandafter\expandafter #1%
622 {%
623   \XINT_chtb_end_b #1%
624   \xint_relax\xint_relax\xint_relax\xint_relax
625   \xint_relax\xint_relax\xint_relax\xint_relax\Z
626 }%

```

## 5 Package *xintbinhex* implementation

```
627 \def\XINT_chtb_end_b #1\W#2\W#3\W#4\W#5\W#6\W#7\W#8\W\endcsname
628 {%
629     \XINT_num_loop
630 }%
631 \XINT_restorecatcodes_endinput%
```

## 6 Package *xintgcd* implementation

.1	Catcodes, $\varepsilon$ -TeX and reload detection . . .	148	.7	<code>\xintBezoutAlgorithm</code> . . . . .	155
.2	Package identification . . . . .	149	.8	<code>\xintGCDof</code> . . . . .	157
.3	<code>\xintGCD</code> , <code>\xintiiGCD</code> . . . . .	149	.9	<code>\xintLCMof</code> . . . . .	157
.4	<code>\xintLCM</code> , <code>\xintiiLCM</code> . . . . .	150	.10	<code>\xintTypesetEuclideanAlgorithm</code> . . . . .	158
.5	<code>\xintBezout</code> . . . . .	150	.11	<code>\xintTypesetBezoutAlgorithm</code> . . . . .	158
.6	<code>\xintEuclideanAlgorithm</code> . . . . .	154			

The commenting is currently (2016/03/12) 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  $\mathbb{X}$ 's `\loop`.

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

### 6.1 Catcodes, $\varepsilon$ -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 [2016/03/12 1.2f Euclidean algorithm with xint package (JFB)]%

```

## 6.3 *\xintGCD*, *\xintiGCD*

```

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 }%
56 \def\xintiGCD {\romannumeral0\xintiigcd}%
57 \def\xintiigcd #1%
58 {%
59   \expandafter\XINT_iigcd\expandafter{\romannumeral0\xintiiabs {#1}}%
60 }%
61 \def\XINT_iigcd #1#2%
62 {%
63   \expandafter\XINT_gcd_fork\romannumeral0\xintiiabs {#2}\Z #1\Z
64 }%

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

65 \def\XINT_gcd_fork #1#2\Z #3#4\Z
66 {%
67   \xint_UDzerofork
68   #1\XINT_gcd_BisZero
69   #3\XINT_gcd_AisZero
70   0\XINT_gcd_loop
71   \krof
72   {#1#2}{#3#4}%
73 }%
74 \def\XINT_gcd_AisZero #1#2{ #1}%
75 \def\XINT_gcd_BisZero #1#2{ #2}%
76 \def\XINT_gcd_CheckRem #1#2\Z
77 {%
78   \xint_gob_til_zero #1\xintgcd_end0\XINT_gcd_loop {#1#2}%

```

```

79 }%
80 \def\xint_gcd_end0\XINT_gcd_loop #1#2{ #2}%

#1=B, #2=A

81 \def\XINT_gcd_loop #1#2%
82 {%
83   \expandafter\expandafter\expandafter
84     \XINT_gcd_CheckRem
85   \expandafter\xint_secondoftwo
86   \romannumeral0\XINT_div_prepare {#1}{#2}\Z
87   {#1}%
88 }%

```

#### 6.4 \xintLCM, \xintiilCM

```

89 \def\xintLCM {\romannumeral0\xintlcm}%
90 \def\xintlcm #1%
91 {%
92   \expandafter\XINT_lcm\expandafter{\romannumeral0\xintiabs {#1}}%
93 }%
94 \def\XINT_lcm #1#2%
95 {%
96   \expandafter\XINT_lcm_fork\romannumeral0\xintiabs {#2}\Z #1\Z
97 }%
98 \def\xintiilCM {\romannumeral0\xintiilcm}%
99 \def\xintiilcm #1%
100 {%
101   \expandafter\XINT_iilcm\expandafter{\romannumeral0\xintiabs {#1}}%
102 }%
103 \def\XINT_iilcm #1#2%
104 {%
105   \expandafter\XINT_lcm_fork\romannumeral0\xintiabs {#2}\Z #1\Z
106 }%
107 \def\XINT_lcm_fork #1#2\Z #3#4\Z
108 {%
109   \xint_UDzerofork
110   #1\XINT_lcm_BisZero
111   #3\XINT_lcm_AisZero
112   0\expandafter
113   \krof
114   \XINT_lcm_notzero\expandafter{\romannumeral0\XINT_gcd_loop {#1#2}{#3#4}}%
115   {#1#2}{#3#4}%
116 }%
117 \def\XINT_lcm_AisZero #1#2#3#4#5{ 0}%
118 \def\XINT_lcm_BisZero #1#2#3#4#5{ 0}%
119 \def\XINT_lcm_notzero #1#2#3{\xintiimul {#2}{\xintiiQuo{#3}{#1}}}%

```

#### 6.5 \xintBezout

```

120 \def\xintBezout {\romannumeral0\xintbezout }%
121 \def\xintbezout #1%
122 {%
123   \expandafter\xint_bezout\expandafter {\romannumeral0\xintnum{#1}}%
124 }%

```

```

125 \def\xint_bezout #1#2%
126 {%
127   \expandafter\XINT_bezout_fork \romannumeral0\xintnum{#2}\Z #1\Z
128 }%

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

129 \def\XINT_bezout_fork #1#2\Z #3#4\Z
130 {%
131   \xint_UDzerosfork
132   #1#3\XINT_bezout_botharezero
133   #10\XINT_bezout_secondiszero
134   #30\XINT_bezout_firstiszero
135   00{\xint_UDsignsfork
136     #1#3\XINT_bezout_minusminus % A < 0, B < 0
137     #1-\XINT_bezout_minusplus % A > 0, B < 0
138     #3-\XINT_bezout_plusminus % A < 0, B > 0
139     --\XINT_bezout_plusplus % A > 0, B > 0
140   \krof }%
141   \krof
142   {#2}{#4}#1#3{#3#4}{#1#2}% #1#2=B, #3#4=A
143 }%
144 \edef\XINT_bezout_botharezero #1#2#3#4#5#6%
145 {%
146   \noexpand\xintError:NoBezoutForZeros\space {0}{0}{0}{0}{0}%
147 }%

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

148 \def\XINT_bezout_firstiszero #1#2#3#4#5#6%
149 {%
150   \xint_UDsignfork
151   #3{ {0}{#3#1}{0}{1}{#1}}%
152   -{ {0}{#3#1}{0}{-1}{#1}}%
153   \krof
154 }%

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

155 \def\XINT_bezout_secondiszero #1#2#3#4#5#6%
156 {%
157   \xint_UDsignfork
158   #4{ {#4#2}{0}{-1}{0}{#2}}%
159   -{ {#4#2}{0}{1}{0}{#2}}%
160   \krof
161 }%

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

162 \def\XINT_bezout_minusminus #1#2#3#4%
163 {%
164   \expandafter\XINT_bezout_mm_post
165   \romannumeral0\XINT_bezout_loop_a 1{#1}{#2}1001%
166 }%
167 \def\XINT_bezout_mm_post #1#2%

```

```

168 {%
169   \expandafter\XINT_bezout_mm_postb\expandafter
170   {\romannumeral0\xintiopp{#2}}{\romannumeral0\xintiopp{#1}}%
171 }%
172 \def\XINT_bezout_mm_postb #1#2%
173 {%
174   \expandafter\XINT_bezout_mm_postc\expandafter {#2}{#1}%
175 }%
176 \edef\XINT_bezout_mm_postc #1#2#3#4#5%
177 {%
178   \space {#4}{#5}{#1}{#2}{#3}%
179 }%

```

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

```

180 \def\XINT_bezout_minusplus #1#2#3#4%
181 {%
182   \expandafter\XINT_bezout_mp_post
183   \romannumeral0\XINT_bezout_loop_a 1{#1}{#4#2}1001%
184 }%
185 \def\XINT_bezout_mp_post #1#2%
186 {%
187   \expandafter\XINT_bezout_mp_postb\expandafter
188   {\romannumeral0\xintiopp {#2}}{#1}%
189 }%
190 \edef\XINT_bezout_mp_postb #1#2#3#4#5%
191 {%
192   \space {#4}{#5}{#2}{#1}{#3}%
193 }%

```

*plusminus* A < 0, B > 0

```

194 \def\XINT_bezout_plusminus #1#2#3#4%
195 {%
196   \expandafter\XINT_bezout_pm_post
197   \romannumeral0\XINT_bezout_loop_a 1{#3#1}{#2}1001%
198 }%
199 \def\XINT_bezout_pm_post #1%
200 {%
201   \expandafter \XINT_bezout_pm_postb \expandafter
202   {\romannumeral0\xintiopp{#1}}%
203 }%
204 \edef\XINT_bezout_pm_postb #1#2#3#4#5%
205 {%
206   \space {#4}{#5}{#1}{#2}{#3}%
207 }%

```

*plusplus*

```

208 \def\XINT_bezout_plusplus #1#2#3#4%
209 {%
210   \expandafter\XINT_bezout_pp_post
211   \romannumeral0\XINT_bezout_loop_a 1{#3#1}{#4#2}1001%
212 }%

```

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



```

213 \edef\XINT_bezout_pp_post #1#2#3#4#5%
214 {%
215     \space {#4}{#5}{#1}{#2}{#3}%
216 }%

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

217 \def\XINT_bezout_loop_a #1#2#3%
218 {%
219     \expandafter\XINT_bezout_loop_b
220     \expandafter{\the\numexpr -#1\expandafter}%
221     \romannumeral0\XINT_div_prepare {#2}{#3}{#2}%
222 }%

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

223 \def\XINT_bezout_loop_b #1#2#3#4#5#6#7#8%
224 {%
225     \expandafter \XINT_bezout_loop_c \expandafter
226     {\romannumeral0\xinttiadd{\XINT_mul_fork #5\Z #2\Z}{#7}}%
227     {\romannumeral0\xinttiadd{\XINT_mul_fork #6\Z #2\Z}{#8}}%
228     {#1}{#3}{#4}{#5}{#6}%
229 }%

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

230 \def\XINT_bezout_loop_c #1#2%
231 {%
232     \expandafter \XINT_bezout_loop_d \expandafter
233     {#2}{#1}%
234 }%

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

235 \def\XINT_bezout_loop_d #1#2#3#4#5%
236 {%
237     \XINT_bezout_loop_e #4\Z {#3}{#5}{#2}{#1}%
238 }%

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

239 \def\XINT_bezout_loop_e #1#2\Z
240 {%
241     \xint_gob_til_zero #1\xint_bezout_loop_exit0\XINT_bezout_loop_f
242     {#1#2}%
243 }%

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

244 \def\XINT_bezout_loop_f #1#2%
245 {%
246     \XINT_bezout_loop_a {#2}{#1}%
247 }%
```

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

```
248 \def\xint_bezout_loop_exit0\xint_bezout_loop_f #1#2%
249 {%
250   \ifcase #2
251   \or \expandafter\xint_bezout_exiteven
252   \else\expandafter\xint_bezout_exitodd
253   \fi
254 }%
255 \edef\xint_bezout_exiteven #1#2#3#4#5%
256 {%
257   \space {#5}{#4}{#1}%
258 }%
259 \edef\xint_bezout_exitodd #1#2#3#4#5%
260 {%
261   \space {-#5}{-#4}{#1}%
262 }%
```

## 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

```
263 \def\xintEuclideanAlgorithm {\romannumeral0\xinteucclideanalgorithm }%
264 \def\xinteucclideanalgorithm #1%
265 {%
266   \expandafter \XINT_euc \expandafter{\romannumeral0\xintiabs {#1}}%
267 }%
268 \def\XINT_euc #1#2%
269 {%
270   \expandafter\XINT_euc_fork \romannumeral0\xintiabs {#2}\Z #1\Z
271 }%
```

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

```
272 \def\XINT_euc_fork #1#2\Z #3#4\Z
273 {%
274   \xint_UDzerofork
275   #1\XINT_euc_BisZero
276   #3\XINT_euc_AisZero
277   0\XINT_euc_a
278   \krof
279   {0}{#1#2}{#3#4}{#3#4}{#1#2}}\Z
280 }%
```

Le  $\{\}$  pour protéger  $\{A\}\{B\}$  si on s'arrête après une étape ( $B$  divise  $A$ ). On va renvoyer:  
 $\{N\}\{A\}\{D=r(n)\}\{B\}\{q_1\}\{r_1\}\{q_2\}\{r_2\}\{q_3\}\{r_3\}\dots\{q_N\}\{r_N=0\}$

```
281 \def\XINT_euc_AisZero #1#2#3#4#5#6{ {1}{0}{#2}{#2}{0}{0}}%
282 \def\XINT_euc_BisZero #1#2#3#4#5#6{ {1}{0}{#3}{#3}{0}{0}}%

{n}{rn}{an}{q_n}{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
```

```

283 \def\XINT_euc_a #1#2#3%
284 {%
285   \expandafter\XINT_euc_b
286   \expandafter {\the\numexpr #1+1\expandafter}%
287   \romannumeral0\XINT_div_prepare {#2}{#3}{#2}%
288 }%

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

289 \def\XINT_euc_b #1#2#3#4%
290 {%
291   \XINT_euc_c #3\Z {#1}{#3}{#4}{{#2}{#3}}%
292 }%

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.

293 \def\XINT_euc_c #1#2\Z
294 {%
295   \xint_gob_til_zero #1\xint_euc_end0\XINT_euc_a
296 }%

{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}

297 \def\xint_euc_end0\XINT_euc_a #1#2#3#4\Z%
298 {%
299   \expandafter\xint_euc_end_
300   \romannumeral0%
301   \XINT_rord_main {}#4{{#1}{#3}}%
302   \xint_relax
303   \xint_bye\xint_bye\xint_bye\xint_bye
304   \xint_bye\xint_bye\xint_bye\xint_bye
305   \xint_relax
306 }%
307 \edef\xint_euc_end_ #1#2#3%
308 {%
309   \space {#1}{#3}{#2}%
310 }%

```

## 6.7 \xintBezoutAlgorithm

Pour Bezout: objectif, renvoyer

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

```

311 \def\xintBezoutAlgorithm {\romannumeral0\xintbezoutalgorithm}%
312 \def\xintbezoutalgorithm #1%
313 {%
314   \expandafter \XINT_bezalg \expandafter{\romannumeral0\xintiabs {#1}}%
315 }%
316 \def\XINT_bezalg #1#2%

```

```

317 {%
318     \expandafter\XINT_bezalg_fork \romannumeral0\xintiabs {#2}\Z #1\Z
319 }%

```

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

```

320 \def\XINT_bezalg_fork #1#2\Z #3#4\Z
321 {%
322     \xint_UDzerofork
323     #1\XINT_bezalg_BisZero
324     #3\XINT_bezalg_AisZero
325     0\XINT_bezalg_a
326     \krof
327     0{#1#2}{#3#4}1001{{#3#4}{#1#2}}{}}\Z
328 }%
329 \def\XINT_bezalg_AisZero #1#2#3\Z{ {1}{0}{0}{1}{#2}{#2}{1}{0}{0}{0}{0}{1}}%
330 \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  $\{n\}\{r(n)\}\{r(n-1)\}\{\alpha(n)\}\{\beta(n)\}\{\alpha(n-1)\}\{\beta(n-1)\}\{q(n)\}\{r(n)\}\{\alpha(n)\}$   
division de #3 par #2

```

331 \def\XINT_bezalg_a #1#2#3%
332 {%
333     \expandafter\XINT_bezalg_b
334     \expandafter {\the\numexpr #1+1\expandafter}%
335     \romannumeral0\XINT_div_prepare {#2}{#3}{#2}%
336 }%

```

$$\{n+1\}\{q(n+1)\}\{r(n+1)\}\{r(n)\}\{\alpha(n)\}\{\beta(n)\}\{\alpha(n-1)\}\{\beta(n-1)\} \dots$$

```

337 \def\XINT_bezalg_b #1#2#3#4#5#6#7#8%
338 {%
339     \expandafter\XINT_bezalg_c\expandafter
340     {\romannumeral0\xintiiadd {\xintiiMul {#6}{#2}}{#8}}%
341     {\romannumeral0\xintiiadd {\xintiiMul {#5}{#2}}{#7}}%
342     {#1}{#2}{#3}{#4}{#5}{#6}%
343 }%

```

$$\{\beta(n+1)\}\{\alpha(n+1)\}\{n+1\}\{q(n+1)\}\{r(n+1)\}\{r(n)\}\{\alpha(n)\}\{\beta(n)\}$$

```

344 \def\XINT_bezalg_c #1#2#3#4#5#6%
345 {%
346     \expandafter\XINT_bezalg_d\expandafter {#2}{#3}{#4}{#5}{#6}{#1}%
347 }%

```

$$\{\alpha(n+1)\}\{n+1\}\{q(n+1)\}\{r(n+1)\}\{r(n)\}\{\beta(n+1)\}$$

```

348 \def\XINT_bezalg_d #1#2#3#4#5#6#7#8%
349 {%
350     \XINT_bezalg_e #4\Z {#2}{#4}{#5}{#1}{#6}{#7}{#8}{{#3}{#4}{#1}{#6}}%
351 }%

```

```

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.

```

```

352 \def\XINT_bezalg_e #1#2\Z
353 {%
354   \xint_gob_til_zero #1\xint_bezalg_end0\XINT_bezalg_a
355 }%

Ici  $r(n+1) = 0$ . On arrête on se prépare à inverser.
 $\{n+1\}\{r(n+1)\}\{r(n)\}\{\alpha(n+1)\}\{\beta(n+1)\}\{\alpha(n)\}\{\beta(n)\}$ 
 $\{q,r,\alpha,\beta(n+1)\} \dots \{A\}\{B\}\}\{Z$ 
On veut renvoyer
 $\{N\}\{A\}\{0\}\{1\}\{D=r(n)\}\{B\}\{1\}\{0\}\{q_1\}\{r_1\}\{\alpha_1=q_1\}\{\beta_1=1\}$ 
 $\{q_2\}\{r_2\}\{\alpha_2\}\{\beta_2\} \dots \{q_N\}\{r_N=0\}\{\alpha_N=A/D\}\{\beta_N=B/D\}$ 

356 \def\xint_bezalg_end0\XINT_bezalg_a #1#2#3#4#5#6#7#8\Z
357 {%
358   \expandafter\xint_bezalg_end_
359   \romannumeral0%
360   \XINT_rord_main {}#8{{#1}\{#3\}}%
361   \xint_relax
362   \xint_bye\xint_bye\xint_bye\xint_bye
363   \xint_bye\xint_bye\xint_bye\xint_bye
364   \xint_relax
365 }%

 $\{N\}\{D\}\{A\}\{B\}\{q_1\}\{r_1\}\{\alpha_1=q_1\}\{\beta_1=1\}\{q_2\}\{r_2\}\{\alpha_2\}\{\beta_2\}$ 
 $\dots \{q_N\}\{r_N=0\}\{\alpha_N=A/D\}\{\beta_N=B/D\}$ 
On veut renvoyer
 $\{N\}\{A\}\{0\}\{1\}\{D=r(n)\}\{B\}\{1\}\{0\}\{q_1\}\{r_1\}\{\alpha_1=q_1\}\{\beta_1=1\}$ 
 $\{q_2\}\{r_2\}\{\alpha_2\}\{\beta_2\} \dots \{q_N\}\{r_N=0\}\{\alpha_N=A/D\}\{\beta_N=B/D\}$ 

366 \edef\xint_bezalg_end_ #1#2#3#4%
367 {%
368   \space {}#1}\{#3\}\{0\}\{1\}\{#2\}\{#4\}\{1\}\{0\}%
369 }%
```

## 6.8 \xintGCDof

```

370 \def\xintGCDof      {\romannumeral0\xintgcdof }%
371 \def\xintgcdof      #1{\expandafter\XINT_gcdof_a\romannumeral`&&@#1\relax }%
372 \def\XINT_gcdof_a #1{\expandafter\XINT_gcdof_b\romannumeral`&&@#1\Z }%
373 \def\XINT_gcdof_b #1\Z #2{\expandafter\XINT_gcdof_c\romannumeral`&&@#2\Z {#1}\Z}%
374 \def\XINT_gcdof_c #1{\xint_gob_til_relax #1\XINT_gcdof_e\relax\XINT_gcdof_d #1}%
375 \def\XINT_gcdof_d #1\Z {\expandafter\XINT_gcdof_b\romannumeral0\xintgcd {#1}}%
376 \def\XINT_gcdof_e #1\Z #2\Z { #2}%
```

## 6.9 \xintLCMof

New with 1.09a

```

377 \def\xintLCMof      {\romannumeral0\xintlcmof }%
378 \def\xintlcmof      #1{\expandafter\XINT_lcmof_a\romannumeral`&&@#1\relax }%
379 \def\XINT_lcmof_a #1{\expandafter\XINT_lcmof_b\romannumeral`&&@#1\Z }%
380 \def\XINT_lcmof_b #1\Z #2{\expandafter\XINT_lcmof_c\romannumeral`&&@#2\Z {#1}\Z}%
381 \def\XINT_lcmof_c #1{\xint_gob_til_relax #1\XINT_lcmof_e\relax\XINT_lcmof_d #1}%
382 \def\XINT_lcmof_d #1\Z {\expandafter\XINT_lcmof_b\romannumeral0\xintlcm {#1}}%
383 \def\XINT_lcmof_e #1\Z #2\Z { #2}%
```

## 6.10 `\xintTypesetEuclideanAlgorithm`

TYPESETTING

Organisation:

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

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

$r(n-2) = q(n)r(n-1)+r(n)$  (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`

```

384 \def\xintTypesetEuclideanAlgorithm {%
385   \unless\ifdefined\xintAssignArray
386     \errmessage
387       {xintgcd: package xinttools is required for \string\xintTypesetEuclideanAlgorithm}%
388     \expandafter\xint_gobble_iii
389   \fi
390   \XINT_TypesetEuclideanAlgorithm
391 }%
392 \def\XINT_TypesetEuclideanAlgorithm #1#2%
393 {% l'algo remplace #1 et #2 par |#1| et |#2|
394   \par
395   \begingroup
396     \xintAssignArray\xintEuclideanAlgorithm {#1}{#2}\to\U
397     \edef\A{\U2}\edef\B{\U4}\edef\N{\U1}%
398     \setbox 0 \vbox{\halign {$##$\cr \A\cr \B \cr}}%
399     \count 255 1
400     \xintloop
401       \indent\hbox to \wd 0 {\hfil$\U{\numexpr 2*\count255\relax}$}%
402       ${} = \U{\numexpr 2*\count255 + 3\relax}
403       \times \U{\numexpr 2*\count255 + 2\relax}
404       + \U{\numexpr 2*\count255 + 4\relax}$%
405     \ifnum \count255 < \N
406       \par
407       \advance \count255 1
408     \repeat
409   \endgroup
410 }%

```

## 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:  $U1 = N$ ,  $U2= A$ ,  $U5=D$ ,  $U6=B$ ,  $q_1 = U9$ ,  $q_n = U\{4n+5\}$ , 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`

```

411 \def\xintTypesetBezoutAlgorithm {%
412   \unless\ifdefined\xintAssignArray
413     \errmessage
414       {xintgcd: package xinttools is required for \string\xintTypesetBezoutAlgorithm}%

```

```

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

```

## 7 Package *xintfrac* implementation

.1	Catcodes, $\varepsilon$ -T <sub>E</sub> X and reload detection . .	160	.39	<code>\xintPow</code> . . . . .	191
.2	Package identification . . . . .	161	.40	<code>\xintiFac</code> . . . . .	192
.3	<code>\XINT_cntSgnFork</code> . . . . .	161	.41	<code>\xintiBinomial</code> . . . . .	192
.4	<code>\xintLen</code> . . . . .	162	.42	<code>\xintiPFactorial</code> . . . . .	193
.5	<code>\XINT_lenrord_loop</code> . . . . .	162	.43	<code>\xintPrd</code> . . . . .	193
.6	<code>\XINT_outfrac</code> . . . . .	163	.44	<code>\xintDiv</code> . . . . .	193
.7	<code>\XINT_inFrac</code> . . . . .	163	.45	<code>\xintDivFloor</code> . . . . .	194
.8	<code>\XINT_frac_gen</code> . . . . .	165	.46	<code>\xintDivTrunc</code> . . . . .	194
.9	<code>\XINT_factoritens</code> , <code>\XINT_cuz_cnt</code> . .	167	.47	<code>\xintDivRound</code> . . . . .	194
.10	<code>\XINT_addm_A</code> . . . . .	169	.48	<code>\xintMod</code> . . . . .	194
.11	<code>\xintRaw</code> . . . . .	170	.49	<code>\xintIsOne</code> . . . . .	195
.12	<code>\xintPRaw</code> . . . . .	171	.50	<code>\xintGeq</code> . . . . .	195
.13	<code>\xintRawWithZeros</code> . . . . .	171	.51	<code>\xintMax</code> . . . . .	196
.14	<code>\xintFloor</code> , <code>\xintiFloor</code> . . . . .	172	.52	<code>\xintMaxof</code> . . . . .	197
.15	<code>\xintCeil</code> , <code>\xintiCeil</code> . . . . .	172	.53	<code>\xintMin</code> . . . . .	197
.16	<code>\xintNumerator</code> . . . . .	172	.54	<code>\xintMinof</code> . . . . .	198
.17	<code>\xintDenominator</code> . . . . .	172	.55	<code>\xintCmp</code> . . . . .	198
.18	<code>\xintFrac</code> . . . . .	173	.56	<code>\xintAbs</code> . . . . .	199
.19	<code>\xintSignedFrac</code> . . . . .	173	.57	<code>\xintOpp</code> . . . . .	199
.20	<code>\xintFwOver</code> . . . . .	174	.58	<code>\xintSgn</code> . . . . .	199
.21	<code>\xintSignedFwOver</code> . . . . .	174	.59	Floating point macros . . . . .	199
.22	<code>\xintREZ</code> . . . . .	175	.60	<code>\xintFloat</code> . . . . .	200
.23	<code>\xintE</code> . . . . .	176	.61	<code>\XINTinFloat</code> . . . . .	205
.24	<code>\xintIrr</code> . . . . .	176	.62	<code>\xintPFloat</code> . . . . .	209
.25	<code>\xintifInt</code> . . . . .	177	.63	<code>\XINTinFloatFracdigits</code> . . . . .	211
.26	<code>\xintJrr</code> . . . . .	177	.64	<code>\xintFloatAdd</code> , <code>\XINTinFloatAdd</code> . .	211
.27	<code>\xintTFrac</code> . . . . .	178	.65	<code>\xintFloatSub</code> , <code>\XINTinFloatSub</code> . .	212
.28	<code>\xintTrunc</code> , <code>\xintiTrunc</code> . . . . .	179	.66	<code>\xintFloatMul</code> , <code>\XINTinFloatMul</code> . .	213
.29	<code>\xintTTrunc</code> . . . . .	181	.67	<code>\xintFloatDiv</code> , <code>\XINTinFloatDiv</code> . .	214
.30	<code>\xintNum</code> . . . . .	181	.68	<code>\xintFloatPower</code> , <code>\XINTinFloatPower</code> .	218
.31	<code>\xintRound</code> , <code>\xintiRound</code> . . . . .	181	.69	<code>\xintFloatFac</code> , <code>\XINTFloatFac</code> . . .	221
.32	<code>\xintXTrunc</code> . . . . .	182	.70	<code>\xintFloatPFactorial</code> , <code>\XINTinFloatP-</code> <code>Factorial</code> . . . . .	226
.33	<code>\xintDigits</code> . . . . .	188	.71	<code>\xintFloatBinomial</code> , <code>\XINTinFloatBino-</code> <code>mial</code> . . . . .	229
.34	<code>\xintAdd</code> . . . . .	188	.72	<code>\xintFloatSqrt</code> , <code>\XINTinFloatSqrt</code> . .	230
.35	<code>\xintSub</code> . . . . .	190	.73	<code>\xintFloatE</code> , <code>\XINTinFloatE</code> . . . .	232
.36	<code>\xintSum</code> . . . . .	190	.74	<code>\XINTinFloatMod</code> . . . . .	233
.37	<code>\xintMul</code> . . . . .	190			
.38	<code>\xintSqr</code> . . . . .	191			

The commenting is currently (2016/03/12) very sparse.

### 7.1 Catcodes, $\varepsilon$ -T<sub>E</sub>X 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 [2016/03/12 1.2f Expandable operations on fractions (JFB)]%

```

## 7.3 `\XINT_cntSgnFork`

1.09i. Used internally, #1 must expand to `\m@ne`, `\z@`, or `\@ne` or equivalent. `\XINT_cntSgnFork` does not insert a romannumeral stopper.

```

47 \def\XINT_cntSgnFork #1%
48 {%
49 \ifcase #1\expandafter\xint_secondofthree
50 \or\expandafter\xint_thirdofthree

```

```

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

```

## 7.4 `\xintLen`

The used formula is disputable, the idea is that  $A/1$  and  $A$  should have same length.

```

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

```

## 7.5 `\XINT_lenrord_loop`

Faire `\romannumeral`&&@\XINT_lenrord_loop 0.{}`foobar`\Z\W\W\W\W\W\W\W\Z`, et en sortie on aura : longueur.raboof`\Z`. C'est une vieille routine, employée uniquement par `\xintFloat` et `\XINTinFloat`, et uniquement pour des chiffres. Donc j'ai plus rapide maintenant mais elle n'est utilisée que pour des choses assez courtes, alors à voir.

```

64 \def\XINT_lenrord_loop #1.#2#3#4#5#6#7#8#9%
65 {%
66     \xint_gob_til_W #9\XINT_lenrord_W\W
67     \expandafter\XINT_lenrord_loop\the\numexpr #1+\xint_c_vii.{#9#8#7#6#5#4#3#2}%
68 }%
69 \def\XINT_lenrord_W\W\expandafter\XINT_lenrord_loop #1.#2#3\Z
70 {%
71     \expandafter\XINT_lenrord_X #1.#2\Z
72 }%
73 \def\XINT_lenrord_X #1.#2\Z
74 {%
75     \XINT_lenrord_Y #2\R\R\R\R\R\R\T {#1}%
76 }%
77 \def\XINT_lenrord_Y #1#2#3#4#5#6#7#8\T
78 {%
79     \xint_gob_til_W
80         #7\XINT_lenrord_Z \xint_c_viii
81         #6\XINT_lenrord_Z \xint_c_vii
82         #5\XINT_lenrord_Z \xint_c_vi
83         #4\XINT_lenrord_Z \xint_c_v
84         #3\XINT_lenrord_Z \xint_c_iv
85         #2\XINT_lenrord_Z \xint_c_iii
86         \W\XINT_lenrord_Z \xint_c_ii \Z
87 }%
88 \def\XINT_lenrord_Z #1#2\Z #3{\the\numexpr #3-#1.}%

```

## 7.6 `\XINT_outfrac`

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

```

89 \def\XINT_outfrac #1#2#3%
90 {%
91   \ifcase\XINT_cntSgn #3\Z
92     \expandafter \XINT_outfrac_divisionbyzero
93   \or
94     \expandafter \XINT_outfrac_P
95   \else
96     \expandafter \XINT_outfrac_N
97   \fi
98   {#2}{#3}[#1]%
99 }%
100 \def\XINT_outfrac_divisionbyzero #1#2{\xintError:DivisionByZero\space #1/0}%
101 \edef\XINT_outfrac_P #1#2%
102 {%
103   \noexpand\if0\noexpand\XINT_Sgn #1\noexpand\Z
104     \noexpand\expandafter\noexpand\XINT_outfrac_Zero
105   \noexpand\fi
106   \space #1/#2%
107 }%
108 \def\XINT_outfrac_Zero #1[#2]{ 0/1[0]}%
109 \def\XINT_outfrac_N #1#2%
110 {%
111   \expandafter\XINT_outfrac_N_a\expandafter
112   {\romannumeral0\XINT_opp #2}{\romannumeral0\XINT_opp #1}%
113 }%
114 \def\XINT_outfrac_N_a #1#2%
115 {%
116   \expandafter\XINT_outfrac_P\expandafter {#2}{#1}%
117 }%

```

## 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.

2015/10/09: this venerable macro from the very early days (1.03, 2013/04/14) has gotten a lift-ing for release 1.2. There were two kinds of issues:

- 1) use of `\W`, `\Z`, `\T` delimiters was very poor choice as this could clash with user input,
- 2) the new `\XINT_frac_gen` handles macros (possibly empty) in the input as general as `\A.\Be\C/\D.\Ee\F`. The earlier version would not have expanded the `\B` or `\E`: digits after decimal mark were constrained to arise from expansion of the first token. Thus the 1.03 original code would have expanded only `\A`, `\D`, `\C`, and `\F` for this input.

This reminded me think I should revisit the remaining earlier portions of code, as I was still learning TeX coding when I wrote them.

## 7 Package *xintfrac* implementation

Also I thought about parsing even faster the  $A/B[N]$  input, not expanding  $B$ , but this turned out to clash with some established uses in the documentation such as  $1/\text{xintiiSqr}\{...\}[0]$ . For the implementation, careful here about potential brace removals with parameter patterns such as like  $\#1/\#2\#3[\#4]$  for example.

While I was at it 1.2 added `\numexpr` parsing of the  $N$ , which earlier was restricted to be only explicit digits. I allowed  $[]$  with empty  $N$ , but the way I did it in 1.2 with `\the\numexpr 0\#1` was buggy, as it did not allow  $\#1$  to be a `\count` for example or itself a `\numexpr` (although such inputs were not previously allowed, I later turned out to use them in the code itself, e.g. the float factorial of version 1.2f). The better way would be `\the\numexpr\#1+\xint_c_` but 1.2f finally does only `\the\numexpr \#1` and  $\#1$  is not allowed to be empty.

The 1.2 `\XINT_frac_gen` had two locations with such a problematic `\numexpr 0\#1` which I replaced for 1.2f with `\numexpr\#1+\xint_c_`.

Note: regarding calling the macro with  $A[\text{<expression>}]$ , the  $/$  must be suitably hidden for example in `\firstofone` type constructs.

```

118 \def\XINT_inFrac {\romannumeral0\XINT_infrac }%
119 \def\XINT_infrac #1%
120 {%
121   \expandafter\XINT_infrac_fork\romannumeral`&&@#1/\XINT_W[\XINT_W\XINT_T
122 }%
123 \def\XINT_infrac_fork #1[#2%
124 {%
125   \xint_UDXINTWfork
126   #2\XINT_frac_gen
127   \XINT_W\XINT_infrac_res_a % strict A[N] or A/B[N] input
128   \krof
129   #1[#2%
130 }%
131 \def\XINT_infrac_res_a #1%
132 {%
133   \xint_gob_til_zero #1\XINT_infrac_res_zero 0\XINT_infrac_res_b #1%
134 }%
135 \def\XINT_infrac_res_zero 0\XINT_infrac_res_b #1\XINT_T {{0}{0}{1}}%
136 \def\XINT_infrac_res_b #1/#2%
137 {%
138   \xint_UDXINTWfork
139   #2\XINT_infrac_res_ca
140   \XINT_W\XINT_infrac_res_cb
141   \krof
142   #1/#2%
143 }%

```

An empty  $[]$  is not allowed. (this was authorized in 1.2, removed in 1.2f). As nobody reads `xint` documentation, noone will have noticed the fleeting possibility.

```

144 \def\XINT_infrac_res_ca #1[#2]/\XINT_W[\XINT_W\XINT_T
145   {\expandafter{\the\numexpr #2}\#1}\{1}}%
146 \def\XINT_infrac_res_cb #1/#2[%
147   {\expandafter\XINT_infrac_res_cc\romannumeral`&&@#2~#1}%
148 \def\XINT_infrac_res_cc #1~#2[#3]/\XINT_W[\XINT_W\XINT_T
149   {\expandafter{\the\numexpr #3}\#2}\{#1}}%

```

## 7.8 \XINT\_frac\_gen

Extended in 1.07 to recognize and accept scientific notation both at the numerator and (possible) denominator. Only a lowercase e will do here, but uppercase E is possible within an `\xintexpr`..`\relax`

Completely rewritten for 1.2 2015/10/10. The parsing handles inputs such as `\A.\Be\C/\D.\Ee\F` where each of `\A`, `\B`, `\D`, and `\E` may need `\fexpan` sion and `\C` and `\F` will end up in `\numexpr`.

1.2f corrects an issue to allow `\C` and `\F` to be `\count` variable (or expressions with `\numexpr`): 1.2 did a bad `\numexpr0#1` which allowed only explicit digits for expanded `#1`.

```

150 \def\xint_frac_gen #1/#2%
151 {%
152   \xint_UDXINTWfork
153   #2\xint_frac_gen_A
154   \XINT_W\xint_frac_gen_B
155   \krof
156   #1/#2%
157 }%
158 \def\xint_frac_gen_A #1/\XINT_W [\XINT_W {\xint_frac_gen_C 0~1!#1ee.\XINT_W }%
159 \def\xint_frac_gen_B #1/#2/\XINT_W [%\XINT_W
160 {%
161   \expandafter\xint_frac_gen_Ba
162   \romannumeral`&&@#2ee.\XINT_W\XINT_Z #1ee.%\XINT_W
163 }%
164 \def\xint_frac_gen_Ba #1.#2%
165 {%
166   \xint_UDXINTWfork
167   #2\xint_frac_gen_Bb
168   \XINT_W\xint_frac_gen_Bc
169   \krof
170   #1.#2%
171 }%
172 \def\xint_frac_gen_Bb #1e#2e#3\XINT_Z
173   {\expandafter\xint_frac_gen_C\the\numexpr #2+\xint_c_~#1!}%
174 \def\xint_frac_gen_Bc #1.#2e%
175 {%
176   \expandafter\xint_frac_gen_Bd\romannumeral`&&@#2.#1e%
177 }%

```

Here in `\XINT_frac_gen_Bd`, the 1.2 rewrite of `\XINT_infrac` did `\the\numexpr0#3-`, but if `#3` is a `\count` for example this was bad (although such inputs never have been explicitly allowed in the doc). And why this 0? to handle empty `#3`? But empty `#3` was fine here. Another bug of 1.2 ! Fixed in 1.2f. Same in `\XINT_frac_gen_Bb` above, with the difference however that an empty `#2` there indeed had to be handled properly.

```

178 \def\xint_frac_gen_Bd #1.#2e#3e#4\XINT_Z
179 {%
180   \expandafter\xint_frac_gen_C\the\numexpr #3-\romannumeral0\expandafter
181   \XINT_length_loop
182   0.#1\xint_relax\xint_relax\xint_relax\xint_relax
183   \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye~#2#1!%
184 }%
185 \def\xint_frac_gen_C #1!#2.#3%
186 {%

```

## 7 Package *xintfrac* implementation

```

187 \xint_UDXINTWfork
188   #3\XINT_frac_gen_Ca
189   \XINT_W\XINT_frac_gen_Cb
190   \krof
191   #1!#2.#3%
192 }%
193 \def\XINT_frac_gen_Ca #1~#2!#3e#4e#5\XINT_T
194 {%
195   \expandafter\XINT_frac_gen_F\the\numexpr #4-#1\expandafter
196   ~\romannumeral0\XINT_num_loop
197   #2\xint_relax\xint_relax\xint_relax\xint_relax
198   \xint_relax\xint_relax\xint_relax\xint_relax\Z~#3~%
199 }%
200 \def\XINT_frac_gen_Cb #1.#2e%
201 {%
202   \expandafter\XINT_frac_gen_Cc\romannumeral`&&@#2.#1e%
203 }%
204 \def\XINT_frac_gen_Cc #1.#2~#3!#4e#5e#6\XINT_T
205 {%
206   \expandafter\XINT_frac_gen_F\the\numexpr #5-#2-%
207   \romannumeral0\XINT_length_loop
208   0.#1\xint_relax\xint_relax\xint_relax\xint_relax
209   \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye\expandafter
210   ~\romannumeral0\XINT_num_loop
211   #3\xint_relax\xint_relax\xint_relax\xint_relax
212   \xint_relax\xint_relax\xint_relax\xint_relax\Z
213   ~#4#1~%
214 }%
215 \def\XINT_frac_gen_F #1~#2%
216 {%
217   \xint_UDzerominusfork
218   #2-\XINT_frac_gen_Gdivbyzero
219   0#2{\XINT_frac_gen_G -{}}%
220   0-{\XINT_frac_gen_G {}#2}%
221   \krof #1~%
222 }%
223 \def\XINT_frac_gen_Gdivbyzero #1~~#2~%
224 {%
225   \expandafter\XINT_frac_gen_Gdivbyzero_a
226   \romannumeral0\XINT_num_loop
227   #2\xint_relax\xint_relax\xint_relax\xint_relax
228   \xint_relax\xint_relax\xint_relax\xint_relax\Z~#1~%
229 }%
230 \def\XINT_frac_gen_Gdivbyzero_a #1~#2~%
231 {%
232   \xintError:DivisionByZero {#2}{#1}{0}%
233 }%
234 \def\XINT_frac_gen_G #1#2#3~#4~#5~%
235 {%
236   \expandafter\XINT_frac_gen_Ga
237   \romannumeral0\XINT_num_loop
238   #1#5\xint_relax\xint_relax\xint_relax\xint_relax

```

```

239     \xint_relax\xint_relax\xint_relax\xint_relax\Z~#3~{#2#4}%
240 }%
241 \def\XINT_frac_gen_Ga #1#2~#3~%
242 {%
243     \xint_gob_til_zero #1\XINT_frac_gen_zero 0%
244     {#3}{#1#2}%
245 }%
246 \def\XINT_frac_gen_zero 0#1#2#3{{0}{0}{1}}%

```

## 7.9 \XINT\_factortens, \XINT\_cuz\_cnt

Old routines.

```

247 \def\XINT_factortens #1%
248 {%
249     \expandafter\XINT_cuz_cnt_loop\expandafter
250     {\expandafter}\romannumeral0\XINT_rord_main {#1%
251     \xint_relax
252     \xint_bye\xint_bye\xint_bye\xint_bye
253     \xint_bye\xint_bye\xint_bye\xint_bye
254     \xint_relax
255     \R\R\R\R\R\R\R\R\Z
256 }%
257 \def\XINT_cuz_cnt #1%
258 {%
259     \XINT_cuz_cnt_loop {#1\R\R\R\R\R\R\R\R\Z
260 }%
261 \def\XINT_cuz_cnt_loop #1#2#3#4#5#6#7#8#9%
262 {%
263     \xint_gob_til_R #9\XINT_cuz_cnt_toofara \R
264     \expandafter\XINT_cuz_cnt_checka\expandafter
265     {\the\numexpr #1+8\relax}{#2#3#4#5#6#7#8#9}%
266 }%
267 \def\XINT_cuz_cnt_toofara\R
268     \expandafter\XINT_cuz_cnt_checka\expandafter #1#2%
269 {%
270     \XINT_cuz_cnt_toofarb {#1}#2%
271 }%
272 \def\XINT_cuz_cnt_toofarb #1#2\Z {\XINT_cuz_cnt_toofarc #2\Z {#1}}%
273 \def\XINT_cuz_cnt_toofarc #1#2#3#4#5#6#7#8%
274 {%
275     \xint_gob_til_R #2\XINT_cuz_cnt_toofard 7%
276     #3\XINT_cuz_cnt_toofard 6%
277     #4\XINT_cuz_cnt_toofard 5%
278     #5\XINT_cuz_cnt_toofard 4%
279     #6\XINT_cuz_cnt_toofard 3%
280     #7\XINT_cuz_cnt_toofard 2%
281     #8\XINT_cuz_cnt_toofard 1%
282     \Z #1#2#3#4#5#6#7#8%
283 }%
284 \def\XINT_cuz_cnt_toofard #1#2\Z #3\R #4\Z #5%
285 {%
286     \expandafter\XINT_cuz_cnt_toofare

```

## 7 Package *xintfrac* implementation

```

287 \the\numexpr #3\relax \R\R\R\R\R\R\R\Z
288 {\the\numexpr #5-#1\relax}\R\Z
289 }%
290 \def\XINT_cuz_cnt_toofare #1#2#3#4#5#6#7#8%
291 {%
292 \xint_gob_til_R #2\XINT_cuz_cnt_stopc 1%
293 #3\XINT_cuz_cnt_stopc 2%
294 #4\XINT_cuz_cnt_stopc 3%
295 #5\XINT_cuz_cnt_stopc 4%
296 #6\XINT_cuz_cnt_stopc 5%
297 #7\XINT_cuz_cnt_stopc 6%
298 #8\XINT_cuz_cnt_stopc 7%
299 \Z #1#2#3#4#5#6#7#8%
300 }%
301 \def\XINT_cuz_cnt_checka #1#2%
302 {%
303 \expandafter\XINT_cuz_cnt_checkb\the\numexpr #2\relax \Z {#1}%
304 }%
305 \def\XINT_cuz_cnt_checkb #1%
306 {%
307 \xint_gob_til_zero #1\expandafter\XINT_cuz_cnt_loop\xint_gob_til_Z
308 0\XINT_cuz_cnt_stopa #1%
309 }%
310 \def\XINT_cuz_cnt_stopa #1\Z
311 {%
312 \XINT_cuz_cnt_stopb #1\R\R\R\R\R\R\R\Z %
313 }%
314 \def\XINT_cuz_cnt_stopb #1#2#3#4#5#6#7#8#9%
315 {%
316 \xint_gob_til_R #2\XINT_cuz_cnt_stopc 1%
317 #3\XINT_cuz_cnt_stopc 2%
318 #4\XINT_cuz_cnt_stopc 3%
319 #5\XINT_cuz_cnt_stopc 4%
320 #6\XINT_cuz_cnt_stopc 5%
321 #7\XINT_cuz_cnt_stopc 6%
322 #8\XINT_cuz_cnt_stopc 7%
323 #9\XINT_cuz_cnt_stopc 8%
324 \Z #1#2#3#4#5#6#7#8#9%
325 }%
326 \def\XINT_cuz_cnt_stopc #1#2\Z #3\R #4\Z #5%
327 {%
328 \expandafter\XINT_cuz_cnt_stopd\expandafter
329 {\the\numexpr #5-#1}\#3%
330 }%
331 \def\XINT_cuz_cnt_stopd #1#2\R #3\Z
332 {%
333 \expandafter\space\expandafter
334 {\romannumeral0\XINT_rord_main {}}#2%
335 \xint_relax
336 \xint_bye\xint_bye\xint_bye\xint_bye
337 \xint_bye\xint_bye\xint_bye\xint_bye
338 \xint_relax {#1}%

```



339 }%

## 7.10 \XINT\_addm\_A

This is a routine from xintcore 1.0x, which is now only needed by \xintFloat, \XINTinFloat and \xintRound, for the time being.

```

340 \def\XINT_addm_A #1#2#3#4#5#6%
341 {%
342   \xint_gob_til_W #3\XINT_addm_az\W
343   \XINT_addm_AB #1{#3#4#5#6}{#2}%
344 }%
345 \def\XINT_addm_az\W\XINT_addm_AB #1#2%
346 {%
347   \XINT_addm_AC_checkcarry #1%
348 }%
349 \def\XINT_addm_AB #1#2#3#4\W\X\Y\Z #5#6#7#8%
350 {%
351   \XINT_addm_ABE #1#2{#8#7#6#5}{#3}#4\W\X\Y\Z
352 }%
353 \def\XINT_addm_ABE #1#2#3#4#5#6%
354 {%
355   \expandafter\XINT_addm_ABEA\the\numexpr #1+10#5#4#3#2+#6.%
356 }%
357 \def\XINT_addm_ABEA #1#2#3.#4%
358 {%
359   \XINT_addm_A #2{#3#4}%
360 }%
361 \def\XINT_addm_AC_checkcarry #1%
362 {%
363   \xint_gob_til_zero #1\XINT_addm_AC_nocarry 0\XINT_addm_C
364 }%
365 \def\XINT_addm_AC_nocarry 0\XINT_addm_C #1#2\W\X\Y\Z
366 {%
367   \expandafter
368   \xint_cleanupzeros_andstop
369   \romannumeral0%
370   \XINT_rord_main {}#2%
371   \xint_relax
372   \xint_bye\xint_bye\xint_bye\xint_bye
373   \xint_bye\xint_bye\xint_bye\xint_bye
374   \xint_relax
375   #1%
376 }%
377 \def\XINT_addm_C #1#2#3#4#5%
378 {%
379   \xint_gob_til_W
380   #5\XINT_addm_cw
381   #4\XINT_addm_cx
382   #3\XINT_addm_cy
383   #2\XINT_addm_cz
384   \W\XINT_addm_CD {#5#4#3#2}{#1}%
385 }%
```

```

386 \def\XINT_addm_CD #1%
387 {%
388   \expandafter\XINT_addm_CC\the\numexpr 1+10#1.%
389 }%
390 \def\XINT_addm_CC #1#2#3.#4%
391 {%
392   \XINT_addm_AC_checkcarry #2{#3#4}%
393 }%
394 \def\XINT_addm_cw
395   #1\XINT_addm_cx
396   #2\XINT_addm_cy
397   #3\XINT_addm_cz
398   \W\XINT_addm_CD
399 {%
400   \expandafter\XINT_addm_CDw\the\numexpr 1+#1#2#3.%
401 }%
402 \def\XINT_addm_CDw #1.#2#3\X\Y\Z
403 {%
404   \XINT_addm_end #1#3%
405 }%
406 \def\XINT_addm_cx
407   #1\XINT_addm_cy
408   #2\XINT_addm_cz
409   \W\XINT_addm_CD
410 {%
411   \expandafter\XINT_addm_CDx\the\numexpr 1+#1#2.%
412 }%
413 \def\XINT_addm_CDx #1.#2#3\Y\Z
414 {%
415   \XINT_addm_end #1#3%
416 }%
417 \def\XINT_addm_cy
418   #1\XINT_addm_cz
419   \W\XINT_addm_CD
420 {%
421   \expandafter\XINT_addm_CDy\the\numexpr 1+#1.%
422 }%
423 \def\XINT_addm_CDy #1.#2#3\Z
424 {%
425   \XINT_addm_end #1#3%
426 }%
427 \def\XINT_addm_cz\W\XINT_addm_CD #1#2#3{\XINT_addm_end #1#3}%
428 \edef\XINT_addm_end #1#2#3#4#5%
429   {\noexpand\expandafter\space\noexpand\the\numexpr #1#2#3#4#5\relax}%

```

### 7.11 \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.

```

430 \def\xintRaw {\romannumeral0\xintraw }%
431 \def\xintraw
432 {%

```

```

433 \expandafter\XINT_raw\romannumeral0\XINT_infrac
434 }%
435 \def\XINT_raw #1#2#3{ #2/#3[#1]}%

```

## 7.12 \xintPRaw

### 1.09b

```

436 \def\xintPRaw {\romannumeral0\xintpraw }%
437 \def\xintpraw
438 {%
439 \expandafter\XINT_praw\romannumeral0\XINT_infrac
440 }%
441 \def\XINT_praw #1%
442 {%
443 \ifnum #1=\xint_c_ \expandafter\XINT_praw_a\fi \XINT_praw_A {#1}%
444 }%
445 \def\XINT_praw_A #1#2#3%
446 {%
447 \if\XINT_isOne{#3}1\expandafter\xint_firstoftwo
448 \else\expandafter\xint_secondoftwo
449 \fi { #2[#1]}{ #2/#3[#1]}%
450 }%
451 \def\XINT_praw_a\XINT_praw_A #1#2#3%
452 {%
453 \if\XINT_isOne{#3}1\expandafter\xint_firstoftwo
454 \else\expandafter\xint_secondoftwo
455 \fi { #2}{ #2/#3}%
456 }%

```

## 7.13 \xintRawWithZeros

This was called \xintRaw in versions earlier than 1.07

```

457 \def\xintRawWithZeros {\romannumeral0\xinrawwithzeros }%
458 \def\xinrawwithzeros
459 {%
460 \expandafter\XINT_rawz\romannumeral0\XINT_infrac
461 }%
462 \def\XINT_rawz #1%
463 {%
464 \ifcase\XINT_cntSgn #1\Z
465 \expandafter\XINT_rawz_Ba
466 \or
467 \expandafter\XINT_rawz_A
468 \else
469 \expandafter\XINT_rawz_Ba
470 \fi
471 {#1}%
472 }%
473 \def\XINT_rawz_A #1#2#3{\xint_dsh {#2}{-#1}/#3}%
474 \def\XINT_rawz_Ba #1#2#3{\expandafter\XINT_rawz_Bb
475 \expandafter{\romannumeral0\xint_dsh {#3}{#1}}{#2}}%

```

```
476 \def\XINT_rawz_Bb #1#2{ #2/#1}%
```

### 7.14 `\xintFloor`, `\xintiFloor`

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

```
477 \def\xintFloor {\romannumeral0\xintfloor }%
478 \def\xintfloor #1% devrais-je faire \xintREZ?
479   {\expandafter\XINT_ifloor \romannumeral0\xintraawwithzeros {#1}./1[0]}%
480 \def\xintiFloor {\romannumeral0\xintifloor }%
481 \def\xintifloor #1%
482   {\expandafter\XINT_ifloor \romannumeral0\xintraawwithzeros {#1}.}%
483 \def\XINT_ifloor #1/#2.{\xintiipro {#1}{#2}}%
```

### 7.15 `\xintCeil`, `\xintiCeil`

1.09a

```
484 \def\xintCeil {\romannumeral0\xintceil }%
485 \def\xintceil #1{\xintiioopp {\xintFloor {\xintOpp{#1}}}}%
486 \def\xintiCeil {\romannumeral0\xinticeil }%
487 \def\xinticeil #1{\xintiioopp {\xintiFloor {\xintOpp{#1}}}}%
```

### 7.16 `\xintNumerator`

```
488 \def\xintNumerator {\romannumeral0\xintnumerator }%
489 \def\xintnumerator
490 {%
491   \expandafter\XINT_numer\romannumeral0\XINT_infrac
492 }%
493 \def\XINT_numer #1%
494 {%
495   \ifcase\XINT_cntSgn #1\Z
496     \expandafter\XINT_numer_B
497   \or
498     \expandafter\XINT_numer_A
499   \else
500     \expandafter\XINT_numer_B
501   \fi
502   {#1}%
503 }%
504 \def\XINT_numer_A #1#2#3{\xint_dsh {#2}{-#1}}%
505 \def\XINT_numer_B #1#2#3{ #2}%
```

