

The `xint` source code

JEAN-FRANÇOIS BURNOL

jfbu (at) free (dot) fr

Package version: 1.2e (2015/11/22); documentation date: 2015/11/22.

From source file xint.dtx. Time-stamp: <22-11-2015 à 23:03:12 CET>.

Contents

1	Package <code>xintkernel</code> implementation	1
2	Package <code>xinttools</code> implementation	9
3	Package <code>xintcore</code> implementation	36
4	Package <code>xint</code> implementation	86
5	Package <code>xintbinhex</code> implementation	120
6	Package <code>xintgcd</code> implementation	133
7	Package <code>xintfrac</code> implementation	145
8	Package <code>xintseries</code> implementation	209
9	Package <code>xintcfrac</code> implementation	219
10	Package <code>xintexpr</code> implementation	241

This is 1.2e of 2015/11/22.

- 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 of the style used now for 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, ε -TeX and reload detection	2	.2	Package identification	5
----	---	---	----	----------------------------------	---

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

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

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

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 T_EX3.

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}%
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%

```

1.2 corrects a long-standing somewhat subtle bug, of which the author became aware only on 15/09/2013: earlier releases had `\aftergroup\endinput` above, rather than `\def\z{\endgroup\endinput}` and the `\ifx` test. The `\endinput` token was indeed inserted after the `\endgroup` from `\PrepareCatcodes`, but all material and in particular `\XINT_setupcatcodes` from the macro now called `\PrepareCatcodes` was expanded before the `\endinput` had come into effect ! as a result the catcodes would be modified in unwanted ways, in Plain \TeX , if the source had for example `\input xint.sty` followed by `\input xintkernel.sty`: the catcode changes would be done before the second input of `xintkernel.sty` had been aborted. One didn't see the situation under \LaTeX (in normal circumstances), because a second `\usepackage{xintkernel}` would not do any input of `xintkernel.sty` to start with.

```

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 %

```

1 Package *xintkernel* implementation

```

70      \catcode61=\the\catcode61\relax    % =
71  }%
72  \edef\XINT_restorecatcodes_endinput
73  {%
74      \XINT_restorecatcodes\noexpand\endinput %
75  }%
76  \def\XINT_setcatcodes
77  {%
78      \catcode61=12    % =
79      \catcode32=10    % space
80      \catcode13=5     % ^^M
81      \endlinechar=13 %
82      \catcode123=1    % {
83      \catcode125=2    % }
84      \catcode95=11    % _ LETTER
85      \catcode35=6     % #
86      \catcode44=12    % ,
87      \catcode45=12    % -
88      \catcode46=12    % .
89      \catcode58=11    % : LETTER
90      \catcode60=12    % <
91      \catcode62=12    % >
92      \catcode43=12    % +
93      \catcode42=12    % *
94      \catcode40=12    % (
95      \catcode41=12    % )
96      \catcode47=12    % /
97      \catcode96=12    % `
98      \catcode94=11    % ^ LETTER
99      \catcode36=3     % $
100     \catcode91=12    % [
101     \catcode93=12    % ]
102     \catcode33=12    % !
103     \catcode64=11    % @ LETTER
104     \catcode38=7     % & for \romannumeral`&&@ trick.
105     \catcode124=12   % |
106     \catcode63=11    % ? LETTER
107     \catcode34=12    % "
108     \catcode39=12    % '
109     \catcode126=3    % ~ MATH
110     \catcode59=12    % ;
111     \catcode0=12     % for \romannumeral`&&@ trick
112  }%
113  \XINT_setcatcodes
114  }%
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 HO for some reason escaping me (compatibility with LaTeX 2.09 or other things ??) seems to set extra precautions.

1.09c uses e-TeX `\ifdefined`.

```

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 [2015/11/22 v1.2e Paraphernalia for the xint packages (jfb)]%

```

1.3 Constants

v1.2 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!}%

```

1 Package *xintkernel* implementation

```
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!!!!}%
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}%

```

1 Package *xintkernel* implementation

```
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}%
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}% v1.1
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}% v1.1
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

```



```

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, \mathcal{E} -TeX and reload detection . . .	10	.16	<code>\xintApply</code>	23
.2	Package identification	11	.17	<code>\xintApplyUnbraced</code>	23
.3	<code>\xintgodef</code> , <code>\xintgoodef</code> , <code>\xintgdef</code> . . .	11	.18	<code>\xintSeq</code>	24
.4	<code>\xintRevWithBraces</code>	11	.19	<code>\xintloop</code> , <code>\xintbreakloop</code> , <code>\xintbreak-</code> <code>loopanddo</code> , <code>\xintloopskiptonext</code> . . .	26
.5	<code>\xintZapFirstSpaces</code>	12	.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>	26
.6	<code>\xintZapLastSpaces</code>	12	.21	<code>\XINT_xflet</code>	27
.7	<code>\xintZapSpaces</code>	13	.22	<code>\xintApplyInline</code>	28
.8	<code>\xintZapSpacesB</code>	13	.23	<code>\xintFor</code> , <code>\xintFor*</code> , <code>\xintBreakFor</code> , <code>\xintBreakForAndDo</code>	28
.9	<code>\xintCSVtoList</code> , <code>\xintCSVtoListNon-</code> <code>Stripped</code>	14	.24	<code>\XINT_forever</code> , <code>\xintintegers</code> , <code>\xintdi-</code> <code>mensions</code> , <code>\xinrationals</code>	30
.10	<code>\xintListWithSep</code>	15	.25	<code>\xintForpair</code> , <code>\xintForthree</code> , <code>\xintFor-</code>	
.11	<code>\xintNthElt</code>	15			
.12	<code>\xintKeep</code>	17			
.13	<code>\xintKeepUnbraced</code>	19			
.14	<code>\xintTrim</code>	21			
.15	<code>\xintTrimUnbraced</code>	22			

<i>four</i>	32	<i>DigitsOf</i>	34
.26 <i>\xintAssign</i> , <i>\xintAssignArray</i> , <i>\xint-</i>			

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

2.1 Catcodes, ε -TeX and reload detection

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

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

```

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 [2015/11/22 v1.2e 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 #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

```

2 Package *xinttools* implementation

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

```
103 \long\def\xINT_zapesp_b #1#2#3\xint_relax
104     {\XINT_zapesp_end? #3\xINT_zapesp_e {#2#1}\empty #3\xint_relax }%
```

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

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

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

```
106 \long\def\xINT_zapesp_end\xINT_zapesp_e #1#2\xint_relax { #1}%
```

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

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

2.7 `\xintZapSpaces`

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

```
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_zapesp_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
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.

2 Package *xinttools* implementation

`\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 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 {%

```

2 Package *xinttools* implementation

```

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}%
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\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}}%

```

2 Package *xinttools* implementation

```
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%
```

2 Package `xinttools` implementation

```
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%
448 {%
449   \expandafter\XINT_trim_a\the\numexpr #1\expandafter.%
450   \expandafter{\romannumeral`&&#2}%
451 }%
452 \def\xinttrimnoexpand #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 }%
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

```


2 Package *xinttools* implementation

```

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}%
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_seqo_nd #1#2#3#4{\XINT_seqo_nb {#1}{#2}{#3}{#4{#1}}}%

```

2.19 \xintloop, \xintbreakloop, \xintbreakloopaddo, \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\xintbreakloopaddo #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, \xintbreakiloopaddo, \xintiloopskiptonext, \xintiloopskipandredo

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\xintbreakiloopaddo
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\xintiloopskiptonext #1\xintiloop_again\fi\xint_gobble_iii #2#3{%
709   \expandafter\xintiloop_again_b \the\numexpr#2+#3.#3.%}
710 \long\def\xintiloopskipandredo #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