### 7.17 `\xintDenominator`

```
506 \def\xintDenominator {\romannumeral0\xintdenominator }%
507 \def\xintdenominator
508 {%
509   \expandafter\XINT_denom\romannumeral0\XINT_infrac
510 }%
511 \def\XINT_denom #1%
512 {%
```

```

513 \ifcase\XINT_cntSgn #1\Z
514 \expandafter\XINT_denom_B
515 \or
516 \expandafter\XINT_denom_A
517 \else
518 \expandafter\XINT_denom_B
519 \fi
520 {#1}%
521 }%
522 \def\XINT_denom_A #1#2#3{ #3}%
523 \def\XINT_denom_B #1#2#3{\xint_dsh {#3}{#1}}%

```

## 7.18 \xintFrac

Useless typesetting macro.

```

524 \def\xintFrac {\romannumeral0\xintfrac }%
525 \def\xintfrac #1%
526 {%
527 \expandafter\XINT_fracfrac_A\romannumeral0\XINT_infrac {#1}%
528 }%
529 \def\XINT_fracfrac_A #1{\XINT_fracfrac_B #1\Z }%
530 \catcode\^=7
531 \def\XINT_fracfrac_B #1#2\Z
532 {%
533 \xint_gob_til_zero #1\XINT_fracfrac_C 0\XINT_fracfrac_D {10^{#1#2}}%
534 }%
535 \def\XINT_fracfrac_C 0\XINT_fracfrac_D #1#2#3%
536 {%
537 \if1\XINT_isOne {#3}%
538 \xint_afterfi {\expandafter\xint_firstoftwo_thenstop\xint_gobble_ii }%
539 \fi
540 \space
541 \frac {#2}{#3}%
542 }%
543 \def\XINT_fracfrac_D #1#2#3%
544 {%
545 \if1\XINT_isOne {#3}\XINT_fracfrac_E\fi
546 \space
547 \frac {#2}{#3}#1%
548 }%
549 \def\XINT_fracfrac_E \fi\space\frac #1#2{\fi \space #1\cdot }%

```

## 7.19 \xintSignedFrac

```

550 \def\xintSignedFrac {\romannumeral0\xintsignedfrac }%
551 \def\xintsignedfrac #1%
552 {%
553 \expandafter\XINT_sgnfrac_a\romannumeral0\XINT_infrac {#1}%
554 }%
555 \def\XINT_sgnfrac_a #1#2%
556 {%
557 \XINT_sgnfrac_b #2\Z {#1}%
558 }%

```

```

559 \def\XINT_sgnfrac_b #1%
560 {%
561   \xint_UDsignfork
562   #1\XINT_sgnfrac_N
563   -{\XINT_sgnfrac_P #1}%
564   \krof
565 }%
566 \def\XINT_sgnfrac_P #1\Z #2%
567 {%
568   \XINT_fracfrac_A {#2}{#1}%
569 }%
570 \def\XINT_sgnfrac_N
571 {%
572   \expandafter\xint_minus_thenstop\romannumeral0\XINT_sgnfrac_P
573 }%

```

## 7.20 \xintFwOver

```

574 \def\xintFwOver {\romannumeral0\xintfwover }%
575 \def\xintfwover #1%
576 {%
577   \expandafter\XINT_fwover_A\romannumeral0\XINT_infrac {#1}%
578 }%
579 \def\XINT_fwover_A #1{\XINT_fwover_B #1\Z }%
580 \def\XINT_fwover_B #1#2\Z
581 {%
582   \xint_gob_til_zero #1\XINT_fwover_C 0\XINT_fwover_D {10^{#1#2}}%
583 }%
584 \catcode`^=11
585 \def\XINT_fwover_C #1#2#3#4#5%
586 {%
587   \if0\XINT_isOne {#5}\xint_afterfi { {#4\over #5}}%
588   \else\xint_afterfi { #4}%
589   \fi
590 }%
591 \def\XINT_fwover_D #1#2#3%
592 {%
593   \if0\XINT_isOne {#3}\xint_afterfi { {#2\over #3}}%
594   \else\xint_afterfi { #2\cdot }%
595   \fi
596   #1%
597 }%

```

## 7.21 \xintSignedFwOver

```

598 \def\xintSignedFwOver {\romannumeral0\xintsignedfwover }%
599 \def\xintsignedfwover #1%
600 {%
601   \expandafter\XINT_sgnfwover_a\romannumeral0\XINT_infrac {#1}%
602 }%
603 \def\XINT_sgnfwover_a #1#2%
604 {%
605   \XINT_sgnfwover_b #2\Z {#1}%
606 }%
607 \def\XINT_sgnfwover_b #1%

```

```

608 {%
609   \xint_UDsignfork
610   #1\XINT_sgnfwover_N
611   -{\XINT_sgnfwover_P #1}%
612   \krof
613 }%
614 \def\XINT_sgnfwover_P #1\Z #2%
615 {%
616   \XINT_fwover_A {#2}{#1}%
617 }%
618 \def\XINT_sgnfwover_N
619 {%
620   \expandafter\xint_minus_thenstop\romannumeral0\XINT_sgnfwover_P
621 }%

```

## 7.22 \xintREZ

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

```

622 \def\xintREZ {\romannumeral0\xintrez }%
623 \def\xintrez
624 {%
625   \expandafter\XINT_rez_A\romannumeral0\XINT_infrac
626 }%
627 \def\XINT_rez_A #1#2%
628 {%
629   \XINT_rez_AB #2\Z {#1}%
630 }%
631 \def\XINT_rez_AB #1%
632 {%
633   \xint_UDzerominusfork
634   #1-\XINT_rez_zero
635   0#1\XINT_rez_neg
636   0-{\XINT_rez_B #1}%
637   \krof
638 }%
639 \def\XINT_rez_zero #1\Z #2#3{ 0/1[0]}%
640 \def\XINT_rez_neg {\expandafter\xint_minus_thenstop\romannumeral0\XINT_rez_B }%
641 \def\XINT_rez_B #1\Z
642 {%
643   \expandafter\XINT_rez_C\romannumeral0\XINT_factortens {#1}%
644 }%
645 \def\XINT_rez_C #1#2#3#4%
646 {%
647   \expandafter\XINT_rez_D\romannumeral0\XINT_factortens {#4}{#3}{#2}{#1}%
648 }%
649 \def\XINT_rez_D #1#2#3#4#5%
650 {%
651   \expandafter\XINT_rez_E\expandafter
652   {\the\numexpr #3+#4-#2}{#1}{#5}%
653 }%
654 \def\XINT_rez_E #1#2#3{ #3/#2[#1]}%

```

## 7.23 `\xintE`

1.07: The fraction is the first argument contrarily to `\xintTrunc` and `\xintRound`.  
1.1 modifies and moves `\xintiiE` to `xint.sty`.

```
655 \def\xintE {\romannumeral0\xinte }%
656 \def\xinte #1%
657 {%
658   \expandafter\XINT_e \romannumeral0\XINT_infrac {#1}%
659 }%
660 \def\XINT_e #1#2#3#4%
661 {%
662   \expandafter\XINT_e_end\the\numexpr #1+#4.{#2}{#3}%
663 }%
664 \def\XINT_e_end #1.#2#3{ #2/#3[#1]}%
```

## 7.24 `\xintIrr`

```
665 \def\xintIrr {\romannumeral0\xintirr }%
666 \def\xintirr #1%
667 {%
668   \expandafter\XINT_irr_start\romannumeral0\xintraawithzeros {#1}\Z
669 }%
670 \def\XINT_irr_start #1#2/#3\Z
671 {%
672   \if0\XINT_isOne {#3}%
673     \xint_afterfi
674       {\xint_UDsignfork
675         #1\XINT_irr_negative
676         -{\XINT_irr_nonneg #1}%
677         \krof}%
678   \else
679     \xint_afterfi{\XINT_irr_denomisine #1}%
680     \fi
681     #2\Z {#3}%
682 }%
683 \def\XINT_irr_denomisine #1\Z #2{ #1/1}% changed in 1.08
684 \def\XINT_irr_negative #1\Z #2{\XINT_irr_D #1\Z #2\Z \xint_minus_thenstop}%
685 \def\XINT_irr_nonneg #1\Z #2{\XINT_irr_D #1\Z #2\Z \space}%
686 \def\XINT_irr_D #1#2\Z #3#4\Z
687 {%
688   \xint_UDzerosfork
689     #3#1\XINT_irr_indeterminate
690     #30\XINT_irr_divisionbyzero
691     #10\XINT_irr_zero
692     00\XINT_irr_loop_a
693   \krof
694   {#3#4}{#1#2}{#3#4}{#1#2}%
695 }%
696 \def\XINT_irr_indeterminate #1#2#3#4#5{\xintError:NaN\space 0/0}%
697 \def\XINT_irr_divisionbyzero #1#2#3#4#5{\xintError:DivisionByZero #5#2/0}%
698 \def\XINT_irr_zero #1#2#3#4#5{ 0/1}% changed in 1.08
699 \def\XINT_irr_loop_a #1#2%
```



```

700 {%
701   \expandafter\XINT_irr_loop_d
702   \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
703 }%
704 \def\XINT_irr_loop_d #1#2%
705 {%
706   \XINT_irr_loop_e #2\Z
707 }%
708 \def\XINT_irr_loop_e #1#2\Z
709 {%
710   \xint_gob_til_zero #1\XINT_irr_loop_exit0\XINT_irr_loop_a {#1#2}%
711 }%
712 \def\XINT_irr_loop_exit0\XINT_irr_loop_a #1#2#3#4%
713 {%
714   \expandafter\XINT_irr_loop_exitb\expandafter
715   {\romannumeral0\xintiique {#3}{#2}}%
716   {\romannumeral0\xintiique {#4}{#2}}%
717 }%
718 \def\XINT_irr_loop_exitb #1#2%
719 {%
720   \expandafter\XINT_irr_finish\expandafter {#2}{#1}%
721 }%
722 \def\XINT_irr_finish #1#2#3{#3#1/#2}% changed in 1.08

```

### 7.25 \xintifInt

```

723 \def\xintifInt {\romannumeral0\xintifint }%
724 \def\xintifint #1{\expandafter\XINT_ifint\romannumeral0\xintraewithzeros {#1}.}%
725 \def\XINT_ifint #1/#2.%
726 {%
727   \if 0\xintiiRem {#1}{#2}%
728   \expandafter\xint_firstoftwo_thenstop
729   \else
730   \expandafter\xint_secondoftwo_thenstop
731   \fi
732 }%

```

### 7.26 \xintJrr

```

733 \def\xintJrr {\romannumeral0\xintjrr }%
734 \def\xintjrr #1%
735 {%
736   \expandafter\XINT_jrr_start\romannumeral0\xintraewithzeros {#1}\Z
737 }%
738 \def\XINT_jrr_start #1#2/#3\Z
739 {%
740   \if0\XINT_isOne {#3}\xint_afterfi
741     {\xint_UDsignfork
742       #1\XINT_jrr_negative
743       -{\XINT_jrr_nonneg #1}%
744     \krof}%
745   \else
746     \xint_afterfi{\XINT_jrr_denomisone #1}%
747   \fi
748   #2\Z {#3}%

```

```

749 }%
750 \def\XINT_jrr_denomisine #1\Z #2{ #1/1}% changed in 1.08
751 \def\XINT_jrr_negative #1\Z #2{\XINT_jrr_D #1\Z #2\Z \xint_minus_thenstop }%
752 \def\XINT_jrr_nonneg #1\Z #2{\XINT_jrr_D #1\Z #2\Z \space}%
753 \def\XINT_jrr_D #1#2\Z #3#4\Z
754 {%
755     \xint_UDzerosfork
756     #3#1\XINT_jrr_indeterminate
757     #30\XINT_jrr_divisionbyzero
758     #10\XINT_jrr_zero
759     00\XINT_jrr_loop_a
760     \krof
761     {#3#4}{#1#2}1001%
762 }%
763 \def\XINT_jrr_indeterminate #1#2#3#4#5#6#7{\xintError:NaN\space 0/0}%
764 \def\XINT_jrr_divisionbyzero #1#2#3#4#5#6#7{\xintError:DivisionByZero #7#2/0}%
765 \def\XINT_jrr_zero #1#2#3#4#5#6#7{ 0/1}% changed in 1.08
766 \def\XINT_jrr_loop_a #1#2%
767 {%
768     \expandafter\XINT_jrr_loop_b
769     \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
770 }%
771 \def\XINT_jrr_loop_b #1#2#3#4#5#6#7%
772 {%
773     \expandafter \XINT_jrr_loop_c \expandafter
774     {\romannumeral0\xintiiadd{\XINT_mul_fork #4\Z #1\Z}{#6}}%
775     {\romannumeral0\xintiiadd{\XINT_mul_fork #5\Z #1\Z}{#7}}%
776     {#2}{#3}{#4}{#5}%
777 }%
778 \def\XINT_jrr_loop_c #1#2%
779 {%
780     \expandafter \XINT_jrr_loop_d \expandafter{#2}{#1}%
781 }%
782 \def\XINT_jrr_loop_d #1#2#3#4%
783 {%
784     \XINT_jrr_loop_e #3\Z {#4}{#2}{#1}%
785 }%
786 \def\XINT_jrr_loop_e #1#2\Z
787 {%
788     \xint_gob_til_zero #1\XINT_jrr_loop_exit0\XINT_jrr_loop_a {#1#2}%
789 }%
790 \def\XINT_jrr_loop_exit0\XINT_jrr_loop_a #1#2#3#4#5#6%
791 {%
792     \XINT_irr_finish {#3}{#4}%
793 }%

```

## 7.27 \xintTFrac

1.09i, for frac in \xintexpr. And \xintFrac is already assigned. T for truncation. However, potentially not very efficient with numbers in scientific notations, with big exponents. Will have to think it again some day. I hesitated how to call the macro. Same convention as in maple, but some people reserve fractional part to  $x - \text{floor}(x)$ . 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.

```

794 \def\xintTFrac {\romannumeral0\xinttfrac }%
795 \def\xinttfrac #1{\expandafter\XINT_tfrac_fork\romannumeral0\xintraewithzeros {#1}\Z }%
796 \def\XINT_tfrac_fork #1%
797 {%
798   \xint_UDzerominusfork
799   #1-\XINT_tfrac_zero
800   0#1{\xinttiopp\XINT_tfrac_P }%
801   0-{\XINT_tfrac_P #1}%
802   \krof
803 }%
804 \def\XINT_tfrac_zero #1\Z { 0/1[0]}%
805 \def\XINT_tfrac_P #1/#2\Z {\expandafter\XINT_rez_AB
806   \romannumeral0\xintiirem{#1}{#2}\Z {0}{#2}}%

```

## 7.28 \xintTrunc, \xintiTrunc

```

807 \def\xintTrunc {\romannumeral0\xinttrunc }%
808 \def\xintiTrunc {\romannumeral0\xintitrunc }%
809 \def\xinttrunc #1%
810 {%
811   \expandafter\XINT_trunc\expandafter {\the\numexpr #1}%
812 }%
813 \def\XINT_trunc #1#2%
814 {%
815   \expandafter\XINT_trunc_G
816   \romannumeral0\expandafter\XINT_trunc_A
817   \romannumeral0\XINT_infrac {#2}{#1}{#1}%
818 }%
819 \def\xintitrunc #1%
820 {%
821   \expandafter\XINT_itrunc\expandafter {\the\numexpr #1}%
822 }%
823 \def\XINT_itrunc #1#2%
824 {%
825   \expandafter\XINT_itrunc_G
826   \romannumeral0\expandafter\XINT_trunc_A
827   \romannumeral0\XINT_infrac {#2}{#1}{#1}%
828 }%
829 \def\XINT_trunc_A #1#2#3#4%
830 {%
831   \expandafter\XINT_trunc_checkifzero
832   \expandafter{\the\numexpr #1+#4}#2\Z {#3}%
833 }%
834 \def\XINT_trunc_checkifzero #1#2#3\Z
835 {%
836   \xint_gob_til_zero #2\XINT_trunc_iszero0\XINT_trunc_B {#1}{#2#3}%
837 }%
838 \def\XINT_trunc_iszero0\XINT_trunc_B #1#2#3{ 0\Z 0}%
839 \def\XINT_trunc_B #1%
840 {%
841   \ifcase\XINT_cntSgn #1\Z

```

## 7 Package *xintfrac* implementation

```

842     \expandafter\XINT_trunc_D
843   \or
844     \expandafter\XINT_trunc_D
845   \else
846     \expandafter\XINT_trunc_C
847   \fi
848   {#1}%
849 }%
850 \def\XINT_trunc_C #1#2#3%
851 {%
852   \expandafter\XINT_trunc_CE\expandafter
853   {\romannumeral0\XINT_dsx_zeroloop -#1.{}\Z {#3}}{#2}%
854 }%
855 \def\XINT_trunc_CE #1#2{\XINT_trunc_E #2.{#1}}%
856 \def\XINT_trunc_D #1#2%
857 {%
858   \expandafter\XINT_trunc_E
859   \romannumeral0\XINT_dsx_zeroloop #1.{}\Z {#2}.%
860 }%
861 \def\XINT_trunc_E #1%
862 {%
863   \xint_UDsignfork
864     #1\XINT_trunc_Fneg
865     -{\XINT_trunc_Fpos #1}%
866   \krof
867 }%
868 \def\XINT_trunc_Fneg #1.#2{\expandafter\xint_firstoftwo_thenstop
869   \romannumeral0\XINT_div_prepare {#2}{#1}\Z \xint_minus_thenstop}%
870 \def\XINT_trunc_Fpos #1.#2{\expandafter\xint_firstoftwo_thenstop
871   \romannumeral0\XINT_div_prepare {#2}{#1}\Z \space }%
872 \def\XINT_itrunc_G #1#2\Z #3#4%
873 {%
874   \xint_gob_til_zero #1\XINT_trunc_zero 0#3#1#2%
875 }%
876 \def\XINT_trunc_zero 0#1#20{ 0}%
877 \def\XINT_trunc_G #1\Z #2#3%
878 {%
879   \xint_gob_til_zero #2\XINT_trunc_zero 0%
880   \expandafter\XINT_trunc_H\expandafter
881   {\the\numexpr\romannumeral0\xintlength {#1}-#3}{#3}{#1}#2%
882 }%
883 \def\XINT_trunc_H #1#2%
884 {%
885   \ifnum #1 > \xint_c_
886     \xint_afterfi {\XINT_trunc_Ha {#2}}%
887   \else
888     \xint_afterfi {\XINT_trunc_Hb {-#1}}% -0,--1,--2, ....
889   \fi
890 }%
891 \def\XINT_trunc_Ha
892 {%
893   \expandafter\XINT_trunc_Haa\romannumeral0\xintdecsplit

```

```

894 }%
895 \def\XINT_trunc_Haa #1#2#3%
896 {%
897     #3#1.#2%
898 }%
899 \def\XINT_trunc_Hb #1#2#3%
900 {%
901     \expandafter #3\expandafter0\expandafter.%
902     \romannumeral0\XINT_dsx_zeroloop #1.{ }\Z {}#2% #1=-0 autoris\ 'e !
903 }%

```

## 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.

```

904 \def\xintTTrunc {\romannumeral0\xintttrunc }%
905 \def\xintttrunc #1%
906 {%
907     \expandafter\XINT_itrunc_G
908     \romannumeral0\expandafter\XINT_ttrunc_A
909     \romannumeral0\XINT_infrac {#1}0% this last 0 to let \XINT_itrunc_G be happy
910 }%
911 \def\XINT_ttrunc_A #1#2#3{\XINT_trunc_checkifzero {#1}#2\Z {#3}}%

```

## 7.30 \xintNum

```

912 \let\xintNum \xintTTrunc
913 \let\xintnum \xintttrunc

```

## 7.31 \xintRound, \xintiRound

```

914 \def\xintRound {\romannumeral0\xintround }%
915 \def\xintiRound {\romannumeral0\xintiround }%
916 \def\xintround #1%
917 {%
918     \expandafter\XINT_round\expandafter {\the\numexpr #1}%
919 }%
920 \def\XINT_round
921 {%
922     \expandafter\XINT_trunc_G\romannumeral0\XINT_round_A
923 }%
924 \def\xintiround #1%
925 {%
926     \expandafter\XINT_iround\expandafter {\the\numexpr #1}%
927 }%
928 \def\XINT_iround
929 {%
930     \expandafter\XINT_itrunc_G\romannumeral0\XINT_round_A
931 }%
932 \def\XINT_round_A #1#2%
933 {%
934     \expandafter\XINT_round_B
935     \romannumeral0\expandafter\XINT_trunc_A

```

```

936   \romannumeral0\XINT_infrac {#2}{#1+\xint_c_i}{#1}%
937 }%
938 \def\XINT_round_B #1\Z
939 {%
940   \expandafter\XINT_round_C
941   \romannumeral0\XINT_rord_main {}#1%
942   \xint_relax
943   \xint_bye\xint_bye\xint_bye\xint_bye
944   \xint_bye\xint_bye\xint_bye\xint_bye
945   \xint_relax
946   \Z
947 }%
948 \def\XINT_round_C #1%
949 {%
950   \ifnum #1<\xint_c_v
951     \expandafter\XINT_round_Daa
952   \else
953     \expandafter\XINT_round_Dba
954   \fi
955 }%
956 \def\XINT_round_Daa #1%
957 {%
958   \xint_gob_til_Z #1\XINT_round_Daz\Z \XINT_round_Da #1%
959 }%
960 \def\XINT_round_Daz\Z \XINT_round_Da \Z { 0\Z }%
961 \def\XINT_round_Da #1\Z
962 {%
963   \XINT_rord_main {}#1%
964   \xint_relax
965   \xint_bye\xint_bye\xint_bye\xint_bye
966   \xint_bye\xint_bye\xint_bye\xint_bye
967   \xint_relax \Z
968 }%
969 \def\XINT_round_Dba #1%
970 {%
971   \xint_gob_til_Z #1\XINT_round_Dbz\Z \XINT_round_Db #1%
972 }%
973 \def\XINT_round_Dbz\Z \XINT_round_Db \Z { 1\Z }%
974 \def\XINT_round_Db #1\Z
975 {%
976   \XINT_addm_A 0{}1000\W\X\Y\Z #1000\W\X\Y\Z \Z
977 }%

```

### 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`).

[2015/10/04] Although the explanations above are extremely clear, there are just too complicated for me to be now able to understand them fully. I miraculously managed to do the minimal changes (all happens between `\XINT_xtrunc_Q` and `\XINT_xtrunc_Pa`) in order for `\xintXTrunc` to use the 1.2 division routine. Seems to work. But some thought should be given to how to adapt `\xintXTrunc` for it to better use the abilities and characteristics of the new division routines in `xincore`.

```

978 \def\xintXTrunc #1#2%
979 {%
980   \expandafter\XINT_xtrunc_a\expandafter
981   {\the\numexpr #1\expandafter}\romannumeral0\xintra #2}%
982 }%
983 \def\XINT_xtrunc_a #1%
984 {%
985   \expandafter\XINT_xtrunc_b\expandafter
986   {\the\numexpr #1<\xint_c_i \xint_c_i-\fi #1}%
987 }%
988 \def\XINT_xtrunc_b #1%
989 {%
990   \expandafter\XINT_xtrunc_c\expandafter
991   {\the\numexpr (#1+\xint_c_ii^v)/\xint_c_ii^vi-\xint_c_i}\{#1}%
992 }%
993 \def\XINT_xtrunc_c #1#2%
994 {%
995   \expandafter\XINT_xtrunc_d\expandafter
996   {\the\numexpr #2-\xint_c_ii^vi*#1}\{#1}\{#2}%
997 }%
998 \def\XINT_xtrunc_d #1#2#3#4/#5[#6]%
999 {%
1000   \XINT_xtrunc_e #4.\{#6}\{#5}\{#3}\{#2}\{#1}%
1001 }%

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

1002 \def\XINT_xtrunc_e #1%
1003 {%

```

[illegible]



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

## 7 Package `xintfrac` implementation

[illegible]

## 7 Package `xintfrac` implementation

```

\expandafter\XINT_xtrunc_prepare_e\xint_gob_til_dot #1#2#3#4#5#6#7#8#9!%
1155 }%
1156 \def\XINT_xtrunc_prepare_e #1!#2!#3#4%
1157 {%
1158     \XINT_xtrunc_prepare_f #4#3\X {#1}{#3}%
1159 }%
1160 \def\XINT_xtrunc_prepare_f #1#2#3#4#5#6#7#8#9\X
1161 {%
1162     \expandafter\space\expandafter\XINT_div_prepare_g
1163         \the\numexpr #1#2#3#4#5#6#7#8+\xint_c_i\expandafter
1164         .\the\numexpr (#1#2#3#4#5#6#7#8+\xint_c_i)/\xint_c_ii\expandafter
1165         .\the\numexpr #1#2#3#4#5#6#7#8\expandafter
1166         .\romannumeral0\XINT_sepandrev_andcount
1167         #1#2#3#4#5#6#7#8#9\XINT_rsepybviii_end_A 2345678%
1168             \XINT_rsepybviii_end_B 2345678%
1169     \relax\xint_c_ii\xint_c_iii
1170         \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
1171         \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_W
1172     \X
1173 }%
1174 \def\XINT_xtrunc_Pa #1#2%
1175 {%
1176     \expandafter\XINT_xtrunc_Pb\romannumeral0#1{#2}{#1}%
1177 }%
1178 \def\XINT_xtrunc_Pb #1#2#3#4{#1.\XINT_xtrunc_A {#4}{#2}{#3}}%
1179 \def\XINT_xtrunc_A #1%
1180 {%
1181     \unless\ifnum #1>\xint_c_ \XINT_xtrunc_transition\fi
1182     \expandafter\XINT_xtrunc_B\expandafter{\the\numexpr #1-\xint_c_i}%
1183 }%
1184 \def\XINT_xtrunc_B #1#2#3%
1185 {%
1186     \expandafter\XINT_xtrunc_D\romannumeral0#3%
1187     #{20000000000000000000000000000000000000000000000000000000000000}
1188     {#1}{#3}%
1189 }%
1190 \def\XINT_xtrunc_D #1#2#3%
1191 {%
1192     \romannumeral0\expandafter\XINT_dsx_zero loop
1193         \the\numexpr \xint_c_ii^vi-\xintLength{#1}.{}Z {}#1%
1194     \XINT_xtrunc_A {#3}{#2}%
1195 }%
1196 \def\XINT_xtrunc_transition\fi
1197     \expandafter\XINT_xtrunc_B\expandafter #1#2#3#4%
1198 {%
1199     \fi
1200     \ifnum #4=\xint_c_ \XINT_xtrunc_abort\fi
1201     \expandafter\XINT_xtrunc_x\expandafter
1202     {\romannumeral0\XINT_dsx_zero loop #4.{}Z {#2}}{#3}{#4}%
1203 }%
1204 \def\XINT_xtrunc_x #1#2%
1205 {%

```

```

1206 \expandafter\XINT_xtrunc_y\romannumeral0#2{#1}%
1207 }%
1208 \def\XINT_xtrunc_y #1#2#3%
1209 {%
1210 \romannumeral0\expandafter\XINT_dsx_zeroloop
1211 \the\numexpr #3-\xintLength{#1}.{}Z {}#1%
1212 }%
1213 \def\XINT_xtrunc_abort\fi\expandafter\XINT_xtrunc_x\expandafter #1#2#3{\fi}%

```

### 7.33 `\xintDigits`

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

```

1214 \mathchardef\XINTdigits 16
1215 \def\xintDigits #1#2%
1216 {\afterassignment \xint_gobble_i \mathchardef\XINTdigits=}%
1217 \def\xinttheDigits {\number\XINTdigits }%

```

### 7.34 `\xintAdd`

```

1218 \def\xintAdd {\romannumeral0\xintadd }%
1219 \def\xintadd #1{\expandafter\XINT_fadd\romannumeral0\xintra {#1}}%
1220 \def\XINT_fadd #1{\xint_gob_til_zero #1\XINT_fadd_Azero 0\XINT_fadd_a #1}%
1221 \def\XINT_fadd_Azero #1]{\xintra }%
1222 \def\XINT_fadd_a #1/#2[#3]#4%
1223 {\expandafter\XINT_fadd_b\romannumeral0\xintra {#4}{#3}{#1}{#2}}%
1224 \def\XINT_fadd_b #1{\xint_gob_til_zero #1\XINT_fadd_Bzero 0\XINT_fadd_c #1}%
1225 \def\XINT_fadd_Bzero #1]#2#3#4{ #3/#4[#2]}%
1226 \def\XINT_fadd_c #1/#2[#3]#4%
1227 {%
1228 \expandafter\XINT_fadd_Aa\expandafter{\the\numexpr #4-#3}{#3}{#4}{#1}{#2}%
1229 }%
1230 \def\XINT_fadd_Aa #1%
1231 {%
1232 \ifcase\XINT_cntSgn #1\Z
1233 \expandafter\XINT_fadd_B
1234 \or
1235 \expandafter \XINT_fadd_Ba
1236 \else
1237 \expandafter \XINT_fadd_Bb
1238 \fi {#1}%
1239 }%
1240 \def\XINT_fadd_B #1#2#3#4#5#6#7{\XINT_fadd_C {#4}{#5}{#7}{#6}[#3]}%
1241 \def\XINT_fadd_Ba #1#2#3#4#5#6#7%
1242 {%
1243 \expandafter\XINT_fadd_C\expandafter
1244 {\romannumeral0\XINT_dsx_zeroloop #1.{}Z {}#6}}%
1245 {#7}{#5}{#4}[#2]}%
1246 }%
1247 \def\XINT_fadd_Bb #1#2#3#4#5#6#7%
1248 {%
1249 \expandafter\XINT_fadd_C\expandafter

```

## 7 Package *xintfrac* implementation

```

1250      {\romannumeral0\XINT_dsx_zeroloop -#1.{}\Z {#4}}%
1251      {#5}{#7}{#6}[#3]%
1252 }%
1253 \def\XINT_fadd_C #1#2#3%
1254 {%
1255   \ifcase\romannumeral0\xintiicmp {#2}{#3} %<- intentional space here.
1256     \expandafter\XINT_fadd_eq
1257   \or\expandafter\XINT_fadd_D
1258   \else\expandafter\XINT_fadd_Da
1259   \fi {#2}{#3}{#1}%
1260 }%
1261 \def\XINT_fadd_eq #1#2#3#4#5%
1262 {%
1263   \expandafter\XINT_fadd_G
1264   \romannumeral0\xintiiadd {#3}{#4}/#1[#5]%
1265 }%
1266 \def\XINT_fadd_D #1#2%
1267 {%
1268   \expandafter\XINT_fadd_E\romannumeral0\XINT_div_prepare {#2}{#1}{#1}{#2}%
1269 }%
1270 \def\XINT_fadd_E #1#2%
1271 {%
1272   \if0\XINT_Sgn #2\Z
1273     \expandafter\XINT_fadd_F
1274   \else\expandafter\XINT_fadd_K
1275   \fi {#1}%
1276 }%
1277 \def\XINT_fadd_F #1#2#3#4#5#6%
1278 {%
1279   \expandafter\XINT_fadd_G
1280   \romannumeral0\xintiiadd {\xintiiMul {#5}{#1}}{#4}/#2[#6]%
1281 }%
1282 \def\XINT_fadd_Da #1#2%
1283 {%
1284   \expandafter\XINT_fadd_Ea\romannumeral0\XINT_div_prepare {#1}{#2}{#1}{#2}%
1285 }%
1286 \def\XINT_fadd_Ea #1#2%
1287 {%
1288   \if0\XINT_Sgn #2\Z
1289     \expandafter\XINT_fadd_Fa
1290   \else\expandafter\XINT_fadd_K
1291   \fi {#1}%
1292 }%
1293 \def\XINT_fadd_Fa #1#2#3#4#5#6%
1294 {%
1295   \expandafter\XINT_fadd_G
1296   \romannumeral0\xintiiadd {\xintiiMul {#4}{#1}}{#5}/#3[#6]%
1297 }%
1298 \def\XINT_fadd_G #1{\if0#1\XINT_fadd_iszero\fi\space #1}%
1299 \def\XINT_fadd_K #1#2#3#4#5%
1300 {%
1301   \expandafter\XINT_fadd_L

```

```

1302 \romannumeral0\xintiiadd {\xintiiMul {#2}{#5}}{\xintiiMul {#3}{#4}}.%
1303 {{#2}{#3}}}%
1304 }%
1305 \def\xINT_fadd_L #1{\if0#1\xINT_fadd_iszero\fi \xINT_fadd_M #1}%
1306 \def\xINT_fadd_M #1.#2{\expandafter\xINT_fadd_N \expandafter
1307     {\romannumeral0\xintiimul #2}{#1}}%
1308 \def\xINT_fadd_N #1#2{ #2/#1}%
1309 \edef\xINT_fadd_iszero\fi #1[#2]{\noexpand\fi\space 0/1[0]}% ou [#2] originel?

```

### 7.35 \xintSub

```

1310 \def\xintSub {\romannumeral0\xintsub }%
1311 \def\xintsub #1{\expandafter\xINT_fsub\romannumeral0\xintra {#1}}%
1312 \def\xINT_fsub #1{\xint_gob_til_zero #1\xINT_fsub_Azero 0\xINT_fsub_a #1}%
1313 \def\xINT_fsub_Azero #1{\xintopp }%
1314 \def\xINT_fsub_a #1/#2[#3]#4%
1315     {\expandafter\xINT_fsub_b\romannumeral0\xintra {#4}{#3}{#1}{#2}}%
1316 \def\xINT_fsub_b #1{\xint_UDzerominusfork
1317     #1-\xINT_fadd_Bzero
1318     0#1\xINT_fadd_c
1319     0-{\xINT_fadd_c -#1}%
1320     \krof }%

```

### 7.36 \xintSum

```

1321 \def\xintSum {\romannumeral0\xintsum }%
1322 \def\xintsum #1{\xintsumexpr #1\relax }%
1323 \def\xintSumExpr {\romannumeral0\xintsumexpr }%
1324 \def\xintsumexpr {\expandafter\xINT_fsumexpr\romannumeral`&&@}%
1325 \def\xINT_fsumexpr {\xINT_fsum_loop_a {0/1[0]}}%
1326 \def\xINT_fsum_loop_a #1#2%
1327 {%
1328     \expandafter\xINT_fsum_loop_b \romannumeral`&&@#2\Z {#1}%
1329 }%
1330 \def\xINT_fsum_loop_b #1%
1331 {%
1332     \xint_gob_til_relax #1\xINT_fsum_finished\relax
1333     \xINT_fsum_loop_c #1%
1334 }%
1335 \def\xINT_fsum_loop_c #1\Z #2%
1336 {%
1337     \expandafter\xINT_fsum_loop_a\expandafter{\romannumeral0\xintadd {#2}{#1}}%
1338 }%
1339 \def\xINT_fsum_finished #1\Z #2{ #2}%

```

### 7.37 \xintMul

```

1340 \def\xintMul {\romannumeral0\xintmul }%
1341 \def\xintmul #1{\expandafter\xINT_fmula\romannumeral0\xintra {#1}.}%
1342 \def\xINT_fmula #1{\xint_gob_til_zero #1\xINT_fmula_zero 0\xINT_fmula_a #1}%
1343 \def\xINT_fmula_a #1[#2].#3%
1344     {\expandafter\xINT_fmula_b\romannumeral0\xintra {#3}#1[#2.]}%
1345 \def\xINT_fmula_b #1{\xint_gob_til_zero #1\xINT_fmula_zero 0\xINT_fmula_c #1}%
1346 \def\xINT_fmula_c #1/#2[#3]#4/#5[#6.]%
1347 {%

```

```

1348 \expandafter\XINT_fmud_d
1349 \expandafter{\the\numexpr #3+#6\expandafter}%
1350 \expandafter{\romannumeral0\xintiimul {#5}{#2}}%
1351 {\romannumeral0\xintiimul {#4}{#1}}%
1352 }%
1353 \def\XINT_fmud_d #1#2#3%
1354 {%
1355 \expandafter \XINT_fmud_e \expandafter{#3}{#1}{#2}%
1356 }%
1357 \def\XINT_fmud_e #1#2{\XINT_outfrac {#2}{#1}}%
1358 \def\XINT_fmud_zero #1.#2{ 0/1[0]}%

```

### 7.38 \xintSqr

#### 1.1 modifs comme xintMul.

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

```

1359 \def\xintSqr {\romannumeral0\xintsqr }%
1360 \def\xintsqr #1{\expandafter\XINT_fsqr\romannumeral0\xintraw {#1}}%
1361 \def\XINT_fsqr #1{\xint_gob_til_zero #1\XINT_fsqr_zero 0\XINT_fsqr_a #1}%
1362 \def\XINT_fsqr_a #1/#2[#3]%
1363 {%
1364 \expandafter\XINT_fsqr_b
1365 \expandafter{\the\numexpr #3+#3\expandafter}%
1366 \expandafter{\romannumeral0\xintiisqr {#2}}%
1367 {\romannumeral0\xintiisqr {#1}}%
1368 }%
1369 \def\XINT_fsqr_b #1#2#3{\expandafter \XINT_fmud_e \expandafter{#3}{#1}{#2}}%
1370 \def\XINT_fsqr_zero #1{\ 0/1[0]}%

```

### 7.39 \xintPow

1.2f: to be coherent with the "i" convention \xintiPow should parse also its exponent via \xintNum when xintfrac.sty is loaded. This was not the case so far. Cependant le problème est que le fait d'appliquer \xintNum rend impossible certains inputs qui auraient pu être gérés par \numexpr. Le \numexpr externe est ici pour intercepter trop grand input.

```

1371 \def\xintipow #1#2%
1372 {%
1373 \expandafter\xint_pow\the\numexpr \xintNum{#2}\expandafter.\romannumeral0\xintnum{#1}\Z%
1374 }%
1375 \def\xintPow {\romannumeral0\xintpow }%
1376 \def\xintpow #1%
1377 {%
1378 \expandafter\XINT_fpow\expandafter {\romannumeral0\XINT_infrac {#1}}%
1379 }%
1380 \def\XINT_fpow #1#2%
1381 {%
1382 \expandafter\XINT_fpow_fork\the\numexpr \xintNum{#2}\relax\Z #1%
1383 }%
1384 \def\XINT_fpow_fork #1#2\Z

```

```

1385 {%
1386   \xint_UDzerominusfork
1387   #1-\XINT_fpow_zero
1388   0#1\XINT_fpow_neg
1389   0-\XINT_fpow_pos #1}%
1390   \krof
1391   {#2}%
1392 }%
1393 \def\XINT_fpow_zero #1#2#3#4{ 1/1[0]}%
1394 \def\XINT_fpow_pos #1#2#3#4#5%
1395 {%
1396   \expandafter\XINT_fpow_pos_A\expandafter
1397   {\the\numexpr #1#2*#3\expandafter}\expandafter
1398   {\romannumeral0\xintiipow {#5}{#1#2}}%
1399   {\romannumeral0\xintiipow {#4}{#1#2}}%
1400 }%
1401 \def\XINT_fpow_neg #1#2#3#4%
1402 {%
1403   \expandafter\XINT_fpow_pos_A\expandafter
1404   {\the\numexpr -#1*#2\expandafter}\expandafter
1405   {\romannumeral0\xintiipow {#3}{#1}}%
1406   {\romannumeral0\xintiipow {#4}{#1}}%
1407 }%
1408 \def\XINT_fpow_pos_A #1#2#3%
1409 {%
1410   \expandafter\XINT_fpow_pos_B\expandafter {#3}{#1}{#2}%
1411 }%
1412 \def\XINT_fpow_pos_B #1#2{\XINT_outfrac {#2}{#1}}%

```

## 7.40 \xintiFac

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

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

```

1413 \def\xintifac #1{\expandafter\XINT_fac_fork\the\numexpr \xintNum{#1}.}%

```

## 7.41 \xintiBinomial

### 1.2f. Binomial coefficients.

```

1414 \def\xintibinomial #1#2%
1415 {%
1416   \expandafter\XINT_binom_pre
1417   \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.%
1418 }%

```



## 7.42 `\xintiPFactorial`

### 1.2f. Partial factorial.

```

1419 \def\xintipfactorial #1#2%
1420 {%
1421     \expandafter\XINT_pfac_fork
1422     \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.%
1423 }%
```

## 7.43 `\xintPrd`

```

1424 \def\xintPrd {\romannumeral0\xintprd }%
1425 \def\xintprd #1{\xintprdexpr #1\relax }%
1426 \def\xintPrdExpr {\romannumeral0\xintprdexpr }%
1427 \def\xintprdexpr {\expandafter\XINT_fprdexpr \romannumeral`&&@}%
1428 \def\XINT_fprdexpr {\XINT_fprod_loop_a {1/1[0]}}%
1429 \def\XINT_fprod_loop_a #1#2%
1430 {%
1431     \expandafter\XINT_fprod_loop_b \romannumeral`&&@#2\Z {#1}%
1432 }%
1433 \def\XINT_fprod_loop_b #1%
1434 {%
1435     \xint_gob_til_relax #1\XINT_fprod_finished\relax
1436     \XINT_fprod_loop_c #1%
1437 }%
1438 \def\XINT_fprod_loop_c #1\Z #2%
1439 {%
1440     \expandafter\XINT_fprod_loop_a\expandafter{\romannumeral0\xintmul {#1}{#2}}%
1441 }%
1442 \def\XINT_fprod_finished #1\Z #2{ #2}%
```

## 7.44 `\xintDiv`

```

1443 \def\xintDiv {\romannumeral0\xintdiv }%
1444 \def\xintdiv #1%
1445 {%
1446     \expandafter\XINT_fdiv\expandafter {\romannumeral0\XINT_infrac {#1}}%
1447 }%
1448 \def\XINT_fdiv #1#2%
1449     {\expandafter\XINT_fdiv_A\romannumeral0\XINT_infrac {#2}{#1}}%
1450 \def\XINT_fdiv_A #1#2#3#4#5#6%
1451 {%
1452     \expandafter\XINT_fdiv_B
1453     \expandafter{\the\numexpr #4-#1\expandafter}%
1454     \expandafter{\romannumeral0\xintiimul {#2}{#6}}%
1455     {\romannumeral0\xintiimul {#3}{#5}}%
1456 }%
1457 \def\XINT_fdiv_B #1#2#3%
1458 {%
1459     \expandafter\XINT_fdiv_C
1460     \expandafter{#3}{#1}{#2}%
1461 }%
1462 \def\XINT_fdiv_C #1#2{\XINT_outfrac {#2}{#1}}%
```

## 7.45 `\xintDivFloor`

### 1.1

```
1463 \def\xintDivFloor      {\romannumeral0\xintdivfloor }%
1464 \def\xintdivfloor #1#2{\xintfloor{\xintDiv {#1}{#2}}}%
```

## 7.46 `\xintDivTrunc`

### 1.1. `\xintttrunc` rather than `\xintitrunc0` in 1.1a

```
1465 \def\xintDivTrunc      {\romannumeral0\xintdivtrunc }%
1466 \def\xintdivtrunc #1#2{\xintttrunc {\xintDiv {#1}{#2}}}%
```

## 7.47 `\xintDivRound`

### 1.1

```
1467 \def\xintDivRound      {\romannumeral0\xintdivround }%
1468 \def\xintdivround #1#2{\xintiround 0{\xintDiv {#1}{#2}}}%
```

## 7.48 `\xintMod`

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

```
1469 \def\xintMod {\romannumeral0\xintmod }%
1470 \def\xintmod #1{\expandafter\XINT_mod_a\romannumeral0\xintra{#1}.}%
1471 \def\XINT_mod_a #1#2.#3%
1472   {\expandafter\XINT_mod_b\expandafter #1\romannumeral0\xintra{#3}#2.}%
1473 \def\XINT_mod_b #1#2% #1 de A, #2 de B.
1474 {%
1475   \if0#2\xint_dothis\XINT_mod_divbyzero\fi
1476   \if0#1\xint_dothis\XINT_mod_aiszero\fi
1477   \if-#2\xint_dothis{\XINT_mod_bneg #1}\fi
1478   \xint_orthat{\XINT_mod_bpos #1#2}%
1479 }%
1480 \def\XINT_mod_bpos #1%
1481 {%
1482   \xint_UDsignfork
1483     #1{\xintiiopp\XINT_mod_pos {}}%
1484     -{\XINT_mod_pos #1}%
1485   \krof
1486 }%
1487 \def\XINT_mod_bneg #1%
1488 {%
1489   \xint_UDsignfork
1490     #1{\xintiiopp\XINT_mod_pos {}}%
1491     -{\XINT_mod_pos #1}%
1492   \krof
1493 }%
1494 \def\XINT_mod_divbyzero #1.{\xintError:DivisionByZero\space 0/1[0]}%
```

```

1495 \def\XINT_mod_aiszero #1.{ 0/1[0]}%
1496 \def\XINT_mod_pos #1#2/#3[#4]#5/#6[#7].%
1497 {%
1498   \expandafter\XINT_mod_pos_a
1499   \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.\expandafter
1500   {\romannumeral0\xintiimul {#6}{#3}}%      n fois u
1501   {\xintiiE{\xintiiMul {#1#5}{#3}}{#7-#4}}%  m fois u
1502   {\xintiiE{\xintiiMul {#2}{#6}}{#4-#7}}%    t fois n
1503 }%
1504 \def\XINT_mod_pos_a #1.#2#3#4{\xintiirem {#3}{#4}/#2[#1]}%

```

### 7.49 \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.

```

1505 \def\xintIsOne {\romannumeral0\xintisone }%
1506 \def\xintisone #1{\expandafter\XINT_fracisone
1507   \romannumeral0\xintraewithzeros{#1}\Z }%
1508 \def\XINT_fracisone #1/#2\Z
1509   {\if0\XINT_Cmp {#1}{#2}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%

```

### 7.50 \xintGeq

```

1510 \def\xintGeq {\romannumeral0\xintgeq }%
1511 \def\xintgeq #1%
1512 {%
1513   \expandafter\XINT_fgeq\expandafter {\romannumeral0\xintabs {#1}}%
1514 }%
1515 \def\XINT_fgeq #1#2%
1516 {%
1517   \expandafter\XINT_fgeq_A \romannumeral0\xintabs {#2}#1%
1518 }%
1519 \def\XINT_fgeq_A #1%
1520 {%
1521   \xint_gob_til_zero #1\XINT_fgeq_Zii 0%
1522   \XINT_fgeq_B #1%
1523 }%
1524 \def\XINT_fgeq_Zii 0\XINT_fgeq_B #1[#2]#3[#4]{ 1}%
1525 \def\XINT_fgeq_B #1/#2[#3]#4#5/#6[#7]%
1526 {%
1527   \xint_gob_til_zero #4\XINT_fgeq_Zi 0%
1528   \expandafter\XINT_fgeq_C\expandafter
1529   {\the\numexpr #7-#3\expandafter}\expandafter
1530   {\romannumeral0\xintiimul {#4#5}{#2}}%
1531   {\romannumeral0\xintiimul {#6}{#1}}%
1532 }%
1533 \def\XINT_fgeq_Zi 0#1#2#3#4#5#6#7{ 0}%
1534 \def\XINT_fgeq_C #1#2#3%
1535 {%
1536   \expandafter\XINT_fgeq_D\expandafter
1537   {#3}{#1}{#2}%
1538 }%

```

```

1539 \def\XINT_fgeq_D #1#2#3%
1540 {%
1541   \expandafter\XINT_cntSgnFork\romannumeral`&&\expandafter\XINT_cntSgn
1542   \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\Z
1543   { 0}\XINT_fgeq_E #2\Z {#3}{#1}}{ 1}%
1544 }%
1545 \def\XINT_fgeq_E #1%
1546 {%
1547   \xint_UDsignfork
1548     #1\XINT_fgeq_Fd
1549     -{\XINT_fgeq_Fn #1}%
1550   \krof
1551 }%
1552 \def\XINT_fgeq_Fd #1\Z #2#3%
1553 {%
1554   \expandafter\XINT_fgeq_Fe\expandafter
1555   {\romannumeral0\XINT_dsx_addzerosnofuss {#1}{#3}}{#2}%
1556 }%
1557 \def\XINT_fgeq_Fe #1#2{\XINT_geq_pre {#2}{#1}}%
1558 \def\XINT_fgeq_Fn #1\Z #2#3%
1559 {%
1560   \expandafter\XINT_geq_pre\expandafter
1561   {\romannumeral0\XINT_dsx_addzerosnofuss {#1}{#2}}{#3}%
1562 }%

```

### 7.51 \xintMax

```

1563 \def\xintMax {\romannumeral0\xintmax }%
1564 \def\xintmax #1%
1565 {%
1566   \expandafter\XINT_fmax\expandafter {\romannumeral0\xintra #1}%
1567 }%
1568 \def\XINT_fmax #1#2%
1569 {%
1570   \expandafter\XINT_fmax_A\romannumeral0\xintra #2}%
1571 }%
1572 \def\XINT_fmax_A #1#2/#3[#4]#5#6/#7[#8]%
1573 {%
1574   \xint_UDsignsfork
1575     #1#5\XINT_fmax_minusminus
1576     -#5\XINT_fmax_firstneg
1577     #1-\XINT_fmax_secondneg
1578     --\XINT_fmax_nonneg_a
1579   \krof
1580   #1#5{#2/#3[#4]}{#6/#7[#8]}%
1581 }%
1582 \def\XINT_fmax_minusminus --%
1583   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_fmin_nonneg_b }%
1584 \def\XINT_fmax_firstneg #1-#2#3{ #1#2}%
1585 \def\XINT_fmax_secondneg -#1#2#3{ #1#3}%
1586 \def\XINT_fmax_nonneg_a #1#2#3#4%
1587 {%
1588   \XINT_fmax_nonneg_b {#1#3}{#2#4}%
1589 }%

```

```

1590 \def\XINT_fmax_nonneg_b #1#2%
1591 {%
1592   \if0\romannumeral0\XINT_fgeq_A #1#2%
1593     \xint_afterfi{ #1}%
1594   \else \xint_afterfi{ #2}%
1595   \fi
1596 }%

```

### 7.52 \xintMaxof

```

1597 \def\xintMaxof      {\romannumeral0\xintmaxof }%
1598 \def\xintmaxof      #1{\expandafter\XINT_maxof_a\romannumeral`&&@#1\relax }%
1599 \def\XINT_maxof_a #1{\expandafter\XINT_maxof_b\romannumeral0\xintra{#1}\Z }%
1600 \def\XINT_maxof_b #1\Z #2%
1601   {\expandafter\XINT_maxof_c\romannumeral`&&@#2\Z {#1}\Z}%
1602 \def\XINT_maxof_c #1%
1603   {\xint_gob_til_relax #1\XINT_maxof_e\relax\XINT_maxof_d #1}%
1604 \def\XINT_maxof_d #1\Z
1605   {\expandafter\XINT_maxof_b\romannumeral0\xintmax {#1}}%
1606 \def\XINT_maxof_e #1\Z #2\Z { #2}%

```

### 7.53 \xintMin

```

1607 \def\xintMin {\romannumeral0\xintmin }%
1608 \def\xintmin #1%
1609 {%
1610   \expandafter\XINT_fmin\expandafter {\romannumeral0\xintra{#1}}%
1611 }%
1612 \def\XINT_fmin #1#2%
1613 {%
1614   \expandafter\XINT_fmin_A\romannumeral0\xintra{#2}#1%
1615 }%
1616 \def\XINT_fmin_A #1#2/#3[#4]#5#6/#7[#8]%
1617 {%
1618   \xint_UDsignsfork
1619     #1#5\XINT_fmin_minusminus
1620     -#5\XINT_fmin_firstneg
1621     #1-\XINT_fmin_secondneg
1622     --\XINT_fmin_nonneg_a
1623   \krof
1624   #1#5{#2/#3[#4]}\{#6/#7[#8]}%
1625 }%
1626 \def\XINT_fmin_minusminus --%
1627   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_fmax_nonneg_b }%
1628 \def\XINT_fmin_firstneg #1-#2#3{ -#3}%
1629 \def\XINT_fmin_secondneg -#1#2#3{ -#2}%
1630 \def\XINT_fmin_nonneg_a #1#2#3#4%
1631 {%
1632   \XINT_fmin_nonneg_b {#1#3}\{#2#4}%
1633 }%
1634 \def\XINT_fmin_nonneg_b #1#2%
1635 {%
1636   \if0\romannumeral0\XINT_fgeq_A #1#2%
1637     \xint_afterfi{ #2}%
1638   \else \xint_afterfi{ #1}%

```

```
1639 \fi
1640 }%
```

### 7.54 *\xintMinof*

```
1641 \def\xintMinof      {\romannumeral0\xintminof }%
1642 \def\xintminof      #1{\expandafter\XINT_minof_a\romannumeral`&&@#1\relax }%
1643 \def\XINT_minof_a #1{\expandafter\XINT_minof_b\romannumeral0\xintra{#1}\Z }%
1644 \def\XINT_minof_b #1\Z #2%
1645     {\expandafter\XINT_minof_c\romannumeral`&&@#2\Z {#1}\Z}%
1646 \def\XINT_minof_c #1%
1647     {\xint_gob_til_relax #1\XINT_minof_e\relax\XINT_minof_d #1}%
1648 \def\XINT_minof_d #1\Z
1649     {\expandafter\XINT_minof_b\romannumeral0\xintmin {#1}}%
1650 \def\XINT_minof_e #1\Z #2\Z { #2}%
```

### 7.55 *\xintCmp*

```
1651 %\def\xintCmp {\romannumeral0\xintcmp }%
1652 \def\xintcmp #1%
1653 {%
1654     \expandafter\XINT_fc{#1}\expandafter {\romannumeral0\xintra{#1}}%
1655 }%
1656 \def\XINT_fc{#1}%
1657 {%
1658     \expandafter\XINT_fc{#1}\expandafter {\romannumeral0\xintra{#2}}%
1659 }%
1660 \def\XINT_fc{#1}\XINT_fc{#2}\XINT_fc{#3}\XINT_fc{#4}\XINT_fc{#5}\XINT_fc{#6}\XINT_fc{#7}\XINT_fc{#8}%
1661 {%
1662     \xint_UDsignsfork
1663     #1#5\XINT_fc{#1}\XINT_fc{#2}\XINT_fc{#3}\XINT_fc{#4}\XINT_fc{#5}\XINT_fc{#6}\XINT_fc{#7}\XINT_fc{#8}%
1664     -#5\XINT_fc{#1}\XINT_fc{#2}\XINT_fc{#3}\XINT_fc{#4}\XINT_fc{#5}\XINT_fc{#6}\XINT_fc{#7}\XINT_fc{#8}%
1665     #1-\XINT_fc{#1}\XINT_fc{#2}\XINT_fc{#3}\XINT_fc{#4}\XINT_fc{#5}\XINT_fc{#6}\XINT_fc{#7}\XINT_fc{#8}%
1666     --\XINT_fc{#1}\XINT_fc{#2}\XINT_fc{#3}\XINT_fc{#4}\XINT_fc{#5}\XINT_fc{#6}\XINT_fc{#7}\XINT_fc{#8}%
1667     \kro{#1}\XINT_fc{#2}\XINT_fc{#3}\XINT_fc{#4}\XINT_fc{#5}\XINT_fc{#6}\XINT_fc{#7}\XINT_fc{#8}%
1668     #1#5{#2/#3[4]}{#6/#7[8]}%
1669 }%
1670 \def\XINT_fc{#1}\XINT_fc{#2}\XINT_fc{#3}\XINT_fc{#4}\XINT_fc{#5}\XINT_fc{#6}\XINT_fc{#7}\XINT_fc{#8}%
1671 \def\XINT_fc{#1}\XINT_fc{#2}\XINT_fc{#3}\XINT_fc{#4}\XINT_fc{#5}\XINT_fc{#6}\XINT_fc{#7}\XINT_fc{#8}%
1672 \def\XINT_fc{#1}\XINT_fc{#2}\XINT_fc{#3}\XINT_fc{#4}\XINT_fc{#5}\XINT_fc{#6}\XINT_fc{#7}\XINT_fc{#8}%
1673 \def\XINT_fc{#1}\XINT_fc{#2}\XINT_fc{#3}\XINT_fc{#4}\XINT_fc{#5}\XINT_fc{#6}\XINT_fc{#7}\XINT_fc{#8}%
1674 {%
1675     \xint_UDzerosfork
1676     #1#2\XINT_fc{#1}\XINT_fc{#2}\XINT_fc{#3}\XINT_fc{#4}\XINT_fc{#5}\XINT_fc{#6}\XINT_fc{#7}\XINT_fc{#8}%
1677     0#2\XINT_fc{#1}\XINT_fc{#2}\XINT_fc{#3}\XINT_fc{#4}\XINT_fc{#5}\XINT_fc{#6}\XINT_fc{#7}\XINT_fc{#8}%
1678     #10\XINT_fc{#1}\XINT_fc{#2}\XINT_fc{#3}\XINT_fc{#4}\XINT_fc{#5}\XINT_fc{#6}\XINT_fc{#7}\XINT_fc{#8}%
1679     00\XINT_fc{#1}\XINT_fc{#2}\XINT_fc{#3}\XINT_fc{#4}\XINT_fc{#5}\XINT_fc{#6}\XINT_fc{#7}\XINT_fc{#8}%
1680     \kro{#1}\XINT_fc{#2}\XINT_fc{#3}\XINT_fc{#4}\XINT_fc{#5}\XINT_fc{#6}\XINT_fc{#7}\XINT_fc{#8}%
1681     #1#2%
1682 }%
1683 \def\XINT_fc{#1}\XINT_fc{#2}\XINT_fc{#3}\XINT_fc{#4}\XINT_fc{#5}\XINT_fc{#6}\XINT_fc{#7}\XINT_fc{#8}%
1684 \def\XINT_fc{#1}\XINT_fc{#2}\XINT_fc{#3}\XINT_fc{#4}\XINT_fc{#5}\XINT_fc{#6}\XINT_fc{#7}\XINT_fc{#8}%
1685 \def\XINT_fc{#1}\XINT_fc{#2}\XINT_fc{#3}\XINT_fc{#4}\XINT_fc{#5}\XINT_fc{#6}\XINT_fc{#7}\XINT_fc{#8}%
1686 \def\XINT_fc{#1}\XINT_fc{#2}\XINT_fc{#3}\XINT_fc{#4}\XINT_fc{#5}\XINT_fc{#6}\XINT_fc{#7}\XINT_fc{#8}%
1687 }
```

```

1688 \XINT_fcmp_B #1#3#2#4%
1689 }%
1690 \def\XINT_fcmp_B #1/#2[#3]#4/#5[#6]%
1691 {%
1692 \expandafter\XINT_fcmp_C\expandafter
1693 {\the\numexpr #6-#3\expandafter}\expandafter
1694 {\romannumeral0\xintiimul {#4}{#2}}%
1695 {\romannumeral0\xintiimul {#5}{#1}}%
1696 }%
1697 \def\XINT_fcmp_C #1#2#3%
1698 {%
1699 \expandafter\XINT_fcmp_D\expandafter
1700 {#3}{#1}{#2}%
1701 }%
1702 \def\XINT_fcmp_D #1#2#3%
1703 {%
1704 \expandafter\XINT_cntSgnFork\romannumeral`&&\expandafter\XINT_cntSgn
1705 \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\Z
1706 { -1}{\XINT_fcmp_E #2\Z {#3}{#1}}{ 1}%
1707 }%
1708 \def\XINT_fcmp_E #1%
1709 {%
1710 \xint_UDsignfork
1711 #1\XINT_fcmp_Fd
1712 -{\XINT_fcmp_Fn #1}%
1713 \krof
1714 }%
1715 \def\XINT_fcmp_Fd #1\Z #2#3%
1716 {%
1717 \expandafter\XINT_fcmp_Fe\expandafter
1718 {\romannumeral0\XINT_dsx_addzerosnofuss {#1}{#3}}{#2}%
1719 }%
1720 \def\XINT_fcmp_Fe #1#2{\xintiicmp {#2}{#1}}%
1721 \def\XINT_fcmp_Fn #1\Z #2#3%
1722 {%
1723 \expandafter\xintiicmp\expandafter
1724 {\romannumeral0\XINT_dsx_addzerosnofuss {#1}{#2}}{#3}%
1725 }%

```

### 7.56 \xintAbs

```

1726 \def\xintAbs {\romannumeral0\xintabs }%
1727 \def\xintabs #1{\expandafter\XINT_abs\romannumeral0\xintraw {#1}}%

```

### 7.57 \xintOpp

```

1728 \def\xintOpp {\romannumeral0\xintopp }%
1729 \def\xintopp #1{\expandafter\XINT_opp\romannumeral0\xintraw {#1}}%

```

### 7.58 \xintSgn

```

1730 \def\xintSgn {\romannumeral0\xintsgn }%
1731 \def\xintsgn #1{\expandafter\XINT_sgn\romannumeral0\xintraw {#1}\Z }%

```

## 7.59 Floating point macros

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

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

1.2f added `\xintFloatPFactorial` and `\xintFloatBinomial` and improved the speed of `\xintFloatPow` and `\xintFloatPower`. And its `\xintFloat` tries to be more efficient in handling inputs which are not fractions to start with.

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

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

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

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

## 7.60 `\xintFloat`

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

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

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

The important thing I had forgotten to document was that the initial 1.07 version did the \*exact\* rounding of fractional inputs *A/B* whereas the 1.08 version **first truncated *A* and *B* to *P+2* digits**: to round *A/B*  $10^n$  to *P* digits of precision the routine first truncates *A* and *B* to *P+2* digits and after that does the exact rounding of *A/B*. Naturally this means that it may then not necessarily compute the correct rounding of *A/B*.

Example : `\xintFloat {1/17597472569900621233}`  
 with *xintfrac* 1.07: 5.682634230727187e-20  
 with *xintfrac* 1.08b or later: 5.682634230727188e-20  
 and the exact value is 5.682634230727187499924124...e-20.

1.07 computed the exact rounding for all inputs but as explained from 1.08 on, `\xintFloat` first truncates the denominator to  $16+2=18$  digits, and here this increases  $1/x$  enough to make the final rounding produce a result *1ulp* higher.

In fact already dropping the last digit is enough to make the quotient cross the border:  
 $1/17597472569900621233=5.682634230727187499924124...e-20$   
 $1/1759747256990062123=56.82634230727187500892894...e-20$

1.2f did some minor improvements to the code, there was in particular a never-used branch. And it tries to handle more swiftly the case of inputs which are not fractions.

This routine uses old macros `\XINT_addm_a` and `\XINT_lenrord_loop`. This could now be penalizing for *P* exceeding a few dozens, compared to doing it the 1.2 way.

I have always hesitated about the policy of printing 10.00...0 in case of rounding upwards towards next power of ten. It does make sense because it tells the (higher) precision of the rounding and moreover it is not too hard easy to test on output, although it is a bit cumbersome not to be certain to have exactly *P* digits.

New: since 1.2f `\XINTinFloat` (which is the variant of `\xintFloat` used by float macros when they need to round their input or round some pre-final result) always guarantees that *A* exactly has



P digits. This simplifies things; and goes hand in hand with the fact that I have decided float macros should not aim like MPFR at correct rounding from the exact inputs, but only from inputs rounded to target precision P.

```

1732 \def\xintFloat  {\romannumeral0\xintfloat }%
1733 \def\xintfloat #1{\XINT_float_chkopt #1\xint_relax }%
1734 \def\XINT_float_chkopt #1%
1735 {%
1736   \ifx [#1\expandafter\XINT_float_opt
1737     \else\expandafter\XINT_float_noopt
1738   \fi #1%
1739 }%
1740 \def\XINT_float_noopt #1\xint_relax
1741 {%
1742   \expandafter\XINT_float_a\expandafter\XINTdigits\expandafter.%
1743   \romannumeral0\XINT_infrac {#1}\XINT_float_Q
1744 }%
1745 \def\XINT_float_opt [\xint_relax #1]#2%
1746 {%
1747   \expandafter\XINT_float_a\the\numexpr #1\expandafter.%
1748   \romannumeral0\XINT_infrac {#2}\XINT_float_Q
1749 }%

```

Note 2015/12/02. Le but de ce code de 1.08 (2013), jusqu'à l'exécution de `\XINT_float_Q`, est simplement de tronquer numérateur et dénominateur à au plus  $P+2$  chiffres en ajustant la partie décimale "n".

```

1750 \def\XINT_float_a #1.#2#3% #1=P, #2=n, #3=A, #4=B
1751 {%
1752   \XINT_float_fork #3\Z {#1}{#2}% #1 = precision, #2=n
1753 }%

```

$A \setminus \{P\} \setminus \{n\} \setminus \{B\} \setminus \text{XINT\_float\_Q}$

```

1754 \def\XINT_float_fork #1%
1755 {%
1756   \xint_UDzerominusfork
1757   #1-\XINT_float_zero
1758   0#1\XINT_float_J
1759   0-\XINT_float_K #1}%
1760 \krof
1761 }%
1762 \def\XINT_float_zero #1\Z #2#3#4#5{ 0.e0}%
1763 \def\XINT_float_J {\expandafter\xint_minus_thenstop\romannumeral0\XINT_float_K }%
1764 \def\XINT_float_K #1\Z #2% #1=A, #2=P, #3=n, #4=B
1765 {%
1766   \expandafter\XINT_float_L
1767   \the\numexpr\xintLength{#1}\expandafter.\the\numexpr #2+\xint_c_ii.{#1}{#2}%
1768 }%

```

$|A|.P+2.\{A\}\{P\}\{n\}\{B\}\setminus \text{XINT\_float\_Q}$ . We check if A already has length  $\leq P+2$ .

```

1769 \def\XINT_float_L #1.#2.%
1770 {%

```

## 7 Package *xintfrac* implementation

```

1771 \ifnum #1>#2
1772 \expandafter\XINT_float_Ma
1773 \else
1774 \expandafter\XINT_float_Mb
1775 \fi #1.#2.%
1776 }%

```

$|A|.P+2.\{A\}\{P\}\{n\}\{B\}\backslash XINT\_float\_Q$ . We will keep only the first  $P+2$  digits of  $A$ . We use  $A'$  for notation.

```

1777 \def\XINT_float_Ma #1.#2.#3%
1778 {%
1779 \expandafter\XINT_float_MatoN
1780 \the\numexpr #1-#2\expandafter.%
1781 \romannumeral0\XINT_split_fromleft_loop #2.\{#\W\W\W\W\W\W\W\W\Z
1782 \{#2\}%
1783 }%

```

$|A|-(P+2).\{A'=P+2 \text{ premiers chiffres de } A\}.\{junk\}.\{P+2\}\{P\}\{n\}\{B\}\backslash XINT\_float\_Q$  devient  $|B|.n'=n+|A|-(P+2).P+2.\{B\}\{P+2\}\{A'\}\{P\}$ . Car ici  $P+2=|A'|$ .

```

1784 \def\XINT_float_MatoN #1.#2.#3.#4#5#6#7%
1785 {%
1786 \expandafter\XINT_float_N
1787 \the\numexpr\xintLength\{#7\}\expandafter.\the\numexpr #1+#6.#4.%
1788 \{#7\}\{#4\}\{#2\}\{#5\}%
1789 }%

```

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

```

1790 \def\XINT_float_Mb #1.#2.#3#4#5#6%
1791 {%
1792 \expandafter\XINT_float_N
1793 \romannumeral0\xintlength\{#6\}.#5.#2.\{#6\}\{#1\}\{#3\}\{#4\}%
1794 }%

```

Ce qu'on a fait avec  $A$  on le fait maintenant avec  $B$ . Mais on va repérer  $B=1$  avant de faire une division.

En entrée:  $|B|.n'.P+2.\{B\}\{|A'|\}\{A'\}\{P\}\backslash XINT\_float\_Q$

```

1795 \def\XINT_float_N #1.#2.#3.%
1796 {%
1797 \ifnum #1>#3 \xint_dothis\XINT_float_N_Blong\fi
1798 \ifnum #1=\xint_c_i\xint_dothis\XINT_float_N_Bshort\fi
1799 \xint_orthat{\XINT_float_P 0.}%
1800 #1.#2.#3.%
1801 }%

```

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

```

1802 \def\XINT_float_N_Blong #1.#2.#3.#4%
1803 {%
1804 \expandafter\XINT_float_NaP
1805 \the\numexpr #2-#1+#3\expandafter.%
1806 \romannumeral0\XINT_split_fromleft_loop #3.\{#\W\W\W\W\W\W\W\W\Z \{#3\}%
1807 }%

```

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

```
1808 \def\XINT_float_NaP #1.#2.#3.#4{\XINT_float_P 0.#4.#1.#4.{#2}}%
```

Ici B est court, mais on va repérer les cas avec B=1 pour aller plus vite.

```
1809 \def\XINT_float_N_Bshort 1.#1.#2.#3%
```

```
1810 {%
```

```
1811 \ifnum #3=\xint_c_i \xint_dothis{\XINT_float_P 1.}\fi
```

```
1812 \xint_orthat{\XINT_float_P 0.}1.#1.#2.#3%
```

```
1813 }%
```

Si B est de longueur  $\leq P+2$  on arrive ici avec en entrée 0 ou 1. $|B|.n'.P+2.\{B\}$ , sinon avec  $i.P+2.n'.P+2.\{B'\}$ , suivi dans les deux cas par  $\{A'\}\{A'\}\{P\}$  et  $\#9=\text{\XINT\_float\_Q}$ . On va invoquer  $\text{\XINT\_float\_Q}$ , mais on lui passe l'indicateur i qui repère le cas B=1 que l'on veut traiter plus rapidement (en attendant que j'introduise une vraie notion de type pour les flottants; à laquelle je ne pourrai pas échapper bien plus longtemps). On aura donc  $\text{\XINT\_float\_Q } i|B|-|A|+P+1.\{A\}\{B\}\{P\}\{n\}$  avec les nouveaux A, B, n. Ici on a doit passer au-dessus de A et |A| pour aller chercher #9. à revoir car pas satisfaisant.

```
1814 \def\XINT_float_P #1.#2.#3.#4.#5#6#7#8#9%
```

```
1815 {%
```

```
1816 \expandafter #9\expandafter#1%
```

```
1817 \the\numexpr #2-#6+#4-\xint_c_i.{#7}{#5}{#8}{#3}%
```

```
1818 }%
```

On arrive ici avec  $|B|-|A|+P+1.\{A\}\{B\}\{P\}\{n\}$ , les A et B étant ceux d'origine tronqués à au plus P+2 chiffres, le n a été ajusté si besoin.

On calcule maintenant le quotient euclidien de  $A \cdot 10^{|B|-|A|+P+1}$  (qui a P+1 chiffres de plus que B) par B. Ce quotient Q aura P+1 ou P+2 chiffres.

2015/12/07: nouveau premier token indicateur i=1 si B=1, i=0 sinon.

```
1819 \def\XINT_float_Q #1%
```

```
1820 {%
```

```
1821 \if 1#1\expandafter\XINT_float_Ri\else\expandafter\XINT_float_Rii\fi
```

```
1822 }%
```

```
1823 \def\XINT_float_Ri #1.#2#3%
```

```
1824 {%
```

```
1825 \expandafter\XINT_float_Sa
```

```
1826 \romannumeral0\XINT_dsx_addzerosnofuss {#1}{#2}\Z {#1}%
```

```
1827 }%
```

```
1828 \def\XINT_float_Rii #1.#2#3%
```

```
1829 {%
```

```
1830 \expandafter\XINT_float_Sa
```

```
1831 \romannumeral0\xintiiquo{\XINT_dsx_addzerosnofuss {#1}{#2}}{#3}\Z {#1}%
```

```
1832 }%
```

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

```
1833 \def\XINT_float_Sa #1%
```

```
1834 {%
```

## 7 Package *xintfrac* implementation

```

1835 \if #19\xint_dothis {\XINT_float_Sb\XINT_float_Wb }\fi
1836 \xint_orthat {\XINT_float_Sb\XINT_float_Wa }#1%
1837 }%

```

En entrée  $\backslash\text{XINT\_float\_W}(a \text{ ou } b) \text{ Q}\backslash\text{Z} \{ |B| - |A| + P + 1 \} \{ P \} \{ n \}$ . On va renverser Q en comptant sa longueur  $L = P + 1$  ou  $P + 2$  au passage. Pour le moment avec la vieille routine  $\backslash\text{XINT\_lenrord\_loop}$  mais il faudra voir si je modifie cela.

```

1838 \def\XINT_float_Sb #1#2\Z #3#4%
1839 {%
1840 \expandafter\XINT_float_T
1841 \the\numexpr #4+\xint_c_i\expandafter.%
1842 \romannumeral`&&\XINT_lenrord_loop 0.{}\#2\Z\W\W\W\W\W\W\W\Z #1{\#3}{\#4}%
1843 }%

```

Si  $L > P + 1$ , c'est que  $L = P + 2$ , on laisse tomber le chiffre le moins significatif de Q. En sortie on a  $\backslash\text{token Q à l'envers}\backslash\text{Z} \backslash\text{token} \{ |B| - |A| + P + 1 \} \{ P \} \{ n \}$

```

1844 \def\XINT_float_T #1.#2.#3%
1845 {%
1846 \ifnum #2>#1 \xint_dothis{\XINT_float_U\XINT_float_Xb}\fi
1847 \xint_orthat{\XINT_float_U\XINT_float_Xa #3}%
1848 }%
1849 \def\XINT_float_U #1#2%
1850 {%
1851 \ifnum #2<\xint_c_v
1852 \expandafter\XINT_float_Va
1853 \else
1854 \expandafter\XINT_float_Vb
1855 \fi #1%
1856 }%

```

Chiffre moins significatif de Q est  $< 5$ . Suffit de tronquer. On va ensuite embrayer soit sur  $\backslash\text{XINT\_float\_Xa}$  si on avait  $P + 1$  chiffres, soit sur  $\backslash\text{XINT\_float\_Xb}$  si on en avait  $P + 2$ . Dans ce cas de troncation, on n'a pas à se préoccuper d'un arrondi éventuel vers le haut vers une puissance de dix, donc le #3 est ici remplacé par  $\backslash\text{XINT\_float\_Wa}$ .

```

1857 \def\XINT_float_Va #1#2\Z #3%
1858 {%
1859 \expandafter#1%
1860 \romannumeral0\expandafter\XINT_float_Wa
1861 \romannumeral0\XINT_rord_main {}#2%
1862 \xint_relax
1863 \xint_bye\xint_bye\xint_bye\xint_bye
1864 \xint_bye\xint_bye\xint_bye\xint_bye
1865 \xint_relax \Z
1866 }%

```

Chiffre moins significatif de Q est au moins 5. Faut aussi ajouter 1. On utilise la vieille routine d'addition spéciale  $\backslash\text{XINT\_addm\_A}$  qui renvoie son résultat à l'endroit. De plus on doit ensuite vérifier si l'arrondi s'est fait vers la puissance de 10 supérieure. Cela ne peut se produire que si le premier chiffre était 9, donc dans ce cas on a  $\#3 = \backslash\text{XINT\_float\_Wb}$ , sinon on a  $\#3 = \backslash\text{XINT\_float\_Wa}$ .

```

1867 \def\XINT_float_Vb #1#2\Z #3%
1868 {%

```

```

1869 \expandafter #1%
1870 \romannumeral0\expandafter #3%
1871 \romannumeral0\XINT_addm_A 0{}1000\W\X\Y\Z #2000\W\X\Y\Z \Z
1872 }%

```

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

```

1873 \def\XINT_float_Wa #1{ #1.}%
1874 \def\XINT_float_Wb #1#2{\if #11\xint_dothis{ 10.}\fi\xint_orthat{ #1.#2}}%

```

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

```

1875 \def\XINT_float_Xa #1\Z #2#3#4%
1876 {%
1877 \expandafter\XINT_float_Y\the\numexpr #3+#4-#2.{#1}%
1878 }%
1879 \def\XINT_float_Xb #1\Z #2#3#4%
1880 {%
1881 \expandafter\XINT_float_Y\the\numexpr #3+#4+\xint_c_i-#2.{#1}%
1882 }%
1883 \def\XINT_float_Y #1.#2{ #2e#1}%

```

## 7.61 $\backslash XINTinFloat$

1.07.

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

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

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

As this is surgery on pre-existing code where a more complete rewrite would be needed it is a bit ugly though.

Each time  $\backslash XINTinFloat$  is called it at least computes a length. Naturally if we had some format for floats that would be dispensed of... Something like  $\langle letter P \rangle \langle length of mantissa \rangle . mantissa . exponent$ , etc...

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

```

1884 \def\XINTinFloat {\romannumeral0\XINTinfloat }%
1885 \def\XINTinfloat [#1]#2%
1886 {%

```

## 7 Package *xintfrac* implementation

```

1887 \expandafter\XINT_infloat\the\numexpr #1\expandafter.%
1888 \romannumeral0\expandafter\XINT_infloat_in
1889 \romannumeral`&&@#2/\XINT_W[\XINT_W\XINT_T\XINT_infloat_Q
1890 }%
1891 \def\XINT_infloat #1.#2{#2#1.}%
1892 \def\XINT_infloat_in #1[#2%
1893 {%
1894 \xint_UDXINTWfork
1895 #2\XINT_infloat_gen
1896 \XINT_W\XINT_infloat_res_a
1897 \krof
1898 #1[#2%
1899 }%
1900 \edef\XINT_infloat_gen {\noexpand\expandafter\space\noexpand\expandafter
1901 \noexpand\XINT_infloat_a
1902 \noexpand\romannumeral0\noexpand\XINT_frac_gen }%
1903 \def\XINT_infloat_res_a #1%
1904 {%
1905 \xint_gob_til_zero #1\XINT_infloat_res_zero 0\XINT_infloat_res_b #1%
1906 }%
1907 \def\XINT_infloat_res_zero #1\XINT_infloat_Q { \XINT_infloat_sp_zero}%
1908 \def\XINT_infloat_sp_zero #1.{ 0[0]}%
1909 \def\XINT_infloat_res_b #1/#2%
1910 {%
1911 \xint_UDXINTWfork
1912 #2\XINT_infloat_res_ca
1913 \XINT_W\XINT_infloat_res_cb
1914 \krof
1915 #1/#2%
1916 }%
1917 \def\XINT_infloat_res_ca #1[#2]/\XINT_W[\XINT_W\XINT_T\XINT_infloat_Q
1918 { \XINT_infloat_sp #1.#2.}%
1919 \def\XINT_infloat_res_cb #1/#2[%
1920 {\expandafter\XINT_infloat_res_cc\romannumeral`&&@#2~#1[%
1921 \edef\XINT_infloat_res_cc #1~#2[#3]/\XINT_W[\XINT_W\XINT_T
1922 {\noexpand\expandafter\space\noexpand\expandafter
1923 \noexpand\XINT_infloat_a\noexpand\expandafter
1924 {\noexpand\the\numexpr #3}{#2}{#1}}%

```

1.2f adds (2016/03/11) special quick treatment for A[N] inputs.

```

1925 \def\XINT_infloat_sp #1.#2%
1926 {%
1927 \if-#2\xint_dothis{\expandafter-\romannumeral0\XINT_infloat_sp_a {}}\fi
1928 \xint_orthat{\XINT_infloat_sp_a #2}#1.%
1929 }%
1930 \def\XINT_infloat_sp_a #1#2.#3.%
1931 {%
1932 \expandafter\XINT_infloat_sp_b\the\numexpr#2-\xintLength{#1#3}.#1#3.#2.%
1933 }%
1934 \def\XINT_infloat_sp_b #1%
1935 {%
1936 \xint_UDzerominusfork

```

## 7 Package *xintfrac* implementation

```

1937      #1-\XINT_infloat_sp_quick
1938      0#1\XINT_infloat_sp_c
1939      0-\XINT_infloat_sp_addzeroes #1}%
1940      \krof
1941 }%
1942 \def\XINT_infloat_sp_quick .#1.#2.#3.{ #1[#3]}%
1943 \def\XINT_infloat_sp_addzeroes #1.#2.#3.#4.%
1944 {%
1945     \expandafter\XINT_infloat_sp_done
1946     \the\numexpr #4-#1\expandafter.%
1947     \romannumeral0\XINT_dsx_addzerosnofuss {#1}{#2};%
1948 }%
1949 \def\XINT_infloat_sp_done #1.#2;{ #2[#1]}%

```

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

```

1950 \def\XINT_infloat_sp_c #1.#2%
1951 {%
1952     \if #29\xint_dothis {\XINT_infloat_sp_d\XINT_infloat_Wb }\fi
1953     \xint_orthat {\XINT_infloat_sp_d\xint_c_ }#1.#2%
1954 }%
1955 \def\XINT_infloat_sp_d #1.#2.#3.#4.%#5.%
1956 {%
1957     \expandafter\XINT_infloat_sp_e
1958     \romannumeral0\expandafter\XINT_split_fromleft_loop
1959     \the\numexpr #4+\xint_c_i.{ }#3\W\W\W\W\W\W\W\W\Z #1#2.%
1960 }%

```

Still using old Reverse routine because presumably we are handling numbers up to a few dozens digits. But a faster `\xintReverseDigits` is available. No time now.

#1=first P+1 digits of A, #2=junk

```

1961 \def\XINT_infloat_sp_e #1.#2.%
1962 {%
1963     \expandafter\XINT_infloat_sp_f
1964     \romannumeral0\XINT_rord_main {}#1%
1965     \xint_relax
1966     \xint_bye\xint_bye\xint_bye\xint_bye
1967     \xint_bye\xint_bye\xint_bye\xint_bye
1968     \xint_relax ;%
1969 }%
1970 \def\XINT_infloat_sp_f #1%
1971 {%
1972     \ifnum #1<\xint_c_v
1973     \expandafter\XINT_infloat_sp_ga\else\expandafter\XINT_infloat_sp_h\fi
1974 }%
1975 \def\XINT_infloat_sp_ga #1;#2#3.#4.%
1976 {%
1977     \expandafter\XINT_infloat_sp_done\the\numexpr #3+#4\expandafter.%
1978     \romannumeral0\XINT_rord_main {}#1%
1979     \xint_relax
1980     \xint_bye\xint_bye\xint_bye\xint_bye
1981     \xint_bye\xint_bye\xint_bye\xint_bye

```

## 7 Package *xintfrac* implementation

```

1982 \xint_relax ;%
1983 }%
1984 \def\xINT_infloat_sp_h #1;#2%
1985 {%
1986 \expandafter\xINT_infloat_sp_i
1987 \romannumeral0\expandafter#2%
1988 \romannumeral0\xINT_addm_A 0{}1000\W\X\Y\Z #1000\W\X\Y\Z;%
1989 }%
1990 \def\xINT_infloat_sp_i #1#2;#3.#4.%
1991 {%
1992 \expandafter\xINT_infloat_sp_done\the\numexpr #1+#3+#4.#2;%
1993 }%

```

General branch handling A/B[N] inputs. Still, 1.2f identifies faster B=1 case.

```

1994 \def\xINT_infloat_a #1.#2#3% #1=P, #2=n, #3=A, #4=B
1995 {%
1996 \XINT_infloat_fork #3\Z {#1}{#2}% #1 = precision, #2=n
1997 }%
1998 \def\xINT_infloat_fork #1%
1999 {%
2000 \xint_UDzerominusfork
2001 #1-\XINT_infloat_zero
2002 0#1\XINT_infloat_J
2003 0-\XINT_float_K #1}%
2004 \krof
2005 }%
2006 \def\xINT_infloat_zero #1\Z #2#3#4#5{ 0[0]}%
2007 \def\xINT_infloat_J {\expandafter-\romannumeral0\xINT_float_K }%
2008 \def\xINT_infloat_Q #1%
2009 {%
2010 \if 1#1\expandafter\xINT_infloat_Ri\else\expandafter\xINT_infloat_Rii\fi
2011 }%
2012 \def\xINT_infloat_Ri #1.#2#3%
2013 {%
2014 \expandafter\xINT_infloat_Sa
2015 \romannumeral0\xINT_dsx_addzerosnofuss {#1}{#2}\Z {#1}%
2016 }%
2017 \def\xINT_infloat_Rii #1.#2#3%
2018 {%
2019 \expandafter\xINT_infloat_Sa
2020 \romannumeral0\xintiiquo{\XINT_dsx_addzerosnofuss {#1}{#2}}{#3}\Z {#1}%
2021 }%
2022 \def\xINT_infloat_Sa #1%
2023 {%
2024 \if #19\xint_dothis {\XINT_infloat_Sb\xINT_infloat_Wb }\fi
2025 \xint_orthat {\XINT_infloat_Sb\xint_c_ }#1%
2026 }%
2027 \def\xINT_infloat_Sb #1#2\Z #3#4%
2028 {%
2029 \expandafter\xINT_infloat_T
2030 \the\numexpr #4+\xint_c_i\expandafter.%
2031 \romannumeral &&\XINT_lenrord_loop 0.{#2}\Z\W\W\W\W\W\W\W\W\Z #1{#3}%

```



```

2032 }%
2033 \def\XINT_infloat_T #1.#2.#3%
2034 {%
2035     \ifnum #2>#1 \xint_dothis{\XINT_infloat_U\XINT_infloat_Xb}\fi
2036     \xint_orthat{\XINT_infloat_U\XINT_infloat_Xa #3}%
2037 }%
2038 \def\XINT_infloat_U #1#2%
2039 {%
2040     \ifnum #2<\xint_c_v
2041         \expandafter\XINT_infloat_Va
2042     \else
2043         \expandafter\XINT_infloat_Vb
2044     \fi #1%
2045 }%
2046 \def\XINT_infloat_Va #1#2\Z #3%
2047 {%
2048     \expandafter#1%
2049     \romannumeral0\expandafter\xint_c_
2050     \romannumeral0\XINT_rord_main {}#2%
2051     \xint_relax
2052     \xint_bye\xint_bye\xint_bye\xint_bye
2053     \xint_bye\xint_bye\xint_bye\xint_bye
2054     \xint_relax \Z
2055 }%
2056 \def\XINT_infloat_Vb #1#2\Z #3%
2057 {%
2058     \expandafter #1%
2059     \romannumeral0\expandafter #3%
2060     \romannumeral0\XINT_addm_A 0{}1000\W\X\Y\Z #2000\W\X\Y\Z \Z
2061 }%
2062 \def\XINT_infloat_Wb #1#2%
2063     {\if #11\xint_dothis{\xint_c_il}\fi\xint_orthat{\xint_c_#1#2}}%
2064 \def\XINT_infloat_Xa #1#2\Z #3#4%
2065 {%
2066     \expandafter\XINT_infloat_Y\the\numexpr #1+\xint_c_i+#4-#3.{#2}%
2067 }%
2068 \def\XINT_infloat_Xb #1#2\Z #3#4%
2069 {%
2070     \expandafter\XINT_infloat_Y\the\numexpr #1+\xint_c_ii+#4-#3.{#2}%
2071 }%
2072 \def\XINT_infloat_Y #1.#2{ #2[#1]}%

```

## 7.62 \xintPFloat

1.1. This is a prettifying printing macro for floats.

2016/03/07.

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

The new rule is simply that x.yz...eN will drop scientific notation in favor of pure decimal notation if  $-5 \leq N \leq 5$ . This is the default behaviour of Maple. The N here is as produced on output

by `\xintFloat`. There is the exceptional cases where  $x=10$ , and  $yz..=0000....$  from rounding up to the next power of ten.

```

2073 \def\xintPFloat    {\romannumeral0\xintpfloat }%
2074 \def\xintpfloat #1{\XINT_pfloat_chkopt #1\xint_relax }%
2075 \def\XINT_pfloat_chkopt #1%
2076 {%
2077   \ifx [#1\expandafter\XINT_pfloat_opt
2078     \else\expandafter\XINT_pfloat_noopt
2079   \fi #1%
2080 }%
2081 \def\XINT_pfloat_noopt #1\xint_relax
2082 {%
2083   \expandafter\XINT_pfloat_a
2084   \romannumeral0\xintfloat [\XINTdigits]{#1};\XINTdigits.%
2085 }%
2086 \def\XINT_pfloat_opt [\xint_relax #1]%#2%
2087 {%
2088   \expandafter\XINT_pfloat_opt_a \the\numexpr #1.%
2089 }%
2090 \def\XINT_pfloat_opt_a #1.#2%
2091 {%
2092   \expandafter\XINT_pfloat_a\romannumeral0\xintfloat [#1]{#2};#1.%
2093 }%
2094 \def\XINT_pfloat_a #1%
2095 {%
2096   \xint_UDzerominusfork
2097     #1-\XINT_pfloat_zero
2098     0#1\XINT_pfloat_neg
2099     0-{\XINT_pfloat_pos #1}%
2100   \krof
2101 }%
2102 \def\XINT_pfloat_zero #1;#2.{ 0.}%
2103 \def\XINT_pfloat_neg
2104   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_pfloat_pos }%
2105 \def\XINT_pfloat_pos #1e#2;#3.%
2106 {%
2107   \ifnum #2>\xint_c_v \xint_dothis\XINT_pfloat_no\fi
2108   \ifnum #2<-\xint_c_v \xint_dothis\XINT_pfloat_no\fi
2109   \ifnum #2<\xint_c_ \xint_dothis\XINT_pfloat_N\fi
2110   \ifnum #2>\numexpr #3-\xint_c_i\relax \xint_dothis\XINT_pfloat_Ps\fi
2111   \xint_orthat\XINT_pfloat_P #1e#2;%
2112 }%
2113 \def\XINT_pfloat_no #1;{ #1}%
2114 \def\XINT_pfloat_N #1#2.#3e#4;%
2115 {%
2116   \csname XINT_pfloat_N\romannumeral-#4\endcsname #2#10.#3;%
2117 }%
2118 \def\XINT_pfloat_Ni #1#2#3.#4;{ #2.#1#4}%
2119 \def\XINT_pfloat_Nii #1#2#3.#4;{ 0.#2#1#4}%
2120 \def\XINT_pfloat_Niii#1#2#3.#4;{ 0.0#2#1#4}%
2121 \def\XINT_pfloat_Niv #1#2#3.#4;{ 0.00#2#1#4}%
2122 \def\XINT_pfloat_Nv #1#2#3.#4;{ 0.000#2#1#4}%

```

```

2123 \def\XINT_pfloat_P #1#2.#3e#4;%
2124 {%
2125   \csname XINT_pfloat_P\romannumeral#4\endcsname #3.#1#2;%
2126 }%
2127 \def\XINT_pfloat_P_ #1.#2;{ #2.#1}%
2128 \def\XINT_pfloat_P_i #1#2.#3;{ #3#1.#2}%
2129 \def\XINT_pfloat_P_ii #1#2#3.#4;{ #4#1#2.#3}%
2130 \def\XINT_pfloat_P_iii#1#2#3#4.#5;{ #5#1#2#3.#4}%
2131 \def\XINT_pfloat_P_iv #1#2#3#4#5.#6;{ #6#1#2#3#4.#5}%
2132 \def\XINT_pfloat_P_v #1#2#3#4#5#6.#7;{ #7#1#2#3#4#5.#6}%
2133 \def\XINT_pfloat_Ps #1#2.#3e#4;%
2134 {%
2135   \csname XINT_pfloat_Ps\romannumeral#4\endcsname #300000.#1#2;%
2136 }%
2137 \def\XINT_pfloat_Psi #1#2.#3;{ #3#1.}%
2138 \def\XINT_pfloat_Psii #1#2#3.#4;{ #4#1#2.}%
2139 \def\XINT_pfloat_Psiii#1#2#3#4.#5;{ #5#1#2#3.}%
2140 \def\XINT_pfloat_Psiv #1#2#3#4#5.#6;{ #6#1#2#3#4.}%
2141 \def\XINT_pfloat_Psv #1#2#3#4#5#6.#7;{ #7#1#2#3#4#5.}%

```

### 7.63 \XINTinFloatFracdigits

1.09i, for `frac` function in `\xintfloatexpr`. This version computes exactly from the input the fractional part and then only converts it into a float with the asked-for number of digits. I will have to think it again some day, certainly.

1.1 removes optional argument for which there was anyhow no interface, for technical reasons having to do with `\xintNewExpr`.

1.1a renames the macro as `\XINTinFloatFracdigits` (from `\XINTinFloatFrac`) to be synchronous with the `\XINTinFloatSqrt` and `\XINTinFloat` habits related to `\xintNewExpr` problems.

Note to myself: I still have to rethink the whole thing about what is the best to do, the initial way of going through `\xintfrac` was just a first implementation.

```

2142 \def\XINTinFloatFracdigits {\romannumeral0\XINTinfloatfracdigits }%
2143 \def\XINTinfloatfracdigits #1%
2144 {%
2145   \expandafter\XINT_infloatfracdg_a\expandafter {\romannumeral0\xintfrac{#1}}%
2146 }%
2147 \def\XINT_infloatfracdg_a {\XINTinfloat [\XINTdigits]}%

```

### 7.64 \xintFloatAdd, \XINTinFloatAdd

First included in release 1.07.

1.09ka improved a bit the efficiency. However the `add`, `sub`, `mul`, `div` routines were provisory and supposed to be revised soon.

Which didn't happen until 1.2f. Now, the inputs are first rounded to `P` digits, not `P+2` as earlier.

```

2148 \def\xintFloatAdd {\romannumeral0\xintfloatadd }%
2149 \def\xintfloatadd #1{\XINT_fladd_chkopt \xintfloat #1\xint_relax }%
2150 \def\XINTinFloatAdd {\romannumeral0\XINTinfloatadd }%
2151 \def\XINTinfloatadd #1{\XINT_fladd_chkopt \XINTinfloat #1\xint_relax }%
2152 \def\XINT_fladd_chkopt #1#2%
2153 {%
2154   \ifx [#2\expandafter\XINT_fladd_opt

```

```

2155     \else\expandafter\XINT_fladd_noopt
2156     \fi #1#2%
2157 }%
2158 \def\XINT_fladd_noopt #1#2\xint_relax #3%
2159 {%
2160     #1[\XINTdigits]%
2161     {\expandafter\XINT_FL_add_a
2162     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{#3}}%
2163 }%
2164 \def\XINT_fladd_opt #1[\xint_relax #2]#3#4%
2165 {%
2166     \expandafter\XINT_fladd_opt_a\the\numexpr #2.#1%
2167 }%
2168 \def\XINT_fladd_opt_a #1.#2#3#4%
2169 {%
2170     #2[#1]{\expandafter\XINT_FL_add_a\romannumeral0\XINTinfloat[#1]{#3}#1.{#4}}%
2171 }%
2172 \def\XINT_FL_add_a #1%
2173 {%
2174     \xint_gob_til_zero #1\XINT_FL_add_zero 0\XINT_FL_add_b #1%
2175 }%
2176 \def\XINT_FL_add_zero #1.#2{#2}%
2177 \def\XINT_FL_add_b #1]#2.#3%
2178 {%
2179     \expandafter\XINT_FL_add_c\romannumeral0\XINTinfloat[#2]{#3}#2.#1}%
2180 }%
2181 \def\XINT_FL_add_c #1%
2182 {%
2183     \xint_gob_til_zero #1\XINT_FL_add_zero 0\XINT_FL_add_d #1%
2184 }%
2185 \def\XINT_FL_add_d #1[#2]#3.#4[#5]%
2186 {%
2187     \ifnum\numexpr #2-#3-#5>\xint_c\xint_dothis\xint_firstoftwo\fi
2188     \ifnum\numexpr #5-#3-#2>\xint_c\xint_dothis\xint_secondoftwo\fi
2189     \xint_orthat\xintAdd {#1[#2]}{#4[#5]}%
2190 }%

```

## 7.65 \xintFloatSub, \XINTinFloatSub

First done 1.07.

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

```

2191 \def\xintFloatSub {\romannumeral0\xintfloatsub }%
2192 \def\xintfloatsub #1{\XINT_fsub_chkopt \xintfloat #1\xint_relax }%
2193 \def\XINTinFloatSub {\romannumeral0\XINTinfloatsub }%
2194 \def\XINTinfloatsub #1{\XINT_fsub_chkopt \XINTinfloat #1\xint_relax }%
2195 \def\XINT_fsub_chkopt #1#2%
2196 {%
2197     \ifx [#2\expandafter\XINT_fsub_opt
2198     \else\expandafter\XINT_fsub_noopt
2199     \fi #1#2%
2200 }%
2201 \def\XINT_fsub_noopt #1#2\xint_relax #3%

```

```

2202 {%
2203     #1[\XINTdigits]%
2204     {\expandafter\XINT_FL_add_a
2205       \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{\xintOpp{#3}}}%
2206 }%
2207 \def\XINT_fsub_opt #1[\xint_relax #2]%#3#4%
2208 {%
2209     \expandafter\XINT_fsub_opt_a\the\numexpr #2.#1%
2210 }%
2211 \def\XINT_fsub_opt_a #1.#2#3#4%
2212 {%
2213     #2[#1]{\expandafter\XINT_FL_add_a\romannumeral0\XINTinfloat[#1]{#3}#1.{\xintOpp{#4}}}%
2214 }%

```

## 7.66 \xintFloatMul, \XINTinFloatMul

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

### 1.07.

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

```

2215 \def\xintFloatMul {\romannumeral0\xintfloatmul }%
2216 \def\xintfloatmul #1{\XINT_fmul_chkopt \xintfloat #1\xint_relax }%
2217 \def\XINTinFloatMul {\romannumeral0\XINTinfloatmul }%
2218 \def\XINTinfloatmul #1{\XINT_fmul_chkopt \XINTinfloat #1\xint_relax }%
2219 \def\XINT_fmul_chkopt #1#2%
2220 {%
2221     \ifx [#2\expandafter\XINT_fmul_opt
2222       \else\expandafter\XINT_fmul_noopt
2223       \fi #1#2%
2224 }%
2225 \def\XINT_fmul_noopt #1#2\xint_relax #3%
2226 {%
2227     #1[\XINTdigits]%
2228     {\expandafter\XINT_FL_mul_a
2229       \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{#3}}%
2230 }%
2231 \def\XINT_fmul_opt #1[\xint_relax #2]%#3#4%
2232 {%
2233     \expandafter\XINT_fmul_opt_a\the\numexpr #2.#1%
2234 }%
2235 \def\XINT_fmul_opt_a #1.#2#3#4%
2236 {%
2237     #2[#1]{\expandafter\XINT_FL_mul_a\romannumeral0\XINTinfloat[#1]{#3}#1.{#4}}%
2238 }%
2239 \def\XINT_FL_mul_a #1[#2]#3.#4%
2240 {%
2241     \expandafter\XINT_FL_mul_b\romannumeral0\XINTinfloat [#3]{#4}#1[#2]%
2242 }%
2243 \def\XINT_FL_mul_b #1[#2]#3[#4]{\xintE{\xintiiMul {#3}{#1}}{#4+#2}}%

```

**7.67  $\backslash$ xintFloatDiv,  $\backslash$ XINTinFloatDiv**

1.07.

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

```

2244 \def\xintFloatDiv {\romannumeral0\xintfloatdiv }%
2245 \def\xintfloatdiv #1{\XINT_fldiv_chkopt \xintfloat #1\xint_relax }%
2246 \def\XINTinFloatDiv {\romannumeral0\XINTinfloatdiv }%
2247 \def\XINTinfloatdiv #1{\XINT_fldiv_chkopt \XINTinfloat #1\xint_relax }%
2248 \def\XINT_fldiv_chkopt #1#2%
2249 {%
2250   \ifx [#2\expandafter\XINT_fldiv_opt
2251     \else\expandafter\XINT_fldiv_noopt
2252   \fi #1#2%
2253 }%
2254 \def\XINT_fldiv_noopt #1#2\xint_relax #3%
2255 {%
2256   #1[\XINTdigits]%
2257   {\expandafter\XINT_FL_div_a
2258     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{#3}}%
2259 }%
2260 \def\XINT_fldiv_opt #1[\xint_relax #2]%#3#4%
2261 {%
2262   \expandafter\XINT_fldiv_opt_a\the\numexpr #2.#1%
2263 }%
2264 \def\XINT_fldiv_opt_a #1.#2#3#4%
2265 {%
2266   #2[#1]{\expandafter\XINT_FL_div_a\romannumeral0\XINTinfloat[#1]{#3}#1.{#4}}%
2267 }%
2268 \def\XINT_FL_div_a #1[#2]#3.#4%
2269 {%
2270   \expandafter\XINT_FL_div_b\romannumeral0\XINTinfloat[#3]{#4}#1[#2]%
2271 }%
2272 \def\XINT_FL_div_b #1[#2]#3[#4]{\xintE{#3/#1}{#4-#2}}%
2273 % \subsection{\csh{xintFloatPow}, \csh{XINTinFloatPow}}
2274 % \lverb|1.07: initial version. 1.09j has re-organized the core loop.
2275 %
2276 % 2015/12/07. I have hesitated to maintain the mapping of ^ in expressions to
2277 % \xintFloatPow rather than \xintFloatPower. But for 1.234567890123456 to the
2278 % power 2145678912 with P=16, using Pow rather than Power seems to bring only
2279 % about 5$char$37$space gain.
2280 %
2281 % This routine requires the exponent x to be compatible with \numexpr parsing.
2282 %
2283 % 1.2f has rewritten the code for better efficiency. Also, now the argument A
2284 % for A^x is first rounded to P digits before switching to the increased
2285 % working precision (which depends upon x).
2286 %
2287 % |
2288 % \begin{macrocode}
2289 \def\xintFloatPow {\romannumeral0\xintfloatpow}%
2290 \def\xintfloatpow #1{\XINT_flpow_chkopt \xintfloat #1\xint_relax }%
2291 \def\XINTinFloatPow {\romannumeral0\XINTinfloatpow}%

```

```

2292 \def\XINTinfloatpow #1{\XINT_flpow_chkopt \XINTinfloat #1\xint_relax }%
2293 \def\XINT_flpow_chkopt #1#2%
2294 {%
2295   \ifx [#2\expandafter\XINT_flpow_opt
2296   \else\expandafter\XINT_flpow_noopt
2297   \fi
2298   #1#2%
2299 }%
2300 \def\XINT_flpow_noopt #1#2\xint_relax #3%
2301 {%
2302   \expandafter\XINT_flpow_checkB_a
2303   \the\numexpr #3.\XINTdigits.{#2}{#1[\XINTdigits]]}%
2304 }%
2305 \def\XINT_flpow_opt #1[\xint_relax #2]%
2306 {%
2307   \expandafter\XINT_flpow_opt_a\the\numexpr #2.#1%
2308 }%
2309 \def\XINT_flpow_opt_a #1.#2#3#4%
2310 {%
2311   \expandafter\XINT_flpow_checkB_a\the\numexpr #4.#1.{#3}{#2[#1]}%
2312 }%
2313 \def\XINT_flpow_checkB_a #1%
2314 {%
2315   \xint_UDzerominusfork
2316   #1-\XINT_flpow_BisZero
2317   0#1{\XINT_flpow_checkB_b -}%
2318   0-{\XINT_flpow_checkB_b }{#1}%
2319   \krof
2320 }%
2321 \def\XINT_flpow_BisZero .#1.#2#3{#3{1[0]}}%
2322 \def\XINT_flpow_checkB_b #1#2.#3.%
2323 {%
2324   \expandafter\XINT_flpow_checkB_c
2325   \the\numexpr\xintLength{#2}+\xint_c_iii.#3.#2.{#1}%
2326 }%
2327 \def\XINT_flpow_checkB_c #1.#2.%
2328 {%
2329   \expandafter\XINT_flpow_checkB_d\the\numexpr#1+#2.#1.#2.%
2330 }%

```

1.2f rounds input to P digits, first.

```

2331 \def\XINT_flpow_checkB_d #1.#2.#3.#4.#5#6%
2332 {%
2333   \expandafter \XINT_flpow_aa
2334   \romannumeral0\XINTinfloat [#3]{#6}{#2}{#1}{#4}{#5}%
2335 }%
2336 \def\XINT_flpow_aa #1[#2]#3%
2337 {%
2338   \expandafter\XINT_flpow_ab\the\numexpr #2-#3\expandafter.%
2339   \romannumeral0\XINT_dsx_addzerosnofuss {#3}{#1}.#1.%
2340 }%
2341 \def\XINT_flpow_ab #1.#2.#3.{\XINT_flpow_a #3#2[#1]}%

```

## 7 Package *xintfrac* implementation

```

2342 \def\XINT_flpow_a #1%
2343 {%
2344   \xint_UDzerominusfork
2345   #1-\XINT_flpow_zero
2346   0#1{\XINT_flpow_b \iftrue}%
2347   0-{\XINT_flpow_b \iffalse#1}%
2348   \krof
2349 }%
2350 \def\XINT_flpow_zero #1[#2]#3#4#5#6%
2351 {%
2352   \if 1#51\xint_afterfi {#6{0[0]}}\else
2353   \xint_afterfi {\xintError:DivisionByZero #6{1[2147483648]}}\fi
2354 }%
2355 \def\XINT_flpow_b #1#2[#3]#4#5%
2356 {%
2357   \XINT_flpow_loopI #5.#3.#2.#4.{#1\ifodd #5 \xint_c_i\fi\fi}%
2358 }%
2359 \def\XINT_flpow_truncate #1.#2.#3.%
2360 {%
2361   \expandafter\XINT_flpow_truncate_a
2362   \romannumeral0\XINT_split_fromleft_loop #3.{#2\W\W\W\W\W\W\W\W\Z
2363   #1.#3.%
2364 }%
2365 \def\XINT_flpow_truncate_a #1.#2.#3.{#3+\xintLength{#2}.#1.}%
2366 \def\XINT_flpow_loopI #1.%
2367 {%
2368   \ifnum #1=\xint_c_i\expandafter\XINT_flpow_ItoIII\fi
2369   \ifodd #1
2370     \expandafter\XINT_flpow_loopI_odd
2371   \else
2372     \expandafter\XINT_flpow_loopI_even
2373   \fi
2374   #1.%
2375 }%
2376 \def\XINT_flpow_ItoIII\ifodd #1\fi #2.#3.#4.#5.#6%
2377 {%
2378   \expandafter\XINT_flpow_III\the\numexpr #6+\xint_c_.#3.#4.#5.%
2379 }%
2380 \def\XINT_flpow_loopI_even #1.#2.#3.%#4.%
2381 {%
2382   \expandafter\XINT_flpow_loopI
2383   \the\numexpr #1/\xint_c_ii\expandafter.%
2384   \the\numexpr\expandafter\XINT_flpow_truncate
2385   \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.%
2386 }%
2387 \def\XINT_flpow_loopI_odd #1.#2.#3.#4.%
2388 {%
2389   \expandafter\XINT_flpow_loopII
2390   \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter.%
2391   \the\numexpr\expandafter\XINT_flpow_truncate
2392   \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#2.#3.%
2393 }%

```



## 7 Package *xintfrac* implementation

```

2394 \def\XINT_flpow_loopII #1.%
2395 {%
2396   \ifnum #1 = \xint_c_i\expandafter\XINT_flpow_IItoIII\fi
2397   \ifodd #1
2398     \expandafter\XINT_flpow_loopII_odd
2399   \else
2400     \expandafter\XINT_flpow_loopII_even
2401   \fi
2402   #1.%
2403 }%
2404 \def\XINT_flpow_loopII_even #1.#2.#3.##4.%
2405 {%
2406   \expandafter\XINT_flpow_loopII
2407   \the\numexpr #1/\xint_c_ii\expandafter.%
2408   \the\numexpr\expandafter\XINT_flpow_truncate
2409   \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.%
2410 }%
2411 \def\XINT_flpow_loopII_odd #1.#2.#3.#4.#5.#6.%
2412 {%
2413   \expandafter\XINT_flpow_loopII_odda
2414   \the\numexpr\expandafter\XINT_flpow_truncate
2415   \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2416   #1.#2.#3.%
2417 }%
2418 \def\XINT_flpow_loopII_odda #1.#2.#3.#4.#5.#6.%
2419 {%
2420   \expandafter\XINT_flpow_loopII
2421   \the\numexpr #4/\xint_c_ii-\xint_c_i\expandafter.%
2422   \the\numexpr\expandafter\XINT_flpow_truncate
2423   \the\numexpr\xint_c_ii*#5\expandafter.\romannumeral0\xintiisqr{#6}.#3.%
2424   #1.#2.%
2425 }%
2426 \def\XINT_flpow_IItoIII\ifodd #1\fi #2.#3.#4.#5.#6.#7.#8%
2427 {%
2428   \expandafter\XINT_flpow_III\the\numexpr #8+\xint_c_\expandafter.%
2429   \the\numexpr\expandafter\XINT_flpow_truncate
2430   \the\numexpr#3+#6\expandafter.\romannumeral0\xintiimul{#4}{#7}.#5.%
2431 }%

```

This ending is common with `\xintFloatPower`. In the case of negative exponent we will inverse the Q-digits mantissa, keeping Q significant digits (exceptionally  $10^Q$ ) before the final rounding to P digits. Here Q is working precision. Releases prior to 1.2f trusted the final inverse to `\xintFloat` on output but this worked only with P+2 digits on denominator. Enough for 0.1 ulp extra error, but as our goal is to get <0.6ulp, and there is already 0.5ulp from rounding error, this was not enough. When `\xintFloat` will achieve correct rounding for arbitrary fractions, the step here will not be needed.

```

2432 \def\XINT_flpow_III #1.#2.#3.#4.#5%
2433 {%
2434   \expandafter\XINT_flpow_IIIend
2435   \xint_UDsignfork
2436   #5{\xintNum{1/#3[\xint_c_ii*#4-\xint_c_i]][\xint_c_i-\xint_c_ii*#4-#2]}}%
2437   -{\xintNum{#3[#2]}}%

```

```

2438 \krof #1%
2439 }%
2440 \def\XINT_flpow_IIIend #1#2#3%
2441 {#3{\if#21\xint_afterfi{\expandafter-\romannumeral`&&@\fi#1}}}%

```

## 7.68 \xintFloatPower, \XINTinFloatPower

1.07. The core loop has been re-organized in 1.09j for some slight efficiency gain. The exponent B is given to \xintNum. The ^ in expressions is mapped to this routine.