2 Package *xinttools* implementation

`\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 }%
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\xintcsvtolist {#1}Z%
824 }%
825 \def\XINT_to_forever\fi #1\xintcsvtolist #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_forever\fi
835   \ifx\XINT_token V\XINT_to_forever\fi
836   \ifx\XINT_token D\XINT_to_forever\fi
837   \XINT_forx_empty?
838 }%
839 \def\XINT_to_forever\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 \xintiadd/\xintiimul which do not have this overhead. Also

2 Package *xinttools* implementation

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
892     \expandafter\relax\expandafter}%
893   \expandafter{\the\numexpr #2}}%
894 \def\XINT_?expr_Da #1#2%
895   {\expandafter{\expandafter\dimexpr\the\dimexpr #1\expandafter\relax
896     \expandafter s\expandafter p\expandafter\relax\expandafter}%
897   \expandafter{\the\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 \opt\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\xinttiadd {#1}{#2}}{#2}%
932 }%
933 \def\XINT_?expr_Vy #1#2#3#4%
934 {%
935   \expandafter{\romannumeral0\xinttiadd {#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_secondoftwo
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%

```


2 Package *xinttools* implementation

```

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_secondoftwo
977   \ifx #5j\expandafter\xint_firstoftwo
978     \else\expandafter\xint_secondoftwo
979   \fi
980   {\let\xintifForLast\xint_firstoftwo
981     \xintBreakForAndDo {\XINT_x \xint_gobble_i Z}}%
982   \XINT_x
983   \let\xintifForFirst\xint_secondoftwo\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 {%
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}%

```

2 Package *xinttools* implementation

```
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]
1084   \global
1085   \expandafter\let\csname\xint_arrayname\xintilooopindex\endcsname\relax
1086   \ifnum \xintilooopindex > \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_zapspace}%
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 }%
```

3 Package *xintcore* implementation

```

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
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, \mathcal{E} -TeX and reload detection . . .	37		.2	Package identification	38
----	---	----	--	----	----------------------------------	----

3 Package *xintcore* implementation

.3	Counts for holding needed constants . . .	38	.16	<code>\xintDec</code>	47
.4	<code>\xintNum</code>	38	.17	<code>\xintInc</code>	48
.5	Zeroes	39	.18	Core arithmetic	49
.6	Blocks of eight digits	40	.19	<code>\xintiAdd</code> , <code>\xintiiAdd</code>	49
.7	Blocks of eight, for needs of v1.2 <code>\xinti-</code> <code>iDivision</code>	41	.20	<code>\xintisub</code> , <code>\xintiiSub</code>	53
.8	<code>\xintReverseDigits</code>	43	.21	<code>\xintimul</code> , <code>\xintiiMul</code>	57
.9	<code>\xintSgn</code> , <code>\xintiiSgn</code> , <code>\XINT_Sgn</code> , <code>\XINT_cntSgn</code>	43	.22	<code>\xintisqr</code> , <code>\xintiiSqr</code>	61
.10	<code>\xintioff</code> , <code>\xintiiOpp</code>	44	.23	<code>\xintipow</code> , <code>\xintiiPow</code>	62
.11	<code>\xintiAbs</code> , <code>\xintiiAbs</code>	45	.24	<code>\xintifac</code> , <code>\xintiiFac</code>	65
.12	<code>\xintFDg</code> , <code>\xintiiFDg</code>	45	.25	<code>\xintidivision</code> , <code>\xintiQuo</code> , <code>\xintiRem</code> , <code>\xintiiDivision</code> , <code>\xintiiQuo</code> , <code>\xintiiRem</code>	68
.13	<code>\xintLDg</code> , <code>\xintiiLDg</code>	46	.26	<code>\xintidivRound</code> , <code>\xintiiDivRound</code>	82
.14	<code>\xintDouble</code>	46	.27	<code>\xintidivTrunc</code> , <code>\xintiiDivTrunc</code>	84
.15	<code>\xintHalf</code>	47	.28	<code>\xintimod</code> , <code>\xintiiMod</code>	85
			.29	“Load <code>xintfrac</code> ” macros	85

Got split off from *xint* with release 1.1. Release 1.1 also added the new macro `\xintiiDivRound`. The package does not load *xinttools*.