Same modifications as in \xintFloatPow for 1.2f.

1.2f adds a special macro for allowing half-integral exponents for use with ^ within \xintfloat-texpr. The exponent will be first truncated to either an integer or an half-integer.

```

2442 \def\xintFloatPower {\romannumeral0\xintfloatpower}%
2443 \def\xintfloatpower #1{\XINT_flpower_chkopt \xintfloat #1\xint_relax }%
2444 \def\XINTinFloatPower {\romannumeral0\XINTinfloatpower }%
2445 \def\XINTinfloatpower #1{\XINT_flpower_chkopt \XINTinfloat #1\xint_relax }%
2446 \def\XINTinFloatPowerH {\romannumeral0\XINTinfloatpowerH }%
2447 \def\XINTinfloatpowerH #1#2%
2448 {\expandafter\XINT_flpowerh_a \romannumeral0\xinttrunc 1{#2}.0;%
2449 \XINTdigits.{#1}{\XINTinfloat[\XINTdigits]}}}%
2450 \def\XINT_flpowerh_a #1.#2%
2451 {%
2452 \ifnum#2>\xint_c_iv\xint_dothis\XINT_flpowerh_b\fi
2453 \xint_orthat\XINT_flpowerh_i #1.#2%
2454 }%
2455 \def\XINT_flpowerh_i #1.#2;%
2456 {\expandafter\XINT_flpower_checkB_a\romannumeral0\xintinum{#1}.}%
2457 \def\XINT_flpowerh_b #1%
2458 {%
2459 \if#1-\xint_dothis\XINT_flpowerh_bneg\fi
2460 \xint_orthat{\XINT_flpowerh_bpos #1}%
2461 }%
2462 \def\XINT_flpowerh_bpos #1.#2;\XINTdigits.#3#4%
2463 {%
2464 \expandafter\XINT_flpower_checkB_a
2465 \romannumeral0\xintinc{\xintDouble{#1}}.}%
2466 \XINTdigits.{#3}{\XINTinfloatsqrt[\XINTdigits]}}}%
2467 }%
2468 \def\XINT_flpowerh_bneg #1.#2;\XINTdigits.#3#4%
2469 {%
2470 \expandafter\XINT_flpower_checkB_a
2471 \expandafter-\romannumeral0\xintinc{\xintDouble{#1}}.}%
2472 \XINTdigits.{#3}{\XINTinfloatsqrt[\XINTdigits]}}}%
2473 }%
2474 \def\XINT_flpower_chkopt #1#2%
2475 {%
2476 \ifx [#2\expandafter\XINT_flpower_opt
2477 \else\expandafter\XINT_flpower_noopt
2478 \fi
2479 #1#2%
2480 }%
2481 \def\XINT_flpower_noopt #1#2\xint_relax #3%

```

```

2482 {%
2483   \expandafter\XINT_flpower_checkB_a
2484   \romannumeral0\xintnum{#3}.\XINTdigits.{#2}{#1[\XINTdigits]}%
2485 }%
2486 \def\XINT_flpower_opt #1[\xint_relax #2]%
2487 {%
2488   \expandafter\XINT_flpower_opt_a\the\numexpr #2.#1%
2489 }%
2490 \def\XINT_flpower_opt_a #1.#2#3#4%
2491 {%
2492   \expandafter\XINT_flpower_checkB_a
2493   \romannumeral0\xintnum{#4}.#1.{#3}{#2[#1]}%
2494 }%
2495 \def\XINT_flpower_checkB_a #1%
2496 {%
2497   \xint_UDzerominusfork
2498   #1-\XINT_flpower_BisZero
2499   0#1{\XINT_flpower_checkB_b -}%
2500   0-{\XINT_flpower_checkB_b }{#1}%
2501   \krof
2502 }%
2503 \def\XINT_flpower_BisZero .#1.#2#3{#3{1[0]}}%
2504 \def\XINT_flpower_checkB_b #1#2.#3.%
2505 {%
2506   \expandafter\XINT_flpower_checkB_c
2507   \the\numexpr\xintLength{#2}+\xint_c_iii.#3.#2.{#1}%
2508 }%
2509 \def\XINT_flpower_checkB_c #1.#2.%
2510 {%
2511   \expandafter\XINT_flpower_checkB_d\the\numexpr#1+#2.#1.#2.%
2512 }%
2513 \def\XINT_flpower_checkB_d #1.#2.#3.#4.#5#6%
2514 {%
2515   \expandafter \XINT_flpower_aa
2516   \romannumeral0\XINTinfloat [#3]{#6}{#2}{#1}{#4}{#5}%
2517 }%
2518 \def\XINT_flpower_aa #1[#2]#3%
2519 {%
2520   \expandafter\XINT_flpower_ab\the\numexpr #2-#3\expandafter.%
2521   \romannumeral0\XINT_dsx_addzerosnofuss {#3}{#1}.#1.%
2522 }%
2523 \def\XINT_flpower_ab #1.#2.#3.{\XINT_flpower_a #3#2[#1]}%
2524 \def\XINT_flpower_a #1%
2525 {%
2526   \xint_UDzerominusfork
2527   #1-\XINT_flpow_zero
2528   0#1{\XINT_flpower_b \iftrue}%
2529   0-{\XINT_flpower_b \iffalse#1}%
2530   \krof
2531 }%
2532 \def\XINT_flpower_b #1#2[#3]#4#5%
2533 {%

```

## 7 Package *xintfrac* implementation

```

2534 \XINT_flpower_loopI #5.#3.#2.#4.{#1\xintiiOdd{#5}\fi}%
2535 }%
2536 \def\XINT_flpower_loopI #1.%
2537 {%
2538 \if1\XINT_isOne {#1}\xint_dothis\XINT_flpower_ItoIII\fi
2539 \if1\xintiiOdd {#1}\xint_dothis{\expandafter\XINT_flpower_loopI_odd}\fi
2540 \xint_orthat{\expandafter\XINT_flpower_loopI_even}%
2541 \romannumeral0\xinthalff{#1}.%
2542 }%
2543 \def\XINT_flpower_ItoIII #1.#2.#3.#4.#5%
2544 {%
2545 \expandafter\XINT_flpow_III\the\numexpr #5+\xint_c_.#2.#3.#4.%
2546 }%
2547 \def\XINT_flpower_loopI_even #1.#2.#3.#4.%
2548 {%
2549 \expandafter\XINT_flpower_toloopI
2550 \the\numexpr\expandafter\XINT_flpow_truncate
2551 \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#1.%
2552 }%
2553 \def\XINT_flpower_toloopI #1.#2.#3.#4.{\XINT_flpower_loopI #4.#1.#2.#3.}%
2554 \def\XINT_flpower_loopI_odd #1.#2.#3.#4.%
2555 {%
2556 \expandafter\XINT_flpower_toloopII
2557 \the\numexpr\expandafter\XINT_flpow_truncate
2558 \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.%
2559 #1.#2.#3.%
2560 }%
2561 \def\XINT_flpower_toloopII #1.#2.#3.#4.{\XINT_flpower_loopII #4.#1.#2.#3.}%
2562 \def\XINT_flpower_loopII #1.%
2563 {%
2564 \if1\XINT_isOne{#1}\xint_dothis\XINT_flpower_ItoIII\fi
2565 \if1\xintiiOdd{#1}\xint_dothis{\expandafter\XINT_flpower_loopII_odd}\fi
2566 \xint_orthat{\expandafter\XINT_flpower_loopII_even}%
2567 \romannumeral0\xinthalff{#1}.%
2568 }%
2569 \def\XINT_flpower_loopII_even #1.#2.#3.#4.%
2570 {%
2571 \expandafter\XINT_flpower_toloopII
2572 \the\numexpr\expandafter\XINT_flpow_truncate
2573 \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#1.%
2574 }%
2575 \def\XINT_flpower_loopII_odd #1.#2.#3.#4.#5.#6.%
2576 {%
2577 \expandafter\XINT_flpower_loopII_odda
2578 \the\numexpr\expandafter\XINT_flpow_truncate
2579 \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2580 #1.#2.#3.%
2581 }%
2582 \def\XINT_flpower_loopII_odda #1.#2.#3.#4.#5.#6.%
2583 {%
2584 \expandafter\XINT_flpower_toloopII
2585 \the\numexpr\expandafter\XINT_flpow_truncate

```

```

2586 \the\numexpr\xint_c_ii*#5\expandafter.\romannumeral0\xintiisqr{#6}.#3.%
2587 #4.#1.#2.%
2588 }%
2589 \def\XINT_flpower_IItoIII #1.#2.#3.#4.#5.#6.#7%
2590 {%
2591 \expandafter\XINT_flpow_III\the\numexpr #7+\xint_c_\expandafter.%
2592 \the\numexpr\expandafter\XINT_flpow_truncate
2593 \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2594 }%

```

## 7.69 \xintFloatFac, \XINTFloatFac

```

2595 \def\xintFloatFac {\romannumeral0\xintfloatfac}%
2596 \def\xintfloatfac #1{\XINT_flfac_chkopt \xintfloat #1\xint_relax }%
2597 \def\XINTinFloatFac {\romannumeral0\XINTinfloatfac }%
2598 \def\XINTinfloatfac #1{\XINT_flfac_chkopt \XINTinfloat #1\xint_relax }%
2599 \def\XINT_flfac_chkopt #1#2%
2600 {%
2601 \ifx [#2\expandafter\XINT_flfac_opt
2602 \else\expandafter\XINT_flfac_noopt
2603 \fi
2604 #1#2%
2605 }%
2606 \def\XINT_flfac_noopt #1#2\xint_relax
2607 {%
2608 \expandafter\XINT_FL_fac_fork_a
2609 \the\numexpr \xintNum{#2}.\xint_c_i \XINTdigits\XINT_FL_fac_out{#1[\XINTdigits]]}%
2610 }%
2611 \def\XINT_flfac_opt #1[\xint_relax #2]%
2612 {%
2613 \expandafter\XINT_flfac_opt_a\the\numexpr #2.#1%
2614 }%
2615 \def\XINT_flfac_opt_a #1.#2#3%
2616 {%
2617 \expandafter\XINT_FL_fac_fork_a\the\numexpr \xintNum{#3}.\xint_c_i {#1}\XINT_FL_fac_out{#2[#1]]}%
2618 }%
2619 \def\XINT_FL_fac_fork_a #1%
2620 {%
2621 \xint_UDzerominusfork
2622 #1-\XINT_FL_fac_iszero
2623 0#1\XINT_FL_fac_isneg
2624 0-{\XINT_FL_fac_fork_b #1}%
2625 \krof
2626 }%
2627 \def\XINT_FL_fac_iszero #1.#2#3#4#5{#5{1/1[0]}}%

```

1.2f XINT\_FL\_fac\_isneg returns 0, earlier versions used 1 here.

```

2628 \def\XINT_FL_fac_isneg #1.#2#3#4#5{\expandafter\xintError:FactorialOfNegative #5{0/1[0]}}%
2629 \def\XINT_FL_fac_fork_b #1.%
2630 {%
2631 \ifnum #1>\xint_c_x^viii mone\xint_dothis\XINT_FL_fac_toobig\fi
2632 \ifnum #1>\xint_c_x^iv\xint_dothis\XINT_FL_fac_vbig \fi
2633 \ifnum #1>465 \xint_dothis\XINT_FL_fac_big\fi

```

```

2634 \ifnum #1>101 \xint_dothis\xINT_FL_fac_med\fi
2635 \xint_orthat\xINT_FL_fac_small
2636 #1.%
2637 }%
2638 \def\xINT_FL_fac_toobig #1.#2#3#4#5{\expandafter\xintError:TooBigFactorial #5{1/1[0]}}%

```

Computations are done with  $Q$  blocks of eight digits. When a multiplication has a carry, hence creates  $Q+1$  blocks, the least significant one is dropped. The goal is to compute an approximate value  $X'$  to the exact value  $X$ , such that the final relative error  $(X-X')/X$  will be at most  $10^{-P-1}$  with  $P$  the desired precision. Then, when we round  $X'$  to  $X''$  with  $P$  significant digits, we can prove that the absolute error  $|X-X''|$  is bounded (strictly) by  $0.6 \text{ ulp}(X'')$ . (ulp= unit in the last (significant) place). Let  $N$  be the number of such operations, the formula for  $Q$  deduces from the previous explanations is that  $8Q$  should be at least  $P+9+k$ , with  $k$  the number of digits of  $N$  (in base 10). Note that 1.2 version used  $P+10+k$ , for 1.2f I reduced to  $P+9+k$ . Also,  $k$  should be the number of digits of the number  $N$  of multiplications done, hence for  $n \leq 10000$  we can take  $N=n/2$ , or  $N/3$ , or  $N/4$ . This is rounded above by `numexpr` and always an overestimate of the actual number of approximate multiplications done (the first ones are exact). (vérifier ce que je raconte, j'ai la flemme là).

We then want  $\text{ceil}((P+k+n)/8)$ . Using `\numexpr` rounding division (ARRRRRGGGHHHH), if  $m$  is a positive integer,  $\text{ceil}(m/8)$  can be computed as  $(m+3)/8$ . Thus with  $m=P+10+k$ , this gives  $Q < -(P+13+k)/8$ . The routine actually computes  $8(Q-1)$  for use in `\XINT_FL_fac_addzeros`.

With 1.2f the formula is  $m=P+9+k$ ,  $Q < -(P+12+k)/8$ , and we use now  $4=12-8$  rather than the earlier  $5=13-8$ . Whatever happens, the value computed in `\XINT_FL_fac_increaseP` is at least 8. There will always be an extra block.

Note: with Digits:=32; Maple gives for 200!:

```
> factorial(200.);
```

```
375
```

```
0.78865786736479050355236321393218 10
```

My 1.2f routine (and also 1.2) outputs:

```
7.8865786736479050355236321393219e374
```

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

```
7.886578673647905035523632139321850622951e374
```

Maple's result (contrarily to `xint`) is thus not the correct rounding but still it is less than  $0.6 \text{ ulp}$  wrong.

```

2639 \def\xINT_FL_fac_vbig
2640 {\expandafter\xINT_FL_fac_vbigloop_a
2641 \the\numexpr \XINT_FL_fac_increaseP \xint_c_i }%
2642 \def\xINT_FL_fac_big
2643 {\expandafter\xINT_FL_fac_bigloop_a
2644 \the\numexpr \XINT_FL_fac_increaseP \xint_c_ii }%
2645 \def\xINT_FL_fac_med
2646 {\expandafter\xINT_FL_fac_medloop_a
2647 \the\numexpr \XINT_FL_fac_increaseP \xint_c_iii }%
2648 \def\xINT_FL_fac_small
2649 {\expandafter\xINT_FL_fac_smallloop_a
2650 \the\numexpr \XINT_FL_fac_increaseP \xint_c_iv }%
2651 \def\xINT_FL_fac_increaseP #1#2.#3#4%
2652 {%
2653 #2\expandafter.\the\numexpr\xint_c_viii*%
2654 ((\xint_c_iv+#4+\expandafter\xINT_FL_fac_countdigits
2655 \the\numexpr #2/(#1*#3)\relax 87654321\Z)/\xint_c_viii).%
2656 }%
2657 \def\xINT_FL_fac_countdigits #1#2#3#4#5#6#7#8{\XINT_FL_fac_countdone }%

```

## 7 Package *xintfrac* implementation

```

2658 \def\XINT_FL_fac_countdone #1#2\Z {#1}%
2659 \def\XINT_FL_fac_out #1\Z![#2]#3{#3{\romannumeral0\XINT_mul_out
2660 #1\Z!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W [#2]}}%
2661 \def\XINT_FL_fac_vbigloop_a #1.#2.%
2662 {%
2663 \XINT_FL_fac_bigloop_a \xint_c_x^iv.#2.%
2664 {\expandafter\XINT_FL_fac_vbigloop_loop\the\numexpr 100010001\expandafter.%
2665 \the\numexpr \xint_c_x^viii+#1.}%
2666 }%
2667 \def\XINT_FL_fac_vbigloop_loop #1.#2.%
2668 {%
2669 \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2670 \expandafter\XINT_FL_fac_vbigloop_loop
2671 \the\numexpr #1+\xint_c_i\expandafter.%
2672 \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_mul #1!%
2673 }%
2674 \def\XINT_FL_fac_bigloop_a #1.%
2675 {%
2676 \expandafter\XINT_FL_fac_bigloop_b \the\numexpr
2677 #1+\xint_c_i-\xint_c_ii*((#1-464)/\xint_c_ii).#1.%
2678 }%
2679 \def\XINT_FL_fac_bigloop_b #1.#2.#3.%
2680 {%
2681 \expandafter\XINT_FL_fac_medloop_a
2682 \the\numexpr #1-\xint_c_i.#3.{\XINT_FL_fac_bigloop_loop #1.#2.}%
2683 }%
2684 \def\XINT_FL_fac_bigloop_loop #1.#2.%
2685 {%
2686 \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2687 \expandafter\XINT_FL_fac_bigloop_loop
2688 \the\numexpr #1+\xint_c_ii\expandafter.%
2689 \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_bigloop_mul #1!%
2690 }%
2691 \def\XINT_FL_fac_bigloop_mul #1!%
2692 {%
2693 \expandafter\XINT_FL_fac_mul
2694 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2695 }%
2696 \def\XINT_FL_fac_medloop_a #1.%
2697 {%
2698 \expandafter\XINT_FL_fac_medloop_b
2699 \the\numexpr #1+\xint_c_i-\xint_c_iii*((#1-100)/\xint_c_iii).#1.%
2700 }%
2701 \def\XINT_FL_fac_medloop_b #1.#2.#3.%
2702 {%
2703 \expandafter\XINT_FL_fac_smallloop_a
2704 \the\numexpr #1-\xint_c_i.#3.{\XINT_FL_fac_medloop_loop #1.#2.}%
2705 }%
2706 \def\XINT_FL_fac_medloop_loop #1.#2.%
2707 {%
2708 \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2709 \expandafter\XINT_FL_fac_medloop_loop

```

## 7 Package *xintfrac* implementation

```

2710 \the\numexpr #1+\xint_c_iii\expandafter.%
2711 \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_medloop_mul #1!%
2712 }%
2713 \def\XINT_FL_fac_medloop_mul #1!%
2714 {%
2715 \expandafter\XINT_FL_fac_mul
2716 \the\numexpr
2717 \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2718 }%
2719 \def\XINT_FL_fac_smallloop_a #1.%
2720 {%
2721 \csname
2722 XINT_FL_fac_smallloop_\the\numexpr #1-\xint_c_iv*(#1/\xint_c_iv)\relax
2723 \endcsname #1.%
2724 }%
2725 \expandafter\def\csname XINT_FL_fac_smallloop_1\endcsname #1.#2.%
2726 {%
2727 \XINT_FL_fac_addzeros #2.100000001!.{2.#1.}{#2}%
2728 }%
2729 \expandafter\def\csname XINT_FL_fac_smallloop_-2\endcsname #1.#2.%
2730 {%
2731 \XINT_FL_fac_addzeros #2.100000002!.{3.#1.}{#2}%
2732 }%
2733 \expandafter\def\csname XINT_FL_fac_smallloop_-1\endcsname #1.#2.%
2734 {%
2735 \XINT_FL_fac_addzeros #2.100000006!.{4.#1.}{#2}%
2736 }%
2737 \expandafter\def\csname XINT_FL_fac_smallloop_0\endcsname #1.#2.%
2738 {%
2739 \XINT_FL_fac_addzeros #2.100000024!.{5.#1.}{#2}%
2740 }%
2741 \def\XINT_FL_fac_addzeros #1.%
2742 {%
2743 \ifnum #1=\xint_c_viii \expandafter\XINT_FL_fac_addzeros_exit\fi
2744 \expandafter\XINT_FL_fac_addzeros\the\numexpr #1-\xint_c_viii.100000000!%
2745 }%

```

We will manipulate by successive *\*small\** multiplications *Q* blocks *1<8d*!, terminated by *1\Z*!. We need a custom small multiplication which tells us when it has create a new block, and the least significant one should be dropped.

```

2746 \def\XINT_FL_fac_addzeros_exit #1.#2.#3#4{\XINT_FL_fac_smallloop_loop #3#21\Z![-#4]}%
2747 \def\XINT_FL_fac_smallloop_loop #1.#2.%
2748 {%
2749 \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2750 \expandafter\XINT_FL_fac_smallloop_loop
2751 \the\numexpr #1+\xint_c_iv\expandafter.%
2752 \the\numexpr #2\expandafter.\romannumeral0\XINT_FL_fac_smallloop_mul #1!%
2753 }%
2754 \def\XINT_FL_fac_smallloop_mul #1!%
2755 {%
2756 \expandafter\XINT_FL_fac_mul
2757 \the\numexpr
2758 \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%

```



## 7 Package *xintfrac* implementation

```

2759 }%[[
2760 \def\XINT_FL_fac_loop_exit #1!#2]#3{#3#2}}%
2761 \def\XINT_FL_fac_mul 1#1!%
2762   {\expandafter\XINT_FL_fac_mul_a\the\numexpr\XINT_FL_fac_smallmul 10!{#1}}%
2763 \def\XINT_FL_fac_mul_a #1-#2%
2764 {%
2765   \if#21\xint_afterfi{\expandafter\space\xint_gob_til_exclam}\else
2766   \expandafter\space\fi #11\Z!%
2767 }%
2768 \def\XINT_FL_fac_minimulwc_a #1#2#3#4#5!#6#7#8#9%
2769 {%
2770   \XINT_FL_fac_minimulwc_b {#1#2#3#4}{#5}{#6#7#8#9}%
2771 }%
2772 \def\XINT_FL_fac_minimulwc_b #1#2#3#4!#5%
2773 {%
2774   \expandafter\XINT_FL_fac_minimulwc_c
2775   \the\numexpr \xint_c_x^ix+#5+#2*#4.{#1}{#2}{#3}{#4}}%
2776 }%
2777 \def\XINT_FL_fac_minimulwc_c 1#1#2#3#4#5#6.#7%
2778 {%
2779   \expandafter\XINT_FL_fac_minimulwc_d {#1#2#3#4#5}#7{#6}%
2780 }%
2781 \def\XINT_FL_fac_minimulwc_d #1#2#3#4#5%
2782 {%
2783   \expandafter\XINT_FL_fac_minimulwc_e
2784   \the\numexpr \xint_c_x^ix+#1+#2*#5+#3*#4.{#2}{#4}%
2785 }%
2786 \def\XINT_FL_fac_minimulwc_e 1#1#2#3#4#5#6.#7#8#9%
2787 {%
2788   1#6#9\expandafter!%
2789   \the\numexpr\expandafter\XINT_FL_fac_smallmul
2790   \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#7*#8!%
2791 }%
2792 \def\XINT_FL_fac_smallmul 1#1!#21#3!%
2793 {%
2794   \xint_gob_til_Z #3\XINT_FL_fac_smallmul_end\Z
2795   \XINT_FL_fac_minimulwc_a #2!#3!{#1}{#2}%
2796 }%

```

This is the crucial ending. I note that I used here an `\ifnum` test rather than the `gob_til_eightzeroes` thing. Actually for eight digits there is much less difference than for only four.

The "carry" situation is marked by a final `!-1` rather than `!-2` for no-carry. (a `\numexpr` must be stopped, and leaving a `-` as delimiter is good as it will not arise earlier.)

```

2797 \def\XINT_FL_fac_smallmul_end\Z\XINT_FL_fac_minimulwc_a #1!\Z!#2#3[#4]%
2798 {%
2799   \ifnum #2=\xint_c_
2800     \expandafter\xint_firstoftwo\else
2801     \expandafter\xint_secondoftwo
2802   \fi
2803   {-2\relax[#4]}%
2804   {1#2\expandafter!\expandafter-\expandafter1\expandafter
2805     [\the\numexpr #4+\xint_c_viii]}%
2806 }%

```

**7.70 \xintFloatPFactorial, \XINTinFloatPFactorial**

2015/11/29 for 1.2f. Partial factorial  $\text{pfactorial}(a,b)=(a+1)\dots b$ , only for non-negative integers with  $a\leq b<10^8$ .

```

2807 \def\xintFloatPFactorial {\romannumeral0\xintfloatpfactorial}%
2808 \def\xintfloatpfactorial #1{\XINT_flpfac_chkopt \xintfloat #1\xint_relax }%
2809 \def\XINTinFloatPFactorial {\romannumeral0\XINTinfloatpfactorial }%
2810 \def\XINTinfloatpfactorial #1{\XINT_flpfac_chkopt \XINTinfloat #1\xint_relax }%
2811 \def\XINT_flpfac_chkopt #1#2%
2812 {%
2813   \ifx [#2\expandafter\XINT_flpfac_opt
2814     \else\expandafter\XINT_flpfac_noopt
2815     \fi
2816     #1#2%
2817 }%
2818 \def\XINT_flpfac_noopt #1#2\xint_relax #3%
2819 {%
2820   \expandafter\XINT_FL_pfac_fork
2821   \the\numexpr \xintNum{#2}\expandafter.%
2822   \the\numexpr \xintNum{#3}.\xint_c_i{\XINTdigits}\{#1[\XINTdigits]}\}%
2823 }%
2824 \def\XINT_flpfac_opt #1[\xint_relax #2]%
2825 {%
2826   \expandafter\XINT_flpfac_opt_b\the\numexpr #2.#1%
2827 }%
2828 \def\XINT_flpfac_opt_b #1.#2#3#4%
2829 {%
2830   \expandafter\XINT_FL_pfac_fork
2831   \the\numexpr \xintNum{#3}\expandafter.%
2832   \the\numexpr \xintNum{#4}.\xint_c_i{\#1}\{#2[#1]}\}%
2833 }%
2834 \def\XINT_FL_pfac_fork #1#2.#3.%
2835 {%
2836   \if-#1\xint_dothis\XINT_FL_pfac_outofrange\fi
2837   \ifnum #1#2=#3 \xint_dothis\XINT_FL_pfac_one\fi
2838   \ifnum #1#2>#3 \xint_dothis\XINT_FL_pfac_outofrange\fi
2839   \ifnum #3>\xint_c_x^viii_mone
2840     \xint_dothis\XINT_FL_pfac_outofrange\fi
2841   \xint_orthat \XINT_FL_pfac_increaseP #1#2.#3.%
2842 }%
2843 \def\XINT_FL_pfac_outofrange #1.#2.#3#4#5{\xintError:OutOfRangePFac #5{0/1[0]}}%
2844 \def\XINT_FL_pfac_one #1.#2.#3#4#5{\#5{1/1[0]}}%

```

See the comments for \XINT\_FL\_pfac\_increaseP. Case of  $b=a+1$  should be filtered out perhaps. We only needed here to copy the \xintPFactorial macros and re-use \XINT\_FL\_fac\_mul/\XINT\_FL\_fac\_out. Had to modify a bit \XINT\_FL\_pfac\_addzeroes. We can enter here directly with #3 equal to specify the precision (the calculated value before final rounding has a relative error less than  $3\cdot 10^{-(\#4-1)}$ ), and #5 would hold the macro doing the final rounding (or truncating, if I make a FloatTrunc available) to a given number of digits, possibly not #4. By default the #3 is 1, but FloatBinomial calls it with #3=4.

```

2845 \def\XINT_FL_pfac_increaseP #1.#2.#3#4%
2846 {%

```

## 7 Package *xintfrac* implementation

```

2847 \expandafter\XINT_FL_pfac_a
2848 \the\numexpr \xint_c_viii*((\xint_c_iv+#4+\expandafter
2849 \XINT_FL_fac_countdigits\the\numexpr (#2-#1-\xint_c_i)%
2850 /\ifnum #2>\xint_c_x^iv #3\else(#3*\xint_c_ii)\fi\relax
2851 87654321\Z)/\xint_c_viii).#1.#2.%
2852 }%
2853 \def\XINT_FL_pfac_a #1.#2.#3.%
2854 {%
2855 \expandafter\XINT_FL_pfac_b\the\numexpr \xint_c_i+#2\expandafter.%
2856 \the\numexpr#3\expandafter.%
2857 \romannumeral0\XINT_FL_pfac_addzeroes #1.1000000001!1\Z![-#1]%
2858 }%
2859 \def\XINT_FL_pfac_addzeroes #1.%
2860 {%
2861 \ifnum #1=\xint_c_viii \expandafter\XINT_FL_pfac_addzeroes_exit\fi
2862 \expandafter\XINT_FL_pfac_addzeroes\the\numexpr #1-\xint_c_viii.100000000!%
2863 }%
2864 \def\XINT_FL_pfac_addzeroes_exit #1.{ }%
2865 \def\XINT_FL_pfac_b #1.%
2866 {%
2867 \ifnum #1>9999 \xint_dothis\XINT_FL_pfac_vbigloop \fi
2868 \ifnum #1>463 \xint_dothis\XINT_FL_pfac_bigloop \fi
2869 \ifnum #1>98 \xint_dothis\XINT_FL_pfac_medloop \fi
2870 \xint_orthat\XINT_FL_pfac_smallloop #1.%
2871 }%
2872 \def\XINT_FL_pfac_smallloop #1.#2.%
2873 {%
2874 \ifcase\numexpr #2-#1\relax
2875 \expandafter\XINT_FL_pfac_end_
2876 \or \expandafter\XINT_FL_pfac_end_i
2877 \or \expandafter\XINT_FL_pfac_end_ii
2878 \or \expandafter\XINT_FL_pfac_end_iii
2879 \else\expandafter\XINT_FL_pfac_smallloop_a
2880 \fi #1.#2.%
2881 }%
2882 \def\XINT_FL_pfac_smallloop_a #1.#2.%
2883 {%
2884 \expandafter\XINT_FL_pfac_smallloop_b
2885 \the\numexpr #1+\xint_c_iv\expandafter.%
2886 \the\numexpr #2\expandafter.%
2887 \romannumeral0\expandafter\XINT_FL_fac_mul
2888 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2889 }%
2890 \def\XINT_FL_pfac_smallloop_b #1.%
2891 {%
2892 \ifnum #1>98 \expandafter\XINT_FL_pfac_medloop \else
2893 \expandafter\XINT_FL_pfac_smallloop \fi #1.%
2894 }%
2895 \def\XINT_FL_pfac_medloop #1.#2.%
2896 {%
2897 \ifcase\numexpr #2-#1\relax
2898 \expandafter\XINT_FL_pfac_end_

```

## 7 Package *xintfrac* implementation

```

2899 \or \expandafter\XINT_FL_pfac_end_i
2900 \or \expandafter\XINT_FL_pfac_end_ii
2901 \else\expandafter\XINT_FL_pfac_medloop_a
2902 \fi #1.#2.%
2903 }%
2904 \def\XINT_FL_pfac_medloop_a #1.#2.%
2905 {%
2906 \expandafter\XINT_FL_pfac_medloop_b
2907 \the\numexpr #1+\xint_c_iii\expandafter.%
2908 \the\numexpr #2\expandafter.%
2909 \romannumeral0\expandafter\XINT_FL_fac_mul
2910 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2911 }%
2912 \def\XINT_FL_pfac_medloop_b #1.%
2913 {%
2914 \ifnum #1>463 \expandafter\XINT_FL_pfac_bigloop \else
2915 \expandafter\XINT_FL_pfac_medloop \fi #1.%
2916 }%
2917 \def\XINT_FL_pfac_bigloop #1.#2.%
2918 {%
2919 \ifcase\numexpr #2-#1\relax
2920 \expandafter\XINT_FL_pfac_end_
2921 \or \expandafter\XINT_FL_pfac_end_i
2922 \else\expandafter\XINT_FL_pfac_bigloop_a
2923 \fi #1.#2.%
2924 }%
2925 \def\XINT_FL_pfac_bigloop_a #1.#2.%
2926 {%
2927 \expandafter\XINT_FL_pfac_bigloop_b
2928 \the\numexpr #1+\xint_c_ii\expandafter.%
2929 \the\numexpr #2\expandafter.%
2930 \romannumeral0\expandafter\XINT_FL_fac_mul
2931 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2932 }%
2933 \def\XINT_FL_pfac_bigloop_b #1.%
2934 {%
2935 \ifnum #1>9999 \expandafter\XINT_FL_pfac_vbigloop \else
2936 \expandafter\XINT_FL_pfac_bigloop \fi #1.%
2937 }%
2938 \def\XINT_FL_pfac_vbigloop #1.#2.%
2939 {%
2940 \ifnum #2=#1
2941 \expandafter\XINT_FL_pfac_end_
2942 \else\expandafter\XINT_FL_pfac_vbigloop_a
2943 \fi #1.#2.%
2944 }%
2945 \def\XINT_FL_pfac_vbigloop_a #1.#2.%
2946 {%
2947 \expandafter\XINT_FL_pfac_vbigloop
2948 \the\numexpr #1+\xint_c_i\expandafter.%
2949 \the\numexpr #2\expandafter.%
2950 \romannumeral0\expandafter\XINT_FL_fac_mul

```

```

2951 \the\numexpr\xint_c_x^viii+#1!%
2952 }%
2953 \def\XINT_FL_pfac_end_iii #1.#2.%
2954 {%
2955 \expandafter\XINT_FL_fac_out
2956 \romannumeral0\expandafter\XINT_FL_fac_mul
2957 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2958 }%
2959 \def\XINT_FL_pfac_end_ii #1.#2.%
2960 {%
2961 \expandafter\XINT_FL_fac_out
2962 \romannumeral0\expandafter\XINT_FL_fac_mul
2963 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2964 }%
2965 \def\XINT_FL_pfac_end_i #1.#2.%
2966 {%
2967 \expandafter\XINT_FL_fac_out
2968 \romannumeral0\expandafter\XINT_FL_fac_mul
2969 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2970 }%
2971 \def\XINT_FL_pfac_end_ #1.#2.%
2972 {%
2973 \expandafter\XINT_FL_fac_out
2974 \romannumeral0\expandafter\XINT_FL_fac_mul
2975 \the\numexpr \xint_c_x^viii+#1!%
2976 }%

```

### 7.71 *\xintFloatBinomial*, *\XINTinFloatBinomial*

1.2f. We compute  $\text{binomial}(x,y)$  as  $\text{pfac}(x-y,x)/y!$ , where the numerator and denominator are computed with a relative error at most  $4 \cdot 10^{-P-2}$ , then rounded (once I have a float truncation, I will use truncation rather) to  $P+3$  digits, and finally the quotient is correctly rounded to  $P$  digits. This will guarantee that the exact value  $X$  differs from the computed one  $Y$  by at most  $0.6 \text{ ulp}(Y)$ . (2015/12/01).

```

2977 \def\xintFloatBinomial {\romannumeral0\xintfloatbinomial}%
2978 \def\xintfloatbinomial #1{\XINT_flbinom_chkopt \xintfloat #1\xint_relax }%
2979 \def\XINTinFloatBinomial {\romannumeral0\XINTinfloatbinomial }%
2980 \def\XINTinfloatbinomial #1{\XINT_flbinom_chkopt \XINTinfloat #1\xint_relax }%
2981 \def\XINT_flbinom_chkopt #1#2%
2982 {%
2983 \ifx [#2\expandafter\XINT_flbinom_opt
2984 \else\expandafter\XINT_flbinom_noopt
2985 \fi #1#2%
2986 }%
2987 \def\XINT_flbinom_noopt #1#2\xint_relax #3%
2988 {%
2989 \expandafter\XINT_FL_binom_a
2990 \the\numexpr\xintNum{#2}\expandafter.\the\numexpr\xintNum{#3}.\XINTdigits.#1%
2991 }%
2992 \def\XINT_flbinom_opt #1[\xint_relax #2]#3#4%
2993 {%
2994 \expandafter\XINT_FL_binom_a

```

```

2995 \the\numexpr\xintNum{#3}\expandafter.\the\numexpr\xintNum{#4}\expandafter.%
2996 \the\numexpr #2.#1%
2997 }%
2998 \def\XINT_FL_binom_a #1.#2.%
2999 {%
3000 \expandafter\XINT_FL_binom_fork \the\numexpr #1-#2.#2.#1.%
3001 }%
3002 \def\XINT_FL_binom_fork #1#2.#3#4.#5.%
3003 {%
3004 \if-#1\xint_dothis \XINT_FL_binom_outofrange\fi
3005 \if-#3\xint_dothis \XINT_FL_binom_outofrange\fi
3006 \if0#1\xint_dothis \XINT_FL_binom_one\fi
3007 \if0#3\xint_dothis \XINT_FL_binom_one\fi
3008 \ifnum #5>\xint_c_x^viii_mone \xint_dothis\XINT_FL_binom_outofrange\fi
3009 \ifnum #1#2>#3#4 \xint_dothis\XINT_FL_binom_ab \fi
3010 \xint_orthat\XINT_FL_binom_aa
3011 #1#2.#3#4.#5.%
3012 }%
3013 \def\XINT_FL_binom_outofrange #1.#2.#3.#4.#5%
3014 {\xintError:OutOfRangeBinomial #5[#4]{0/1[0]}}%
3015 \def\XINT_FL_binom_one #1.#2.#3.#4.#5{#5[#4]{1/1[0]}}%
3016 \def\XINT_FL_binom_aa #1.#2.#3.#4.#5%
3017 {%
3018 #5[#4]{\xintDiv{\XINT_FL_pfac_increaseP
3019 #2.#3.\xint_c_iv{#4+\xint_c_i}{\XINTinfloat[#4+\xint_c_iii]}}%
3020 {\XINT_FL_fac_fork_b
3021 #1.\xint_c_iv{#4+\xint_c_i}\XINT_FL_fac_out{\XINTinfloat[#4+\xint_c_iii]}}}%
3022 }%
3023 \def\XINT_FL_binom_ab #1.#2.#3.#4.#5%
3024 {%
3025 #5[#4]{\xintDiv{\XINT_FL_pfac_increaseP
3026 #1.#3.\xint_c_iv{#4+\xint_c_i}{\XINTinfloat[#4+\xint_c_iii]}}%
3027 {\XINT_FL_fac_fork_b
3028 #2.\xint_c_iv{#4+\xint_c_i}\XINT_FL_fac_out{\XINTinfloat[#4+\xint_c_iii]}}}%
3029 }%

```

## 7.72 \xintFloatSqrt, \XINTinFloatSqrt

First done for 1.08.

The float version was developed at the same time as the integer one and even a bit earlier. As a result the integer variant had some sub-optimal parts. Anyway, for 1.2f I have rewritten the integer variant, and the float variant delegates all preparatory work for it until the last step. In particular the very low precisions are not penalized anymore from doing computations for at least 17 or 18 digits. Both the large and small precisions give quite shorter computation times.

Also, after examining more closely the achieved precision I decided to extend the float version in order for it to obtain the correct rounding (for inputs already of at most P digits with P the precision) of the theoretical exact value.

Beyond about 500 digits of precision the efficiency decreases swiftly, as is the case generally speaking with xintcore/xint/xintfrac arithmetic macros.

Final note: with 1.2f the input is always first rounded to P significant places.

```

3030 \def\xintFloatSqrt {\romannumeral0\xintfloatsqrt }%
3031 \def\xintfloatsqrt #1{\XINT_flsqrt_chkopt \xintfloat #1\xint_relax }%

```

## 7 Package *xintfrac* implementation

```

3032 \def\XINTinFloatSqrt {\romannumeral0\XINTinfloatsqr }%
3033 \def\XINTinfloatsqr #1{\XINT_flsqrt_chkopt \XINTinfloat #1\xint_relax }%
3034 \def\XINT_flsqrt_chkopt #1#2%
3035 {%
3036   \ifx [#2\expandafter\XINT_flsqrt_opt
3037     \else\expandafter\XINT_flsqrt_noopt
3038   \fi #1#2%
3039 }%
3040 \def\XINT_flsqrt_noopt #1#2\xint_relax
3041 {%
3042   \expandafter\XINT_FL_sqrt_a
3043     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.#1%
3044 }%
3045 \def\XINT_flsqrt_opt #1[\xint_relax #2]%#3%
3046 {%
3047   \expandafter\XINT_flsqrt_opt_a\the\numexpr #2.#1%
3048 }%
3049 \def\XINT_flsqrt_opt_a #1.#2#3%
3050 {%
3051   \expandafter\XINT_FL_sqrt_a\romannumeral0\XINTinfloat[#1]{#3}#1.#2%
3052 }%
3053 \def\XINT_FL_sqrt_a #1%
3054 {%
3055   \xint_UDzerominusfork
3056   #1-\XINT_FL_sqrt_iszero
3057   0#1\XINT_FL_sqrt_isneg
3058   0-\XINT_FL_sqrt_pos #1}%
3059 \krof
3060 }%[
3061 \def\XINT_FL_sqrt_iszero #1]#2.#3{#3[#2]{0[0]}}%
3062 \def\XINT_FL_sqrt_isneg #1]#2.#3{\xintError:RootOfNegative #3[#2]{0[0]}}%
3063 \def\XINT_FL_sqrt_pos #1[#2]#3.%
3064 {%
3065   \expandafter\XINT_flsqrt
3066   \the\numexpr #3\ifodd #2 \xint_dothis {+\xint_c_iii.(#2+\xint_c_i).0}\fi
3067   \xint_orthat {+\xint_c_ii.#2.{}}#100.#3.%
3068 }%
3069 \def\XINT_flsqrt #1.#2.%
3070 {%
3071   \expandafter\XINT_flsqrt_a
3072   \the\numexpr #2/\xint_c_ii-(#1-\xint_c_i)/\xint_c_ii.#1.%
3073 }%
3074 \def\XINT_flsqrt_a #1.#2.#3#4.#5.%
3075 {%
3076   \expandafter\XINT_flsqrt_b
3077   \the\numexpr (#2-\xint_c_i)/\xint_c_ii\expandafter.%
3078   \romannumeral0\XINT_sqrt_start #2.#4#3.#5.#2.#4#3.#5.#1.%
3079 }%
3080 \def\XINT_flsqrt_b #1.#2#3%
3081 {%
3082   \expandafter\XINT_flsqrt_c
3083   \romannumeral0\xintiisub

```

```

3084     {\XINT_dsx_addzerosnofuss {#1}{#2}}%
3085     {\xintiiDivRound{\XINT_dsx_addzerosnofuss {#1}{#3}}{\XINT_dbl_pos#2\Z}}.%
3086 }%
3087 \def\XINT_flsqrt_c #1.#2.%
3088 {%
3089     \expandafter\XINT_flsqrt_d
3090     \romannumeral0\XINT_split_fromleft_loop #2.{#1\W\W\W\W\W\W\W\W\Z
3091 }%
3092 \def\XINT_flsqrt_d #1.#2#3.%
3093 {%
3094     \ifnum #2=\xint_c_v
3095     \expandafter\XINT_flsqrt_f\else\expandafter\XINT_flsqrt_finish\fi
3096     #2#3.#1.%
3097 }%
3098 \def\XINT_flsqrt_finish #1#2.#3.#4.#5.#6.#7.#8{#8[#6]{#3#1[#7]}}%
3099 \def\XINT_flsqrt_f 5#1.%
3100     {\xintiifGt{\xintiNum{#1}}{2}{\XINT_flsqrt_finish 5.}{\XINT_flsqrt_again}}%
3101 \def\XINT_flsqrt_again #1.#2.%
3102 {%
3103     \expandafter\XINT_flsqrt_again_a\the\numexpr #2+\xint_c_viii.%
3104 }%
3105 \def\XINT_flsqrt_again_a #1.#2.#3.%
3106 {%
3107     \expandafter\XINT_flsqrt_b
3108     \the\numexpr (#1-\xint_c_i)/\xint_c_ii\expandafter.%
3109     \romannumeral0\XINT_sqrt_start #1.#2000000000.#3.%
3110     #1.#2000000000.#3.%
3111 }%

```

### 7.73 \xintFloatE, \XINTinFloatE

```

3112 \def\xintFloatE {\romannumeral0\xintfloate }%
3113 \def\xintfloate #1{\XINT_floate_chkopt #1\xint_relax }%
3114 \def\XINT_floate_chkopt #1%
3115 {%
3116     \ifx [#1\expandafter\XINT_floate_opt
3117     \else\expandafter\XINT_floate_noopt
3118     \fi #1%
3119 }%
3120 \def\XINT_floate_noopt #1\xint_relax
3121 {%
3122     \expandafter\XINT_floate_a\expandafter\XINTdigits
3123     \romannumeral0\XINT_infrac {#1}%
3124 }%
3125 \def\XINT_floate_opt [\xint_relax #1]#2%
3126 {%
3127     \expandafter\XINT_floate_a\expandafter
3128     {\the\numexpr #1\expandafter}\romannumeral0\XINT_infrac {#2}%
3129 }%
3130 \def\XINT_floate_a #1#2#3#4#5%
3131 {%
3132     \expandafter\XINT_float_a\the\numexpr#1\expandafter.%
3133     \expandafter{\the\numexpr #2+#5}{#3}{#4}\XINT_float_Q

```



```

3134 }%
3135 \def\XINTinFloatE {\romannumeral0\XINTinfloate }%
3136 \def\XINTinfloate
3137   {\expandafter\XINT_infloate\romannumeral0\XINTinfloat [\XINTdigits]]}%
3138 \def\XINT_infloate #1[#2]#3%
3139   {\expandafter\XINT_infloate_end\the\numexpr #3+#2.{#1}}%
3140 \def\XINT_infloate_end #1.#2{ #2[#1]}%

```

#### 7.74 \XINTinFloatMod

```

3141 \def\XINTinFloatMod {\romannumeral0\XINTinfloatmod [\XINTdigits]]}%
3142 \def\XINTinfloatmod [#1]#2#3{\expandafter\XINT_infloatmod\expandafter
3143   {\romannumeral0\XINTinfloat[#1]{#2}}}%
3144   {\romannumeral0\XINTinfloat[#1]{#3}}{#1}}%
3145 \def\XINT_infloatmod #1#2{\expandafter\XINT_infloatmod_a\expandafter {#2}{#1}}%
3146 \def\XINT_infloatmod_a #1#2#3{\XINTinfloat [#3]{\xintMod {#2}{#1}}}%
3147 \XINT_restorecatcodes_endinput%

```

## 8 Package *xintseries* implementation

.1	Catcodes, $\varepsilon$ -TeX and reload detection . .	234	.7	<code>\xintRationalSeries</code> . . . . .	237
.2	Package identification . . . . .	235	.8	<code>\xintRationalSeriesX</code> . . . . .	238
.3	<code>\xintSeries</code> . . . . .	235	.9	<code>\xintFxpPtPowerSeries</code> . . . . .	239
.4	<code>\xintiSeries</code> . . . . .	235	.10	<code>\xintFxpPtPowerSeriesX</code> . . . . .	240
.5	<code>\xintPowerSeries</code> . . . . .	236	.11	<code>\xintFloatPowerSeries</code> . . . . .	240
.6	<code>\xintPowerSeriesX</code> . . . . .	237	.12	<code>\xintFloatPowerSeriesX</code> . . . . .	242

The commenting is currently (2016/03/12) very sparse.

### 8.1 Catcodes, $\varepsilon$ -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 [2016/03/12 1.2f Expandable partial sums with xint package (JFB)]%

```

## 8.3 *\xintSeries*

```

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*

```

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\xintiiadd {#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. (this was at a time \xintAdd always multiplied denominators, by the way) The Horner scheme for polynomial evaluation is used in 1.04, this cures the denominator problem and drastically improves the efficiency of the macro. Modified in 1.06 to give the indices first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that. 1.08a adds the forgotten optimization following that previous change.

```

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`&&@#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,ratiofunction,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`&&@#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\xintiiadd {#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`&&@#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`&&@#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, $\varepsilon$ - $\text{\TeX}$ and reload detection . . .	243	.16	<code>\xintiGctoF</code> . . . . .	255
.2	Package identification . . . . .	244	.17	<code>\xintCtoCv</code> , <code>\xintCstoCv</code> . . . . .	256
.3	<code>\xintCFrac</code> . . . . .	244	.18	<code>\xintiCstoCv</code> . . . . .	257
.4	<code>\xintGCFrac</code> . . . . .	245	.19	<code>\xintGctoCv</code> . . . . .	257
.5	<code>\xintGGCFrac</code> . . . . .	247	.20	<code>\xintiGctoCv</code> . . . . .	259
.6	<code>\xintGctoGCx</code> . . . . .	248	.21	<code>\xintFtoCv</code> . . . . .	260
.7	<code>\xintFtoCs</code> . . . . .	248	.22	<code>\xintFtoCCv</code> . . . . .	260
.8	<code>\xintFtoCx</code> . . . . .	249	.23	<code>\xintCntoF</code> . . . . .	260
.9	<code>\xintFtoC</code> . . . . .	249	.24	<code>\xintGCntoF</code> . . . . .	261
.10	<code>\xintFtoGC</code> . . . . .	250	.25	<code>\xintCntoCs</code> . . . . .	262
.11	<code>\xintFGtoC</code> . . . . .	250	.26	<code>\xintCntoGC</code> . . . . .	262
.12	<code>\xintFtoCC</code> . . . . .	251	.27	<code>\xintGCntoGC</code> . . . . .	263
.13	<code>\xintCtoF</code> , <code>\xintCstoF</code> . . . . .	252	.28	<code>\xintCstoGC</code> . . . . .	264
.14	<code>\xintiCstoF</code> . . . . .	253	.29	<code>\xintGctoGC</code> . . . . .	264
.15	<code>\xintGctoF</code> . . . . .	253			

The commenting is currently (2016/03/12) very sparse. Release 1.09m (2014/02/26) has modified a few things: `\xintFtoCs` and `\xintCntoCs` insert spaces after the commas, `\xintCstoF` and `\xintCstoCv` authorize spaces in the input also before the commas, `\xintCntoCs` does not brace the produced coefficients, new macros `\xintFtoC`, `\xintCtoF`, `\xintCtoCv`, `\xintFGtoC`, and `\xintGGCFrac`.

### 9.1 Catcodes, $\varepsilon$ - $\text{\TeX}$ 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 [2016/03/12 1.2f 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\xintraawithzeros {#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`&&@%
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`&&@%
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`&&@#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 $\backslash$ xintFtoGC

```

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

```

### 9.11 $\backslash$ 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`&&@#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_fork #2\Z #6\Z}{\XINT_mul_fork #1\Z #4\Z}}%
460 {\romannumeral0\xintiiadd {\XINT_mul_fork #2\Z #5\Z}{\XINT_mul_fork #1\Z #3\Z}}%
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`&&@#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`&&@#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_fork #1\Z #3\Z}}%
494 {\romannumeral0\xintiiadd {#4}{\XINT_mul_fork #1\Z #2\Z}}%
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`&&@#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_fork #2\Z #6\Z}{\XINT_mul_fork #1\Z #4\Z}}%
522   {\romannumeral0\xintiiadd {\XINT_mul_fork #2\Z #5\Z}{\XINT_mul_fork #1\Z #3\Z}}%
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`&&@#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`&&@#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_fork #1\Z #3\Z}}%
581 {\romannumeral0\xintiiadd {#4}{\XINT_mul_fork #1\Z #2\Z}}%
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`&&@#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`&&@#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_fork #2\Z #6\Z}{\XINT_mul_fork #1\Z #4\Z}}%
635   {\romannumeral0\xintiiadd {\XINT_mul_fork #2\Z #5\Z}{\XINT_mul_fork #1\Z #3\Z}}%
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`&&@#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`&&@#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\xintiiadd {#5}{\XINT_mul_fork #1\Z #3\Z}}%
675   {\romannumeral0\xintiiadd {#4}{\XINT_mul_fork #1\Z #2\Z}}%
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`&&@#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_fork #2\Z #6\Z}{\XINT_mul_fork #1\Z #4\Z}}%
709   {\romannumeral0\xintiiadd {\XINT_mul_fork #2\Z #5\Z}{\XINT_mul_fork #1\Z #3\Z}}%
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`&&@#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`&&@#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_fork #1\Z #3\Z}}%
778   {\romannumeral0\xintiiadd {#4}{\XINT_mul_fork #1\Z #2\Z}}%
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`&&@#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`&&#2{#1}}{#2}}%
836 \else
837 \xint_afterfi
838 {\ifnum #1=\xint_c_
839 \xint_afterfi {\expandafter\space \romannumeral`&&#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`&&@#2{#1}\{#2}\{#3}\}}%
869   \else
870     \xint_afterfi
871     {\ifnum #1=\xint_c_
872       \xint_afterfi {\expandafter\space\romannumeral`&&@#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`&&@#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`&&@#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`&&@#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`&&@#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`&&@#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`&&@#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`&&@#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`&&@#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`&&@#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`&&@#2}{#3}{#1}%
1006 }%
1007 \def\XINT_gctgc_loop_b #1#2%
1008 {%
1009   \expandafter\XINT_gctgc_loop_c\expandafter
1010   {\romannumeral`&&@#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, $\varepsilon$ -TeX and reload detection	269
10.2	Package identification	270
10.3	Locking and unlocking	270
10.4	<code>\XINT_expr_wrap</code> , <code>\XINT_iiexpr_wrap</code>	271
10.5	<code>\XINT_protectii</code> , <code>\XINT_expr_usethe</code>	271
10.6	<code>\XINT_expr_print</code> , <code>\XINT_iiexpr_print</code> , <code>\XINT_boolexpr_print</code>	271
10.7	<code>\xintexpr</code> , <code>\xintiexpr</code> , <code>\xintfloatexpr</code> , <code>\xintiexpr</code> , <code>\xinttheexpr</code> , etc...	271
10.8	<code>\xintthe</code>	271
10.9	<code>\xintthecoords</code>	272
10.10	<code>\xintbareeval</code> , <code>\xintbarefloateval</code> , <code>\xintbareiieval</code>	272
10.11	<code>\xintthebareeval</code> , <code>\xintthebarefloateval</code> , <code>\xintthebareiieval</code>	272
10.12	<code>\xinteval</code> , <code>\xintiieval</code>	272
10.13	<code>\xintieval</code> , <code>\XINT_iexpr_wrap</code>	272
10.14	<code>\xintfloateval</code> , <code>\XINT_flexpr_wrap</code> , <code>\XINT_flexpr_print</code>	273
10.15	<code>\xintboolexpr</code> , <code>\xinttheboolexpr</code>	273
10.16	<code>\xintifboolexpr</code> , <code>\xintifboolfloatexpr</code> , <code>\xintifbooliiexpr</code>	273
10.17	Macros handling csv lists on output (for <code>\XINT_expr_print</code> et al. routines)	274
10.18	<code>\XINT_expr_getnext</code> : fetching some number then an operator	275
10.19	The integer or decimal number or hexa-decimal number or function name or variable name or special hacky things big parser	276
10.20	<code>\XINT_expr_getop</code> : finding the next operator or closing parenthesis or end of expression	283
10.21	Expansion spanning; opening and closing parentheses	284
10.22	<code> </code> , <code>  </code> , <code>&amp;</code> , <code>&amp;&amp;</code> , <code>&lt;</code> , <code>&gt;</code> , <code>=</code> , <code>==</code> , <code>&lt;=</code> , <code>&gt;=</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> , and <code>++</code> operators	286
10.23	Macros for a..b list generation	295
10.24	Macros for a..[d]..b list generation	296
10.25	The comma as binary operator	298
10.26	The minus as prefix operator of variable precedence level	299
10.27	<code>?</code> as two-way and <code>??</code> as three-way conditionals with braced branches	300
10.28	<code>!</code> as postfix factorial operator	300
10.29	The A/B[N] mechanism	300
10.30	<code>\XINT_expr_op_`</code> for recognizing functions	301
10.31	The <code>bool</code> , <code>togl</code> , <code>protect</code> pseudo “functions”	301
10.32	The <code>break</code> function	302
10.33	The <code>qint</code> , <code>qfrac</code> , <code>qfloat</code> “functions”	302
10.34	<code>\XINT_expr_op__</code> for recognizing variables	302
10.35	User defined variables: <code>\xintdefvar</code> , <code>\xintdefiivar</code> , <code>\xintdeffloatvar</code>	302
10.36	<code>\xintunassignvar</code>	303
10.37	<code>seq</code> and the implementation of dummy variables	304
10.38	<code>add</code> , <code>mul</code>	310
10.39	<code>subs</code>	311
10.40	<code>rseq</code>	311
10.41	<code>rrseq</code>	313
10.42	<code>iter</code>	314
10.43	Macros handling csv lists for functions with multiple comma separated arguments in expressions	316

10.44	The num, reduce, abs, sgn, frac, floor, ceil, sqr, sqrt, sqrtr, float, round, trunc, mod, quo, rem, gcd, lcm, max, min, `+`, `*`, `?`, `!`, not, all, any, xor, if, ifsgn, first, last, even, odd, reversed, factorial and binomial functions . . . . .	319
10.45	f-expandable versions of the <code>\xintSeqB::csv</code> and alike routines, for <code>\xintNewExpr</code> . . . . .	326
10.46	User defined functions: <code>\xintdeffunc</code> , <code>\xintdefiifunc</code> , <code>\xintdeffloatfunc</code> . . . . .	329
10.47	<code>\xintNewExpr</code> , <code>\xintNewIExpr</code> , <code>\xintNewFloatExpr</code> , <code>\xintNewIIFExpr</code> . . . . .	330

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 1.2 [2015/10/10] has the following changes:

**not anymore limited to 5000 digits:** 1.2 replaces chains of `\romannumeral-`0` used earlier to gather digits by `\csname` governed expansions. The use of `\csname.=A/B[N]\endcsname` storage has been part of the design from the start, hence it was very natural and not too hard to gather the number directly inside `\csname`. With the chains of `\romannumeral-`0` gone, there is no more a limit at about 5000 (with the standard settings of the maximal expansion depth at 10000) on the maximal number of digits for each gathered number.

**faster gathering of digits:** the previous item and some other changes have accelerated the building up of numbers.

**optional accelerated parsing:** the new functions `qint`, `qfrac`, `qfloat` allow to skip entirely the digit by digit parsing and hand over directly responsibility to `\xintiNum`, `\xintRaw`, or `\xintFloat` respectively.

**float factorial:** the factorial operator `!` maps to the new macro `\xintFloatFac` inside `\xintfloatexpr`.

**isolated dot now illegal:** the decimal mark must have digits either before or after it, an isolated `.` is now illegal input.

**more recognized tokens:** `\ht`, `\dp`, `\wd`, `\fontcharht`, `\fontcharwd`, `\fontchardp` and `\fontcharic` are recognized and prefixed with `\number` automatically.

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

**bug fixes** • `\xintiexpr` did not strip leading zeroes,

- `\xinttheexpr\xintiexpr 1.23\relax\relax` should have produced `1`, but it produced `1.23`
- the catcode of `;` was not set at package launching time.

**breaking changes** • in `\xintiexpr`, `/` does *rounded* division, rather than the Euclidean division (for positive arguments, this is truncated division). The new `//` operator does truncated division,

- the `:` operator for three-way branching is gone, replaced with `??`,
- `1e(3+5)` is now illegal. The number parser identifies `e` and `E` in the same way it does for the decimal mark, earlier versions treated `e` as `E` rather as postfix operators,
- the `add` and `mul` have a new syntax, old syntax is with ``+`` and ``*`` (quotes mandatory), `sum` and `prd` are gone,

- no more special treatment for encountered brace pairs `{..}` by the number scanner, `a/b[N]` notation can be used without use of braces (the `N` will end up as is in a `\numexpr`, it is not parsed by the `\xintexpr`-ession scanner).
- although `&` and `|` are still available as Boolean operators the use of `&&` and `||` is strongly recommended. The single letter operators might be assigned some other meaning in later releases (bitwise operations, perhaps). Do not use them.
- place holders for `\xintNewExpr` could be denoted `#1`, `#2`, ... or also, for special purposes `$1`, `$2`, ... Only the first form is now accepted and the special cases previously treated via the second form are now managed via a `protect(...)` function.

**novelties** They are quite a few.

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

- `n++` potentially infinite index generation for `seq`, `rseq`, `rrseq`, and `iter`, it is advised to use `abort` or `break(..)` at some point,
- the `add`, `mul`, `seq`, ... are nestable,
- `\xintthecoords` converts a comma separated list of an even number of items to the format as expected by the `TikZ coordinates` syntax,
- completely rewritten `\xintNewExpr`, `protect` function to handle external macros. However not all constructs are compatible with `\xintNewExpr`.

Comments dating back to earlier releases:

Roughly speaking, the parser mechanism is as follows: at any given time the last found ``operator'' has its associated `until` macro awaiting some news from the token flow; first `getnext` expands forward in the hope to construct some number, which may come from a parenthesized sub-expression, from some braced material, or from a digit by digit scan. After this number has been formed the next operator is looked for by the `getop` macro. Once `getop` has finished its job, `until` is presented with three tokens: the first one is the precedence level of the new found operator (which may be an end of expression marker), the second is the operator character token (earlier versions had here already some macro name, but in order to keep as much common code to `expr` and `floatexpr` common as possible, this was modified) of the new found operator, and the third one is the newly found number (which was encountered just before the new operator).

The `until` macro of the earlier operator examines the precedence level of the new found one, and either executes the earlier operator (in the case of a binary operation, with the found number and a previously stored one) or it delays execution, giving the hand to the `until` macro of the operator having been found of higher precedence.

A minus sign acting as prefix gets converted into a (unary) operator inheriting the precedence level of the previous operator.

Once the end of the expression is found (it has to be marked by a `\relax`) the final result is output as four tokens (five tokens since 1.09j) the first one a catcode 11 exclamation mark, the second one an error generating macro, the third one is a protection mechanism, the fourth one a printing macro and the fifth is `\.=a/b[n]`. The prefix `\xintthe` makes the output printable by killing the first three tokens.

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

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

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

**1.09i [2013/12/18]** unpacks count and dimen registers and control 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 "x", 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 1.2f of [2016/03/12].

## 10.1 Catcodes, $\varepsilon$ -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 [2016/03/12 1.2f Expandable expression parser (JFB)]%
58 \catcode`! 11

```

## 10.3 Locking and unlocking

Some renaming and modifications here with release 1.2 to switch from using chains of `\romannumeral-`0` in order to gather numbers, possibly hexadecimal, to using a `\csname` governed expansion. In this way no more limit at 5000 digits, and besides this is a logical move because the `\xintexpr` parser is already based on `\csname...\endcsname` storage of numbers as one token.

The limitation at 5000 digits didn't worry me too much because it was not very realistic to launch computations with thousands of digits... such computations are still slow with 1.2 but less so now. Chains or `\romannumeral` are still used for the gathering of function names and other stuff which I have half-forgotten because the parser does many things.

In the earlier versions we used the `lockscan` macro after a chain of `\romannumeral-`0` had ended gathering digits; this uses has been replaced by direct processing inside a `\csname...\endcsname` and the macro is kept only for matters of dummy variables.

Currently, the parsing of hexadecimal numbers needs two nested `\csname...\endcsname`, first to gather the letters (possibly with a hexadecimal fractional part), and in a second stage to apply `\xintHexToDec` to do the actual conversion. This should be faster than updating on the fly the number (which would be hard for the fraction part...). The macro `\xintHexToDec` could probably be made faster by using techniques similar as the ones 1.2 uses in `xintcore.sty`.

```

59 \def\xint_gob_til_! #1!{% catcode 11 ! default in xintexpr.sty code.
60 \edef\XINT_expr_lockscan#1!% not used for decimal numbers in xintexpr 1.2
61   {\noexpand\expandafter\space\noexpand\csname .=#1\endcsname }%
62 \edef\XINT_expr_lockit
63   #1{\noexpand\expandafter\space\noexpand\csname .=#1\endcsname }%
64 \def\XINT_expr_unlock_hex_in #1%   expanded inside \csname..\endcsname
65   {\expandafter\XINT_expr_inhex\romannumeral`&&\XINT_expr_unlock#1;}%
66 \def\XINT_expr_inhex #1.#2#3;%     expanded inside \csname..\endcsname
67 {%
68   \if#2>\xintHexToDec{#1}%
69   \else
70     \xintiiMul{\xintiiPow{625}{\xintLength{#3}}}{\xintHexToDec{#1#3}}%
71     [\the\numexpr-4*\xintLength{#3}]%
72   \fi
73 }%
74 \def\XINT_expr_unlock {\expandafter\XINT_expr_unlock_a\string }%
75 \def\XINT_expr_unlock_a #1.={%
76 \def\XINT_expr_unexpectedtoken {\xintError:ignored }%
77 \let\XINT_expr_done\space

```

#### 10.4 `\XINT_expr_wrap`, `\XINT_iiexpr_wrap`

```

78 \def\XINT_expr_wrap {!\XINT_expr_usethe\XINT_protectii\XINT_expr_print }%
79 \def\XINT_iiexpr_wrap {!\XINT_expr_usethe\XINT_protectii\XINT_iiexpr_print }%

```

#### 10.5 `\XINT_protectii`, `\XINT_expr_usethe`

```

80 \def\XINT_protectii #1{\noexpand\XINT_protectii\noexpand #1\noexpand }%
81 \protected\def\XINT_expr_usethe\XINT_protectii {\xintError:missing_xintthe!}%

```

#### 10.6 `\XINT_expr_print`, `\XINT_iiexpr_print`, `\XINT_boolexpr_print`

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

```

82 \def\XINT_expr_print #1{\xintSPRaw::csv {\XINT_expr_unlock #1}}%
83 \def\XINT_iiexpr_print #1{\xintCSV::csv {\XINT_expr_unlock #1}}%
84 \def\XINT_boolexpr_print #1{\xintIsTrue::csv {\XINT_expr_unlock #1}}%

```

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

```

85 \def\xintexpr {\romannumeral0\xinteval }%
86 \def\xintiexpr {\romannumeral0\xintieval }%
87 \def\xintfloatexpr {\romannumeral0\xintfloateval }%
88 \def\xintiiexpr {\romannumeral0\xintiieval }%
89 \def\xinttheexpr
90   {\romannumeral`&&\expandafter\XINT_expr_print\romannumeral0\xintbareeval }%
91 \def\xinttheiexpr {\romannumeral`&&\xintthe\xintiexpr }%
92 \def\xintthefloatexpr {\romannumeral`&&\xintthe\xintfloatexpr }%
93 \def\xinttheiiexpr
94   {\romannumeral`&&\expandafter\XINT_iiexpr_print\romannumeral0\xintbareiieval }%

```

#### 10.8 `\xintthe`

```

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

```



## 10.9 `\xintthecoords`

1.1 Wraps up an even number of comma separated items into pairs of TikZ coordinates; for use in the following way:

```
coordinates {\xintthecoords\xintfloatexpr ... \relax}
```

The crazyness with the `\csname` and `unlock` is due to TikZ somewhat STRANGE control of the TOTAL number of expansions which should not exceed the very low value of 100 !! As we implemented `\XINT_thecoords_b` in an "inline" style for efficiency, we need to hide its expansions.

Not to be used as `\xintthecoords\xintthefloatexpr`, only as `\xintthecoords\xintfloatexpr` (or `\xintiexpr` etc...). Perhaps `\xintthecoords` could make an extra check, but one should not accustom users to too loose requirements!

```
96 \def\xintthecoords #1{\romannumeral`&&\expandafter\expandafter\expandafter
97     \XINT_thecoords_a
98     \expandafter\xint_gobble_iii\romannumeral0#1}%
99 \def\XINT_thecoords_a #1#2% #1=print macro, indispensable for scientific notation
100     {\expandafter\XINT_expr_unlock\csname.=\expandafter\XINT_thecoords_b
101         \romannumeral`&&@#1#2,!,!,^ \endcsname }%
102 \def\XINT_thecoords_b #1#2,#3#4,%
103     {\xint_gob_til_! #3\XINT_thecoords_c ! (#1#2, #3#4)\XINT_thecoords_b }%
104 \def\XINT_thecoords_c #1^{}%
```

## 10.10 `\xintbareeval`, `\xintbarefloateval`, `\xintbareiieval`

```
105 \def\xintbareeval
106     {\expandafter\XINT_expr_until_end_a\romannumeral`&&\XINT_expr_getnext }%
107 \def\xintbarefloateval
108     {\expandafter\XINT_flexpr_until_end_a\romannumeral`&&\XINT_expr_getnext }%
109 \def\xintbareiieval
110     {\expandafter\XINT_iexpr_until_end_a\romannumeral`&&\XINT_expr_getnext }%
```

## 10.11 `\xintthebareeval`, `\xintthebarefloateval`, `\xintthebareiieval`

```
111 \def\xintthebareeval      {\expandafter\XINT_expr_unlock\romannumeral0\xintbareeval}%
112 \def\xintthebarefloateval {\expandafter\XINT_expr_unlock\romannumeral0\xintbarefloateval}%
113 \def\xintthebareiieval    {\expandafter\XINT_expr_unlock\romannumeral0\xintbareiieval}%
```

## 10.12 `\xinteval`, `\xintiieval`

```
114 \def\xinteval      {\expandafter\XINT_expr_wrap\romannumeral0\xintbareeval }%
115 \def\xintiieval    {\expandafter\XINT_iexpr_wrap\romannumeral0\xintbareiieval }%
```

## 10.13 `\xintieval`, `\XINT_iexpr_wrap`

Optional argument since 1.1.

```
116 \def\xintieval #1%
117     {\ifx [#1\expandafter\XINT_iexpr_withopt\else\expandafter\XINT_iexpr_noopt \fi #1}%
118 \def\XINT_iexpr_noopt
119     {\expandafter\XINT_iexpr_wrap \expandafter 0\romannumeral0\xintbareeval }%
120 \def\XINT_iexpr_withopt [#1]%
121     {%
122     \expandafter\XINT_iexpr_wrap\expandafter
123     {\the\numexpr \xint_zapspace #1 \xint_gobble_i\expandafter}%
124     \romannumeral0\xintbareeval
125     }%
```



```

126 \def\XINT_iexpr_wrap #1#2%
127 {%
128     \expandafter\XINT_expr_wrap
129     \csname .=\xintRound::csv {#1}\XINT_expr_unlock #2}\endcsname
130 }%

```

## 10.14 \xintfloateval, \XINT\_flexpr\_wrap, \XINT\_flexpr\_print

Optional argument since 1.1

```

131 \def\xintfloateval #1%
132 {%
133     \ifx [#1\expandafter\XINT_flexpr_withopt_a\else\expandafter\XINT_flexpr_noopt
134     \fi #1%
135 }%
136 \def\XINT_flexpr_noopt
137 {%
138     \expandafter\XINT_flexpr_withopt_b\expandafter\xinttheDigits
139     \romannumeral0\xintbarefloateval
140 }%
141 \def\XINT_flexpr_withopt_a [#1]%
142 {%
143     \expandafter\XINT_flexpr_withopt_b\expandafter
144     {\the\numexpr\xint_zapspace #1 \xint_gobble_i\expandafter}%
145     \romannumeral0\xintbarefloateval
146 }%
147 \def\XINT_flexpr_withopt_b #1#2%
148 {%
149     \expandafter\XINT_flexpr_wrap\csname .;#1.=% ; and not : as before b'cause NewExpr
150     \XINTinFloat::csv {#1}\XINT_expr_unlock #2}\endcsname
151 }%
152 \def\XINT_flexpr_wrap { !\XINT_expr_usethe\XINT_protectii\XINT_flexpr_print }%
153 \def\XINT_flexpr_print #1%
154 {%
155     \expandafter\xintPFloat::csv
156     \romannumeral`&&\expandafter\XINT_expr_unlock_sp\string #1!%
157 }%
158 \catcode`: 12
159     \def\XINT_expr_unlock_sp #1.;#2.=#3!{{#2}{#3}}%
160 \catcode`: 11

```

## 10.15 \xintboolexpr, \xinttheboolexpr

```

161 \def\xintboolexpr      {\romannumeral0\expandafter\expandafter\expandafter
162     \XINT_boolexpr_done \expandafter\xint_gobble_iv\romannumeral0\xinteval }%
163 \def\xinttheboolexpr  {\romannumeral`&&\expandafter\expandafter\expandafter
164     \XINT_boolexpr_print\expandafter\xint_gobble_iv\romannumeral0\xinteval }%
165 \def\XINT_boolexpr_done { !\XINT_expr_usethe\XINT_protectii\XINT_boolexpr_print }%

```

## 10.16 \xintifboolexpr, \xintifboolfloatexpr, \xintifboolliexpr

Do not work with comma separated expressions.

```

166 \def\xintifboolexpr      #1{\romannumeral0\xintifnotzero {\xinttheexpr #1\relax}}%

```

```
167 \def\xintifboolfloatexpr #1{\romannumeral0\xintifnotzero {\xintthefloatexpr #1\relax}}%
168 \def\xintifbooliiexpr #1{\romannumeral0\xintifnotzero {\xinttheiiexpr #1\relax}}%
```

## 10.17 Macros handling csv lists on output (for `\XINT_expr_print` et al. routines)

10.17.1	<code>\XINT::_end</code>	274
10.17.2	<code>\xintCSV::csv</code>	274
10.17.3	<code>\xintSPRaw, \xintSPRaw::csv</code>	274
10.17.4	<code>\xintIsTrue::csv</code>	274
10.17.5	<code>\xintRound::csv</code>	275
10.17.6	<code>\XINTinFloat::csv</code>	275
10.17.7	<code>\xintPFloat::csv</code>	275

Changed completely for 1.1, which adds the optional arguments to `\xintiexpr` and `\xintfloatexpr`.

### 10.17.1 `\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`

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

### 10.17.2 `\xintCSV::csv`

```
170 \def\xintCSV::csv #1{\expandafter\XINT_csv::_a\romannumeral`&&@#1,^}%
171 \def\XINT_csv::_a {\XINT_csv::_b {}}%
172 \def\XINT_csv::_b #1#2,{\expandafter\XINT_csv::_c \romannumeral`&&@#2,{#1}}%
173 \def\XINT_csv::_c #1{\if ^#1\expandafter\XINT::_end\fi\XINT_csv::_d #1}%
174 \def\XINT_csv::_d #1,#2{\XINT_csv::_b {#2, #1}}% possibly, item #1 is empty.
```

### 10.17.3 `\xintSPRaw, \xintSPRaw::csv`

```
175 \def\xintSPRaw {\romannumeral0\xintspraw}%
176 \def\xintspraw #1{\expandafter\XINT_spraw\romannumeral`&&@#1[\W]}%
177 \def\XINT_spraw #1[#2#3]{\xint_gob_til_W #2\XINT_spraw_a\W\XINT_spraw_p #1[#2#3]}%
178 \def\XINT_spraw_a\W\XINT_spraw_p #1[\W]{ #1}%
179 \def\XINT_spraw_p #1[\W]{\xintprap {#1}}%
180 \def\xintSPRaw::csv #1{\romannumeral0\expandafter\XINT_spraw::_a\romannumeral`&&@#1,^}%
181 \def\XINT_spraw::_a {\XINT_spraw::_b {}}%
182 \def\XINT_spraw::_b #1#2,{\expandafter\XINT_spraw::_c \romannumeral`&&@#2,{#1}}%
183 \def\XINT_spraw::_c #1{\if ,#1\xint_dothis\XINT_spraw::_e\fi
184     \if ^#1\xint_dothis\XINT::_end\fi
185     \xint_orthat\XINT_spraw::_d #1}%
186 \def\XINT_spraw::_d #1,{\expandafter\XINT_spraw::_e\romannumeral0\XINT_spraw #1[\W],}%
187 \def\XINT_spraw::_e #1,#2{\XINT_spraw::_b {#2, #1}}%
```

### 10.17.4 `\xintIsTrue::csv`

```
188 \def\xintIsTrue::csv #1{\romannumeral0\expandafter\XINT_istrue::_a\romannumeral`&&@#1,^}%
189 \def\XINT_istrue::_a {\XINT_istrue::_b {}}%
190 \def\XINT_istrue::_b #1#2,{\expandafter\XINT_istrue::_c \romannumeral`&&@#2,{#1}}%
191 \def\XINT_istrue::_c #1{\if ,#1\xint_dothis\XINT_istrue::_e\fi
192     \if ^#1\xint_dothis\XINT::_end\fi
193     \xint_orthat\XINT_istrue::_d #1}%
194 \def\XINT_istrue::_d #1,{\expandafter\XINT_istrue::_e\romannumeral0\xintisnotzero {#1},}%

```

```
195 \def\XINT_istrue::_e #1,#2{\XINT_istrue::_b {#2, #1}}%
```

### 10.17.5 \xintRound::csv

```
196 \def\XINT_::_end #1,#2#3{\xint_gobble_i #3}%
197 \def\xintRound::csv #1#2{\romannumeral0\expandafter\XINT_round::_b\expandafter
198   {\the\numexpr#1\expandafter}\expandafter{\expandafter}\romannumeral`&&@#2,^,%}
199 \def\XINT_round::_b #1#2#3,{\expandafter\XINT_round::_c \romannumeral`&&@#3,{#1}{#2}}%
200 \def\XINT_round::_c #1{\if ,#1\xint_dothis\XINT_round::_e\fi
201   \if ^#1\xint_dothis\XINT_::_end\fi
202   \xint_orthat\XINT_round::_d #1}%
203 \def\XINT_round::_d #1,#2{%
204   \expandafter\XINT_round::_e\romannumeral0\ifnum#2>\xint_c_
205   \expandafter\xintround\else\expandafter\xintiround\fi {#2}{#1},{#2}}%
206 \def\XINT_round::_e #1,#2#3{\XINT_round::_b {#2}{#3, #1}}%
```

### 10.17.6 \XINTinFloat::csv

```
207 \def\XINTinFloat::csv #1#2{\romannumeral0\expandafter\XINT_infloat::_b\expandafter
208   {\the\numexpr #1\expandafter}\expandafter{\expandafter}\romannumeral`&&@#2,^,%}
209 \def\XINT_infloat::_b #1#2#3,{\XINT_infloat::_c #3,{#1}{#2}}%
210 \def\XINT_infloat::_c #1{\if ,#1\xint_dothis\XINT_infloat::_e\fi
211   \if ^#1\xint_dothis\XINT_::_end\fi
212   \xint_orthat\XINT_infloat::_d #1}%
213 \def\XINT_infloat::_d #1,#2{%
214   {\expandafter\XINT_infloat::_e\romannumeral0\XINTinfloat [#2]{#1},{#2}}%
215 \def\XINT_infloat::_e #1,#2#3{\XINT_infloat::_b {#2}{#3, #1}}%
```

### 10.17.7 \xintPFloat::csv

```
216 \def\xintPFloat::csv #1#2{\romannumeral0\expandafter\XINT_pfloat::_b\expandafter
217   {\the\numexpr #1\expandafter}\expandafter{\expandafter}\romannumeral`&&@#2,^,%}
218 \def\XINT_pfloat::_b #1#2#3,{\expandafter\XINT_pfloat::_c \romannumeral`&&@#3,{#1}{#2}}%
219 \def\XINT_pfloat::_c #1{\if ,#1\xint_dothis\XINT_pfloat::_e\fi
220   \if ^#1\xint_dothis\XINT_::_end\fi
221   \xint_orthat\XINT_pfloat::_d #1}%
222 \def\XINT_pfloat::_d #1,#2{%
223   {\expandafter\XINT_pfloat::_e\romannumeral0\XINT_pfloat_opt [\xint_relax #2]{#1},{#2}}%
224 \def\XINT_pfloat::_e #1,#2#3{\XINT_pfloat::_b {#2}{#3, #1}}%
```

## 10.18 \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.

```
225 \def\XINT_expr_getnext #1%
226 {%
227   \expandafter\XINT_expr_getnext_a\romannumeral`&&@#1%
228 }%
229 \def\XINT_expr_getnext_a #1%
230 {% screens out sub-expressions and \count or \dimen registers/variables
231   \xint_gob_til_! #1\XINT_expr_subexpr !% recall this ! has catcode 11
232   \ifcat\relax#1% \count or \numexpr etc... token or count, dimen, skip cs
233     \expandafter\XINT_expr_countetc
234   \else
235     \expandafter\expandafter\expandafter\XINT_expr_getnextfork\expandafter\string
```

```

236 \fi
237 #1%
238 }%
239 \def\XINT_expr_subexpr !#1\fi !{\expandafter\XINT_expr_getop\xint_gobble_iii }%

```

1.2 adds \ht, \dp, \wd and the eTeX font things.

```

240 \def\XINT_expr_countetc #1%
241 {%
242 \ifx\count#1\else\ifx\dimen#1\else\ifx\numexpr#1\else\ifx\dimexpr#1\else
243 \ifx\skip#1\else\ifx\glueexpr#1\else\ifx\fontdimen#1\else\ifx\ht#1\else
244 \ifx\dp#1\else\ifx\wd#1\else\ifx\fontcharht#1\else\ifx\fontcharwd#1\else
245 \ifx\fontcharhp#1\else\ifx\fontcharic#1\else
246 \XINT_expr_unpackvar
247 \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
248 \expandafter\XINT_expr_getnext\number #1%
249 }%
250 \def\XINT_expr_unpackvar\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
251 \expandafter\XINT_expr_getnext\number #1%
252 {\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
253 \expandafter\XINT_expr_getop\csname .=#1\endcsname }%
254 \begingroup
255 \lccode`*=`#
256 \lowercase{\endgroup
257 \def\XINT_expr_getnextfork #1{%
258 \if#1*\xint_dothis {\XINT_expr_scan_macropar *}\fi
259 \if#1[\xint_dothis {\xint_c_xviii ({} }\fi
260 \if#1+\xint_dothis \XINT_expr_getnext \fi
261 \if#1.\xint_dothis {\XINT_expr_startdec}\fi
262 \if#1-\xint_dothis -\fi
263 \if#1(\xint_dothis {\xint_c_xviii ({} }\fi
264 \xint_orthat {\XINT_expr_scan_nbr_or_func #1}%
265 }%
266 \def\XINT_expr_scan_macropar #1#2{\expandafter\XINT_expr_getop\csname .=#1#2\endcsname }%

```

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

10.19.1	Integral part (skipping zeroes)	277
10.19.2	Fractional part	279
10.19.3	Scientific notation	280
10.19.4	Hexadecimal numbers	280
10.19.5	Parsing names of functions and variables	282

1.2 release has replaced chains of \romannumeral-`0 by \csname governed expansion. Thus there is no more the limit at about 5000 digits for parsed numbers.

In order to avoid having to lock and unlock in succession to handle the scientific part and adjust the exponent according to the number of digits of the decimal part, the parsing of this decimal part counts on the fly the number of digits it encounters.

There is some slight annoyance with \xintiexpr which should never be given a [n] inside its \csname.<=<digits>\endcsname storage of numbers (because its arithmetic uses the ii macros which know nothing about the [N] notation). Hence if the parser has only seen digits when hitting something else than the dot or e (or E), it will not insert a [0]. Thus we very slightly compromise

the efficiency of `\xintexpr` and `\xintfloatexpr` in order to be able to share the same code with `\xintiexpr`.

Indeed, the parser at this location is completely common to all, it does not know if it is working inside `\xintexpr` or `\xintiexpr`. On the other hand if a dot or a e (or E) is met, then the (common) parser has no scruples ending this number with a [n], this will provoke an error later if that was within an `\xintiexpr`, as soon as an arithmetic macro is used.

As the gathered numbers have no spaces, no pluses, no minuses, the only remaining issue is with leading zeroes, which are discarded on the fly. The hexadecimal numbers leading zeroes are stripped in a second stage by the `\xintHexToDec` macro.

With 1.2, `\xinttheexpr . \relax` does not work anymore (it did in earlier releases). There must be digits either before or after the decimal mark. Thus both `\xinttheexpr 1.\relax` and `\xinttheexpr .1\relax` are legal.

The ``` syntax is here used for special constructs like ``+`(..)`, ``*`(..)` where + or \* will be treated as functions. Current implementation pick only one token (could have been braced stuff), thus here it will be + or \*, and via `\XINT_expr_op_`` this into becomes a suitable `\XINT_{expr|iiexpr|fexpr}_func_+` (or \*). Documentation of 1.1 said to use ``+`(...)`, but ``+`(...)` is also valid. The opening parenthesis must be there, it is not allowed to come from expansion.

```

267 \catcode96 11 % `
268 \def\xINT_expr_scan_nbr_or_func #1% this #1 has necessarily here catcode 12
269 {%
270   \if "#1\xint_dothis \XINT_expr_scanhex_I\fi
271   \if `#1\xint_dothis {\XINT_expr_onlitteral_}\fi
272   \ifnum \xint_c_ix<1#1 \xint_dothis \XINT_expr_startint\fi
273   \xint_orthat \XINT_expr_scanfunc #1%
274 }%
275 \def\xINT_expr_onlitteral_` #1#2#3({\xint_c_xviii `{#2}}%
276 \catcode96 12 % `
277 \def\xINT_expr_startint #1%
278 {%
279   \if #10\expandafter\xINT_expr_gobz_a\else\xINT_expr_scanint_a\fi #1%
280 }%
281 \def\xINT_expr_scanint_a #1#2%
282   {\expandafter\xINT_expr_getop\csname.=#1%
283     \expandafter\xINT_expr_scanint_b\romannumeral`&&@#2}%
284 \def\xINT_expr_gobz_a #1%
285   {\expandafter\xINT_expr_getop\csname.=#1%
286     \expandafter\xINT_expr_gobz_scanint_b\romannumeral`&&@#1}%
287 \def\xINT_expr_startdec #1%
288   {\expandafter\xINT_expr_getop\csname.=#1%
289     \expandafter\xINT_expr_scandec_a\romannumeral`&&@#1}%

```

### 10.19.1 Integral part (skipping zeroes)

1.2 has modified the code to give highest priority to digits, the accelerating impact is non-negligeable. I don't think the doubled `\string` is a serious penalty.

```

290 \def\xINT_expr_scanint_b #1%
291 {%
292   \ifcat \relax #1\expandafter\xINT_expr_scanint_endbycs\expandafter #1\fi
293   \ifnum\xint_c_ix<1\string#1 \else\expandafter\xINT_expr_scanint_c\fi
294   \string#1\xINT_expr_scanint_d
295 }%

```

```

296 \def\XINT_expr_scanint_d #1%
297 {%
298   \expandafter\XINT_expr_scanint_b\romannumeral`&&@#1%
299 }%
300 \def\XINT_expr_scanint_endbycs#1#2\XINT_expr_scanint_d{\endcsname #1}%

```

With 1.2d the tacit multiplication in front of a variable name or function name is now done with a higher precedence, intermediate between the common one of  $*$  and  $/$  and the one of  $^$ . Thus  $x/2y$  is like  $x/(2y)$ , but  $x^2y$  is like  $x^2*y$  and  $2y!$  is not  $(2y)!$  but  $2*y!$ .

Finally, 1.2d has moved away from the `_scan` macros all the business of the tacit multiplication in one unique place via `\XINT_expr_getop`. For this, the ending token is not first given to `\string` as was done earlier before handing over back control to `\XINT_expr_getop`. Earlier we had to identify the catcode 11 ! signaling a sub-expression here. With no `\string` applied we can do it in `\XINT_expr_getop`. As a corollary of this displacement, parsing of big numbers should be a tiny bit faster now.

```

301 \def\XINT_expr_scanint_c\string #1\XINT_expr_scanint_d
302 {%
303   \if e#1\xint_dothis{[\the\numexpr0\XINT_expr_scanexp_a +}\fi
304   \if E#1\xint_dothis{[\the\numexpr0\XINT_expr_scanexp_a +}\fi
305   \if .#1\xint_dothis{\XINT_expr_startdec_a .}\fi
306   \xint_orthat {\endcsname #1}%
307 }%
308 \def\XINT_expr_startdec_a .#1%
309 {%
310   \expandafter\XINT_expr_scandec_a\romannumeral`&&@#1%
311 }%
312 \def\XINT_expr_scandec_a #1%
313 {%
314   \if .#1\xint_dothis{\endcsname .}\fi
315   \xint_orthat {\XINT_expr_scandec_b 0.#1}%
316 }%
317 \def\XINT_expr_gobz_scanint_b #1%
318 {%
319   \ifcat \relax #1\expandafter\XINT_expr_gobz_scanint_endbycs\expandafter #1\fi
320   \ifnum\xint_c_x<1\string#1 \else\expandafter\XINT_expr_gobz_scanint_c\fi
321   \string#1\XINT_expr_scanint_d
322 }%
323 \def\XINT_expr_gobz_scanint_endbycs#1#2\XINT_expr_scanint_d{0\endcsname #1}%
324 \def\XINT_expr_gobz_scanint_c\string #1\XINT_expr_scanint_d
325 {%
326   \if e#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
327   \if E#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
328   \if .#1\xint_dothis{\XINT_expr_gobz_startdec_a .}\fi
329   \if 0#1\xint_dothis\XINT_expr_gobz_scanint_d\fi
330   \xint_orthat {0\endcsname #1}%
331 }%
332 \def\XINT_expr_gobz_scanint_d #1%
333 {%
334   \expandafter\XINT_expr_gobz_scanint_b\romannumeral`&&@#1%
335 }%
336 \def\XINT_expr_gobz_startdec_a .#1%
337 {%

```

```

338 \expandafter\XINT_expr_gobz_scandec_a\romannumeral`&&@#1%
339 }%
340 \def\XINT_expr_gobz_scandec_a #1%
341 {%
342 \if .#1\xint_dothis{0\endcsname.}\fi
343 \xint_orthat {\XINT_expr_gobz_scandec_b 0.#1}%
344 }%

```

### 10.19.2 Fractional part

Annoying duplication of code to allow 0. as input.

1.2a corrects a very bad bug in 1.2 \XINT\_expr\_gobz\_scandec\_b which should have stripped leading zeroes in the fractional part but didn't; as a result \xinttheexpr 0.01\relax returned 0 =:-((( Thanks to Kroum Tzanev who reported the issue. Does it improve things if I say the bug was introduced in 1.2, it wasn't present before ?

```

345 \def\XINT_expr_scandec_b #1.#2%
346 {%
347 \ifcat \relax #2\expandafter\XINT_expr_scandec_endbycs\expandafter#2\fi
348 \ifnum\xint_c_ix<1\string#2 \else\expandafter\XINT_expr_scandec_c\fi
349 \string#2\expandafter\XINT_expr_scandec_d\the\numexpr #1-\xint_c_i.%
350 }%
351 \def\XINT_expr_scandec_endbycs #1#2\XINT_expr_scandec_d
352 \the\numexpr#3-\xint_c_i.{[#3]\endcsname #1}%
353 \def\XINT_expr_scandec_d #1.#2%
354 {%
355 \expandafter\XINT_expr_scandec_b
356 \the\numexpr #1\expandafter.\romannumeral`&&@#2%
357 }%
358 \def\XINT_expr_scandec_c\string #1#2\the\numexpr#3-\xint_c_i.%
359 {%
360 \if e#1\xint_dothis{[\the\numexpr#3\XINT_expr_scanexp_a +}\fi
361 \if E#1\xint_dothis{[\the\numexpr#3\XINT_expr_scanexp_a +}\fi
362 \xint_orthat {[#3]\endcsname #1}%
363 }%
364 \def\XINT_expr_gobz_scandec_b #1.#2%
365 {%
366 \ifcat \relax #2\expandafter\XINT_expr_gobz_scandec_endbycs\expandafter#2\fi
367 \ifnum\xint_c_ix<1\string#2 \else\expandafter\XINT_expr_gobz_scandec_c\fi
368 \if0#2\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
369 {\expandafter\XINT_expr_gobz_scandec_b}%
370 {\string#2\expandafter\XINT_expr_scandec_d}\the\numexpr#1-\xint_c_i.%
371 }%
372 \def\XINT_expr_gobz_scandec_endbycs #1#2\xint_c_i.{0[0]\endcsname #1}%
373 \def\XINT_expr_gobz_scandec_c\if0#1#2\fi #3\xint_c_i.%
374 {%
375 \if e#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
376 \if E#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
377 \xint_orthat {0[0]\endcsname #1}%
378 }%

```

### 10.19.3 Scientific notation

Some pluses and minuses are allowed at the start of the scientific part, however not later, and no parenthesis.

```

379 \def\XINT_expr_scanexp_a #1#2%
380 {%
381   #1\expandafter\XINT_expr_scanexp_b\romannumeral`&&@#2%
382 }%
383 \def\XINT_expr_scanexp_b #1%
384 {%
385   \ifcat \relax #1\expandafter\XINT_expr_scanexp_endbycs\expandafter #1\fi
386   \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanexp_c\fi
387   \string#1\XINT_expr_scanexp_d
388 }%
389 \def\XINT_expr_scanexpr_endbycs#1#2\XINT_expr_scanexp_d {}]\endcsname #1}%
390 \def\XINT_expr_scanexp_d #1%
391 {%
392   \expandafter\XINT_expr_scanexp_bb\romannumeral`&&@#1%
393 }%
394 \def\XINT_expr_scanexp_c\string #1\XINT_expr_scanexp_d
395 {%
396   \if +#1\xint_dothis {\XINT_expr_scanexp_a +}\fi
397   \if -#1\xint_dothis {\XINT_expr_scanexp_a -}\fi
398   \xint_orthat {}]\endcsname #1}%
399 }%
400 \def\XINT_expr_scanexp_bb #1%
401 {%
402   \ifcat \relax #1\expandafter\XINT_expr_scanexp_endbycs_b\expandafter #1\fi
403   \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanexp_cb\fi
404   \string#1\XINT_expr_scanexp_db
405 }%
406 \def\XINT_expr_scanexp_endbycs_b#1#2\XINT_expr_scanexp_db {}]\endcsname #1}%
407 \def\XINT_expr_scanexp_db #1%
408 {%
409   \expandafter\XINT_expr_scanexp_bb\romannumeral`&&@#1%
410 }%
411 \def\XINT_expr_scanexp_cb\string #1\XINT_expr_scanexp_db {}]\endcsname #1}%

```

### 10.19.4 Hexadecimal numbers

1.2d has moved most of the handling of tacit multiplication to `\XINT_expr_getop`, but we have to do some of it here, because we apply `\string` before calling `\XINT_expr_scanhexI_aa`. I do not insert the `*` in `\XINT_expr_scanhexI_a`, because it is its higher precedence variant which will be expected, to do the same as when a non-hexadecimal number prefixes a sub-expression. Tacit multiplication in front of variable or function names will not work (because of this `\string`).

```

412 \def\XINT_expr_scanhex_I #1% #1="
413 {%
414   \expandafter\XINT_expr_getop\csname.=\expandafter
415   \XINT_expr_unlock_hex_in\csname.=\XINT_expr_scanhexI_a
416 }%
417 \def\XINT_expr_scanhexI_a #1%

```



```

418 {%
419   \ifcat #1\relax\xint_dothis{.>\endcsname\endcsname #1}\fi
420   \ifx !#1\xint_dothis{.>\endcsname\endcsname !}\fi
421   \xint_orthat {\expandafter\XINT_expr_scanhexI_aa\string #1}%
422 }%
423 \def\XINT_expr_scanhexI_aa #1%
424 {%
425   \if\ifnum`#1>`/
426     \ifnum`#1>`9
427       \ifnum`#1>`@
428         \ifnum`#1>`F
429           0\else1\fi\else0\fi\else1\fi\else0\fi 1%
430         \expandafter\XINT_expr_scanhexI_b
431       \else
432         \if .#1%
433           \expandafter\xint_firstoftwo
434         \else % gather what we got so far, leave catcode 12 #1 in stream
435           \expandafter\xint_secondoftwo
436         \fi
437         {\expandafter\XINT_expr_scanhex_transition}%
438         {\xint_afterfi {.>\endcsname\endcsname}}%
439       \fi
440     #1%
441 }%
442 \def\XINT_expr_scanhexI_b #1#2%
443 {%
444   #1\expandafter\XINT_expr_scanhexI_a\romannumeral`&&@#2%
445 }%
446 \def\XINT_expr_scanhex_transition .#1%
447 {%
448   \expandafter.\expandafter.\expandafter
449   \XINT_expr_scanhexII_a\romannumeral`&&@#1%
450 }%
451 \def\XINT_expr_scanhexII_a #1%
452 {%
453   \ifcat #1\relax\xint_dothis{\endcsname\endcsname#1}\fi
454   \ifx !#1\xint_dothis{\endcsname\endcsname !}\fi
455   \xint_orthat {\expandafter\XINT_expr_scanhexII_aa\string #1}%
456 }%
457 \def\XINT_expr_scanhexII_aa #1%
458 {%
459   \if\ifnum`#1>`/
460     \ifnum`#1>`9
461       \ifnum`#1>`@
462         \ifnum`#1>`F
463           0\else1\fi\else0\fi\else1\fi\else0\fi 1%
464         \expandafter\XINT_expr_scanhexII_b
465       \else
466         \xint_afterfi {\endcsname\endcsname}%
467       \fi
468     #1%
469 }%

```

```

470 \def\XINT_expr_scanhexII_b #1#2%
471 {%
472   #1\expandafter\XINT_expr_scanhexII_a\romannumeral`&&@#2%
473 }%

```

### 10.19.5 Parsing names of functions and variables

```

474 \def\XINT_expr_scanfunc
475 {%
476   \expandafter\XINT_expr_func\romannumeral`&&\XINT_expr_scanfunc_a
477 }%
478 \def\XINT_expr_scanfunc_a #1#2%
479 {%
480   \expandafter #1\romannumeral`&&\expandafter\XINT_expr_scanfunc_b\romannumeral`&&@#2%
481 }%

```

This handles: 1) (indirectly) tacit multiplication by a variable in front a of sub-expression, 2) (indirectly) tacit multiplication in front of a \count etc..., 3) functions which are recognized via an encountered opening parenthesis (but later this must be disambiguated from variables with tacit multiplication) 4) 5) 6) 7) acceptable components of a variable or function names: @, underscore, digits, letters (or chars of category code letter.)

The short lived 1.2d which followed the even shorter lived 1.2c managed to introduce a bug here as it removed the check for catcode 11 !, which must be recognized if ! is not to be taken as part of a variable name. Don't know what I was thinking, it was the time when I was moving the handling of tacit multiplication entirely to the \XINT\_expr\_getop side. Fixed in 1.2e.

I almost decided to remove the \ifcat\relax test whose rôle is to avoid the \string#1 to do something bad is the escape char is a digit! Perhaps I will remove it at some point ! I truly almost did it, but also the case of no escape char is a problem (\string\0, if \0 is a count ...)

The (indirectly) above means that via \XINT\_expr\_func then \XINT\_expr\_op\_\_ one goes back to \XINT\_expr\_getop then \XINT\_expr\_getop\_b which is the location where tacit multiplication is now centralized. This makes the treatment of tacit multiplication for situations such as <variable>\count or <variable>\xintexpr..\relax, perhaps a bit sub-optimal, but first the variable name must be gathered, second the variable must expand to its value.

```

482 \def\XINT_expr_scanfunc_b #1%
483 {%
484   \ifx !#1\xint_dothis{(_}\fi
485   \ifcat \relax#1\xint_dothis{(_}\fi
486   \if (#1\xint_dothis{\xint_firstoftwo{(`}}\fi
487   \if @#1\xint_dothis \XINT_expr_scanfunc_a \fi
488   \if _#1\xint_dothis \XINT_expr_scanfunc_a \fi
489   \ifnum \xint_c_ix<1\string#1 \xint_dothis \XINT_expr_scanfunc_a \fi
490   \ifcat a#1\xint_dothis \XINT_expr_scanfunc_a \fi
491   \xint_orthat {(_}%
492   #1%
493 }%

```

Comments written 2015/11/12: earlier there was an \ifcsname test for checking if we had a variable in front of a (, for tacit multiplication for example in x(y+z(x+w)) to work. But after I had implemented functions (that was yesterday...), I had the problem if was impossible to re-declare a variable name such as "f" as a function name. The problem is that here we can not test if the function is available because we don't know if we are in expr, iiexpr or floatexpr. The \xint\_c\_xviii causes all fetching operations to stop and control is handed over to the routines which will be expr, iiexpr ou floatexpr specific, i.e. the \XINT\_{expr|iiexpr|flexpr}\_op\_{`|\_} which are invoked by the until<op>\_b macros earlier in the stream. Functions may exist for one but not the

two other parsers. Variables are declared via one parser and usable in the others, but naturally `\xintiexpr` has its restrictions.

Thinking about this again I decided to treat a priori cases such as `x(...)` as functions, after having assigned to each variable a low-weight macro which will convert this into `_getop\.=<value of x>*(...)`. To activate that macro at the right time I could for this exploit the "onlitteral" intercept, which is parser independent (1.2c).

This led to me necessarily to rewrite partially the `seq`, `add`, `mul`, `subs`, `iter` ... routines as now the variables fetch only one token. I think the thing is more efficient.

1.2c had `\def\XINT_expr_func #1(#2{\xint_c_xviii #2{#1}}`

In `\XINT_expr_func` the `#2` is `_` if `#1` must be a variable name, or `#2=` if `#1` must be either a function name or possibly a variable name which will then have to be followed by tacit multiplication before the opening parenthesis.

The `\xint_c_xviii` is there because `_op_` must know in which parser it works. Dispendious for `_`. Hence I modify for 1.2d.

```
494 \def\XINT_expr_func #1(#2{\if _#2\xint_dothis\XINT_expr_op__\fi
495 \xint_orthat{\xint_c_xviii #2}{#1}}%
```

## 10.20 `\XINT_expr_getop`: finding the next operator or closing parenthesis or end of expression

Release 1.1 implements multi-character operators.

1.2d adds tacit multiplication also in front of variable or functions names starting with a letter, not only a `@` or a `_` as was already the case. This is for `(x+y)z` situations. It also applies higher precedence in cases like `x/2y` or `x/2@`, or `x/2max(3,5)`, or `x/2\xintexpr 3\relax`.

In fact, finally I decide that all sorts of tacit multiplication will always use the higher precedence.

Indeed I hesitated somewhat: with the current code one does not know if `\XINT_expr_getop` as invoked after a closing parenthesis or because a number parsing ended, and I felt distinguishing the two was unneeded extra stuff. This means cases like `(a+b)/(c+d)(e+f)` will first multiply the last two parenthesized terms.

The `!` starting a sub-expression must be distinguished from the post-fix `!` for factorial, thus we must not do a too early `\string`. In versions `< 1.2c`, the catcode `11 !` had to be identified in all branches of the number or function scans. Here it is simply treated as a special case of a letter.

```
496 \def\XINT_expr_getop #1#2% this #1 is the current locked computed value
497 {%
498   \expandafter\XINT_expr_getop_a\expandafter #1\romannumeral`&&@#2%
499 }%
500 \catcode`* 11
501 \def\XINT_expr_getop_a #1#2%
502 {%
503   \ifx \relax #2\xint_dothis\xint_firstofthree\fi
504   \ifcat \relax #2\xint_dothis\xint_secondofthree\fi
505   \if _#2\xint_dothis \xint_secondofthree\fi
506   \if @#2\xint_dothis \xint_secondofthree\fi
507   \if (#2\xint_dothis \xint_secondofthree\fi
508   \ifcat a#2\xint_dothis \xint_secondofthree\fi
509   \xint_orthat \xint_thirdofthree
510   {\XINT_expr_founded #1}%
511   {\XINT_expr_precedence_*** *#1#2}% tacit multiplication with higher precedence
512   {\expandafter\XINT_expr_getop_b \string#2#1}%
513 }%
514 \catcode`* 12
```

```
515 \def\XINT_expr_foundend {\xint_c_ \relax }% \relax is a place holder here.
```

? is a very special operator with top precedence which will check if the next token is another ?, while avoiding removing a brace pair from token stream due to its syntax. Pre 1.1 releases used : rather than ??, but we need : for Python like slices of lists.

```
516 \def\XINT_expr_getop_b #1%
517 {%
518     \if '#1\xint_dothis{\XINT_expr_binopwrdrd } \fi
519     \if ?#1\xint_dothis{\XINT_expr_precedence_? ?} \fi
520     \xint_orthat      {\XINT_expr_scanop_a #1}%
521 }%
522 \def\XINT_expr_binopwrdrd #1#2'\{\expandafter\XINT_expr_foundop_a
523     \csname XINT_expr_itself_\xint_zapspace #2 \xint_gobble_i\endcsname #1}%
524 \def\XINT_expr_scanop_a #1#2#3%
525     {\expandafter\XINT_expr_scanop_b\expandafter #1\expandafter #2\romannumeral`&&@#3}%
526 \def\XINT_expr_scanop_b #1#2#3%
527 {%
528     \ifcat#3\relax\xint_dothis{\XINT_expr_foundop_a #1#2#3}\fi
529     \ifcsname XINT_expr_itself_#1#3\endcsname
530     \xint_dothis
531         {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#3\endcsname #2}\fi
532     \xint_orthat {\XINT_expr_foundop_a #1#2#3}%
533 }%
534 \def\XINT_expr_scanop_c #1#2#3%
535 {%
536     \expandafter\XINT_expr_scanop_d\expandafter #1\expandafter #2\romannumeral`&&@#3%
537 }%
538 \def\XINT_expr_scanop_d #1#2#3%
539 {%
540     \ifcat#3\relax \xint_dothis{\XINT_expr_foundop #1#2#3}\fi
541     \ifcsname XINT_expr_itself_#1#3\endcsname
542     \xint_dothis
543         {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#3\endcsname #2}\fi
544     \xint_orthat {\csname XINT_expr_precedence_#1\endcsname #1#2#3}%
545 }%
546 \def\XINT_expr_foundop_a #1%
547 {%
548     \ifcsname XINT_expr_precedence_#1\endcsname
549     \csname XINT_expr_precedence_#1\endcsname\expandafter\endcsname
550     \expandafter #1%
551     \else
552     \xint_afterfi{\XINT_expr_unknown_operator {#1}\XINT_expr_getop}%
553     \fi
554 }%
555 \def\XINT_expr_unknown_operator #1{\xintError:removed \xint_gobble_i {#1}}%
556 \def\XINT_expr_foundop #1{\csname XINT_expr_precedence_#1\endcsname #1}%
```

## 10.21 Expansion spanning; opening and closing parentheses

Version 1.1 had a hack inside the until macros for handling the omit and abort in iterations over dummy variables. This has been removed by 1.2c, see the subsection where omit and abort are discussed.

```

557 \catcode`) 11
558 \def\XINT_tmpa #1#2#3#4%
559 {%
560   \def#1##1%
561   {%
562     \xint_UDsignfork
563       ##1{\expandafter#1\romannumeral`&&@#3}%
564       -{#2##1}%
565     \krof
566   }%
567 \def#2##1##2%
568 {%
569   \ifcase ##1\expandafter\XINT_expr_done
570   \or\xint_afterfi{\XINT_expr_extra_)
571     \expandafter #1\romannumeral`&&\XINT_expr_getop }%
572   \else
573   \xint_afterfi{\expandafter#1\romannumeral`&&\csname XINT_#4_op_##2\endcsname }%
574   \fi
575 }%
576 }%
577 \def\XINT_expr_extra_) {\xintError:removed }%
578 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
579   \expandafter\XINT_tmpa
580   \csname XINT_#1_until_end_a\expandafter\endcsname
581   \csname XINT_#1_until_end_b\expandafter\endcsname
582   \csname XINT_#1_op_-vi\endcsname
583   {#1}%
584 }%
585 \def\XINT_tmpa #1#2#3#4#5#6%
586 {%
587   \def #1##1{\expandafter #3\romannumeral`&&\XINT_expr_getnext }%
588   \def #2{\expandafter #3\romannumeral`&&\XINT_expr_getnext }%
589   \def #3##1{\xint_UDsignfork
590     ##1{\expandafter #3\romannumeral`&&@#5}%
591     -{#4##1}%
592     \krof }%
593   \def #4##1##2{\ifcase ##1\expandafter\XINT_expr_missing_)
594   \or \csname XINT_#6_op_##2\expandafter\endcsname
595   \else
596   \xint_afterfi{\expandafter #3\romannumeral`&&\csname XINT_#6_op_##2\endcsname }%
597   \fi
598 }%
599 }%
600 \def\XINT_expr_missing_) {\xintError:inserted \xint_c_ \XINT_expr_done }%

We should be using until_( notation to stay synchronous with until_+, until_* etc..., but I found
that until_) was more telling.