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

The commenting continues (2015/11/22) to be very sparse: actually it got worse than ever with release 1.2. I will possibly add comments at a later date, but for the time being the new routines are not commented at all.

Also, with 1.2, `\xintAdd` etc... have been left undefined control sequences: only `\xintiAdd` and `\xintiiAdd` (etc...) are provided via *xintcore*. It was announced a long time ago that `\xintAdd` etc... were to be removed from *xint* and only defined by *xintfrac*.

3.1 Catcodes, ε -TEX and reload detection

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

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

```

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

```

```

21 \expandafter
22 \ifx\csname numexpr\endcsname\relax
23   \y{xintcore}{\numexpr not available, aborting input}%
24   \aftergroup\endinput
25 \else
26   \ifx\x\relax % plain-TeX, first loading of xintcore.sty
27     \ifx\w\relax % but xintkernel.sty not yet loaded.
28       \def\z{\endgroup\input xintkernel.sty\relax}%
29     \fi
30   \else
31     \def\empty {}%
32     \ifx\x\empty % LaTeX, first loading,
33     % variable is initialized, but \ProvidesPackage not yet seen
34       \ifx\w\relax % xintkernel.sty not yet loaded.
35         \def\z{\endgroup\RequirePackage{xintkernel}}%
36       \fi
37     \else
38       \aftergroup\endinput % 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 [2015/11/22 v1.2e 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 100000000
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 1200000000
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}`

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

Note (10/2015): I should take time to revisit this.

```

55 \def\xintiNum {\romannumeral0\xintinum }%
56 \def\xintinum #1%

```

3 Package *xintcore* implementation

```
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_relax\Z
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

Changed for 1.2 which has a base model of eight digits rather than four for the basic operations.

```
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 }%
98 %%%%%%%%%%
99 \def\XINT_cuz #1#2#3#4#5#6#7#8#9%
100 {%
101   \xint_gob_til_R #9\XINT_cuz_e \R
102   \xint_gob_til_eightzeroes #1#2#3#4#5#6#7#8\XINT_cuz_z 00000000%
103   \XINT_cuz_clean #1#2#3#4#5#6#7#8#9%
104 }%

```

3 Package *xintcore* implementation

```
105 \edef\XINT_cuz_clean #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_clean #2\R
108   {\noexpand\expandafter\space\noexpand\the\numexpr #2\relax }%
109 \def\XINT_cuz_z 00000000\XINT_cuz_clean 00000000{\XINT_cuz }%
110 %%%%%%%%%%
111 \def\XINT_cuz_byviii #1#2#3#4#5#6#7#8#9%
112 {%
113   \xint_gob_til_R #9\XINT_cuz_byviii_e \R
114   \xint_gob_til_eightzeroes #1#2#3#4#5#6#7#8\XINT_cuz_byviii_z 00000000%
115   \XINT_cuz_byviii_clean #1#2#3#4#5#6#7#8#9%
116 }%
117 \def\XINT_cuz_byviii_clean #1\R { #1}%
118 \def\XINT_cuz_byviii_e\R #1\XINT_cuz_byviii_clean #2\R{ #2}%
119 \def\XINT_cuz_byviii_z 00000000\XINT_cuz_byviii_clean 00000000{\XINT_cuz_byviii}%
```

3.6 Blocks of eight digits

Lingua of release 1.2.

```
120 \def\XINT_zeroes_forviii #1#2#3#4#5#6#7#8%
121 {%
122   \xint_gob_til_R #8\XINT_zeroes_forviii_end\R\XINT_zeroes_forviii
123 }%
124 \edef\XINT_zeroes_forviii_end\R\XINT_zeroes_forviii #1#2#3#4#5#6#7#8#9\W
125 {%
126   \noexpand\expandafter\space\noexpand\xint_gob_til_one #2#3#4#5#6#7#8%
127 }%
128 %%%%%%%%%%
129 \def\XINT_rsepbyviii #1#2#3#4#5#6#7#8%
130 {%
131   \XINT_rsepbyviii_b {#1#2#3#4#5#6#7#8}%
132 }%
133 \def\XINT_rsepbyviii_b #1#2#3#4#5#6#7#8#9%
134 {%
135   #2#3#4#5#6#7#8#9\expandafter!\the\numexpr
136   1#1\expandafter.\the\numexpr 1\XINT_rsepbyviii
137 }%
138 \def\XINT_rsepbyviii_end_B #1\relax #2#3{#2.}%
139 \def\XINT_rsepbyviii_end_A #1#2\expandafter #3\relax #4#5{#2.1#5.}%
140 %%%%%%%%%%
141 \def\XINT_sepandrev
142 {%
143   \expandafter\XINT_sepandrev_a\the\numexpr 1\XINT_rsepbyviii
144 }%
145 \def\XINT_sepandrev_a {\XINT_sepandrev_b {}}%
146 \def\XINT_sepandrev_b #1#2.#3.#4.#5.#6.#7.#8.#9.%
147 {%
148   \xint_gob_til_R #9\XINT_sepandrev_end\R
149   \XINT_sepandrev_b {#9!#8!#7!#6!#5!#4!#3!#2!#1}%
150 }%
151 \def\XINT_sepandrev_end\R\XINT_sepandrev_b #1#2\W {\XINT_sepandrev_done #1}%
152 \def\XINT_sepandrev_done #1#2!{ }%
```


3 Package *xintcore* implementation

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

Needed ending pattern: 1\Z!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W. The \romannumeral in unrevbyviii_a is for special effects. I must document when needed and used.

```
170 \def\XINT_unrevbyviii #11#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
171 {%
172   \xint_gob_til_R #9\XINT_unrevbyviii_a\R
173   \XINT_unrevbyviii {#9#8#7#6#5#4#3#2#1}%
174 }%
175 \edef\XINT_unrevbyviii_a\R\XINT_unrevbyviii #1#2\W
176   {\noexpand\expandafter\space
177   \noexpand\romannumeral`&&@\noexpand\xint_gob_til_Z #1}%
```

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

```
178 \def\XINT_smallunrevbyviii 1#1!1#2!1#3!1#4!1#5!1#6!1#7!1#8!#9\W%
179 {%
180   \expandafter\XINT_cuz_small\xint_gob_til_Z #8#7#6#5#4#3#2#1%
181 }%
```