601 \catcode`) 12
602 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
603   \expandafter\XINT_tmpa
604   \csname XINT_#1_op_(\expandafter\endcsname
605   \csname XINT_#1_oparen\expandafter\endcsname

```

```

606 \csname XINT_#1_until_)_a\expandafter\endcsname
607 \csname XINT_#1_until_)_b\expandafter\endcsname
608 \csname XINT_#1_op_-vi\endcsname
609 {#1}%
610 }%
611 \expandafter\let\csname XINT_expr_precedence_)\endcsname\xint_c_i

```

## 10.22 |, ||, &, &&, <, >, =, ==, <=, >=, !=, +, -, \*, /, ^, \*\*, //, /:, ..[, ].., ][, ][:, :], and ++ operators

10.22.1	Square brackets for lists, the !? for omit and abort, and the ++ postfix construct . . . .	286
10.22.2	The  , &, xor, <, >, =, <=, >=, !=, //, /:, .., +, -, *, /, ^, ..[, and ].. operators for expr, floatexpr and iexpr operators . . . . .	287
10.22.3	The ]+, ]-, ]*, ]/, ]^, +[, -[, *[, /[, and ^[ list operators . . . . .	289
10.22.4	The 'and', 'or', 'xor', and 'mod' as infix operator words . . . . .	291
10.22.5	The   , &&, **, **[, ]** operators as synonyms . . . . .	291
10.22.6	List selectors: [list][N], [list][:b], [list][a:], [list][a:b] . . . . .	292
10.22.7	\xintListSel::csv . . . . .	294

### 10.22.1 Square brackets for lists, the !? for omit and abort, and the ++ postfix construct

This is all very clever and only need setting some suitable precedence levels, if only I could understand what I did in 2014... just joking. Notice that op\_) macros are defined here in the \xintFor loop.

There is some clever business going on here with the letter a for handling constructs such as [3..5]\*2 (I think...).

1.2c has replaced 1.1's private dealings with "^C" (which was done before dummy variables got implemented) by use of "!?". See discussion of omit and abort.

```

612 \expandafter\let\csname XINT_expr_precedence_]\endcsname\xint_c_i
613 \expandafter\let\csname XINT_expr_precedence_;\endcsname\xint_c_i
614 \let\XINT_expr_precedence_a \xint_c_xviii
615 \let\XINT_expr_precedence_!? \xint_c_ii
616 \expandafter\let\csname XINT_expr_precedence_++)\endcsname \xint_c_i

```

Comments added 2015/11/13 Here we have in particular the mechanism for post action on lists via op\_] The precedence\_] is the one of a closing parenthesis. We need the closing parenthesis to do its job, hence we can not define a op\_]+ operator for example, as we want to assign it the precedence of addition not the one of closing parenthesis. The trick I used in 1.1 was to let the op\_] insert the letter a, this letter exceptionnally also being a legitimate operator, launch the \_getop and let it find a \*, +, a/, a-, a^, a\*\* operator standing for ], +, ], ^, ]\*\* postfix item by item list operator. I thought I had in mind an example to show that having defined op\_a and precedence\_a for the letter a caused a reduction in syntax for this letter, but it seems I am lacking now an example.

2015/11/18: for 1.2d I accelerate \XINT\_expr\_op\_] to jump over the \XINT\_expr\_getop\_a which now does tacit multiplications also in front of letters, for reasons of things like, (x+y)z, hence it must not see the "a". I could have used a catcode12 a possibly, but anyhow jumping straight to \XINT\_expr\_scanop\_a skips a few expansion steps (up to the potential price of less conceptual programming if I change things in the future.)

```

617 \catcode`. 11 \catcode`= 11 \catcode`+ 11
618 \xintFor #1 in {expr,flexpr,iexpr} \do {%
619 \expandafter\let\csname XINT_#1_op_)\endcsname \XINT_expr_getop
620 \expandafter\let\csname XINT_#1_op_;\endcsname \space

```

```

621 \expandafter\def\csname XINT_#1_op_]\endcsname ##1{\XINT_expr_scanop_a a##1}%
622 \expandafter\let\csname XINT_#1_op_a]\endcsname \XINT_expr_getop

```

1.1 2014/10/29 did `\expandafter\.=\xintiCeil` which transformed it into `\romannumeral0\xintiCeil`, which seems a bit weird. This exploited the fact that dummy variables macros could back then pick braced material (which in the case at hand here ended being `{\romannumeral0\xintiCeil...}` and were submitted to two expansions. The result of this was to provide a not value which got expanded only in the first loop of the `:A` and following macros of `seq`, `iter`, `rseq`, etc...

Anyhow with 1.2c I have changed the implementation of dummy variables which now need to fetch a single locked token, which they do not expand.

The `\xintiCeil` appears a bit dispensious, but I need the starting value in a `\numexpr` compatible form in the iteration loops.

```

623 \expandafter\def\csname XINT_#1_op_++\endcsname ##1##2\relax
624 {\expandafter\XINT_expr_foundend \expandafter
625   {\expandafter\.=\csname .=\xintiCeil{\XINT_expr_unlock ##1}\endcsname }}%
626 }%
627 \catcode`. 12 \catcode`= 12 \catcode`+ 12

```

1.2d adds the `***` for tying via tacit multiplication, for example  $x/2y$ . Actually I don't need the `itself` mechanism for `***`, only a precedence.

```
628 \catcode\& 12
629 \xintFor* #1 in {{==}{<=}{>=}{!=}{&&}{||}{**}{//}{/:}{..}{..[]{}]}{..}%
630      {+[-[*[/[**[^[{a+]{a-}{a*}{a/}{a**}{a^}%
631      [][][:]{:}]!}?{++}{++}]]%{***}}
632     \do {\expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}}%
633 \catcode\& 7
634 \expandafter\let\csname XINT_expr_precedence_***\endcsname \xint_c_viii
```

### 10.22.2 The |, &, xor, <, >, =, <=, >=, !=, //, /:, .., +, -, \*, /, ^, ..[, and ].. operators for expr, floatexpr and iixpr operators

1.2d needed some room between /, \* and ^. Hence precedence for ^ is now at 9

```

635 \def\XINT_expr_defbin_c #1#2#3#4#5#6#7#8%
636 {%
637   \def #1##1% \XINT_expr_op_<op> ou flexpr ou iexpr
638   {% keep value, get next number and operator, then do until
639     \expandafter #2\expandafter ##1%
640     \romannumeral`&&@\expandafter\XINT_expr_getnext }%
641   \def #2##1##2% \XINT_expr_until_<op>_a ou flexpr ou iexpr
642   {\xint_UDsignfork ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
643     -{#3##1##2}%
644     \krof }%
645   \def #3##1##2##3##4% \XINT_expr_until_<op>_b ou flexpr ou iexpr
646   {% either execute next operation now, or first do next (possibly unary)
647     \ifnum ##2>#7%
648       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
649         \csname XINT_#8_op_##3\endcsname {##4}}}%
650     \else \xint_afterfi {\expandafter #2\expandafter ##3%
651       \csname .=#6{\XINT_expr_unlock ##1}{\XINT_expr_unlock ##4}\endcsname }%
652       \fi }%
653   \let #7#5%

```

```

654 }%
655 \def\XINT_expr_defbin_b #1#2#3#4#5%
656 {%
657   \expandafter\XINT_expr_defbin_c
658   \csname XINT_#1_op_#2\expandafter\endcsname
659   \csname XINT_#1_until_#2_a\expandafter\endcsname
660   \csname XINT_#1_until_#2_b\expandafter\endcsname
661   \csname XINT_#1_op_#4\expandafter\endcsname
662   \csname xint_c_#3\expandafter\endcsname
663   \csname #5\expandafter\endcsname
664   \csname XINT_expr_precedence_#2\endcsname {#1}%
665 }%
666 \XINT_expr_defbin_b {expr} | {iii}{vi} {xintOR}%
667 \XINT_expr_defbin_b {flexpr} | {iii}{vi} {xintOR}%
668 \XINT_expr_defbin_b {iiexpr} | {iii}{vi} {xintOR}%
669 \XINT_expr_defbin_b {expr} & {iv}{vi} {xintAND}%
670 \XINT_expr_defbin_b {flexpr} & {iv}{vi} {xintAND}%
671 \XINT_expr_defbin_b {iiexpr} & {iv}{vi} {xintAND}%
672 \XINT_expr_defbin_b {expr} {xor}{iii}{vi} {xintXOR}%
673 \XINT_expr_defbin_b {flexpr}{xor}{iii}{vi} {xintXOR}%
674 \XINT_expr_defbin_b {iiexpr}{xor}{iii}{vi} {xintXOR}%
675 \XINT_expr_defbin_b {expr} < {v}{vi} {xintLt}%
676 \XINT_expr_defbin_b {flexpr} < {v}{vi} {xintLt}%
677 \XINT_expr_defbin_b {iiexpr} < {v}{vi} {xintiiLt}%
678 \XINT_expr_defbin_b {expr} > {v}{vi} {xintGt}%
679 \XINT_expr_defbin_b {flexpr} > {v}{vi} {xintGt}%
680 \XINT_expr_defbin_b {iiexpr} > {v}{vi} {xintiiGt}%
681 \XINT_expr_defbin_b {expr} = {v}{vi} {xintEq}%
682 \XINT_expr_defbin_b {flexpr} = {v}{vi} {xintEq}%
683 \XINT_expr_defbin_b {iiexpr} = {v}{vi} {xintiiEq}%
684 \XINT_expr_defbin_b {expr} {<=} {v}{vi} {xintLtorEq}%
685 \XINT_expr_defbin_b {flexpr}{<=} {v}{vi} {xintLtorEq}%
686 \XINT_expr_defbin_b {iiexpr}{<=} {v}{vi} {xintiiLtorEq}%
687 \XINT_expr_defbin_b {expr} {>=} {v}{vi} {xintGtorEq}%
688 \XINT_expr_defbin_b {flexpr}{>=} {v}{vi} {xintGtorEq}%
689 \XINT_expr_defbin_b {iiexpr}{>=} {v}{vi} {xintiiGtorEq}%
690 \XINT_expr_defbin_b {expr} {!=} {v}{vi} {xintNeq}%
691 \XINT_expr_defbin_b {flexpr}{!=} {v}{vi} {xintNeq}%
692 \XINT_expr_defbin_b {iiexpr}{!=} {v}{vi} {xintiiNeq}%
693 \XINT_expr_defbin_b {expr} {...} {iii}{vi} {xintSeq::csv}%
694 \XINT_expr_defbin_b {flexpr}{...} {iii}{vi} {xintSeq::csv}%
695 \XINT_expr_defbin_b {iiexpr}{...} {iii}{vi} {xintiiSeq::csv}%
696 \XINT_expr_defbin_b {expr} {//} {vii}{vii}{xintDivTrunc}%
697 \XINT_expr_defbin_b {flexpr}{//} {vii}{vii}{xintDivTrunc}%
698 \XINT_expr_defbin_b {iiexpr}{//} {vii}{vii}{xintiiDivTrunc}%
699 \XINT_expr_defbin_b {expr} {/:} {vii}{vii}{xintMod}%
700 \XINT_expr_defbin_b {flexpr}{/:} {vii}{vii}{xintMod}%
701 \XINT_expr_defbin_b {iiexpr}{/:} {vii}{vii}{xintiiMod}%
702 \XINT_expr_defbin_b {expr} + {vi}{vi} {xintAdd}%
703 \XINT_expr_defbin_b {flexpr} + {vi}{vi} {XINTinFloatAdd}%
704 \XINT_expr_defbin_b {iiexpr} + {vi}{vi} {xintiiAdd}%
705 \XINT_expr_defbin_b {expr} - {vi}{vi} {xintSub}%

```



```

706 \XINT_expr_defbin_b {flexpr} - {vi}{vi} {XINTinFloatSub}%
707 \XINT_expr_defbin_b {iiexpr} - {vi}{vi} {xintiiSub}%
708 \XINT_expr_defbin_b {expr} * {vii}{vii}{xintMul}%
709 \XINT_expr_defbin_b {flexpr} * {vii}{vii}{XINTinFloatMul}%
710 \XINT_expr_defbin_b {iiexpr} * {vii}{vii}{xintiiMul}%
711 \XINT_expr_defbin_b {expr} / {vii}{vii}{xintDiv}%
712 \XINT_expr_defbin_b {flexpr} / {vii}{vii}{XINTinFloatDiv}%
713 \XINT_expr_defbin_b {iiexpr} / {vii}{vii}{xintiiDivRound}% CHANGED IN 1.1!
714 \XINT_expr_defbin_b {expr} ^ {ix}{ix} {xintPow}%
715 \XINT_expr_defbin_b {flexpr} ^ {ix}{ix} {XINTinFloatPowerH}%
716 \XINT_expr_defbin_b {iiexpr} ^ {ix}{ix} {xintiiPow}%
717 \XINT_expr_defbin_b {expr} {..[]}{iii}{vi} {xintSeqA::csv}%
718 \XINT_expr_defbin_b {flexpr}{..[]}{iii}{vi} {XINTinFloatSeqA::csv}%
719 \XINT_expr_defbin_b {iiexpr}{..[]}{iii}{vi} {xintiiSeqA::csv}%
720 \XINT_expr_defbin_b {expr} {[]}{}{iii}{vi} {xintSeqB::csv}%
721 \XINT_expr_defbin_b {flexpr}{[]}{}{iii}{vi} {XINTinFloatSeqB::csv}%
722 \XINT_expr_defbin_b {iiexpr}{[]}{}{iii}{vi} {xintiiSeqB::csv}%

```

### 10.22.3 The `+`, `-`, `*`, `/`, `^`, `+`, `-`, `*`, `/`, and `^` list operators

**`\XINT_expr_binop_inline_b`** This handles acting on comma separated values (no need to bother about spaces in this context; expansion in a `\csname...\endcsname`).

```

723 \def\XINT_expr_binop_inline_a
724   {\expandafter\xint_gobble_i\romannumeral`&&\XINT_expr_binop_inline_b}%
725 \def\XINT_expr_binop_inline_b #1#2,{\XINT_expr_binop_inline_c #2,{#1}}%
726 \def\XINT_expr_binop_inline_c #1{%
727   \if ,#1\xint_dothis\XINT_expr_binop_inline_e\fi
728   \if ^#1\xint_dothis\XINT_expr_binop_inline_end\fi
729   \xint_orthat\XINT_expr_binop_inline_d #1}%
730 \def\XINT_expr_binop_inline_d #1,#2,{#2{#1}\XINT_expr_binop_inline_b {#2}}%
731 \def\XINT_expr_binop_inline_e #1,#2,{\XINT_expr_binop_inline_b {#2}}%
732 \def\XINT_expr_binop_inline_end #1,#2{%
733 \def\XINT_expr_deflistopr_c #1#2#3#4#5#6#7#8%
734 {%
735   \def #1#1% \XINT_expr_op_<op> ou flexpr ou iiexpr
736   {% keep value, get next number and operator, then do until
737     \expandafter #2\expandafter ##1%
738     \romannumeral`&&\expandafter\XINT_expr_getnext}%
739   \def #2#1##2% \XINT_expr_until_<op>_a ou flexpr ou iiexpr
740   {\xint_UDsignfork ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
741     -{#3##1##2}%
742     \krof}%
743   \def #3#1##2##3##4% \XINT_expr_until_<op>_b ou flexpr ou iiexpr
744   {% either execute next operation now, or first do next (possibly unary)
745     \ifnum ##2>#7%
746     \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
747       \csname XINT_#8_op_##3\endcsname {##4}}%
748     \else \xint_afterfi {\expandafter ##2\expandafter ##3%
749       \csname .=\expandafter\XINT_expr_binop_inline_a\expandafter
750       {\expandafter\expandafter\expandafter#6\expandafter
751       \xint_exchangetwo_keepbraces\expandafter
752       {\expandafter\XINT_expr_unlock\expandafter ##4\expandafter}\expandafter}%

```

```

753      \romannumeral`&&\XINT_expr_unlock ##1,^,\endcsname }%
754    \fi }%
755    \let #7#5%
756 }%
757 \def\XINT_expr_deflistopr_b #1#2#3#4%
758 {%
759   \expandafter\XINT_expr_deflistopr_c
760   \csname XINT_#1_op_#2\expandafter\endcsname
761   \csname XINT_#1_until_#2_a\expandafter\endcsname
762   \csname XINT_#1_until_#2_b\expandafter\endcsname
763   \csname XINT_#1_op_#3\expandafter\endcsname
764   \csname xint_c_#3\expandafter\endcsname
765   \csname #4\expandafter\endcsname
766   \csname XINT_expr_precedence_#2\endcsname {#1}%
767 }%

```

This is for  $[x..y]^z$  syntax etc.... Attention that with 1.2d, precedence level of  $^$  raised to ix to make room for  $***$ .

```

768 \XINT_expr_deflistopr_b {expr} {a+}{vi} {xintAdd}%
769 \XINT_expr_deflistopr_b {expr} {a-}{vi} {xintSub}%
770 \XINT_expr_deflistopr_b {expr} {a*}{vii} {xintMul}%
771 \XINT_expr_deflistopr_b {expr} {a/}{vii} {xintDiv}%
772 \XINT_expr_deflistopr_b {expr} {a^}{ix} {xintPow}%
773 \XINT_expr_deflistopr_b {iiexpr}{a+}{vi} {xintiiAdd}%
774 \XINT_expr_deflistopr_b {iiexpr}{a-}{vi} {xintiiSub}%
775 \XINT_expr_deflistopr_b {iiexpr}{a*}{vii} {xintiiMul}%
776 \XINT_expr_deflistopr_b {iiexpr}{a/}{vii} {xintiiDivRound}%
777 \XINT_expr_deflistopr_b {iiexpr}{a^}{ix} {xintiiPow}%
778 \XINT_expr_deflistopr_b {flexpr}{a+}{vi} {XINTinFloatAdd}%
779 \XINT_expr_deflistopr_b {flexpr}{a-}{vi} {XINTinFloatSub}%
780 \XINT_expr_deflistopr_b {flexpr}{a*}{vii} {XINTinFloatMul}%
781 \XINT_expr_deflistopr_b {flexpr}{a/}{vii} {XINTinFloatDiv}%
782 \XINT_expr_deflistopr_b {flexpr}{a^}{ix} {XINTinFloatPowerH}%
783 \def\XINT_expr_deflistopl_c #1#2#3#4#5#6#7%
784 {%
785   \def #1##1{\expandafter#2\expandafter##1\romannumeral`&&%
786     \expandafter #3\romannumeral`&&\XINT_expr_getnext }%
787   \def #2##1##2##3##4%
788   {% either execute next operation now, or first do next (possibly unary)
789     \ifnum ##2>#6%
790       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&%
791         \csname XINT_#7_op_##3\endcsname {##4}}%
792       \else \xint_afterfi {\expandafter ##2\expandafter ##3%
793         \csname .=\expandafter\XINT_expr_binop_inline_a\expandafter
794           {\expandafter#5\expandafter
795             {\expandafter\XINT_expr_unlock\expandafter ##1\expandafter}\expandafter}%
796           \romannumeral`&&\XINT_expr_unlock ##4,^,\endcsname }%
797       \fi }%
798   \let #6#4%
799 }%
800 \def\XINT_expr_deflistopl_b #1#2#3#4%
801 {%

```

```

802 \expandafter\XINT_expr_deflistopl_c
803 \csname XINT_#1_op_#2\expandafter\endcsname
804 \csname XINT_#1_until_#2\expandafter\endcsname
805 \csname XINT_#1_until_)_a\expandafter\endcsname
806 \csname xint_c_#3\expandafter\endcsname
807 \csname #4\expandafter\endcsname
808 \csname XINT_expr_precedence_#2\endcsname {#1}%
809 }%

```

This is for  $z*[x..y]$  syntax etc...

```

810 \XINT_expr_deflistopl_b {expr} {+[]{}vi} {xintAdd}%
811 \XINT_expr_deflistopl_b {expr} {-[]{}vi} {xintSub}%
812 \XINT_expr_deflistopl_b {expr} {*[]{}vii} {xintMul}%
813 \XINT_expr_deflistopl_b {expr} {/[]{}vii} {xintDiv}%
814 \XINT_expr_deflistopl_b {expr} {^[]{}ix} {xintPow}%
815 \XINT_expr_deflistopl_b {iiexpr}{+[]{}vi} {xintiiAdd}%
816 \XINT_expr_deflistopl_b {iiexpr}{-[]{}vi} {xintiiSub}%
817 \XINT_expr_deflistopl_b {iiexpr}{*[]{}vii} {xintiiMul}%
818 \XINT_expr_deflistopl_b {iiexpr}{/[]{}vii} {xintiiDivRound}%
819 \XINT_expr_deflistopl_b {iiexpr}{^[]{}ix} {xintiiPow}%
820 \XINT_expr_deflistopl_b {fexpr}{+[]{}vi} {XINTinFloatAdd}%
821 \XINT_expr_deflistopl_b {fexpr}{-[]{}vi} {XINTinFloatSub}%
822 \XINT_expr_deflistopl_b {fexpr}{*[]{}vii} {XINTinFloatMul}%
823 \XINT_expr_deflistopl_b {fexpr}{/[]{}vii} {XINTinFloatDiv}%
824 \XINT_expr_deflistopl_b {fexpr}{^[]{}ix} {XINTinFloatPowerH}%

```

#### 10.22.4 The 'and', 'or', 'xor', and 'mod' as infix operator words

```

825 \xintFor #1 in {and,or,xor,mod} \do {%
826   \expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}}%
827 \expandafter\let\csname XINT_expr_precedence_and\expandafter\endcsname
828   \csname XINT_expr_precedence_&\endcsname
829 \expandafter\let\csname XINT_expr_precedence_or\expandafter\endcsname
830   \csname XINT_expr_precedence_| \endcsname
831 \expandafter\let\csname XINT_expr_precedence_mod\expandafter\endcsname
832   \csname XINT_expr_precedence_/: \endcsname
833 \xintFor #1 in {expr, fexpr, iiexpr} \do {%
834   \expandafter\let\csname XINT_#1_op_and\expandafter\endcsname
835     \csname XINT_#1_op_&\endcsname
836   \expandafter\let\csname XINT_#1_op_or\expandafter\endcsname
837     \csname XINT_#1_op_| \endcsname
838   \expandafter\let\csname XINT_#1_op_mod\expandafter\endcsname
839     \csname XINT_#1_op_/: \endcsname
840 }%

```

#### 10.22.5 The ||, &&, \*\*, \*\*[, ]\*\* operators as synonyms

```

841 \expandafter\let\csname XINT_expr_precedence_==\expandafter\endcsname
842   \csname XINT_expr_precedence_=\endcsname
843 \expandafter\let\csname XINT_expr_precedence_&\string&\expandafter\endcsname
844   \csname XINT_expr_precedence_&\endcsname
845 \expandafter\let\csname XINT_expr_precedence_| |\expandafter\endcsname
846   \csname XINT_expr_precedence_| \endcsname
847 \expandafter\let\csname XINT_expr_precedence_**\expandafter\endcsname

```

```

848         \csname XINT_expr_precedence_^\endcsname
849 \expandafter\let\csname XINT_expr_precedence_a**\expandafter\endcsname
850         \csname XINT_expr_precedence_a^\endcsname
851 \expandafter\let\csname XINT_expr_precedence_**[\expandafter\endcsname
852         \csname XINT_expr_precedence_^\endcsname
853 \xintFor #1 in {expr, flexpr, iiexpr} \do {%
854   \expandafter\let\csname XINT_#1_op_==\expandafter\endcsname
855         \csname XINT_#1_op_=\endcsname
856   \expandafter\let\csname XINT_#1_op_&\string&\expandafter\endcsname
857         \csname XINT_#1_op_&\endcsname
858   \expandafter\let\csname XINT_#1_op_||\expandafter\endcsname
859         \csname XINT_#1_op_|\endcsname
860   \expandafter\let\csname XINT_#1_op_**\expandafter\endcsname
861         \csname XINT_#1_op_^\endcsname
862   \expandafter\let\csname XINT_#1_op_a**\expandafter\endcsname
863         \csname XINT_#1_op_a^\endcsname
864   \expandafter\let\csname XINT_#1_op_**[\expandafter\endcsname
865         \csname XINT_#1_op_^\endcsname
866 }%
```

### 10.22.6 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>. I do not implement third argument giving the step. Also, Python [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 do that. For reversing I could allow [::-1] syntax 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.

The \xintListSel::csv was named \xintListSel:csv, but as it not only extracts one item but may produce csv, I renamed it.

```

867 \def\XINT_tmpa #1#2#3#4#5#6%
868 {%
869   \def #1##1% \XINT_expr_op_[
870   {%
871     \expandafter #2\expandafter ##1\romannumeral`&&\XINT_expr_getnext
872   }%
873   \def #2##1##2% \XINT_expr_until_[_a
874   {\xint_UDsignfork
875     ##2{\expandafter #2\expandafter ##1\romannumeral`&&#4}%
876     -{#3##1##2}%
877   \krof }%
878   \def #3##1##2##3##4% \XINT_expr_until_[_b
879   {%
880     \ifnum ##2>#5%
881       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
882         \csname XINT_#6_op_##3\endcsname {##4}}%
883     \else
884       \xint_afterfi
885       {\expandafter ##2\expandafter ##3\csname
886         .=\expandafter\xintListSel::csv \romannumeral`&&\XINT_expr_unlock ##4;%
887         \XINT_expr_unlock ##1;\endcsname % unlock added for \xintNewExpr
888       }%
```

```

889     \fi
890   }%
891   \let #5\xint_c_ii
892 }%
893 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
894 \expandafter\XINT_tmpa
895   \csname XINT_#1_op_][\expandafter\endcsname
896   \csname XINT_#1_until_][_a\expandafter\endcsname
897   \csname XINT_#1_until_][_b\expandafter\endcsname
898   \csname XINT_#1_op_-vi\expandafter\endcsname
899   \csname XINT_expr_precedence_][\endcsname {#1}%
900 }%
901 \def\XINT_tmpa #1#2#3#4#5#6%
902 {%
903   \def #1##1% \XINT_expr_op_:
904   {%
905     \expandafter #2\expandafter ##1\romannumeral`&&\XINT_expr_getnext
906   }%
907   \def #2##1##2% \XINT_expr_until_:_a
908   {\xint_UDsignfork
909     ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
910     -{##3##1##2}%
911   \krof }%
912   \def #3##1##2##3##4% \XINT_expr_until_:_b
913   {%
914     \ifnum ##2>#5%
915       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&%
916         \csname XINT_#6_op_##3\endcsname {##4}}%
917     \else
918       \xint_afterfi
919       {\expandafter ##2\expandafter ##3\csname
920         .:=\xintiiifSgn{\XINT_expr_unlock ##1}NPP.%
921         \xintiiifSgn{\XINT_expr_unlock ##4}NPP.%
922         \xintNum{\XINT_expr_unlock ##1};\xintNum{\XINT_expr_unlock ##4}\endcsname
923       }%
924     \fi
925   }%
926   \let #5\xint_c_iii
927 }%
928 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
929 \expandafter\XINT_tmpa
930   \csname XINT_#1_op_:\expandafter\endcsname
931   \csname XINT_#1_until_:_a\expandafter\endcsname
932   \csname XINT_#1_until_:_b\expandafter\endcsname
933   \csname XINT_#1_op_-vi\expandafter\endcsname
934   \csname XINT_expr_precedence_:\endcsname {#1}%
935 }%
936 \catcode`[ 11 \catcode`] 11
937 \let\XINT_expr_precedence_:\ \xint_c_iii
938 \def\XINT_expr_op_:] #1{\expandafter\xint_c_i\expandafter }%
939   \csname .:=\xintiiifSgn{\XINT_expr_unlock #1}npp\XINT_expr_unlock #1\endcsname }%
940 \let\XINT_flexpr_op_:] \XINT_expr_op_:]

```

```
941 \let\XINT_iiexpr_op_[] \XINT_expr_op_[]
942 \let\XINT_expr_precedence_[][: \xint_c_iii
```

At the end of the replacement text of `\XINT_expr_op_[][:`, the `:` after index 0 must be catcode 12, else will be mistaken for the start of variable by expression parser (as `<digits><variable>` is allowed by the syntax and does tacit multiplication).

```
943 \edef\XINT_expr_op_[][: #1{\xint_c_ii \expandafter\noexpand
944 \csname XINT_expr_itself_[][\endcsname #10\string :}%
```

## 10.22.7 `\xintListSel::csv`

Some complications here are due to `\xintNewExpr` matters.

```
945 \let\XINT_flexpr_op_[][: \XINT_expr_op_[][:
946 \let\XINT_iiexpr_op_[][: \XINT_expr_op_[][:
947 \catcode`[ 12 \catcode`] 12
948 \def\xintListSel::csv #1{%
949   \if ]\noexpand#1\xint_dothis{\expandafter\XINT_listsel:_s\romannumeral`&&@}\fi
950   \if : \noexpand#1\xint_dothis{\XINT_listsel:_:}\fi
951   \xint_orthat {\XINT_listsel:_nth #1}%
952 }%
953 \def\XINT_listsel:_s #1{\if p#1\expandafter\XINT_listsel:_trim\else
954   \expandafter\XINT_listsel:_keep\fi }%
955 \def\XINT_listsel:_: #1.#2.{\csname XINT_listsel:_#1#2\endcsname }%
956 \def\XINT_listsel:_trim #1;#2;%
957   {\xintListWithSep,{\xintTrim {\xintNum{#1}}{\xintCSVtoListNonStripped{#2}}}}}%
958 \def\XINT_listsel:_keep #1;#2;%
959   {\xintListWithSep,{\xintKeep {\xintNum{#1}}{\xintCSVtoListNonStripped{#2}}}}}%
960 \def\XINT_listsel:_nth#1;#2;%
961   {\xintNthElt {\xintNum{#1}}{\xintCSVtoListNonStripped{#2}}}}}%
962 \def\XINT_listsel:_PP #1;#2;#3;%
963   {\xintListWithSep,%
964     {\xintTrim {\xintNum{#1}}{\xintKeep {\xintNum{#2}}{\xintCSVtoListNonStripped{#3}}}}}%
965   }%
966 \def\XINT_listsel:_NN #1;#2;#3;%
967   {\xintListWithSep,%
968     {\xintTrim {\xintNum{#2}}{\xintKeep {\xintNum{#1}}{\xintCSVtoListNonStripped{#3}}}}}%
969   }%
970 \def\XINT_listsel:_NP #1;#2;#3;%
971   {\expandafter\XINT_listsel:_NP_a \the\numexpr #1+%
972     \xintNthElt{0}{\xintCSVtoListNonStripped{#3}};#2;#3;%
973 \def\XINT_listsel:_NP_a #1#2;{\if -#1\expandafter\XINT_listsel:_OP\fi
974   \XINT_listsel:_PP #1#2;%
975 \def\XINT_listsel:_OP\XINT_listsel:_PP #1;{\XINT_listsel:_PP 0;%
976 \def\XINT_listsel:_PN #1;#2;#3;%
977   {\expandafter\XINT_listsel:_PN_a \the\numexpr #2+%
978     \xintNthElt{0}{\xintCSVtoListNonStripped{#3}};#1;#3;%
979 \def\XINT_listsel:_PN_a #1#2;#3;{\if -#1\expandafter\XINT_listsel:_PO\fi
980   \XINT_listsel:_PP #3;#1#2;%
981 \def\XINT_listsel:_PO\XINT_listsel:_PP #1;#2;{\XINT_listsel:_PP #1;0;%
```

## 10.23 Macros for a..b list generation

10.23.1	<code>\xintSeq::csv</code>	295
10.23.2	<code>\xintiiSeq::csv</code>	296

Ne produit que des listes d'entiers inférieurs à la borne de TeX ! mais sous la forme  $N/1[0]$  en ce qui concerne `\xintSeq::csv`.

### 10.23.1 `\xintSeq::csv`

Commence par remplacer a par `ceil(a)` et b par `floor(b)` et renvoie ensuite les entiers entre les deux, possiblement en décroissant, et extrémités comprises. Si  $a=b$  est non entier en obtient donc `ceil(a)` et `floor(a)`. Ne renvoie jamais une liste vide.

Note: le a..b dans `\xintfloatexpr` utilise cette routine.

```

982 \def\xintSeq::csv {\romannumeral0\xintseq::csv}%
983 \def\xintseq::csv #1#2%
984 {%
985   \expandafter\XINT_seq::csv\expandafter
986     {\the\numexpr \xintiCeil{#1}\expandafter}\expandafter
987     {\the\numexpr \xintiFloor{#2}}}%
988}%
989 \def\XINT_seq::csv #1#2%
990 {%
991   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
992     \expandafter\XINT_seq::csv_z
993   \or
994     \expandafter\XINT_seq::csv_p
995   \else
996     \expandafter\XINT_seq::csv_n
997   \fi
998   {#2}{#1}%
999}%
1000 \def\XINT_seq::csv_z #1#2{ #1/1[0]}%
1001 \def\XINT_seq::csv_p #1#2%
1002 {%
1003   \ifnum #1>#2
1004     \expandafter\expandafter\expandafter\XINT_seq::csv_p
1005   \else
1006     \expandafter\XINT_seq::csv_e
1007   \fi
1008   \expandafter{\the\numexpr #1-\xint_c_i}{#2},#1/1[0]}%
1009}%
1010 \def\XINT_seq::csv_n #1#2%
1011 {%
1012   \ifnum #1<#2
1013     \expandafter\expandafter\expandafter\XINT_seq::csv_n
1014   \else
1015     \expandafter\XINT_seq::csv_e
1016   \fi
1017   \expandafter{\the\numexpr #1+\xint_c_i}{#2},#1/1[0]}%
1018}%
1019 \def\XINT_seq::csv_e #1,{ }%
```

**10.23.2 `\xintiiSeq::csv`**

```

1020 \def\xintiiSeq::csv {\romannumeral0\xintiiseq::csv}%
1021 \def\xintiiseq::csv #1#2%
1022 {%
1023   \expandafter\XINT_iiseq::csv\expandafter
1024     {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
1025}%
1026 \def\XINT_iiseq::csv #1#2%
1027 {%
1028   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
1029   \expandafter\XINT_iiseq::csv_z
1030   \or
1031   \expandafter\XINT_iiseq::csv_p
1032   \else
1033   \expandafter\XINT_iiseq::csv_n
1034   \fi
1035   {#2}{#1}%
1036}%
1037 \def\XINT_iiseq::csv_z #1#2{ #1}%
1038 \def\XINT_iiseq::csv_p #1#2%
1039 {%
1040   \ifnum #1>#2
1041     \expandafter\expandafter\expandafter\XINT_iiseq::csv_p
1042   \else
1043     \expandafter\XINT_seq::csv_e
1044   \fi
1045   \expandafter{\the\numexpr #1-\xint_c_i}{#2},#1%
1046}%
1047 \def\XINT_iiseq::csv_n #1#2%
1048 {%
1049   \ifnum #1<#2
1050     \expandafter\expandafter\expandafter\XINT_iiseq::csv_n
1051   \else
1052     \expandafter\XINT_seq::csv_e
1053   \fi
1054   \expandafter{\the\numexpr #1+\xint_c_i}{#2},#1%
1055}%
1056 \def\XINT_seq::csv_e #1,{ }%

```

**10.24 Macros for a..**[d]**..b list generation**

10.24.1	<code>\xintSeqA::csv, \xintiiSeqA::csv, \XINTinFloatSeqA::csv</code>	296
10.24.2	<code>\xintSeqB::csv</code>	297
10.24.3	<code>\xintiiSeqB::csv</code>	297
10.24.4	<code>\XINTinFloatSeqB::csv</code>	298

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.

**10.24.1 `\xintSeqA::csv, \xintiiSeqA::csv, \XINTinFloatSeqA::csv`**

```

1057 \def\xintSeqA::csv #1%
1058   {\expandafter\XINT_seqa::csv\expandafter{\romannumeral0\xintra {#1}}}%

```



```

1059 \def\XINT_seqa::csv #1#2{\expandafter\XINT_seqa::csv_a \romannumeral0\xintra {#2};#1;}%
1060 \def\xintiiSeqA::csv #1{\expandafter\XINT_iiseqa::csv\expandafter{\romannumeral`&&@#1}}%
1061 \def\XINT_iiseqa::csv #1#2{\expandafter\XINT_seqa::csv_a\romannumeral`&&@#2;#1;}%
1062 \def\XINTinFloatSeqA::csv #1{\expandafter\XINT_flseqa::csv\expandafter
1063   {\romannumeral0\XINTinfloat [\XINTdigits]{#1}}}%
1064 \def\XINT_flseqa::csv #1#2%
1065   {\expandafter\XINT_seqa::csv_a\romannumeral0\XINTinfloat [\XINTdigits]{#2};#1;}%
1066 \def\XINT_seqa::csv_a #1{\xint_UDzerominusfork
1067   #1-{z}%
1068   0#1{n}%
1069   0-{p}%
1070   \krof #1}%

```

### 10.24.2 \xintSeqB::csv

With one year late documentation, let's just say, the #1 is \XINT\_expr\_unlock\.=Ua;b; with U=z or n or p, a=step, b=start.

```

1071 \def\xintSeqB::csv #1#2%
1072   {\expandafter\XINT_seqb::csv \expandafter{\romannumeral0\xintra{#2}}{#1}}%
1073 \def\XINT_seqb::csv #1#2{\expandafter\XINT_seqb::csv_a\romannumeral`&&@#2#1!}%
1074 \def\XINT_seqb::csv_a #1#2;#3;#4!\expandafter\XINT_expr_seq_empty?
1075   \romannumeral0\csname XINT_seqb::csv_#1\endcsname {#3}{#4}{#2}}%
1076 \def\XINT_seqb::csv_p #1#2#3%
1077 {%
1078   \xintifCmp {#1}{#2}{, #1\expandafter\XINT_seqb::csv_p\expandafter}%
1079   {, #1\xint_gobble_iii}{\xint_gobble_iii}%

```

\romannumeral0 stopped by \endcsname, XINT\_expr\_seq\_empty? constructs "nil".

```

1080   {\romannumeral0\xintadd {#3}{#1}}{#2}{#3}%
1081 }%
1082 \def\XINT_seqb::csv_n #1#2#3%
1083 {%
1084   \xintifCmp {#1}{#2}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1085   {, #1\expandafter\XINT_seqb::csv_n\expandafter}%
1086   {\romannumeral0\xintadd {#3}{#1}}{#2}{#3}%
1087 }%
1088 \def\XINT_seqb::csv_z #1#2#3{, #1}%

```

### 10.24.3 \xintiiSeqB::csv

```

1089 \def\xintiiSeqB::csv #1#2{\XINT_iiseqb::csv #1#2}%
1090 \def\XINT_iiseqb::csv #1#2#3#4%
1091   {\expandafter\XINT_iiseqb::csv_a
1092     \romannumeral`&&\expandafter \XINT_expr_unlock\expandafter#2%
1093     \romannumeral`&&\XINT_expr_unlock #4!}%
1094 \def\XINT_iiseqb::csv_a #1#2;#3;#4!\expandafter\XINT_expr_seq_empty?
1095   \romannumeral`&&\csname XINT_iiseqb::csv_#1\endcsname {#3}{#4}{#2}}%
1096 \def\XINT_iiseqb::csv_p #1#2#3%
1097 {%
1098   \xintSgnFork{\XINT_Cmp {#1}{#2}}{, #1\expandafter\XINT_iiseqb::csv_p\expandafter}%
1099   {, #1\xint_gobble_iii}{\xint_gobble_iii}%
1100   {\romannumeral0\xintiiadd {#3}{#1}}{#2}{#3}%

```

```

1101 }%
1102 \def\XINT_iiseqb::csv_n #1#2#3%
1103 {%
1104   \xintSgnFork{\XINT_Cmp {#1}{#2}}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1105   {, #1\expandafter\XINT_iiseqb::csv_n\expandafter}%
1106   {\romannumeral0\xintiadd {#3}{#1}}{#2}{#3}%
1107 }%
1108 \def\XINT_iiseqb::csv_z #1#2#3{, #1}%

```

#### 10.24.4 \XINTinFloatSeqB::csv

```

1109 \def\XINTinFloatSeqB::csv #1#2{\expandafter\XINT_flseqb::csv \expandafter
1110   {\romannumeral0\XINTinfloat [\XINTdigits]{#2}}{#1}}%
1111 \def\XINT_flseqb::csv #1#2{\expandafter\XINT_flseqb::csv_a\romannumeral`&&@#2#1!}%
1112 \def\XINT_flseqb::csv_a #1#2;#3;#4!\expandafter\XINT_expr_seq_empty?
1113   \romannumeral`&&@\csname XINT_flseqb::csv_#1\endcsname {#3}{#4}{#2}}%
1114 \def\XINT_flseqb::csv_p #1#2#3%
1115 {%
1116   \xintifCmp {#1}{#2}{, #1\expandafter\XINT_flseqb::csv_p\expandafter}%
1117   {, #1\xint_gobble_iii}{\xint_gobble_iii}%
1118   {\romannumeral0\XINTinfloatadd {#3}{#1}}{#2}{#3}%
1119 }%
1120 \def\XINT_flseqb::csv_n #1#2#3%
1121 {%
1122   \xintifCmp {#1}{#2}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1123   {, #1\expandafter\XINT_flseqb::csv_n\expandafter}%
1124   {\romannumeral0\XINTinfloatadd {#3}{#1}}{#2}{#3}%
1125 }%
1126 \def\XINT_flseqb::csv_z #1#2#3{, #1}%

```

## 10.25 The comma as binary operator

New with 1.09a. Suffices to set its precedence level to two.

```

1127 \def\XINT_tmpa #1#2#3#4#5#6%
1128 {%
1129   \def #1##1% \XINT_expr_op_,
1130   {%
1131     \expandafter #2\expandafter ##1\romannumeral`&&@\XINT_expr_getnext
1132   }%
1133   \def #2##1##2% \XINT_expr_until_,_a
1134   {\xint_UDsignfork
1135     ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
1136     -{#3##1##2}%
1137   \krof }%
1138   \def #3##1##2##3##4% \XINT_expr_until_,_b
1139   {%
1140     \ifnum ##2>\xint_c_ii
1141       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
1142         \csname XINT_#6_op_##3\endcsname {##4}}%
1143     \else
1144       \xint_afterfi
1145       {\expandafter #2\expandafter ##3%
1146        \csname .=\XINT_expr_unlock ##1,\XINT_expr_unlock ##4\endcsname }%
1147     \fi

```

```

1148 }%
1149 \let #5\xint_c_ii
1150 }%
1151 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1152 \expandafter\XINT_tmpa
1153 \csname XINT_#1_op_,\expandafter\endcsname
1154 \csname XINT_#1_until_,_a\expandafter\endcsname
1155 \csname XINT_#1_until_,_b\expandafter\endcsname
1156 \csname XINT_#1_op_-vi\expandafter\endcsname
1157 \csname XINT_expr_precedence_,\endcsname {#1}%
1158 }%

```

## 10.26 The minus as prefix operator of variable precedence level

Inherits the precedence level of the previous infix operator.

```

1159 \def\XINT_tmpa #1#2#3%
1160 {%
1161 \expandafter\XINT_tmph
1162 \csname XINT_#1_op_-#3\expandafter\endcsname
1163 \csname XINT_#1_until_-#3_a\expandafter\endcsname
1164 \csname XINT_#1_until_-#3_b\expandafter\endcsname
1165 \csname xint_c_#3\endcsname {#1}#2%
1166 }%
1167 \def\XINT_tmph #1#2#3#4#5#6%
1168 {%
1169 \def #1% \XINT_expr_op_-<level>
1170 {% get next number+operator then switch to _until macro
1171 \expandafter #2\romannumeral`&&\XINT_expr_getnext
1172 }%
1173 \def #2##1% \XINT_expr_until_-<l>_a
1174 {\xint_UDsignfork
1175 ##1{\expandafter #2\romannumeral`&&@#1}%
1176 -{##1}%
1177 \krof }%
1178 \def #3##1##2##3% \XINT_expr_until_-<l>_b
1179 {% _until tests precedence level with next op, executes now or postpones
1180 \ifnum ##1>#4%
1181 \xint_afterfi {\expandafter #2\romannumeral`&&@%
1182 \csname XINT_#5_op_##2\endcsname {##3}}%
1183 \else
1184 \xint_afterfi {\expandafter ##1\expandafter ##2%
1185 \csname .=#6{\XINT_expr_unlock ##3}\endcsname }%
1186 \fi
1187 }%
1188 }%

```

1.2d needs precedence 8 for \*\*\* and 9 for ^. Earlier, precedence level for ^ was only 8 but nevertheless the code did also "ix" here, which I think was unneeded back then.

```

1189 \xintApplyInline{\XINT_tmpa {expr}\xintOpp}{vi}{vii}{viii}{ix}}%
1190 \xintApplyInline{\XINT_tmpa {flexpr}\xintOpp}{vi}{vii}{viii}{ix}}%
1191 \xintApplyInline{\XINT_tmpa {iiexpr}\xintiiOpp}{vi}{vii}{viii}{ix}}%

```

## 10.27 ? 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.

syntax: ?{yes}{no} and ??{<0}{=0}{>0}.

The difficulty is to recognize the second ? without removing braces as would be the case with standard parsing of operators. Hence the ? operator is intercepted in \XINT\_expr\_getop\_b.

```

1192 \let\XINT_expr_precedence_? \xint_c_x
1193 \def\XINT_expr_op_? #1#2{\if ?#2\expandafter \XINT_expr_op_??\fi
1194         \XINT_expr_op_?a #1{#2}}%
1195 \def\XINT_expr_op_?a #1#2#3%
1196 {%
1197     \xintiiifNotZero{\XINT_expr_unlock #1}{\XINT_expr_getnext #2}{\XINT_expr_getnext #3}%
1198 }%
1199 \let\XINT_flexpr_op_?\XINT_expr_op_?
1200 \let\XINT_iiexpr_op_?\XINT_expr_op_?
1201 \def\XINT_expr_op_?? #1#2#3#4#5#6%
1202 {%
1203     \xintiiifSgn {\XINT_expr_unlock #2}{\XINT_expr_getnext #4}{\XINT_expr_getnext #5}%
1204     {\XINT_expr_getnext #6}%
1205 }%

```

## 10.28 ! as postfix factorial operator

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

```

1206 \let\XINT_expr_precedence_! \xint_c_x
1207 \def\XINT_expr_op_! #1{\expandafter\XINT_expr_getop
1208         \csname .=\xintiFac{\XINT_expr_unlock #1}\endcsname }%
1209 \def\XINT_flexpr_op_! #1{\expandafter\XINT_expr_getop
1210         \csname .=\XINTinFloatFac{\XINT_expr_unlock #1}\endcsname }%
1211 \def\XINT_iiexpr_op_! #1{\expandafter\XINT_expr_getop
1212         \csname .=\xintiiFac{\XINT_expr_unlock #1}\endcsname }%

```

## 10.29 The A/B[N] mechanism

Releases earlier than 1.1 required the use of braces around A/B[N] input. The [N] is now implemented directly. \*BUT\* this uses a delimited macro! thus N is not allowed to be itself an expression (I could add it...). \xintE, \xintiiE, and \XINTinFloatE all put #2 in a \numexpr. But attention to the fact that \numexpr stops at spaces separating digits: \the\numexpr 3 + 7 9\relax gives 109\relax !! Hence we have to be careful.

\numexpr will not handle catcode 11 digits, but adding a \detokenize will suddenly make illicit for N to rely on macro expansion.

```

1213 \catcode`[ 11
1214 \catcode`* 11
1215 \let\XINT_expr_precedence_[ \xint_c_vii
1216 \def\XINT_expr_op_[ #1#2]{\expandafter\XINT_expr_getop
1217         \csname .=\xintE{\XINT_expr_unlock #1}%
1218         {\xint_zapspace #2 \xint_gobble_i}\endcsname}%
1219 \def\XINT_iiexpr_op_[ #1#2]{\expandafter\XINT_expr_getop

```

```

1220      \csname .=\xintiiE{\XINT_expr_unlock #1}%
1221      {\xint_zapspaces #2 \xint_gobble_i}\endcsname}%
1222 \def\XINT_flexpr_op_[ #1#2]{\expandafter\XINT_expr_getop
1223      \csname .=\XINTinFloatE{\XINT_expr_unlock #1}%
1224      {\xint_zapspaces #2 \xint_gobble_i}\endcsname}%
1225 \catcode`[ 12
1226 \catcode`* 12

```

### 10.30 `\XINT_expr_op_`` for recognizing functions

The "onlitteral" intercepts is for bool, togl, protect, ... but also for add, mul, seq, etc... Genuine functions have expr, iiexpr and flexpr versions (or only one or two of the three).

With 1.2c "onlitteral" is also used to disambiguate variables from functions. However as I use only a `\ifcsname` test, in order to be able to re-define a variable as function, I move the check for being a function first. Each variable name now has its `onlitteral_<name>` associated macro which is the new way tacit multiplication in front of a parenthesis is implemented. This used to be decided much earlier at the time of `\XINT_expr_func`.

The advantage of our choices for 1.2c is that the same name can be used for a variable or a function, the parser will apply the correct interpretation which is decided by the presence or not of an opening parenthesis next.

```

1227 \def\XINT_tmpa #1#2#3{%
1228   \def #1##1%
1229   {%
1230     \ifcsname XINT_#3_func_##1\endcsname
1231     \xint_dothis{\expandafter\expandafter
1232       \csname XINT_#3_func_##1\endcsname\romannumeral`&&@#2}\fi
1233     \ifcsname XINT_expr_onlitteral_##1\endcsname
1234     \xint_dothis{\csname XINT_expr_onlitteral_##1\endcsname}\fi
1235     \xint_orthat{\XINT_expr_unknown_function {##1}%
1236       \expandafter\XINT_expr_func_unknown\romannumeral`&&@#2}%
1237   }%
1238 }%
1239 \def\XINT_expr_unknown_function #1{\xintError:removed \xint_gobble_i {#1}}%
1240 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1241   \expandafter\XINT_tmpa
1242     \csname XINT_#1_op_`\expandafter\endcsname
1243     \csname XINT_#1_oparen\endcsname
1244     {#1}%
1245 }%
1246 \def\XINT_expr_func_unknown #1#2#3%
1247   {\expandafter #1\expandafter #2\csname .=\0\endcsname }%

```

### 10.31 The bool, togl, protect pseudo "functions"

`bool`, `togl` and `protect` use delimited macros. They are not true functions, they turn off the parser to gather their "variable".

```

1248 \def\XINT_expr_onlitteral_bool #1)%
1249   {\expandafter\XINT_expr_getop\csname .=\xintBool{#1}\endcsname }%
1250 \def\XINT_expr_onlitteral_togl #1)%
1251   {\expandafter\XINT_expr_getop\csname .=\xintToggle{#1}\endcsname }%
1252 \def\XINT_expr_onlitteral_protect #1)%

```

```
1253      {\expandafter\XINT_expr_getop\csname .=\detokenize{#1}\endcsname }%
```

## 10.32 The break function

*break* is a true function, the parsing via expansion of the succeeding material proceeded via *\_oparen* macros as with any other function.

```
1254 \def\XINT_expr_func_break #1#2#3%
1255     {\expandafter #1\expandafter #2\csname .=?\romannumeral`&&\XINT_expr_unlock #3\endcsname }%
1256 \let\XINT_flexpr_func_break \XINT_expr_func_break
1257 \let\XINT_iiexpr_func_break \XINT_expr_func_break
```

## 10.33 The qint, qfrac, qfloat “functions”

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

```
1258 \def\XINT_expr_onlitteral_qint #1)%
1259     {\expandafter\XINT_expr_getop\csname .=\xintiNum{#1}\endcsname }%
1260 \def\XINT_expr_onlitteral_qfrac #1)%
1261     {\expandafter\XINT_expr_getop\csname .=\xintRaw{#1}\endcsname }%
1262 \def\XINT_expr_onlitteral_qfloat #1)%
1263     {\expandafter\XINT_expr_getop\csname .=\XINTinFloatdigits{#1}\endcsname }%
```

## 10.34 *\XINT\_expr\_op\_\_* for recognizing variables

The 1.1 mechanism for *\XINT\_expr\_var\_<varname>* has been modified in 1.2c. The *<varname>* associated macro is now only expanded once, not twice. We arrive here via *\XINT\_expr\_func*.

```
1264 \def\XINT_expr_op__ #1% op__ with two _'s
1265     {%
1266         \ifcsname XINT_expr_var_#1\endcsname
1267             \expandafter\xint_firstoftwo
1268         \else
1269             \expandafter\xint_secondoftwo
1270         \fi
1271         {\expandafter\expandafter\expandafter
1272          \XINT_expr_getop\csname XINT_expr_var_#1\endcsname}%
1273         {\XINT_expr_unknown_variable {#1}%
1274          \expandafter\XINT_expr_getop\csname .:=0\endcsname}%
1275     }%
1276 \def\XINT_expr_unknown_variable #1{\xintError:removed \xint_gobble_i {#1}}%
1277 \let\XINT_flexpr_op__ \XINT_expr_op__
1278 \let\XINT_iiexpr_op__ \XINT_expr_op__
```

## 10.35 User defined variables: *\xintdefvar*, *\xintdefiivar*, *\xintdeffloatvar*

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

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

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

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

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

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

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

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

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

```

1279 \catcode`* 11
1280 \def\xINT_expr_defvar #1#2#3;%
1281   \edef\xINT_expr_tmpa{#2}%
1282   \edef\xINT_expr_tmpa {\xint_zapspace_o\xINT_expr_tmpa}%
1283   \ifnum\expandafter\xintLength\expandafter{\xINT_expr_tmpa}=\z@
1284     \xintMessage {xintexpr}{Warning}
1285     {Error: impossible to declare variable with empty name.}%
1286   \else
1287     \edef\xINT_expr_tmpb {\romannumeral0#1#3\relax }%
1288     \expandafter\edef\csname XINT_expr_var_\xINT_expr_tmpa\endcsname
1289       {\expandafter\noexpand\xINT_expr_tmpb}%
1290     \expandafter\edef\csname XINT_expr_onlitteral_\xINT_expr_tmpa\endcsname
1291       {\xINT_expr_precedence_*** *\expandafter\noexpand\xINT_expr_tmpb {}}%
1292     \ifxintverbose\xintMessage {xintexpr}{Info}
1293       {Variable "\xINT_expr_tmpa" defined with value
1294         \expandafter\xINT_expr_unlock\xINT_expr_tmpb.}%
1295   \fi
1296 \fi
1297 }%
1298 \catcode`* 12
1299 \catcode`: 12
1300 \def\xintdefvar #1:={\xINT_expr_defvar\xintbareeval {#1}}%
1301 \def\xintdefiivar #1:={\xINT_expr_defvar\xintbareiieval {#1}}%
1302 \def\xintdeffloatvar #1:={\xINT_expr_defvar\xintbarefloateval {#1}}%
1303 \catcode`: 11

```

## 10.36 `\xintunassignvar`

1.2e. Currently not possible to genuinely `\undefine` a variable, all we can do is to let it stand for zero and generate an error. The reason is that I chose to use `\ifcsname` tests in `\xINT_expr_op__` and `\xINT_expr_op_`.

```

1304 \def\xintunassignvar #1{%
1305   \edef\XINT_expr_tmpa{#1}%
1306   \edef\XINT_expr_tmpa {\xint_zapspace\XINT_expr_tmpa}%
1307   \ifcsname XINT_expr_var_\XINT_expr_tmpa\endcsname
1308     \ifnum\expandafter\xintLength\expandafter{\XINT_expr_tmpa}=\@ne
1309       \expandafter\XINT_expr_makedummy \XINT_expr_tmpa
1310       \ifxintverbose\xintMessage {xintexpr}{Info}
1311         {Character \XINT_expr_tmpa\space usable as dummy variable (if letter).}%
1312       \fi
1313     \else
1314       \expandafter\edef\csname XINT_expr_var_\XINT_expr_tmpa\endcsname
1315         {\csname .=\0\endcsname\noexpand\XINT_expr_undefined {\XINT_expr_tmpa}}%
1316       \expandafter\edef\csname XINT_expr_onlitteral_\XINT_expr_tmpa\endcsname
1317         {\csname .=\0\endcsname\noexpand\XINT_expr_undefined {\XINT_expr_tmpa}*}%
1318       \ifxintverbose\xintMessage {xintexpr}{Info}
1319         {Variable \XINT_expr_tmpa\space has been ``unassigned''.}%
1320       \fi
1321     \fi
1322   \else
1323     \xintMessage {xintexpr}{Warning}
1324     {Error: there was no such variable \XINT_expr_tmpa\space to unassign.}%
1325   \fi
1326 }%
1327 \def\XINT_expr_undefined #1{\xintError:replaced_by_zero\xint_gobble_i {#1}}%

```

### 10.37 seq and the implementation of dummy variables

10.37.1	All letters usable as dummy variables . . . . .	305
10.37.2	omit and abort . . . . .	305
10.37.3	The special variables @, @1, @2, @3, @4, @@, @@(1), . . . , @@@, @@@(1), . . . for recursion	306
10.37.4	<code>\XINT_expr_onlitteral_seq</code> . . . . .	307
10.37.5	<code>\XINT_expr_onlitteral_seq_a</code> . . . . .	307
10.37.6	<code>\XINT_isbalanced_a</code> for <code>\XINT_expr_onlitteral_seq_a</code> . . . . .	308
10.37.7	<code>\XINT_allexpr_func_seqx</code> . . . . .	308
10.37.8	Evaluation over list, <code>\XINT_expr_seq:_a</code> with break, abort, omit . . . . .	309
10.37.9	Evaluation over ++ generated lists with <code>\XINT_expr_seq:_A</code> . . . . .	309

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 `iiexpr`.



### 10.37.1 All letters usable as dummy variables

The nil variable was introduced in 1.1 but isn't used under that name. However macros handling a..*[d]*..*b*, or for seq with dummy variable where omit has omitted everything may in practice inject a nil value as current number.

1.2c has changed the way variables are disambiguated from functions and for this it has added here the definitions of `\XINT_expr_onlitteral_<name>`.

In 1.1 a letter variable say X was acting as a delimited macro looking for `!X{stuff}` and then would expand the stuff inside a `\csname.=...\endcsname`. I don't think I used the possibilities this opened and the 1.2c version has stuff `_already_` encapsulated thus a single token. Only one expansion, not two is then needed in `\XINT_expr_op__`.

I had to accordingly modify seq, add, mul and subs, but fortunately realized that the @, @1, etc... variables for rseq, rrseq and iter already had been defined in the way now also followed by the Latin letters as dummy variables.

The 1.2e `\XINT_expr_makedummy` should not be used with multi-letter argument. The add, mul, seq, etc... can only work with one-letter long dummy variable. And this will almost certainly not change. Public interface is via `\xintunassignvar` which one can use with a letter to re-declare it as dummy variable. Also 1.2e does the tacit multiplication `x(stuff)->x*(stuff)` in its higher precedence form. Things are easy now that variables always fetch a single already locked value `\.=<number>`.

The tacit multiplication in case of the `''nil''` variable doesn't make much sense but we do it anyhow.

```

1328 \catcode`* 11
1329 \def\XINT_expr_makedummy #1%
1330 {%
1331   \expandafter\def\csname XINT_expr_var_#1\endcsname ##1\relax !#1##2%
1332   {##2##1\relax !#1##2}%
1333   \expandafter\def\csname XINT_expr_onlitteral_#1\endcsname ##1\relax !#1##2%
1334   {\XINT_expr_precedence_*** *##2(##1\relax !#1##2)%
1335 }%
1336 \xintApplyUnbraced \XINT_expr_makedummy {abcdefghijklmnopqrstuvwxyz}%
1337 \xintApplyUnbraced \XINT_expr_makedummy {ABCDEFGHIJKLMNOPQRSTUVWXYZ}%
1338 \edef\XINT_expr_var_nil {\expandafter\noexpand\csname . = \endcsname}%
1339 \edef\XINT_expr_onlitteral_nil
1340   {\XINT_expr_precedence_*** *\expandafter\noexpand\csname . = \endcsname {}%
1341 \catcode`* 12

```

### 10.37.2 omit and abort

June 24 and 25, 2014.

Added comments 2015/11/13:

Et la documentation ? on n'y comprend plus rien. Trop rusé.

```
\def\XINT_expr_var_omit #1\relax !{1^C!{}}{}}\.=!\relax !}
```

```
\def\XINT_expr_var_abort #1\relax !{1^C!{}}{}}\.=^\relax !}
```

C'était accompagné de `\XINT_expr_precedence_^C=0` et d'un hack au sein même des macros until de plus bas niveau.

Le mécanisme sioux était le suivant: `^C` est déclaré comme un opérateur de précedence nulle. Lorsque le parseur trouve un "omit" dans un seq ou autre, il va insérer dans le stream `\XINT_expr_getop` suivi du texte de remplacement. Donc ici on avait un 1 comme place holder, puis l'opérateur `^C`. Celui-ci étant de précedence zéro provoque la finalisation de tous les calculs antérieurs dans le sous-bareeval. Mais j'ai dû hacker le `until_end_b` (et le `until_*)_b`) qui confronté à `^C`, va se relancer à zéro, le `getnext` va trouver le `!{}}{}}\.=!` et ensuite il y aura `\relax`, et le résultat

sera \.=! pour omit ou \.=^ pour abort. Les routines des boucles seq, iter, etc... peuvent alors repérer le ! ou ^ et agir en conséquence (un long paragraphe pour ne décrire que partiellement une ou deux lignes de codes...).

Mais ^C a été fait alors que je n'avais pas encore les variables muettes. Je dois trouver autre chose, car seq(2^C, C=1..5) est alors impossible. De toute façon ce ^C était à usage interne uniquement.

Il me faut un symbole d'opérateur qui ne rentre pas en conflit. Bon je vais prendre !?. Ensuite au lieu de hacker until\_end, il vaut mieux lui donner précedence 2 (mais ça ne pourra pas marcher à l'intérieur de parenthèses il faut d'abord les fermer manuellement) et lui associer un simplement un op spécial. Je n'avais pas fait cela peut-être pour éviter d'avoir à définir plusieurs macros. Le #1 dans la définition de \XINT\_expr\_op\_!? est le résultat de l'évaluation forcée précédente.

Attention que les premier ! doivent être de catcode 12 sinon ils signalent une sous-expression qui déclenche une multiplication tacite.

```
1342 \edef\XINT_expr_var_omit #1\relax !{1\string !?!\relax !}%
1343 \edef\XINT_expr_var_abort #1\relax !{1\string !?^\relax !}%
1344 \def\XINT_expr_op_!? #1#2\relax {\expandafter\XINT_expr_foundend\csname .=#2\endcsname}%
1345 \let\XINT_iiexpr_op_!? \XINT_expr_op_!?
1346 \let\XINT_flexpr_op_!? \XINT_expr_op_!?
```

### 10.37.3 The special variables @, @1, @2, @3, @4, @@, @@(1), ..., @@@, @@@(1), ... for recursion

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

1.2c adds the needed "onlitteral" now that tacit multiplication between a variable and a ( has a new mechanism. 1.2e does this tacit multiplication with higher precedence.

For the record, the ~ has catcode 3 in this code.

```
1347 \catcode`? 3 \catcode`* 11
1348 \def\XINT_expr_var_@ #1~#2{#2#1~#2}%
1349 \expandafter\let\csname XINT_expr_var_@1\endcsname \XINT_expr_var_@
1350 \expandafter\def\csname XINT_expr_var_@2\endcsname #1~#2#3{#3#1~#2#3}%
1351 \expandafter\def\csname XINT_expr_var_@3\endcsname #1~#2#3#4{#4#1~#2#3#4}%
1352 \expandafter\def\csname XINT_expr_var_@4\endcsname #1~#2#3#4#5{#5#1~#2#3#4#5}%
1353 \def\XINT_expr_onlitteral_@ #1~#2{\XINT_expr_precedence_*** *#2(#1~#2)%
1354 \expandafter\let\csname XINT_expr_onlitteral_@1\endcsname \XINT_expr_onlitteral_@
1355 \expandafter\def\csname XINT_expr_onlitteral_@2\endcsname #1~#2#3%
1356         {\XINT_expr_precedence_*** *#3(#1~#2#3)%
1357 \expandafter\def\csname XINT_expr_onlitteral_@3\endcsname #1~#2#3#4%
1358         {\XINT_expr_precedence_*** *#4(#1~#2#3#4)%
1359 \expandafter\def\csname XINT_expr_onlitteral_@4\endcsname #1~#2#3#4#5%
1360         {\XINT_expr_precedence_*** *#5(#1~#2#3#4#5)%
1361 \catcode`* 12
1362 \def\XINT_expr_func_@@ #1#2#3#4~#5?%
1363 {%
1364   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1365         {\xintNum{\XINT_expr_unlock#3}}{#5}#4~#5?%
1366 }%
1367 \def\XINT_expr_func_@@@ #1#2#3#4~#5~#6?%
1368 {%
1369   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1370   {\xintNum{\XINT_expr_unlock#3}}{#6}#4~#5~#6?%
```

```

1371 }%
1372 \def\XINT_expr_func_@@@ #1#2#3#4~#5~#6~#7?%
1373 {%
1374   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1375   {\xintNum{\XINT_expr_unlock#3}}{#7}#4~#5~#6~#7?%
1376 }%
1377 \let\XINT_flexpr_func_@@\XINT_expr_func_@@
1378 \let\XINT_flexpr_func_@@@\XINT_expr_func_@@@
1379 \let\XINT_flexpr_func_@@@@\XINT_expr_func_@@@
1380 \def\XINT_iiexpr_func_@@ #1#2#3#4~#5?%
1381 {%
1382   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1383   {\XINT_expr_unlock#3}{#5}#4~#5?%
1384 }%
1385 \def\XINT_iiexpr_func_@@@ #1#2#3#4~#5~#6?%
1386 {%
1387   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1388   {\XINT_expr_unlock#3}{#6}#4~#5~#6?%
1389 }%
1390 \def\XINT_iiexpr_func_@@@ @ #1#2#3#4~#5~#6~#7?%
1391 {%
1392   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1393   {\XINT_expr_unlock#3}{#7}#4~#5~#6~#7?%
1394 }%
1395 \catcode`? 11

```

#### 10.37.4 \XINT\_expr\_onlitteral\_seq

```

1396 \def\XINT_expr_onlitteral_seq
1397 {\expandafter\XINT_expr_onlitteral_seq_f\romannumeral`&&\XINT_expr_onlitteral_seq_a {}}%
1398 \def\XINT_expr_onlitteral_seq_f #1#2{\xint_c_xviii `{seqx}#2)\relax #1}%

```

#### 10.37.5 \XINT\_expr\_onlitteral\_seq\_a

```

1399 \def\XINT_expr_onlitteral_seq_a #1#2,%
1400 {%
1401   \ifcase\XINT_isbalanced_a \relax #1#2(\xint_bye)\xint_bye
1402   \expandafter\XINT_expr_onlitteral_seq_c
1403   \or\expandafter\XINT_expr_onlitteral_seq_b
1404   \else\expandafter\xintError:we_are_doomed
1405   \fi {#1#2},%
1406 }%
1407 \def\XINT_expr_onlitteral_seq_b #1,{\XINT_expr_onlitteral_seq_a {#1,}}%
1408 \def\XINT_expr_onlitteral_seq_c #1,#2#3% #3 pour absorber le =
1409 {%
1410   \XINT_expr_onlitteral_seq_d {#2{#1}}}%
1411 }%
1412 \def\XINT_expr_onlitteral_seq_d #1#2#3)%
1413 {%
1414   \ifcase\XINT_isbalanced_a \relax #2#3(\xint_bye)\xint_bye
1415   \or\expandafter\XINT_expr_onlitteral_seq_e
1416   \else\expandafter\xintError:we_are_doomed
1417   \fi
1418   {#1}{#2#3}%

```

```
1419 }%
1420 \def\XINT_expr_onlitteral_seq_e #1#2{\XINT_expr_onlitteral_seq_d {#1}{#2}}}%
```

### 10.37.6 \XINT\_isbalanced\_a for \XINT\_expr\_onlitteral\_seq\_a

Expands to \xint\_c\_mone in case a closing ) had no opening ( matching it, to \@ne if opening ) had no closing ) matching it, to \z@ if expression was balanced.

```
1421 % use as \XINT_isbalanced_a \relax #1(\xint_bye)\xint_bye
1422 \def\XINT_isbalanced_a #1({\XINT_isbalanced_b #1)\xint_bye }%
1423 \def\XINT_isbalanced_b #1)#2%
1424   {\xint_bye #2\XINT_isbalanced_c\xint_bye\XINT_isbalanced_error }%

  if #2 is not \xint_bye, a ) was found, but there was no (. Hence error -> -1

1425 \def\XINT_isbalanced_error #1)\xint_bye {\xint_c_mone}%

  #2 was \xint_bye, was there a ) in original #1?

1426 \def\XINT_isbalanced_c\xint_bye\XINT_isbalanced_error #1%
1427   {\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.

1428 \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.

1429 \def\XINT_isbalanced_d #1)#2%
1430   {\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.

1431 \def\XINT_isbalanced_no\xint_bye #1\xint_bye\xint_bye {\xint_c_i }%
```

### 10.37.7 \XINT\_allexpr\_func\_seqx

1.2c uses \xintthebareval, ... which strangely were not available at 1.1 time. This spares some tokens from \XINT\_expr\_seq:\_d and cousins. Also now variables have changed their mode of operation they pick only one token which must be an already encapsulated value.

In \XINT\_allexp\_seqx, #2 is the list, evaluated and encapsulated, #3 is the dummy variable, #4 is the expression to evaluate repeatedly.

A special case is a list generated by <variable>+: then #2 is {\.=+\.=<start>}.

```
1432 \def\XINT_expr_func_seqx #1#2{\XINT_allexpr_seqx \xintthebareeval }%
1433 \def\XINT_flexpr_func_seqx #1#2{\XINT_allexpr_seqx \xintthebarefloateval}%
1434 \def\XINT_iexpr_func_seqx #1#2{\XINT_allexpr_seqx \xintthebareiieval }%
1435 \def\XINT_allexpr_seqx #1#2#3#4%
1436 {%
1437   \expandafter \XINT_expr_getop
1438   \csname .=\expandafter\XINT_expr_seq:_aa
1439     \romannumeral`&&@\XINT_expr_unlock #2!{#1#4\relax !#3}\endcsname
1440 }%
1441 \def\XINT_expr_seq:_aa #1{\if +#1\expandafter\XINT_expr_seq:_A\else
1442   \expandafter\XINT_expr_seq:_a\fi #1}%

```

**10.37.8 Evaluation over list, `\XINT_expr_seq:_a` with `break`, `abort`, `omit`**

The #2 here is `\...bareeval <expression>\relax !<variable name>`. The #1 is a comma separated list of values to assign to the dummy variable. The `\XINT_expr_seq_empty?` intervenes immediately after handling of firstvalue.

1.2c has rewritten to a large extent this and other similar loops because the dummy variables now fetch a single encapsulated token (apart from a good means to lose a few hours needlessly -- as I have had to rewrite and review most everything, this change could make the thing more efficient if the same variable is used many times in an expression, but we are talking micro-seconds here anyhow.)

```

1443 \def\XINT_expr_seq:_a #1!#2{\expandafter\XINT_expr_seq_empty?
1444         \romannumeral0\XINT_expr_seq:_b {#2}#1,^,}%
1445 \def\XINT_expr_seq:_b #1#2#3,{%
1446     \if ,#2\xint_dothis\XINT_expr_seq:_noop\fi
1447     \if ^#2\xint_dothis\XINT_expr_seq:_end\fi
1448     \xint_orthat{\expandafter\XINT_expr_seq:_c}\csname.=#2#3\endcsname {#1}%
1449 }%
1450 \def\XINT_expr_seq:_noop\csname.=, #1\endcsname #2{\XINT_expr_seq:_b {#2}#1,}%
1451 \def\XINT_expr_seq:_end \csname.=^ \endcsname #1{%
1452 \def\XINT_expr_seq:_c #1#2{\expandafter\XINT_expr_seq:_d\romannumeral`&&@#2#1{#2}}%
1453 \def\XINT_expr_seq:_d #1{\if #1^ \xint_dothis\XINT_expr_seq:_abort\fi
1454     \if #1? \xint_dothis\XINT_expr_seq:_break\fi
1455     \if #1! \xint_dothis\XINT_expr_seq:_omit\fi
1456     \xint_orthat{\XINT_expr_seq:_goon #1}}%
1457 \def\XINT_expr_seq:_abort #1!#2#3#4#5^,{}%
1458 \def\XINT_expr_seq:_break #1!#2#3#4#5^,{, #1}%
1459 \def\XINT_expr_seq:_omit #1!#2#3#4{\XINT_expr_seq:_b {#4}}%
1460 \def\XINT_expr_seq:_goon #1!#2#3#4{, #1\XINT_expr_seq:_b {#4}}%

```

If all is omitted or list is empty, `_empty?` will fetch with `##1 \endcsname` and construct "nil" via `<space>\endcsname`, if not `##1` will be a comma and the gobble will swallow the space token and the extra `\endcsname`.

```

1461 \def\XINT_expr_seq_empty? #1{%
1462 \def\XINT_expr_seq_empty? ##1{\if ,##1\expandafter\xint_gobble_i\fi #1\endcsname }}%
1463 \XINT_expr_seq_empty? { }%

```

**10.37.9 Evaluation over ++ generated lists with `\XINT_expr_seq:_A`**

This is for index lists generated by `n++`. The starting point will have been replaced by its ceil (added: in fact with version 1.1. the ceil was not yet evaluated, but `_var_<letter>` did an expansion of what they fetch). We use `\numexpr` rather than `\xintInc`, hence the indexing is limited to small integers.

The 1.2c version of `n++` produces a #1 here which is already a single `\.=<value>` token.

```

1464 \def\XINT_expr_seq:_A +#1!%
1465     {\expandafter\XINT_expr_seq_empty?\romannumeral0\XINT_expr_seq:_D #1}%
1466 \def\XINT_expr_seq:_D #1#2{\expandafter\XINT_expr_seq:_E\romannumeral`&&@#2#1{#2}}%
1467 \def\XINT_expr_seq:_E #1{\if #1^ \xint_dothis\XINT_expr_seq:_Abort\fi
1468     \if #1? \xint_dothis\XINT_expr_seq:_Break\fi
1469     \if #1! \xint_dothis\XINT_expr_seq:_Omit\fi
1470     \xint_orthat{\XINT_expr_seq:_Goon #1}}%
1471 \def\XINT_expr_seq:_Abort #1!#2#3#4{%

```

```

1472 \def\XINT_expr_seq:_Break #1!#2#3#4{,#1}%
1473 \def\XINT_expr_seq:_Omit #1!#2#3%
1474     {\expandafter\XINT_expr_seq:_D
1475         \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
1476 \def\XINT_expr_seq:_Goon #1!#2#3%
1477     {,#1\expandafter\XINT_expr_seq:_D
1478         \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%

```

## 10.38 add, mul

1.2c uses more directly the `\xintiiAdd` etc... macros and has `opxadd/opxmulo` rather than a single `opx`. This is less conceptual as I use explicitly the associated macro names for `+`, `*` but this makes other things more efficient, and the code more readable.

```

1479 \def\XINT_expr_onlitteral_add
1480 {\expandafter\XINT_expr_onlitteral_add_f\romannumeral`&&\XINT_expr_onlitteral_seq_a {}}%
1481 \def\XINT_expr_onlitteral_add_f #1#2{\xint_c_xviii `{opxadd}#2)\relax #1}%
1482 \def\XINT_expr_onlitteral_mul
1483 {\expandafter\XINT_expr_onlitteral_mul_f\romannumeral`&&\XINT_expr_onlitteral_seq_a {}}%
1484 \def\XINT_expr_onlitteral_mul_f #1#2{\xint_c_xviii `{opxmulo}#2)\relax #1}%

```

### 10.38.1 `\XINT_expr_func_opxadd`, `\XINT_flexpr_func_opxadd`, `\XINT_iiexpr_func_opxadd` and same for mul

modified 1.2c.

```

1485 \def\XINT_expr_func_opxadd #1#2{\XINT_allexpr_opx \xintbareeval {\xintAdd 0}}%
1486 \def\XINT_flexpr_func_opxadd #1#2{\XINT_allexpr_opx \xintbarefloateval {\XINTinFloatAdd 0}}%
1487 \def\XINT_iiexpr_func_opxadd #1#2{\XINT_allexpr_opx \xintbareiieval {\xintiiAdd 0}}%
1488 \def\XINT_expr_func_opxmulo #1#2{\XINT_allexpr_opx \xintbareeval {\xintMul 1}}%
1489 \def\XINT_flexpr_func_opxmulo #1#2{\XINT_allexpr_opx \xintbarefloateval {\XINTinFloatMul 1}}%
1490 \def\XINT_iiexpr_func_opxmulo #1#2{\XINT_allexpr_opx \xintbareiieval {\xintiiMul 1}}%

```

*#1=bareval etc, #2={Add0} ou {Mul1}, #3=liste encapsulée, #4=la variable, #5=expression*

```

1491 \def\XINT_allexpr_opx #1#2#3#4#5%
1492 {%
1493     \expandafter\XINT_expr_getop
1494     \csname.=\romannumeral`&&\expandafter\XINT_expr_op:_a
1495         \romannumeral`&&\XINT_expr_unlock #3!{#1#5\relax !#4}{#2}\endcsname
1496 }%
1497 \def\XINT_expr_op:_a #1!#2#3{\XINT_expr_op:_b #3{#2}#1,^,%}

```

*#2 in `\XINT_expr_op:_b` is the partial result of computation so far, not locked. A noop with have `#4=`, and `#5` the next item which we need to recover. No need to be very efficient for that in `op:_noop`. In `op:_d`, `#4` is `\xintAdd` or similar.*

```

1498 \def\XINT_expr_op:_b #1#2#3#4#5,{%
1499     \if ,#4\xint_dothis\XINT_expr_op:_noop\fi
1500     \if ^#4\xint_dothis\XINT_expr_op:_end\fi
1501     \xint_orthat{\expandafter\XINT_expr_op:_c}\csname.=#4#5\endcsname {#3}#1{#2}%
1502 }%
1503 \def\XINT_expr_op:_c #1#2#3#4{\expandafter\XINT_expr_op:_d\romannumeral0#2#1#3{#4}{#2}}%
1504 \def\XINT_expr_op:_d #1!#2#3#4#5%
1505     {\expandafter\XINT_expr_op:_b\expandafter #4\expandafter

```

```

1506      {\romannumeral`&&@#4{\XINT_expr_unlock#1}{#5}}}%
1507 \def\XINT_expr_op:_noop\csname.=,#1\endcsname #2#3#4{\XINT_expr_seq:_b #3{#4}{#2}#1,}%
1508 \def\XINT_expr_op:_end \csname.=^\endcsname #1#2#3{#3}%

```

### 10.39 subs

Got simpler with 1.2c as now the dummy variable fetches an already encapsulated value, which is anyhow the form in which we get it.

```

1509 \def\XINT_expr_onlitteral_subs
1510 {\expandafter\XINT_expr_onlitteral_subs_f\romannumeral`&&@\XINT_expr_onlitteral_seq_a {}}%
1511 \def\XINT_expr_onlitteral_subs_f #1#2{\xint_c_xviii `{subx}#2)\relax #1}%
1512 \def\XINT_expr_func_subx #1#2{\XINT_allexpr_subx \xintbareeval }%
1513 \def\XINT_flexpr_func_subx #1#2{\XINT_allexpr_subx \xintbarefloateval}%
1514 \def\XINT_iiexpr_func_subx #1#2{\XINT_allexpr_subx \xintbareiieval }%
1515 \def\XINT_allexpr_subx #1#2#3#4% #2 is the value to assign to the dummy variable
1516 {% #3 is the dummy variable, #4 is the expression to evaluate
1517   \expandafter\expandafter\expandafter\XINT_expr_getop
1518   \expandafter\XINT_expr_subx:_end\romannumeral0#1#4\relax !#3#2%
1519 }%
1520 \def\XINT_expr_subx:_end #1!#2#3{#1}%

```

### 10.40 rseq

10.40.1	<code>\XINT_expr_rseqx</code>	...	311
10.40.2	<code>\XINT_expr_rseqy</code>	...	312
10.40.3	<code>\XINT_expr_rseq:_a</code>	etc...	312
10.40.4	<code>\XINT_expr_rseq:_A</code>	etc...	312

When `func_rseq` has its turn, initial segment has been scanned by `oparen`, the `;` mimicking the rôle of a closing parenthesis, and stopping further expansion. Notice that the `;` is discovered during standard parsing mode, it may be for example `{;}` or arise from expansion as `rseq` does not use a delimited macro to locate it.

Here and in `rrseq` and `iter`, 1.2c adds also use of `\xintthebareeval`, etc...

```

1521 \def\XINT_expr_func_rseq {\XINT_allexpr_rseq \xintbareeval \xintthebareeval }%
1522 \def\XINT_flexpr_func_rseq {\XINT_allexpr_rseq \xintbarefloateval \xintthebarefloateval }%
1523 \def\XINT_iiexpr_func_rseq {\XINT_allexpr_rseq \xintbareiieval \xintthebareiieval }%
1524 \def\XINT_allexpr_rseq #1#2#3%
1525 {%
1526   \expandafter\XINT_expr_rseqx\expandafter #1\expandafter#2\expandafter
1527   #3\romannumeral`&&@\XINT_expr_onlitteral_seq_a {}}%
1528 }%

```

#### 10.40.1 `\XINT_expr_rseqx`

The `(#5)` is for ++ mechanism which must have its closing parenthesis.

```

1529 \def\XINT_expr_rseqx #1#2#3#4#5%
1530 {%
1531   \expandafter\XINT_expr_rseqy\romannumeral0#1(#5)\relax #3#4#2%
1532 }%

```



**10.40.2 \XINT\_expr\_rseq**

#1=valeurs pour variable (locked), #2=toutes les valeurs initiales (csv,locked), #3=variable, #4=expr, #5=\xintthebareeval ou \xintthebarefloateval ou \xintthebareiieval

```

1533 \def\XINT_expr_rseq #1#2#3#4#5%
1534 {%
1535     \expandafter \XINT_expr_getop
1536     \csname .=\XINT_expr_unlock #2%
1537     \expandafter\XINT_expr_rseq:_aa
1538         \romannumeral`&&@\XINT_expr_unlock #1!{#5#4\relax !#3}#2\endcsname
1539 }%
1540 \def\XINT_expr_rseq:_aa #1{\if +#1\expandafter\XINT_expr_rseq:_A\else
1541     \expandafter\XINT_expr_rseq:_a\fi #1}%

```

**10.40.3 \XINT\_expr\_rseq:\_a etc...**

```

1542 \def\XINT_expr_rseq:_a #1!#2#3{\XINT_expr_rseq:_b {#3}{#2}#1,^,%}
1543 \def\XINT_expr_rseq:_b #1#2#3#4,{%
1544     \if ,#3\xint_dothis\XINT_expr_rseq:_noop\fi
1545     \if ^#3\xint_dothis\XINT_expr_rseq:_end\fi
1546     \xint_orthat{\expandafter\XINT_expr_rseq:_c}\csname .=#3#4\endcsname
1547     {#1}{#2}%
1548 }%
1549 \def\XINT_expr_rseq:_noop\csname .=#1\endcsname #2#3{\XINT_expr_rseq:_b {#2}{#3}#1,%}
1550 \def\XINT_expr_rseq:_end \csname .=#1\endcsname #1#2{%}
1551 \def\XINT_expr_rseq:_c #1#2#3%
1552     {\expandafter\XINT_expr_rseq:_d\romannumeral`&&@#3#1~#2{#3}}%
1553 \def\XINT_expr_rseq:_d #1{%
1554     \if ^#1\xint_dothis\XINT_expr_rseq:_abort\fi
1555     \if ?#1\xint_dothis\XINT_expr_rseq:_break\fi
1556     \if !#1\xint_dothis\XINT_expr_rseq:_omit\fi
1557     \xint_orthat{\XINT_expr_rseq:_goon #1}%}
1558 \def\XINT_expr_rseq:_goon #1!#2#3~#4#5{,#1\expandafter\XINT_expr_rseq:_b
1559     \romannumeral0\XINT_expr_lockit {#1}{#5}}%
1560 \def\XINT_expr_rseq:_omit #1!#2#3~{\XINT_expr_rseq:_b }%
1561 \def\XINT_expr_rseq:_abort #1!#2#3~#4#5#6^,{}%
1562 \def\XINT_expr_rseq:_break #1!#2#3~#4#5#6^,{,#1}%

```

**10.40.4 \XINT\_expr\_rseq:\_A etc...**

n++ for rseq. With 1.2c dummy variables pick a single token.

```

1563 \def\XINT_expr_rseq:_A +#1!#2#3{\XINT_expr_rseq:_D #1#3{#2}}%
1564 \def\XINT_expr_rseq:_D #1#2#3%
1565     {\expandafter\XINT_expr_rseq:_E\romannumeral`&&@#3#1~#2{#3}}%
1566 \def\XINT_expr_rseq:_E #1{\if #1^\xint_dothis\XINT_expr_rseq:_Abort\fi
1567     \if #1?\xint_dothis\XINT_expr_rseq:_Break\fi
1568     \if #1!\xint_dothis\XINT_expr_rseq:_Omit\fi
1569     \xint_orthat{\XINT_expr_rseq:_Goon #1}%}
1570 \def\XINT_expr_rseq:_Goon #1!#2#3~#4#5%
1571     {,#1\expandafter\XINT_expr_rseq:_D
1572     \csname .=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1573     \romannumeral0\XINT_expr_lockit{#1}{#5}}%

```



```

1574 \def\XINT_expr_rseq:_Omit #1!#2#3~%#4#5%
1575     {\expandafter\XINT_expr_rseq:_D
1576         \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname }%
1577 \def\XINT_expr_rseq:_Abort #1!#2#3~#4#5{}%
1578 \def\XINT_expr_rseq:_Break #1!#2#3~#4#5{,#1}%

```

## 10.41 rrseq

10.41.1	<code>\XINT_expr_rrseqx</code> . . . . .	313
10.41.2	<code>\XINT_expr_rrseqy</code> . . . . .	313
10.41.3	<code>\XINT_expr_rrseq:_a</code> etc. . . . .	314
10.41.4	<code>\XINT_expr_rrseq:_A</code> etc. . . . .	314

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.

```

1579 \def\XINT_expr_func_rrseq {\XINT_allexpr_rrseq \xintbareeval \xintthebareeval }%
1580 \def\XINT_flexpr_func_rrseq {\XINT_allexpr_rrseq \xintbarefloateval \xintthebarefloateval }%
1581 \def\XINT_iiepr_func_rrseq {\XINT_allexpr_rrseq \xintbareiieval \xintthebareiieval }%
1582 \def\XINT_allexpr_rrseq #1#2#3%
1583 {%
1584     \expandafter\XINT_expr_rrseqx\expandafter #1\expandafter#2\expandafter
1585     #3\romannumeral`&&\XINT_expr_onlitteral_seq_a {}%
1586 }%

```

### 10.41.1 `\XINT_expr_rrseqx`

The `(#5)` is for `++` mechanism which must have its closing parenthesis.

```

1587 \def\XINT_expr_rrseqx #1#2#3#4#5%
1588 {%
1589     \expandafter\XINT_expr_rrseqy\romannumeral0#1(#5)\expandafter\relax
1590     \expandafter{\romannumeral0\xintapply \XINT_expr_lockit
1591         {\xintRevWithBraces{\xintCSVtoListNonStripped{\XINT_expr_unlock #3}}}%
1592     #3#4#2%
1593 }%

```

### 10.41.2 `\XINT_expr_rrseqy`

`#1`=valeurs pour variable (locked), `#2`=initial values (reversed, one (braced) token each) `#3`=toutes les valeurs initiales (csv,locked), `#4`=variable, `#5`=expr, `#6`=`\xintthebareeval` ou `\xintthebarefloateval` ou `\xintthebareiieval`

```

1594 \def\XINT_expr_rrseqy #1#2#3#4#5#6%
1595 {%
1596     \expandafter \XINT_expr_getop
1597     \csname .=\XINT_expr_unlock #3%
1598     \expandafter\XINT_expr_rrseq:_aa
1599         \romannumeral`&&\XINT_expr_unlock #1!{#6#5\relax !#4}{#2}\endcsname
1600 }%
1601 \def\XINT_expr_rrseq:_aa #1{\if +#1\expandafter\XINT_expr_rrseq:_A\else
1602     \expandafter\XINT_expr_rrseq:_a\fi #1}%

```

### 10.41.3 `\XINT_expr_rrseq:_a` etc...

Attention que ? a catcode 3 ici et dans iter.

```

1603 \catcode`? 3
1604 \def\XINT_expr_rrseq:_a #1!#2#3{\XINT_expr_rrseq:_b {#3}{#2}#1,^,%}
1605 \def\XINT_expr_rrseq:_b #1#2#3#4,{%
1606     \if ,#3\xint_dothis\XINT_expr_rrseq:_noop\fi
1607     \if ^#3\xint_dothis\XINT_expr_rrseq:_end\fi
1608     \xint_orthat{\expandafter\XINT_expr_rrseq:_c}\csname.#3#4\endcsname
1609     {#1}{#2}%
1610 }%
1611 \def\XINT_expr_rrseq:_noop\csname.#, #1\endcsname #2#3{\XINT_expr_rrseq:_b {#2}{#3}#1,%}
1612 \def\XINT_expr_rrseq:_end \csname.^ \endcsname #1#2{%}
1613 \def\XINT_expr_rrseq:_c #1#2#3%
1614     {\expandafter\XINT_expr_rrseq:_d\romannumeral`&&@#3#1~#2?{#3}}%
1615 \def\XINT_expr_rrseq:_d #1{%
1616     \if ^#1\xint_dothis\XINT_expr_rrseq:_abort\fi
1617     \if ?#1\xint_dothis\XINT_expr_rrseq:_break\fi
1618     \if !#1\xint_dothis\XINT_expr_rrseq:_omit\fi
1619     \xint_orthat{\XINT_expr_rrseq:_goon #1}%
1620 }%
1621 \def\XINT_expr_rrseq:_goon #1!#2#3~#4?#5{,#1\expandafter\XINT_expr_rrseq:_b\expandafter
1622     {\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}{#4}}{#5}}}%
1623 \def\XINT_expr_rrseq:_omit #1!#2#3~{\XINT_expr_rrseq:_b }%
1624 \def\XINT_expr_rrseq:_abort #1!#2#3~#4?#5#6^,{,%}
1625 \def\XINT_expr_rrseq:_break #1!#2#3~#4?#5#6^,{,#1}%

```

### 10.41.4 `\XINT_expr_rrseq:_A` etc...

`n++` for `rrseq`. With 1.2C, the `#1` in `\XINT_expr_rrseq:_A` is a single token.

```

1626 \def\XINT_expr_rrseq:_A +#1!#2#3{\XINT_expr_rrseq:_D #1{#3}{#2}}%
1627 \def\XINT_expr_rrseq:_D #1#2#3%
1628     {\expandafter\XINT_expr_rrseq:_E\romannumeral`&&@#3#1~#2?{#3}}%
1629 \def\XINT_expr_rrseq:_Goon #1!#2#3~#4?#5%
1630     {,#1\expandafter\XINT_expr_rrseq:_D
1631     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1632     \expandafter{\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}{#4}}{#5}}}%
1633 \def\XINT_expr_rrseq:_Omit #1!#2#3~#4?#5%
1634     {\expandafter\XINT_expr_rrseq:_D
1635     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
1636 \def\XINT_expr_rrseq:_Abort #1!#2#3~#4?#5{%}
1637 \def\XINT_expr_rrseq:_Break #1!#2#3~#4?#5{,#1}%
1638 \def\XINT_expr_rrseq:_E #1{\if #1^ \xint_dothis\XINT_expr_rrseq:_Abort\fi
1639     \if #1? \xint_dothis\XINT_expr_rrseq:_Break\fi
1640     \if #1! \xint_dothis\XINT_expr_rrseq:_Omit\fi
1641     \xint_orthat{\XINT_expr_rrseq:_Goon #1}}%

```

## 10.42 iter

10.42.1	<code>\XINT_expr_iterx</code>	...	315
10.42.2	<code>\XINT_expr_ity</code>	...	315

10.42.3	<code>\XINT_expr_iter:_a</code> etc...	315
10.42.4	<code>\XINT_expr_iter:_A</code> etc...	316

```

1642 \def\XINT_expr_func_iter    {\XINT_allexpr_iter \xintbareeval      \xintthebareeval      }%
1643 \def\XINT_flexpr_func_iter {\XINT_allexpr_iter \xintbarefloateval \xintthebarefloateval }%
1644 \def\XINT_iiexpr_func_iter {\XINT_allexpr_iter \xintbareiieval   \xintthebareiieval   }%
1645 \def\XINT_allexpr_iter #1#2#3%
1646 {%
1647   \expandafter\XINT_expr_iterx\expandafter #1\expandafter #2\expandafter
1648   #3\romannumeral`&&\XINT_expr_onlitteral_seq_a {}%
1649 }%

```

### 10.42.1 `\XINT_expr_iterx`

The (#5) is for ++ mechanism which must have its closing parenthesis.

```

1650 \def\XINT_expr_iterx #1#2#3#4#5%
1651 {%
1652   \expandafter\XINT_expr_itery\romannumeral0#1(#5)\expandafter\relax
1653   \expandafter{\romannumeral0\xintapply \XINT_expr_lockit
1654     {\xintRevWithBraces{\xintCSVtoListNonStripped{\XINT_expr_unlock #3}}}%
1655   #3#4#2%
1656 }%

```

### 10.42.2 `\XINT_expr_itery`

#1=valeurs pour variable (locked), #2=initial values (reversed, one (braced) token each) #3=toutes les valeurs initiales (csv,locked), #4=variable, #5=expr, #6=\xintbareeval ou \xintbarefloateval ou \xintbareiieval

```

1657 \def\XINT_expr_itery #1#2#3#4#5#6%
1658 {%
1659   \expandafter \XINT_expr_getop
1660   \csname .=%
1661   \expandafter\XINT_expr_iter:_aa
1662   \romannumeral`&&\XINT_expr_unlock #1!{#6#5\relax !#4}{#2}\endcsname
1663 }%
1664 \def\XINT_expr_iter:_aa #1{\if +#1\expandafter\XINT_expr_iter:_A\else
1665   \expandafter\XINT_expr_iter:_a\fi #1}%

```

### 10.42.3 `\XINT_expr_iter:_a` etc...

```

1666 \def\XINT_expr_iter:_a #1!#2#3{\XINT_expr_iter:_b {#3}{#2}#1,^,%}
1667 \def\XINT_expr_iter:_b #1#2#3#4,{%
1668   \if ,#3\xint_dothis\XINT_expr_iter:_noop\fi
1669   \if ^#3\xint_dothis\XINT_expr_iter:_end\fi
1670   \xint_orthat{\expandafter\XINT_expr_iter:_c}\csname .=#3#4\endcsname
1671   {#1}{#2}%
1672 }%
1673 \def\XINT_expr_iter:_noop\csname .=#1\endcsname #2#3{\XINT_expr_iter:_b {#2}{#3}#1,%}
1674 \def\XINT_expr_iter:_end \csname .=#1\endcsname #1#2%
1675   {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1676     {\XINT_expr:_unlock}{\xintReverseOrder{#1\space}}}%

```

```

1677 \def\XINT_expr_iter:_c #1#2#3%
1678   {\expandafter\XINT_expr_iter:_d\romannumeral`&&@#3#1~#2?{#3}}%
1679 \def\XINT_expr_iter:_d #1{%
1680   \if ^#1\xint_dothis\XINT_expr_iter:_abort\fi
1681   \if ?#1\xint_dothis\XINT_expr_iter:_break\fi
1682   \if !#1\xint_dothis\XINT_expr_iter:_omit\fi
1683   \xint_orthat{\XINT_expr_iter:_goon #1}%
1684 }%
1685 \def\XINT_expr_iter:_goon #1!#2#3~#4?#5{\expandafter\XINT_expr_iter:_b\expandafter
1686   {\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
1687 \def\XINT_expr_iter:_omit #1!#2#3~{\XINT_expr_iter:_b }%
1688 \def\XINT_expr_iter:_abort #1!#2#3~#4?#5#6^,%
1689   {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1690     {\XINT_expr:_unlock}{\xintReverseOrder{#4\space}}}%
1691 \def\XINT_expr_iter:_break #1!#2#3~#4?#5#6^,%
1692   {\expandafter\xint_gobble_iv\romannumeral0\xintapplyunbraced
1693     {\XINT_expr:_unlock}{\xintReverseOrder{#4\space}},#1}%
1694 \def\XINT_expr:_unlock #1{\XINT_expr_unlock #1}%

```

#### 10.42.4 \XINT\_expr\_iter:\_A etc...

*n++ for iter. ? is of catcode 3 here.*

```

1695 \def\XINT_expr_iter:_A +#1!#2#3{\XINT_expr_iter:_D #1{#3}{#2}}%
1696 \def\XINT_expr_iter:_D #1#2#3%
1697   {\expandafter\XINT_expr_iter:_E\romannumeral`&&@#3#1~#2?{#3}}%
1698 \def\XINT_expr_iter:_Goon #1!#2#3~#4?#5%
1699   {\expandafter\XINT_expr_iter:_D
1700     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1701     \expandafter{\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
1702 \def\XINT_expr_iter:_Omit #1!#2#3~%#4?#5%
1703   {\expandafter\XINT_expr_iter:_D
1704     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
1705 \def\XINT_expr_iter:_Abort #1!#2#3~#4?#5%
1706   {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1707     {\XINT_expr:_unlock}{\xintReverseOrder{#4\space}}}%
1708 \def\XINT_expr_iter:_Break #1!#2#3~#4?#5%
1709   {\expandafter\xint_gobble_iv\romannumeral0\xintapplyunbraced
1710     {\XINT_expr:_unlock}{\xintReverseOrder{#4\space}},#1}%
1711 \def\XINT_expr_iter:_E #1{\if #1^\xint_dothis\XINT_expr_iter:_Abort\fi
1712   \if #1?\xint_dothis\XINT_expr_iter:_Break\fi
1713   \if #1!\xint_dothis\XINT_expr_iter:_Omit\fi
1714   \xint_orthat{\XINT_expr_iter:_Goon #1}}%
1715 \catcode`? 11

```

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

10.43.1	\xintANDof:csv	317
10.43.2	\xintORof:csv	317
10.43.3	\xintXORof:csv	317
10.43.4	Generic csv routine	318
10.43.5	\xintMaxof:csv, \xintiiMaxof:csv	318
10.43.6	\xintMinof:csv, \xintiiMinof:csv	318

10.43.7	<code>\xintSum:csv, \xintiiSum:csv</code>	318
10.43.8	<code>\xintPrd:csv, \xintiiPrd:csv</code>	318
10.43.9	<code>\xintGCDof:csv, \xintLCMof:csv</code>	319
10.43.10	<code>\xintiiGCDof:csv, \xintiiLCMof:csv</code>	319
10.43.11	<code>\XINTinFloatdigits, \XINTinFloatSqrtdigits, \XINTinFloatFacdigits</code>	319
10.43.12	<code>\XINTinFloatMaxof:csv, \XINTinFloatMinof:csv</code>	319
10.43.13	<code>\XINTinFloatSum:csv, \XINTinFloatPrd:csv</code>	319

These 17 macros are used inside `\csname...\endcsname`. These things are not initiated by a `\romannumeral` in general, but in some cases they are, especially when involved in an `\xintNewExpr`. They will then be protected against expansion and expand only later in contexts governed by an initial `\romannumeral-`0`. There each new item may need to be expanded, which would not be the case in the use for the `_func_` things.

### 10.43.1 `\xintANDof:csv`

1.09a. For use by `\xintexpr` inside `\csname. 1.1`, je remplace `ifTrueAelseB` par `iiNotZero` pour des raisons d'optimisations.

```

1716 \def\xintANDof:csv #1{\expandafter\XINT_andof:_a\romannumeral`&&@#1,,^}%
1717 \def\XINT_andof:_a #1{\if ,#1\expandafter\XINT_andof:_e
1718     \else\expandafter\XINT_andof:_c\fi #1}%
1719 \def\XINT_andof:_c #1,{\xintiiifNotZero {#1}{\XINT_andof:_a}{\XINT_andof:_no}}%
1720 \def\XINT_andof:_no #1^{0}%
1721 \def\XINT_andof:_e #1^{1}% works with empty list

```

### 10.43.2 `\xintORof:csv`

1.09a. For use by `\xintexpr`.

```

1722 \def\xintORof:csv #1{\expandafter\XINT_orof:_a\romannumeral`&&@#1,,^}%
1723 \def\XINT_orof:_a #1{\if ,#1\expandafter\XINT_orof:_e
1724     \else\expandafter\XINT_orof:_c\fi #1}%
1725 \def\XINT_orof:_c #1,{\xintiiifNotZero{#1}{\XINT_orof:_yes}{\XINT_orof:_a}}%
1726 \def\XINT_orof:_yes #1^{1}%
1727 \def\XINT_orof:_e #1^{0}% works with empty list

```

### 10.43.3 `\xintXORof:csv`

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

```

1728 \def\xintXORof:csv #1{\expandafter\XINT_xorof:_a\expandafter 0\romannumeral`&&@#1,,^}%
1729 \def\XINT_xorof:_a #1#2,{\XINT_xorof:_b #2,#1}%
1730 \def\XINT_xorof:_b #1{\if ,#1\expandafter\XINT_xorof:_e
1731     \else\expandafter\XINT_xorof:_c\fi #1}%
1732 \def\XINT_xorof:_c #1,#2%
1733     {\xintiiifNotZero {#1}{\if #20\xint_afterfi{\XINT_xorof:_a 1}%
1734         \else\xint_afterfi{\XINT_xorof:_a 0}\fi}%
1735     {\XINT_xorof:_a #2}%
1736     }%
1737 \def\XINT_xorof:_e ,#1#2^{#1}% allows empty list (then returns 0)

```

#### 10.43.4 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.

```

1738 \def\XINT_oncsv:_empty #1,^,#2{#2}%
1739 \def\XINT_oncsv:_end ^,#1#2#3#4{#1}%
1740 \def\XINT_oncsv:_a #1#2#3%
1741 {\if ,#3\expandafter\XINT_oncsv:_empty\else\expandafter\XINT_oncsv:_b\fi #1#2#3}%
1742 \def\XINT_oncsv:_b #1#2#3,%
1743 {\expandafter\XINT_oncsv:_c \expandafter{\romannumeral`&&@#2{#3}}#1#2}%
1744 \def\XINT_oncsv:_c #1#2#3#4,{\expandafter\XINT_oncsv:_d \romannumeral`&&@#4,{#1}#2#3}%
1745 \def\XINT_oncsv:_d #1%
1746 {\if ^#1\expandafter\XINT_oncsv:_end\else\expandafter\XINT_oncsv:_e\fi #1}%
1747 \def\XINT_oncsv:_e #1,#2#3#4%
1748 {\expandafter\XINT_oncsv:_c\expandafter {\romannumeral`&&@#3{#4{#1}}{#2}}#3#4}%

```

#### 10.43.5 \xintMaxof:csv, \xintiiMaxof:csv

1.09i. Rewritten for 1.1. Compatible avec liste vide donnant valeur par défaut. Pas compatible avec items manquants. ah je m'aperçois au dernier moment que je n'ai pas en effet de \xintiiMax. Je devrais le rajouter. En tout cas ici c'est uniquement pour xintiiexpr, dans il faut bien sûr ne pas faire de xintNum, donc il faut un iimax.

```

1749 \def\xintMaxof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmax
1750 \expandafter\xint_firstofone\romannumeral`&&@#1,^{0/1[0]}}%
1751 \def\xintiiMaxof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiimax
1752 \expandafter\xint_firstofone\romannumeral`&&@#1,^{0}%

```

#### 10.43.6 \xintMinof:csv, \xintiiMinof:csv

1.09i. Rewritten for 1.1. For use by \xintiiexpr.

```

1753 \def\xintMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmin
1754 \expandafter\xint_firstofone\romannumeral`&&@#1,^{0/1[0]}}%
1755 \def\xintiiMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiimin
1756 \expandafter\xint_firstofone\romannumeral`&&@#1,^{0}%

```

#### 10.43.7 \xintSum:csv, \xintiiSum:csv

1.09a. Rewritten for 1.1. For use by \xintexpr.

```

1757 \def\xintSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintadd
1758 \expandafter\xint_firstofone\romannumeral`&&@#1,^{0/1[0]}}%
1759 \def\xintiiSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiadd
1760 \expandafter\xint_firstofone\romannumeral`&&@#1,^{0}%

```

#### 10.43.8 \xintPrd:csv, \xintiiPrd:csv

1.09a. Rewritten for 1.1. For use by \xintexpr.

```

1761 \def\xintPrd:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmul
1762 \expandafter\xint_firstofone\romannumeral`&&@#1,^{1/1[0]}}%
1763 \def\xintiiPrd:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiimul
1764 \expandafter\xint_firstofone\romannumeral`&&@#1,^{1}%

```

### 10.43.9 `\xintGCDof:csv`, `\xintLCMof:csv`

1.09a. Rewritten for 1.1. For use by `\xintexpr`. Expansion réinstaurée pour besoins de `xintNewExpr` de version 1.1

```
1765 \def\xintGCDof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintgcd
1766           \expandafter\xint_firstofone\romannumeral`&&@#1,^,1}%
1767 \def\xintLCMof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintlcm
1768           \expandafter\xint_firstofone\romannumeral`&&@#1,^,0}%
```

### 10.43.10 `\xintiiGCDof:csv`, `\xintiiLCMof:csv`

1.1a pour `\xintiiexpr`. Ces histoires de `ii` sont pénibles à la fin.

```
1769 \def\xintiiGCDof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiigcd
1770           \expandafter\xint_firstofone\romannumeral`&&@#1,^,1}%
1771 \def\xintiiLCMof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiilcm
1772           \expandafter\xint_firstofone\romannumeral`&&@#1,^,0}%
```

### 10.43.11 `\XINTinFloatdigits`, `\XINTinFloatSqrtdigits`, `\XINTinFloatFacdigits`

for `\xintNewExpr` matters, mainly.

```
1773 \def\XINTinFloatdigits {\XINTinFloat [\XINTdigits]}%
1774 \def\XINTinFloatSqrtdigits {\XINTinFloatSqrt[\XINTdigits]}%
1775 \def\XINTinFloatFacdigits {\XINTinFloatFac [\XINTdigits]}%
```

### 10.43.12 `\XINTinFloatMaxof:csv`, `\XINTinFloatMinof:csv`

1.09a. Rewritten for 1.1. For use by `\xintfloatexpr`. Name changed in 1.09h

```
1776 \def\XINTinFloatMaxof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmax
1777           \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^{0[0]}}%
1778 \def\XINTinFloatMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmin
1779           \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^{0[0]}}%
```

### 10.43.13 `\XINTinFloatSum:csv`, `\XINTinFloatPrd:csv`

1.09a. Rewritten for 1.1. For use by `\xintfloatexpr`.

```
1780 \def\XINTinFloatSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\XINTinfloatadd
1781           \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^{0[0]}}%
1782 \def\XINTinFloatPrd:csv #1{\expandafter\XINT_oncsv:_a\expandafter\XINTinfloatmul
1783           \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^{1[0]}}%
```

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

```
1784 \def\XINT_expr_twoargs #1,#2,{#{#1}{#2}}%
1785 \def\XINT_expr_argadopt #1,#2,#3.#4#5%
1786 {%
```



```

1787 \if\relax#3\relax\expandafter\xint_firstoftwo\else
1788 \expandafter\xint_secondoftwo\fi
1789 {#4}{#5[\xintNum {#2}]}{#1}%
1790 }%
1791 \def\xint_expr_oneortwo #1#2#3,#4,#5.%
1792 {%
1793 \if\relax#5\relax\expandafter\xint_firstoftwo\else
1794 \expandafter\xint_secondoftwo\fi
1795 {#1{0}}{#2{\xintNum {#4}}}{#3}%
1796 }%
1797 \def\xint_iiexpr_oneortwo #1#2,#3,#4.%
1798 {%
1799 \if\relax#4\relax\expandafter\xint_firstoftwo\else
1800 \expandafter\xint_secondoftwo\fi
1801 {#1{0}}{#1{#3}}{#2}%
1802 }%
1803 \def\xint_expr_func_num #1#2#3%
1804 {\expandafter #1\expandafter #2\csname.\xintNum {\xint_expr_unlock #3}\endcsname }%
1805 \let\xint_flexpr_func_num\xint_expr_func_num
1806 \let\xint_iiexpr_func_num\xint_expr_func_num
1807 % [0] added Oct 25. For interaction with SPRaw::csv
1808 \def\xint_expr_func_reduce #1#2#3%
1809 {\expandafter #1\expandafter #2\csname.\xintIrr {\xint_expr_unlock #3}[0]\endcsname }%
1810 \let\xint_flexpr_func_reduce\xint_expr_func_reduce
1811 % no \xint_iiexpr_func_reduce
1812 \def\xint_expr_func_abs #1#2#3%
1813 {\expandafter #1\expandafter #2\csname.\xintAbs {\xint_expr_unlock #3}\endcsname }%
1814 \let\xint_flexpr_func_abs\xint_expr_func_abs
1815 \def\xint_iiexpr_func_abs #1#2#3%
1816 {\expandafter #1\expandafter #2\csname.\xintiAbs {\xint_expr_unlock #3}\endcsname }%
1817 \def\xint_expr_func_sgn #1#2#3%
1818 {\expandafter #1\expandafter #2\csname.\xintSgn {\xint_expr_unlock #3}\endcsname }%
1819 \let\xint_flexpr_func_sgn\xint_expr_func_sgn
1820 \def\xint_iiexpr_func_sgn #1#2#3%
1821 {\expandafter #1\expandafter #2\csname.\xintiiSgn {\xint_expr_unlock #3}\endcsname }%
1822 \def\xint_expr_func_frac #1#2#3%
1823 {\expandafter #1\expandafter #2\csname.\xintTFrac {\xint_expr_unlock #3}\endcsname }%
1824 \def\xint_flexpr_func_frac #1#2#3{\expandafter #1\expandafter #2\csname
1825 .=\xintinFloatFracdigits {\xint_expr_unlock #3}\endcsname }%

no \xint_iiexpr_func_frac

1826 \def\xint_expr_func_floor #1#2#3%
1827 {\expandafter #1\expandafter #2\csname .=\xintFloor {\xint_expr_unlock #3}\endcsname }%
1828 \let\xint_flexpr_func_floor\xint_expr_func_floor

The floor and ceil functions in \xintiexpr require protect(a/b) or, better, \qfrac(a/b); else
the / will be executed first and do an integer rounded division.

1829 \def\xint_iiexpr_func_floor #1#2#3%
1830 {%
1831 \expandafter #1\expandafter #2\csname.\xintiFloor {\xint_expr_unlock #3}\endcsname }%
1832 \def\xint_expr_func_ceil #1#2#3%
1833 {\expandafter #1\expandafter #2\csname .=\xintCeil {\xint_expr_unlock #3}\endcsname }%

```



```

1834 \let\XINT_flexpr_func_ceil\XINT_expr_func_ceil
1835 \def\XINT_iiexpr_func_ceil #1#2#3%
1836 {%
1837   \expandafter #1\expandafter #2\csname.=\xintiCeil {\XINT_expr_unlock #3}\endcsname }%
1838 \def\XINT_expr_func_sqr #1#2#3%
1839   {\expandafter #1\expandafter #2\csname.=\xintSqr {\XINT_expr_unlock #3}\endcsname }%
1840 \def\XINT_flexpr_func_sqr #1#2#3%
1841 {%
1842   \expandafter #1\expandafter #2\csname
1843     .=\XINTinFloatMul{\XINT_expr_unlock #3}{\XINT_expr_unlock #3}\endcsname
1844 }%
1845 \def\XINT_expr_func_factorial #1#2#3%
1846 {%
1847   \expandafter #1\expandafter #2\csname .=%
1848   \expandafter\XINT_expr_argandopt
1849   \romannumeral`&&\XINT_expr_unlock#3,,.\xintiFac\XINTinFloatFac
1850   \endcsname
1851 }%
1852 \def\XINT_flexpr_func_factorial #1#2#3%
1853 {%
1854   \expandafter #1\expandafter #2\csname .=%
1855   \expandafter\XINT_expr_argandopt
1856   \romannumeral`&&\XINT_expr_unlock#3,,.\XINTinFloatFacdigits\XINTinFloatFac
1857   \endcsname
1858 }%
1859 \def\XINT_iiexpr_func_factorial #1#2#3%
1860 {%
1861   \expandafter #1\expandafter #2\csname.=\xintiiFac{\XINT_expr_unlock #3}\endcsname
1862 }%
1863 \def\XINT_iiexpr_func_sqr #1#2#3%
1864   {\expandafter #1\expandafter #2\csname.=\xintiiSqr {\XINT_expr_unlock #3}\endcsname }%
1865 \def\XINT_expr_func_sqrt #1#2#3%
1866 {%
1867   \expandafter #1\expandafter #2\csname .=%
1868   \expandafter\XINT_expr_argandopt
1869   \romannumeral`&&\XINT_expr_unlock#3,,.\XINTinFloatSqrtdigits\XINTinFloatSqrt
1870   \endcsname
1871 }%
1872 \let\XINT_flexpr_func_sqrt\XINT_expr_func_sqrt
1873 \def\XINT_iiexpr_func_sqrt #1#2#3%
1874   {\expandafter #1\expandafter #2\csname.=\xintiiSqrt {\XINT_expr_unlock #3}\endcsname }%
1875 \def\XINT_iiexpr_func_sqrtr #1#2#3%
1876   {\expandafter #1\expandafter #2\csname.=\xintiiSqrR {\XINT_expr_unlock #3}\endcsname }%
1877 \def\XINT_expr_func_round #1#2#3%
1878 {%
1879   \expandafter #1\expandafter #2\csname .=%
1880   \expandafter\XINT_expr_oneortwo
1881   \expandafter\xintiRound\expandafter\xintRound
1882   \romannumeral`&&\XINT_expr_unlock #3,,.\endcsname
1883 }%
1884 \let\XINT_flexpr_func_round\XINT_expr_func_round
1885 \def\XINT_iiexpr_func_round #1#2#3%

```

```

1886 {%
1887   \expandafter #1\expandafter #2\csname .=%
1888   \expandafter\XINT_iiexpr_oneortwo\expandafter\xintiRound
1889   \romannumeral`&&\XINT_expr_unlock #3,,.\endcsname
1890}%
1891 \def\XINT_expr_func_trunc #1#2#3%
1892 {%
1893   \expandafter #1\expandafter #2\csname .=%
1894   \expandafter\XINT_expr_oneortwo
1895   \expandafter\xintiTrunc\expandafter\xintTrunc
1896   \romannumeral`&&\XINT_expr_unlock #3,,.\endcsname
1897}%
1898 \let\XINT_flexpr_func_trunc\XINT_expr_func_trunc
1899 \def\XINT_iiexpr_func_trunc #1#2#3%
1900 {%
1901   \expandafter #1\expandafter #2\csname .=%
1902   \expandafter\XINT_iiexpr_oneortwo\expandafter\xintiTrunc
1903   \romannumeral`&&\XINT_expr_unlock #3,,.\endcsname
1904}%
1905 \def\XINT_expr_func_float #1#2#3%
1906 {%
1907   \expandafter #1\expandafter #2\csname .=%
1908   \expandafter\XINT_expr_argandopt
1909   \romannumeral`&&\XINT_expr_unlock #3,,.\XINTinFloatdigits\XINTinFloat
1910   \endcsname
1911}%
1912 \let\XINT_flexpr_func_float\XINT_expr_func_float
1913 % \XINT_iiexpr_func_float not defined
1914 \def\XINT_expr_func_mod #1#2#3%
1915 {%
1916   \expandafter #1\expandafter #2\csname .=%
1917   \expandafter\expandafter\expandafter\xintMod
1918   \expandafter\XINT_expr_twoargs
1919   \romannumeral`&&\XINT_expr_unlock #3,\endcsname
1920}%
1921 \def\XINT_flexpr_func_mod #1#2#3%
1922 {%
1923   \expandafter #1\expandafter #2\csname .=%
1924   \expandafter\XINTinFloatMod
1925   \romannumeral`&&\expandafter\XINT_expr_twoargs
1926   \romannumeral`&&\XINT_expr_unlock #3,\endcsname
1927}%
1928 \def\XINT_iiexpr_func_mod #1#2#3%
1929 {%
1930   \expandafter #1\expandafter #2\csname .=%
1931   \expandafter\expandafter\expandafter\xintiiMod
1932   \expandafter\XINT_expr_twoargs
1933   \romannumeral`&&\XINT_expr_unlock #3,\endcsname
1934}%
1935 \def\XINT_expr_func_binomial #1#2#3%
1936 {%
1937   \expandafter #1\expandafter #2\csname .=%

```

```

1938 \expandafter\expandafter\expandafter\xintiBinomial
1939 \expandafter\XINT_expr_twoargs
1940 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
1941 }%
1942 \def\XINT_flexpr_func_binomial #1#2#3%
1943 {%
1944 \expandafter #1\expandafter #2\csname .=%
1945 \expandafter\expandafter\expandafter\XINTinFloatBinomial
1946 \expandafter\XINT_expr_twoargs
1947 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
1948 }%
1949 \def\XINT_iiexpr_func_binomial #1#2#3%
1950 {%
1951 \expandafter #1\expandafter #2\csname .=%
1952 \expandafter\expandafter\expandafter\xintiiBinomial
1953 \expandafter\XINT_expr_twoargs
1954 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
1955 }%
1956 \def\XINT_expr_func_pfactorial #1#2#3%
1957 {%
1958 \expandafter #1\expandafter #2\csname .=%
1959 \expandafter\expandafter\expandafter\xintiPFactorial
1960 \expandafter\XINT_expr_twoargs
1961 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
1962 }%
1963 \def\XINT_flexpr_func_pfactorial #1#2#3%
1964 {%
1965 \expandafter #1\expandafter #2\csname .=%
1966 \expandafter\expandafter\expandafter\XINTinFloatPFactorial
1967 \expandafter\XINT_expr_twoargs
1968 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
1969 }%
1970 \def\XINT_iiexpr_func_pfactorial #1#2#3%
1971 {%
1972 \expandafter #1\expandafter #2\csname .=%
1973 \expandafter\expandafter\expandafter\xintiiPFactorial
1974 \expandafter\XINT_expr_twoargs
1975 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
1976 }%
1977 \def\XINT_expr_func_quo #1#2#3%
1978 {%
1979 \expandafter #1\expandafter #2\csname .=%
1980 \expandafter\expandafter\expandafter\xintiQuo
1981 \expandafter\XINT_expr_twoargs
1982 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
1983 }%
1984 \let\XINT_flexpr_func_quo\XINT_expr_func_quo
1985 \def\XINT_iiexpr_func_quo #1#2#3%
1986 {%
1987 \expandafter #1\expandafter #2\csname .=%
1988 \expandafter\expandafter\expandafter\xintiiQuo
1989 \expandafter\XINT_expr_twoargs

```

```

1990 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
1991 }%
1992 \def\XINT_expr_func_rem #1#2#3%
1993 {%
1994 \expandafter #1\expandafter #2\csname .=%
1995 \expandafter\expandafter\expandafter\xintiRem
1996 \expandafter\XINT_expr_twoargs
1997 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
1998 }%
1999 \let\XINT_flexpr_func_rem\XINT_expr_func_rem
2000 \def\XINT_iiexpr_func_rem #1#2#3%
2001 {%
2002 \expandafter #1\expandafter #2\csname .=%
2003 \expandafter\expandafter\expandafter\xintiiRem
2004 \expandafter\XINT_expr_twoargs
2005 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2006 }%
2007 \def\XINT_expr_func_gcd #1#2#3%
2008 {\expandafter #1\expandafter #2\csname
2009 .=\xintGCDof:csv{\XINT_expr_unlock #3}\endcsname }%
2010 \let\XINT_flexpr_func_gcd\XINT_expr_func_gcd
2011 \def\XINT_iiexpr_func_gcd #1#2#3%
2012 {\expandafter #1\expandafter #2\csname
2013 .=\xintiiGCDof:csv{\XINT_expr_unlock #3}\endcsname }%
2014 \def\XINT_expr_func_lcm #1#2#3%
2015 {\expandafter #1\expandafter #2\csname
2016 .=\xintLCMof:csv{\XINT_expr_unlock #3}\endcsname }%
2017 \let\XINT_flexpr_func_lcm\XINT_expr_func_lcm
2018 \def\XINT_iiexpr_func_lcm #1#2#3%
2019 {\expandafter #1\expandafter #2\csname
2020 .=\xintiiLCMof:csv{\XINT_expr_unlock #3}\endcsname }%
2021 \def\XINT_expr_func_max #1#2#3%
2022 {\expandafter #1\expandafter #2\csname
2023 .=\xintMaxof:csv{\XINT_expr_unlock #3}\endcsname }%
2024 \def\XINT_iiexpr_func_max #1#2#3%
2025 {\expandafter #1\expandafter #2\csname
2026 .=\xintiiMaxof:csv{\XINT_expr_unlock #3}\endcsname }%
2027 \def\XINT_flexpr_func_max #1#2#3%
2028 {\expandafter #1\expandafter #2\csname
2029 .=\XINTinFloatMaxof:csv{\XINT_expr_unlock #3}\endcsname }%
2030 \def\XINT_expr_func_min #1#2#3%
2031 {\expandafter #1\expandafter #2\csname
2032 .=\xintMinof:csv{\XINT_expr_unlock #3}\endcsname }%
2033 \def\XINT_iiexpr_func_min #1#2#3%
2034 {\expandafter #1\expandafter #2\csname
2035 .=\xintiiMinof:csv{\XINT_expr_unlock #3}\endcsname }%
2036 \def\XINT_flexpr_func_min #1#2#3%
2037 {\expandafter #1\expandafter #2\csname
2038 .=\XINTinFloatMinof:csv{\XINT_expr_unlock #3}\endcsname }%
2039 \expandafter\def\csname XINT_expr_func_+\endcsname #1#2#3%
2040 {\expandafter #1\expandafter #2\csname
2041 .=\xintSum:csv{\XINT_expr_unlock #3}\endcsname }%

```

```

2042 \expandafter\def\csname XINT_flexpr_func_+\endcsname #1#2#3%
2043   {\expandafter #1\expandafter #2\csname
2044     .=\XINTinFloatSum:csv{\XINT_expr_unlock #3}\endcsname }%
2045 \expandafter\def\csname XINT_iiexpr_func_+\endcsname #1#2#3%
2046   {\expandafter #1\expandafter #2\csname
2047     .=\xintiiSum:csv{\XINT_expr_unlock #3}\endcsname }%
2048 \expandafter\def\csname XINT_expr_func_*\endcsname #1#2#3%
2049   {\expandafter #1\expandafter #2\csname
2050     .=\xintPrd:csv{\XINT_expr_unlock #3}\endcsname }%
2051 \expandafter\def\csname XINT_flexpr_func_*\endcsname #1#2#3%
2052   {\expandafter #1\expandafter #2\csname
2053     .=\XINTinFloatPrd:csv{\XINT_expr_unlock #3}\endcsname }%
2054 \expandafter\def\csname XINT_iiexpr_func_*\endcsname #1#2#3%
2055   {\expandafter #1\expandafter #2\csname
2056     .=\xintiiPrd:csv{\XINT_expr_unlock #3}\endcsname }%
2057 \def\XINT_expr_func_? #1#2#3%
2058   {\expandafter #1\expandafter #2\csname
2059     .=\xintiiIsNotZero {\XINT_expr_unlock #3}\endcsname }%
2060 \let\XINT_flexpr_func_? \XINT_expr_func_?
2061 \let\XINT_iiexpr_func_? \XINT_expr_func_?
2062 \def\XINT_expr_func_! #1#2#3%
2063   {\expandafter #1\expandafter #2\csname .=\xintiiIsZero {\XINT_expr_unlock #3}\endcsname }%
2064 \let\XINT_flexpr_func_! \XINT_expr_func_!
2065 \let\XINT_iiexpr_func_! \XINT_expr_func_!
2066 \def\XINT_expr_func_not #1#2#3%
2067   {\expandafter #1\expandafter #2\csname .=\xintiiIsZero {\XINT_expr_unlock #3}\endcsname }%
2068 \let\XINT_flexpr_func_not \XINT_expr_func_not
2069 \let\XINT_iiexpr_func_not \XINT_expr_func_not
2070 \def\XINT_expr_func_all #1#2#3%
2071   {\expandafter #1\expandafter #2\csname
2072     .=\xintANDof:csv{\XINT_expr_unlock #3}\endcsname }%
2073 \let\XINT_flexpr_func_all\XINT_expr_func_all
2074 \let\XINT_iiexpr_func_all\XINT_expr_func_all
2075 \def\XINT_expr_func_any #1#2#3%
2076   {\expandafter #1\expandafter #2\csname
2077     .=\xintORof:csv{\XINT_expr_unlock #3}\endcsname }%
2078 \let\XINT_flexpr_func_any\XINT_expr_func_any
2079 \let\XINT_iiexpr_func_any\XINT_expr_func_any
2080 \def\XINT_expr_func_xor #1#2#3%
2081   {\expandafter #1\expandafter #2\csname
2082     .=\xintXORof:csv{\XINT_expr_unlock #3}\endcsname }%
2083 \let\XINT_flexpr_func_xor\XINT_expr_func_xor
2084 \let\XINT_iiexpr_func_xor\XINT_expr_func_xor
2085 \def\xintifNotZero: #1,#2,#3,{\xintiiifNotZero{#1}{#2}{#3}}%
2086 \def\XINT_expr_func_if #1#2#3%
2087   {\expandafter #1\expandafter #2\csname
2088     .=\expandafter\xintifNotZero:\romannumeral`&&\XINT_expr_unlock #3,\endcsname }%
2089 \let\XINT_flexpr_func_if\XINT_expr_func_if
2090 \let\XINT_iiexpr_func_if\XINT_expr_func_if
2091 \def\xintifSgn: #1,#2,#3,#4,{\xintiiifSgn{#1}{#2}{#3}{#4}}%
2092 \def\XINT_expr_func_ifsgn #1#2#3%
2093 {}%

```

```

2094 \expandafter #1\expandafter #2\csname
2095 .=\expandafter\xintifSgn:\romannumeral`&&\XINT_expr_unlock #3,\endcsname
2096 }%
2097 \let\XINT_flexpr_func_ifsgn\XINT_expr_func_ifsgn
2098 \let\XINT_iiexpr_func_ifsgn\XINT_expr_func_ifsgn
2099 \def\XINT_expr_func_first #1#2#3%
2100 {\expandafter #1\expandafter #2\csname.=\expandafter\XINT_expr_func_firsta
2101 \romannumeral`&&\XINT_expr_unlock #3,^\endcsname }%
2102 \def\XINT_expr_func_firsta #1,#2^{#1}%
2103 \let\XINT_flexpr_func_first\XINT_expr_func_first
2104 \let\XINT_iiexpr_func_first\XINT_expr_func_first
2105 \def\XINT_expr_func_last #1#2#3% will not work in \xintNewExpr if macro param involved
2106 {\expandafter #1\expandafter #2\csname.=\expandafter\XINT_expr_func_lasta
2107 \romannumeral`&&\XINT_expr_unlock #3,^\endcsname }%
2108 \def\XINT_expr_func_lasta #1,#2%
2109 {\if ^#2 #1\expandafter\xint_gobble_ii\fi \XINT_expr_func_lasta #2}%
2110 \let\XINT_flexpr_func_last\XINT_expr_func_last
2111 \let\XINT_iiexpr_func_last\XINT_expr_func_last
2112 \def\XINT_expr_func_odd #1#2#3%
2113 {\expandafter #1\expandafter #2\csname.=\xintOdd{\XINT_expr_unlock #3}\endcsname}%
2114 \let\XINT_flexpr_func_odd\XINT_expr_func_odd
2115 \def\XINT_iiexpr_func_odd #1#2#3%
2116 {\expandafter #1\expandafter #2\csname.=\xintiiOdd{\XINT_expr_unlock #3}\endcsname}%
2117 \def\XINT_expr_func_even #1#2#3%
2118 {\expandafter #1\expandafter #2\csname.=\xintEven{\XINT_expr_unlock #3}\endcsname}%
2119 \let\XINT_flexpr_func_even\XINT_expr_func_even
2120 \def\XINT_iiexpr_func_even #1#2#3%
2121 {\expandafter #1\expandafter #2\csname.=\xintiiEven{\XINT_expr_unlock #3}\endcsname}%
2122 \def\XINT_expr_func_nuple #1#2#3%
2123 {\expandafter #1\expandafter #2\csname .=\XINT_expr_unlock #3\endcsname }%
2124 \let\XINT_flexpr_func_nuple\XINT_expr_func_nuple
2125 \let\XINT_iiexpr_func_nuple\XINT_expr_func_nuple

```

1.2c I hesitated but left the function "reversed" from 1.1 with this name, not "reverse". But the inner not public macro got renamed into `\xintReverse::csv`.

```

2126 \def\XINT_expr_func_reversed #1#2#3%
2127 {\expandafter #1\expandafter #2\csname .=%
2128 \xintReverse::csv {\XINT_expr_unlock #3}\endcsname }%
2129 \let\XINT_flexpr_func_reversed\XINT_expr_func_reversed
2130 \let\XINT_iiexpr_func_reversed\XINT_expr_func_reversed
2131 \def\xintReverse::csv #1% should be done directly, of course
2132 {\xintListWithSep,{\xintRevWithBraces {\xintCSVtoListNonStripped{#1}}}}%

```

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

10.45.1	<code>\xintSeqB:f:csv</code>	327
10.45.2	<code>\xintiiSeqB:f:csv</code>	327
10.45.3	<code>\XINTinFloatSeqB:f:csv</code>	328

**10.45.1 \xintSeqB:f:csv**

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

```

2133 \def\xintSeqB:f:csv #1#2%
2134   {\expandafter\xint_seqb:f:csv \expandafter{\romannumeral0\xintra{#2}}{#1}}%
2135 \def\xint_seqb:f:csv #1#2{\expandafter\xint_seqb:f:csv_a\romannumeral`&&@#2#1!}%
2136 \def\xint_seqb:f:csv_a #1#2;#3;#4!{%
2137   \expandafter\xint_gobble_i\romannumeral`&&@%
2138   \xintifCmp {#3}{#4}\XINT_seqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_seqb:f:csv_bg
2139   #1{#3}{#4}}{#2}}%
2140 \def\xint_seqb:f:csv_be #1#2#3#4#5{,#2}%
2141 \def\xint_seqb:f:csv_bl #1{\if #1p\expandafter\xint_seqb:f:csv_pa\else
2142   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2143 \def\xint_seqb:f:csv_pa #1#2#3#4{\expandafter\xint_seqb:f:csv_p\expandafter
2144   {\romannumeral0\xintadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2145 \def\xint_seqb:f:csv_p #1#2%
2146 {%
2147   \xintifCmp {#1}{#2}\XINT_seqb:f:csv_pa\XINT_seqb:f:csv_pb\XINT_seqb:f:csv_pc
2148   {#1}{#2}%
2149 }%
2150 \def\xint_seqb:f:csv_pb #1#2#3#4{#3,#1}%
2151 \def\xint_seqb:f:csv_pc #1#2#3#4{#3}%
2152 \def\xint_seqb:f:csv_bg #1{\if #1n\expandafter\xint_seqb:f:csv_na\else
2153   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2154 \def\xint_seqb:f:csv_na #1#2#3#4{\expandafter\xint_seqb:f:csv_n\expandafter
2155   {\romannumeral0\xintadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2156 \def\xint_seqb:f:csv_n #1#2%
2157 {%
2158   \xintifCmp {#1}{#2}\XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_seqb:f:csv_na
2159   {#1}{#2}%
2160 }%
2161 \def\xint_seqb:f:csv_nb #1#2#3#4{#3,#1}%
2162 \def\xint_seqb:f:csv_nc #1#2#3#4{#3}%

```

**10.45.2 \xintiiSeqB:f:csv**

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

2015/11/11. I correct a typo dating back to release 1.1 (2014/10/29): the macro name had a "b" rather than "B", hence was not functional (causing \xintNewIIExpr to fail on inputs such as #1..[1]..#2).

```

2163 \def\xintiiSeqB:f:csv #1#2%
2164   {\expandafter\xint_iiSeqb:f:csv \expandafter{\romannumeral`&&@#2}{#1}}%
2165 \def\xint_iiSeqb:f:csv #1#2{\expandafter\xint_iiSeqb:f:csv_a\romannumeral`&&@#2#1!}%
2166 \def\xint_iiSeqb:f:csv_a #1#2;#3;#4!{%
2167   \expandafter\xint_gobble_i\romannumeral`&&@%
2168   \xintSgnFork{\XINT_Cmp {#3}{#4}}%
2169   \XINT_iiSeqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_iiSeqb:f:csv_bg
2170   #1{#3}{#4}}{#2}}%
2171 \def\xint_iiSeqb:f:csv_bl #1{\if #1p\expandafter\xint_iiSeqb:f:csv_pa\else

```



```

2172 \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2173 \def\xINT_iiseqb:f:csv_pa #1#2#3#4{\expandafter\xINT_iiseqb:f:csv_p\expandafter
2174 {\romannumeral0\xintiiadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2175 \def\xINT_iiseqb:f:csv_p #1#2%
2176 {%
2177 \xintSgnFork{\XINT_Cmp {#1}{#2}}%
2178 \XINT_iiseqb:f:csv_pa\xINT_iiseqb:f:csv_pb\xINT_iiseqb:f:csv_pc {#1}{#2}%
2179 }%
2180 \def\xINT_iiseqb:f:csv_pb #1#2#3#4{#3,#1}%
2181 \def\xINT_iiseqb:f:csv_pc #1#2#3#4{#3}%
2182 \def\xINT_iiseqb:f:csv_bg #1{\if #1n\expandafter\xINT_iiseqb:f:csv_na\else
2183 \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2184 \def\xINT_iiseqb:f:csv_na #1#2#3#4{\expandafter\xINT_iiseqb:f:csv_n\expandafter
2185 {\romannumeral0\xintiiadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2186 \def\xINT_iiseqb:f:csv_n #1#2%
2187 {%
2188 \xintSgnFork{\XINT_Cmp {#1}{#2}}%
2189 \XINT_seqb:f:csv_nc\xINT_seqb:f:csv_nb\xINT_iiseqb:f:csv_na {#1}{#2}%
2190 }%

```

### 10.45.3 \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.

```

2191 \def\xINTinFloatSeqB:f:csv #1#2{\expandafter\xINT_flseqb:f:csv \expandafter
2192 {\romannumeral0\xINTinfloat [\XINTdigits]{#2}}{#1}}%
2193 \def\xINT_flseqb:f:csv #1#2{\expandafter\xINT_flseqb:f:csv_a\romannumeral`&&@#2#1!}%
2194 \def\xINT_flseqb:f:csv_a #1#2;#3;#4!{%
2195 \expandafter\xint_gobble_i\romannumeral`&&@%
2196 \xintifCmp {#3}{#4}\XINT_flseqb:f:csv_bl\xINT_seqb:f:csv_be\xINT_flseqb:f:csv_bg
2197 #1{#3}{#4}}{#2}}%
2198 \def\xINT_flseqb:f:csv_bl #1{\if #1p\expandafter\xINT_flseqb:f:csv_pa\else
2199 \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2200 \def\xINT_flseqb:f:csv_pa #1#2#3#4{\expandafter\xINT_flseqb:f:csv_p\expandafter
2201 {\romannumeral0\xINTinfloatadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2202 \def\xINT_flseqb:f:csv_p #1#2%
2203 {%
2204 \xintifCmp {#1}{#2}%
2205 \XINT_flseqb:f:csv_pa\xINT_flseqb:f:csv_pb\xINT_flseqb:f:csv_pc {#1}{#2}%
2206 }%
2207 \def\xINT_flseqb:f:csv_pb #1#2#3#4{#3,#1}%
2208 \def\xINT_flseqb:f:csv_pc #1#2#3#4{#3}%
2209 \def\xINT_flseqb:f:csv_bg #1{\if #1n\expandafter\xINT_flseqb:f:csv_na\else
2210 \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2211 \def\xINT_flseqb:f:csv_na #1#2#3#4{\expandafter\xINT_flseqb:f:csv_n\expandafter
2212 {\romannumeral0\xINTinfloatadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2213 \def\xINT_flseqb:f:csv_n #1#2%
2214 {%
2215 \xintifCmp {#1}{#2}%
2216 \XINT_seqb:f:csv_nc\xINT_seqb:f:csv_nb\xINT_flseqb:f:csv_na {#1}{#2}%
2217 }%

```



## 10.46 User defined functions: `\xintdeffunc`, `\xintdefiifunc`, `\xintdeffloatfunc`

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

2015/11/13: function candidates names first completely expanded, then detokenized and cleaned of spaces.

2015/11/21: no more `\detokenize` on the function names. Also I use `#1(#2)#3:=#4` rather than `#1(#2):=#3`. Ah, rather `#1(#2)#3=#4`, then I don't have to worry about active :.

2016/02/22: 1.2f la macro associée à la fonction ne débute plus par un `\romannumeral`, de toute façon est pour emploi dans `\csname..\endcsname`.

2016/03/08: adding a pair of braces thus allowing comma separated expressions; until then the user had to do `\xintdeffunc foo(x,..):=(..., ..., ..)\relax`.

```

2218 \catcode`: 12
2219 \def\XINT_tmpa #1#2#3#4%
2220 {%
2221   \def #1##1(##2)##3==#4;%
2222   \edef\XINT_expr_tmpa {##1}%
2223   \edef\XINT_expr_tmpa {\xint_zapspace_o \XINT_expr_tmpa}%
2224   \def\XINT_expr_tmpb {0}%
2225   \def\XINT_expr_tmppc {(##4)}%
2226   \xintFor ###1 in {##2} \do
2227     {\edef\XINT_expr_tmppb {\the\numexpr\XINT_expr_tmppb+\xint_c_i}%
2228     \edef\XINT_expr_tmppc {subs(\unexpanded\expandafter{\XINT_expr_tmppc},%
2229     #####\XINT_expr_tmppb)}}%
2230   }%
2231   \expandafter#3\csname XINT_#2_userfunc_\XINT_expr_tmpa\endcsname
2232   [\XINT_expr_tmppb]{\XINT_expr_tmppc}%
2233   \expandafter\XINT_expr_defuserfunc
2234   \csname XINT_#2_func_\XINT_expr_tmpa\expandafter\endcsname
2235   \csname XINT_#2_userfunc_\XINT_expr_tmpa\endcsname
2236   \ifxintverbose\xintMessage {xintexpr}{Info}
2237     {Function \XINT_expr_tmpa\space for \string\xint #4 parser
2238     associated to \string\XINT_#2_userfunc_\XINT_expr_tmpa\space
2239     with meaning \expandafter\meaning
2240     \csname XINT_#2_userfunc_\XINT_expr_tmpa\endcsname}%
2241   \fi
2242 }%
2243 }%
2244 \catcode`: 11
2245 \XINT_tmpa\xintdeffunc {expr} \XINT_NewFunc {expr}%
2246 \XINT_tmpa\xintdefiifunc {iiexpr}\XINT_NewIIFunc {iiexpr}%
2247 \XINT_tmpa\xintdeffloatfunc{fexpr}\XINT_NewFloatFunc{floatexpr}%
2248 \def\XINT_expr_defuserfunc #1#2{%
2249   \def #1##1##2##3{\expandafter ##1\expandafter ##2%
2250   \csname .=\expandafter #2\romannumeral-`0\XINT_expr_unlock ##3,\endcsname
2251   }%
2252 }%

```

**10.47 \xintNewExpr, \xintNewIExpr, \xintNewFloatExpr, \xintNewIIExpr**

10.47.1	\xintApply::csv	330
10.47.2	\xintApply:::csv	330
10.47.3	\XINT_expr_RApply::csv, \XINT_expr_LApply::csv, \XINT_expr_RLApply:::csv	330
10.47.4	Mysterious stuff	330
10.47.5	\xintNewExpr, ..., at last.	334

**10.47.1 \xintApply::csv**

```

2253 \def\xintApply::csv #1#2%
2254   {\expandafter\xint_applyon::_a\expandafter {\romannumeral`&&#2}{#1}}%
2255 \def\xint_applyon::_a #1#2{\XINT_applyon::_b {#2}{#1},,%
2256 \def\xint_applyon::_b #1#2#3,{\expandafter\xint_applyon::_c \romannumeral`&&#3,{#1}{#2}}%
2257 \def\xint_applyon::_c #1{\if #1,\expandafter\xint_applyon::_end
2258   \else\expandafter\xint_applyon::_d\fi #1}%
2259 \def\xint_applyon::_d #1,#2{\expandafter\xint_applyon::_e\romannumeral`&&#2{#1},{#2}}%
2260 \def\xint_applyon::_e #1,#2#3{\XINT_applyon::_b {#2}{#3}, #1}%
2261 \def\xint_applyon::_end #1,#2#3{\xint_secondoftwo #3}%

```

**10.47.2 \xintApply:::csv**

```

2262 \def\xintApply:::csv #1#2#3%
2263   {\expandafter\xint_applyon::_a\expandafter{\romannumeral`&&#2}{#1}{#3}}%
2264 \def\xint_applyon::_a #1#2#3{\XINT_applyon::_b {#2}{#3}{#1},,%
2265 \def\xint_applyon::_b #1#2#3#4,%
2266   {\expandafter\xint_applyon::_c \romannumeral`&&#4,{#1}{#2}{#3}}%
2267 \def\xint_applyon::_c #1{\if #1,\expandafter\xint_applyon::_end
2268   \else\expandafter\xint_applyon::_d\fi #1}%
2269 \def\xint_applyon::_d #1,#2#3%
2270   {\expandafter\xint_applyon::_e\expandafter
2271     {\romannumeral`&&\xintApply::csv {#2}{#1}{#3}},{#2}{#3}}%
2272 \def\xint_applyon::_e #1,#2#3#4{\XINT_applyon::_b {#2}{#3}{#4}, #1}%
2273 \def\xint_applyon::_end #1,#2#3#4{\xint_secondoftwo #4}%

```

**10.47.3 \XINT\_expr\_RApply::csv, \XINT\_expr\_LApply::csv, \XINT\_expr\_RLApply:::csv**

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

Attention ici et dans la suite ~ avec catcode 12 (et non pas 3 comme ailleurs dans xintexpr).

```

2274 \catcode`~ 12
2275 \def\xint_expr_RApply::csv #1#2#3#4%
2276   {\~xintApply::csv{\~expandafter#1~xint_exchangetwo_keepbraces{#4}{#3}}%
2277 \def\xint_expr_LApply::csv #1#2#3#4{\~xintApply::csv{#1}{#3}{#4}}%
2278 \def\xint_expr_RLApply:::csv #1#2{\~xintApply::csv{#1}}%

```

**10.47.4 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.

Comments finally added 2015/12/11: the whole point here is to either expand completely a macro such as \xintAdd if it is has numeric arguments (the original macro is stored by \xintNewExpr in \xintNEAdd, which is not a good name anyhow, as I was reading it as Not Expand, whereas it is exactly

the opposite and NE stands for New Expr), or handle the case when one of the two parameters only is numerical (the other being either a macro parameter, or a previously found macro not be expanded -- this is recursive), or none of them. In that case though, the non-numerical parameter may well stand ultimately for a whole list. Hence the various \xintApply one finds above.

Indeed the catcode 12 dollar sign is used to signal a "virtual list" argument, a catcode 3 ~ will signal a "non-expandable". This happens when something add a "virtual list" argument, we must now leave the macros with a ~ meaning that it will become a real macro during the final \scantokens. Such "~" macro names will be created when they have a macro parameter as argument, or another "~" macro name, or a dollar prefixed argument as created by the NewExpr version of the \xintSeq::csv type macros which create comma separated lists.

This 1.1 (2014/10/29) code works amazingly well allowing frankly amazing things such as the parsing by \xintNewExpr of [#1..[#2]..#3][#4:#5]. But we need to have a translation into exclusively f-expandable macros (avoiding \csname...\endcsname, but if absolutely needed perhaps I will do it ultimately.) And for the iterating loops seq, iter, etc..., we have no equivalent if the list of indices is not explicit. We succeed in doing it with explicit list, but if start, step, or end contains a macro parameter we are stuck (how could test for omit, abort, break work?). Or we would need macro versions.

~ and of catcode 12 in what follows.

1.2f adds the forgotten iiLt, iiGt, iiEq, iiLtorEq, iiGtorEq, iiNeq.

```

2279 \catcode`$ 12 % $
2280 \def\xINT_xptwo_getab_b #1#2!#3%
2281   {\expandafter\xINT_xptwo_getab_c\romannumeral`&&@#3!#1{#1#2}}%
2282 \def\xINT_xptwo_getab_c #1#2!#3#4#5#6{#1#3{#5}{#6}{#1#2}{#4}}%
2283 \def\xint_ddfork #1$#2#3\krof {#2}% $$
2284 \def\xINT_NEfork #1#2{\xint_ddfork
2285   #1#2\xINT_expr_RLApply:::csv
2286   #1$\xINT_expr_RLApply:::csv% $
2287   $#2\xINT_expr_RLApply:::csv% $
2288   $$\xINT_NEfork_nn #1#2}% $$
2289   \krof }%
2290 \def\xINT_NEfork_nn #1#2#3#4{%
2291   \if #1##\xint_dothis{#3}\fi
2292   \if #1~\xint_dothis{#3}\fi
2293   \if #2##\xint_dothis{#3}\fi
2294   \if #2~\xint_dothis{#3}\fi
2295   \xint_orthat {\csname #4NE\endcsname }%
2296   }%
2297 \def\xINT_NEfork_one #1#2!#3#4#5#6{%
2298   \if ##1\xint_dothis {#3}\fi
2299   \if ~#1\xint_dothis {#3}\fi
2300   \if $#1\xint_dothis {\xintApply:::csv{#3#5}}\fi %$
2301   \xint_orthat {\csname #4NE\endcsname #6}{#1#2}%
2302 }%
2303 \toks0 {}%
2304 \xintFor #1 in
2305   {DivTrunc,Mod,Round,Trunc,iRound,iTrunc,iQuo,iRem,
2306   iiDivTrunc,iiDivRound,iiMod,iiQuo,iiRem,%
2307   Lt,Gt,Eq,LtorEq,GtorEq,Neq,%
2308   iiLt,iiGt,iiEq,iiLtorEq,iiGtorEq,iiNeq,%
2309   Add,Sub,Mul,Div,Pow,E,%
2310   iiAdd,iiSub,iiMul,iiPow,iiE,%
2311   AND,OR,XOR,%

```

```

2312      SeqA::csv,iiSeqA::csv}\do
2313 {\toks0
2314 \expandafter{\the\toks0% no space!
2315 \expandafter\let\csname xint#1NE\expandafter\endcsname\csname xint#1\expandafter
2316 \endcsname\expandafter\def\csname xint#1\endcsname ####1####2{%
2317 \expandafter\XINT_NEfork
2318 \romannumeral`&&@\expandafter\XINT_xptwo_getab_b
2319 \romannumeral`&&@####2!{####1}{~xint#1}{xint#1}}}%
2320 }%
2321 }%
2322 % cela aurait-il un sens d'ajouter Raw et iNum (à cause de qint, qfrac,
2323 % qfloat?). Pas le temps d'y réfléchir. Je ne fais rien.
2324 \xintFor #1 in {Num,Irr,Abs,iiAbs,Sgn,iiSgn,TFrac,Floor,iFloor,Ceil,iCeil,%
2325 Sqr,iiSqr,iiSqrt,iiSqrtR,iiIsZero,iiIsNotZero,iiifNotZero,iiifSgn,%
2326 Odd,Even,iiOdd,iiEven,Opp,iiOpp,iiifZero,iFac,iBinomial,%
2327 iPFactorial,iiFac,iiBinomial,iiPFactorial,Bool,Toggle}\do
2328 {\toks0 \expandafter{\the\toks0
2329 \expandafter\let\csname xint#1NE\expandafter\endcsname\csname
2330 xint#1\expandafter
2331 \endcsname\expandafter\def\csname xint#1\endcsname ####1{%
2332 \expandafter\XINT_NEfork_one\romannumeral`&&@####1!{~xint#1}{xint#1}}{}}}%
2333 }%
2334 }%
2335 \toks0
2336 \expandafter{\the\toks0
2337 \let\XINTinFloatFacNE\XINTinFloatFac
2338 \def\XINTinFloatFac ##1{%
2339 \expandafter\XINT_NEfork_one
2340 \romannumeral`&&@##1!{~XINTinFloatFac}{XINTinFloatFac}}{}}}%
2341 }%
2342 \xintFor #1 in {Add,Sub,Mul,Div,Binomial,PFactorial,PowerH,E,Mod,SeqA::csv}\do
2343 {\toks0
2344 \expandafter{\the\toks0%
2345 \expandafter\let\csname XINTinFloat#1NE\expandafter\endcsname
2346 \csname XINTinFloat#1\expandafter\endcsname
2347 \expandafter\def\csname XINTinFloat#1\endcsname ####1####2{%
2348 \expandafter\XINT_NEfork
2349 \romannumeral`&&@\expandafter\XINT_xptwo_getab_b
2350 \romannumeral`&&@####2!{####1}{~XINTinFloat#1}{XINTinFloat#1}}}%
2351 }%
2352 }%
2353 \xintFor #1 in {XINTinFloatdigits,XINTinFloatFracdigits,XINTinFloatSqrtdigits,XINTinFloatFacdigits}\do
2354 {\toks0
2355 \expandafter{\the\toks0%
2356 \expandafter\let\csname #1NE\expandafter\endcsname\csname #1\expandafter
2357 \endcsname\expandafter\def\csname #1\endcsname ####1{\expandafter
2358 \XINT_NEfork_one\romannumeral`&&@####1!{~#1}{#1}}{}}}%
2359 }%
2360 }%
2361 \xintFor #1 in {xintSeq::csv,xintiiSeq::csv,XINTinFloatSeq::csv}\do
2362 {\toks0
2363 \expandafter{\the\toks0% no space

```

```

2364 \expandafter\let\csname #1NE\expandafter\endcsname\csname #1\expandafter
2365 \endcsname\expandafter\def\csname #1\endcsname #####2{%
2366     \expandafter\XINT_NEfork
2367     \romannumeral`&&\expandafter\XINT_xptwo_getab_b
2368     \romannumeral`&&#####2!{#####1}{$noexpand$#1}{#1}}%
2369 }%
2370 }%
2371 \xintFor #1 in {xintSeqB,xintiiSeqB,XINTinFloatSeqB}\do
2372 {\toks0
2373 \expandafter{\the\toks0% no space
2374 \expandafter\let\csname #1::csvNE\expandafter\endcsname\csname #1::csv\expandafter
2375 \endcsname\expandafter\def\csname #1::csv\endcsname #####1#####2{%
2376     \expandafter\XINT_NEfork
2377     \romannumeral`&&\expandafter\XINT_xptwo_getab_b
2378     \romannumeral`&&#####2!{#####1}{$noexpand$#1:f:csv}{#1::csv}}%
2379 }%
2380 }%
2381 \toks0
2382 \expandafter{\the\toks0
2383 \let\XINTinFloatNE\XINTinFloat
2384 \def\XINTinFloat [##1]##2{% not ultimately general, but got tired
2385     \expandafter\XINT_NEfork_one
2386     \romannumeral`&&#####2!{~XINTinFloat[##1]}{XINTinFloat}{#{[##1]}}%
2387 \let\XINTinFloatSqrtNE\XINTinFloatSqrt
2388 \def\XINTinFloatSqrt [##1]##2{%
2389     \expandafter\XINT_NEfork_one
2390     \romannumeral`&&#####2!{~XINTinFloatSqrt[##1]}{XINTinFloatSqrt}{#{[##1]}}%
2391 \let\XINTinFloatFacNE\XINTinFloatFac
2392 \def\XINTinFloatFac [##1]##2{%
2393     \expandafter\XINT_NEfork_one
2394     \romannumeral`&&#####2!{~XINTinFloatFac[##1]}{XINTinFloatFac}{#{[##1]}}%
2395 }%
2396 \xintFor #1 in {ANDof,ORof,XORof,iiMaxof,iiMinof,iiSum,iiPrd,
2397     GCDof,LCMof,Sum,Prd,Maxof,Minof}\do
2398 {\toks0
2399 \expandafter{\the\toks0\expandafter\def\csname xint#1:csv\endcsname {~xint#1:csv}}%
2400 }%
2401 \xintFor #1 in
2402 {XINTinFloatMaxof,XINTinFloatMinof,XINTinFloatSum,XINTinFloatPrd}\do
2403 {\toks0
2404 \expandafter{\the\toks0\expandafter\def\csname #1:csv\endcsname {~#1:csv}}%
2405 }%

```

~xintListSel::csv must have space after it, the reason being in \XINT\_expr\_until\_:\_b which inserts a : as fist token of something which will reappear later following ~xintListSel::csv.

Notice that 1.1 was chicken and did not even try to expand the Reverse and ListSel by fear of the complexities and overhead of checking whether it contained macro parameters (possibly embedded in sub-macros).

```

2406 \toks0 \expandafter{\the\toks0
2407     \def\xintReverse::csv {~xintReverse::csv }%
2408     \def\xintListSel::csv {~xintListSel::csv }%
2409 }%

```

```

2410 \odef\XINT_expr_redefinemacros {\the\toks0}% Not \edef ! (subtle)
2411 \def\XINT_expr_redefineprints
2412 {%
2413   \def\XINT_flexpr_noopt
2414     {\expandafter\XINT_flexpr_withopt_b\expandafter-\romannumeral0\xintbarefloateval }%
2415   \def\XINT_flexpr_withopt_b ##1##2%
2416     {\expandafter\XINT_flexpr_wrap\csname .;##1.=\XINT_expr_unlock ##2\endcsname }%
2417   \def\XINT_expr_unlock_sp ##1.;##2##3.==##4!%
2418     {\if -##2\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
2419     \XINTdigits{\##2##3}{\##4}}%
2420   \def\XINT_expr_print      ##1{\expandafter\xintSPraw::csv\expandafter
2421     {\romannumeral`&&\XINT_expr_unlock ##1}}%
2422   \def\XINT_iexpr_print    ##1{\expandafter\xintCSV::csv\expandafter
2423     {\romannumeral`&&\XINT_expr_unlock ##1}}%
2424   \def\XINT_boolexpr_print ##1{\expandafter\xintIsTrue::csv\expandafter
2425     {\romannumeral`&&\XINT_expr_unlock ##1}}%
2426   \def\xintCSV::csv      {\xintCSV::csv }%
2427   \def\xintSPraw::csv    {\xintSPraw::csv }%
2428   \def\xintPFloat::csv   {\xintPFloat::csv }%
2429   \def\xintIsTrue::csv   {\xintIsTrue::csv }%
2430   \def\xintRound::csv    {\xintRound::csv }%
2431 }%
2432 \toks0 {}%

```

#### 10.47.5 *\xintNewExpr*, ..., at last.

1.2c modifications to accomodate *\XINT\_expr\_deffunc\_newexpr* etc..

1.2f adds token *\XINT\_newexpr\_clean* to be able to have a different *\XINT\_newfunc\_clean*

```

2433 \def\xintNewExpr      {\XINT_NewExpr}\XINT_expr_redefineprints\xint_firstofone
2434                        \xinttheexpr\XINT_newexpr_clean}%
2435 \def\xintNewFloatExpr{\XINT_NewExpr}\XINT_expr_redefineprints\xint_firstofone
2436                        \xintthefloatexpr\XINT_newexpr_clean}%
2437 \def\xintNewIExpr     {\XINT_NewExpr}\XINT_expr_redefineprints\xint_firstofone
2438                        \xinttheiexpr\XINT_newexpr_clean}%
2439 \def\xintNewIIExpr    {\XINT_NewExpr}\XINT_expr_redefineprints\xint_firstofone
2440                        \xinttheiexpr\XINT_newexpr_clean}%
2441 \def\xintNewBoolExpr {\XINT_NewExpr}\XINT_expr_redefineprints\xint_firstofone
2442                        \xinttheboolexpr\XINT_newexpr_clean}%
2443 \def\XINT_newexpr_clean #1>{\noexpand\romannumeral`&&}%

```

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

```

2444 \def\XINT_NewFunc
2445   {\XINT_NewExpr,{}\xint_gobble_i\xintthebareeval \XINT_newfunc_clean }%
2446 \def\XINT_NewFloatFunc
2447   {\XINT_NewExpr,{}\xint_gobble_i\xintthebarefloateval\XINT_newfunc_clean }%
2448 \def\XINT_NewIIFunc
2449   {\XINT_NewExpr,{}\xint_gobble_i\xintthebareiieval \XINT_newfunc_clean }%
2450 \def\XINT_newfunc_clean #1>{}%

```

1.2c adds optional logging. For this needed to pass to *\_NewExpr\_a* the macro name as parameter. And *\_NewExpr* itself receives two new parameters to treat both *\xintNewExpr* and *\xintdeffunc*.

Up to and including 1.2c the definition was global. Starting with 1.2d it is done locally.

```

2451 \def\XINT_NewExpr #1#2#3#4#5#6#7[#8]%
2452 {%
2453 \begingroup
2454 \ifcase #8\relax
2455 \toks0 {\endgroup\def#6}%
2456 \or \toks0 {\endgroup\def#6##1#1}%
2457 \or \toks0 {\endgroup\def#6##1#1##2#1}%
2458 \or \toks0 {\endgroup\def#6##1#1##2#1##3#1}%
2459 \or \toks0 {\endgroup\def#6##1#1##2#1##3#1##4#1}%
2460 \or \toks0 {\endgroup\def#6##1#1##2#1##3#1##4#1##5#1}%
2461 \or \toks0 {\endgroup\def#6##1#1##2#1##3#1##4#1##5#1##6#1}%
2462 \or \toks0 {\endgroup\def#6##1#1##2#1##3#1##4#1##5#1##6#1##7#1}%
2463 \or \toks0 {\endgroup\def#6##1#1##2#1##3#1##4#1##5#1##6#1##7#1##8#1}%
2464 \or \toks0 {\endgroup\def#6##1#1##2#1##3#1##4#1##5#1##6#1##7#1##8#1##9#1}%
2465 \fi
2466 \xintexprSafeCatcodes
2467 \XINT_expr_redefinmacros
2468 #2%
2469 \XINT_NewExpr_a #3#4#5#6%
2470 }%

```

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

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

```

2471 \catcode`~ 13 \catcode`@ 14 \catcode`\% 6 \catcode`# 12 \catcode`$ 11 @ $
2472 \def\XINT_NewExpr_a %1%2%3%4%5@
2473 {@
2474 \def\XINT_tmpa %1%2%3%4%5%6%7%8%9{%5}@
2475 \def~{$\noexpand$}@
2476 \catcode`_ 11 \catcode`_ 11
2477 \catcode`# 12 \catcode`~ 13 \escapechar 126
2478 \endlinechar -1 \everyeof {\noexpand }@
2479 \edef\XINT_tmppb
2480 {\scantokens\expandafter
2481 {\romannumeral`&&@\expandafter%2\XINT_tmpa {#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}\relax}@
2482 }@
2483 \escapechar 92 \catcode`# 6 \catcode`$ 0 @ $
2484 \edef\XINT_tmpa %1%2%3%4%5%6%7%8%9@
2485 {\scantokens\expandafter{\expandafter%3\meaning\XINT_tmppb}}@
2486 \the\toks0\expandafter{\XINT_tmpa{%1}{%2}{%3}{%4}{%5}{%6}{%7}{%8}{%9}}@
2487 %1{\ifxintverbose
2488 \xintMessage{xintexpr}{Info}@
2489 {\string%4\space now with meaning \meaning%4}@
2490 \fi}@
2491 }@
2492 \catcode`% 14
2493 \let\xintexprRestoreCatcodes\empty

```



```

2494 \def\xintexprSafeCatcodes
2495 {%
2496   \edef\xintexprRestoreCatcodes {%
2497     \catcode59=\the\catcode59 % ;
2498     \catcode34=\the\catcode34 % "
2499     \catcode63=\the\catcode63 % ?
2500     \catcode124=\the\catcode124 % |
2501     \catcode38=\the\catcode38 % &
2502     \catcode33=\the\catcode33 % !
2503     \catcode93=\the\catcode93 % ]
2504     \catcode91=\the\catcode91 % [
2505     \catcode94=\the\catcode94 % ^
2506     \catcode95=\the\catcode95 % _
2507     \catcode47=\the\catcode47 % /
2508     \catcode41=\the\catcode41 % )
2509     \catcode40=\the\catcode40 % (
2510     \catcode42=\the\catcode42 % *
2511     \catcode43=\the\catcode43 % +
2512     \catcode62=\the\catcode62 % >
2513     \catcode60=\the\catcode60 % <
2514     \catcode58=\the\catcode58 % :
2515     \catcode46=\the\catcode46 % .
2516     \catcode45=\the\catcode45 % -
2517     \catcode44=\the\catcode44 % ,
2518     \catcode61=\the\catcode61 % =
2519     \catcode32=\the\catcode32\relax % space
2520   }%
2521   \catcode59=12 % ;
2522   \catcode34=12 % "
2523   \catcode63=12 % ?
2524   \catcode124=12 % |
2525   \catcode38=4 % &
2526   \catcode33=12 % !
2527   \catcode93=12 % ]
2528   \catcode91=12 % [
2529   \catcode94=7 % ^
2530   \catcode95=8 % _
2531   \catcode47=12 % /
2532   \catcode41=12 % )
2533   \catcode40=12 % (
2534   \catcode42=12 % *
2535   \catcode43=12 % +
2536   \catcode62=12 % >
2537   \catcode60=12 % <
2538   \catcode58=12 % :
2539   \catcode46=12 % .
2540   \catcode45=12 % -
2541   \catcode44=12 % ,
2542   \catcode61=12 % =
2543   \catcode32=10 % space
2544 }%
2545 \let\XINT_tmpa\relax \let\XINT_tmpb\relax \let\XINT_tmpe\relax

```



2546 \XINT\_restorecatcodes\_endinput%

`xintkernel`: 285. Total number of code lines: 13548. (but 3266 lines among them  
`xinttools`:1160. start either with {% or with }%.)  
`xintcore`:2048. Each package starts with circa 50 lines dealing with cat-  
`xint`:1869. codes, package identification and reloading management,  
`xintbinhex`: 631. also for Plain  $\TeX$ . Version 1.2f of 2016/03/12.  
`xintgcd`: 455.  
`xintfrac`:3147.  
`xintseries`: 386.  
`xintcfraction`:1021.  
`xintexpr`:2546.