3.7 Blocks of eight, for needs of v1.2 \xintiiDivision.

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

3 Package *xintcore* implementation

```

195 !\the\numexpr#5\expandafter!\the\numexpr#6\expandafter!\the\numexpr
196 #7\expandafter
197 !\the\numexpr#8\expandafter!\the\numexpr#9\expandafter!\the\numexpr
198 \expandafter\XINT_sepybyviii_andcount_b\the\numexpr #1+\xint_c_viii.%
199 }%
200 \def\XINT_sepybyviii_andcount_end #1\XINT_sepybyviii_andcount_b\the\numexpr
201 #2+\xint_c_viii.#3#4\W {\expandafter.\the\numexpr #2+#3.}%
202 %%%%%%%%%%
203 \def\XINT_rev_nounsep #1#2!#3!#4!#5!#6!#7!#8!#9!%
204 {%
205 \xint_gob_til_R #9\XINT_rev_nounsep_end\R
206 \XINT_rev_nounsep {#9!#8!#7!#6!#5!#4!#3!#2!#1}%
207 }%
208 \def\XINT_rev_nounsep_end\R\XINT_rev_nounsep #1#2\W {\XINT_rev_nounsep_done #1}%
209 \def\XINT_rev_nounsep_done #11{ 1}%
210 %%%%%%%%%%
211 \def\XINT_sepybyviii_Z #1#2#3#4#5#6#7#8%
212 {%
213 1#1#2#3#4#5#6#7#8\expandafter!\the\numexpr\XINT_sepybyviii_Z
214 }%
215 \def\XINT_sepybyviii_Z_end #1\relax {\relax\Z!}%
216 %%%%%%%%%%
217 \def\XINT_unsep_cuzsmall #11#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
218 {%
219 \xint_gob_til_R #9\XINT_unsep_cuzsmall_end\R
220 \XINT_unsep_cuzsmall {#1#2#3#4#5#6#7#8#9}%
221 }%
222 \def\XINT_unsep_cuzsmall_end\R
223 \XINT_unsep_cuzsmall #1{\XINT_unsep_cuzsmall_done #1}%
224 \def\XINT_unsep_cuzsmall_done #1\R #2\W{\XINT_cuz_small #1}%
225 \def\XINT_unsep_delim {1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W}%
226 %%%%%%%%%%
227 \def\XINT_div_unsepQ #11#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
228 {%
229 \xint_gob_til_R #9\XINT_div_unsepQ_end\R
230 \XINT_div_unsepQ {#1#2#3#4#5#6#7#8#9}%
231 }%
232 \def\XINT_div_unsepQ_end\R\XINT_div_unsepQ #1{\XINT_div_unsepQ_x #1}%
233 \def\XINT_div_unsepQ_x #1#2#3#4#5#6#7#8#9%
234 {%
235 \xint_gob_til_R #9\XINT_div_unsepQ_e \R
236 \xint_gob_til_eightzeroes #1#2#3#4#5#6#7#8\XINT_div_unsepQ_y 00000000%
237 \expandafter\XINT_div_unsepQ_done \the\numexpr #1#2#3#4#5#6#7#8.#9%
238 }%
239 \def\XINT_div_unsepQ_e\R\xint_gob_til_eightzeroes #1\XINT_div_unsepQ_y #2\W
240 {\the\numexpr #1\relax \Z}%
241 \def\XINT_div_unsepQ_y #1.#2\R #3\W{\XINT_cuz_small #2\Z}%
242 \def\XINT_div_unsepQ_done #1.#2\R #3\W { #1#2\Z}%
243 %%%%%%%%%%
244 \def\XINT_div_unsepR #11#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
245 {%
246 \xint_gob_til_R #9\XINT_div_unsepR_end\R

```

```

247 \XINT_div_unsepR {#1#2#3#4#5#6#7#8#9}%
248 }%
249 \def\XINT_div_unsepR_end\R\XINT_div_unsepR #1{\XINT_div_unsepR_done #1}%
250 \def\XINT_div_unsepR_done #1\R #2\W {\XINT_cuz #1\R}%

```

3.8 \xintReverseDigits

v1.2. Needed now by \xintLDg.

```

251 \def\XINT_microrevsep #1#2#3#4#5#6#7#8%
252 {%
253   1#8#7#6#5#4#3#2#1\expandafter!\the\numexpr\XINT_microrevsep
254 }%
255 \def\XINT_microrevsep_end #1\W #2\expandafter #3\Z{#2!}%
256 \def\xintReverseDigits {\romannumeral0\xintreversedigits }%
257 \def\xintreversedigits #1{\expandafter\XINT_reversedigits\romannumeral`&&@#1\Z}%
258 \def\XINT_reversedigits #1%
259 {%
260   \xint_UDsignfork
261   #1{\expandafter\xint_minus_thenstop\romannumeral0\XINT_reversedigits_a}%
262   -{\XINT_reversedigits_a #1}%
263   \krof
264 }%
265 \def\XINT_reversedigits_a #1\Z
266 {%
267   \expandafter\XINT_revdigits_a\the\numexpr\expandafter\XINT_microrevsep
268   \romannumeral`&&@#1{\XINT_microrevsep_end\W}\XINT_microrevsep_end
269   \XINT_microrevsep_end\XINT_microrevsep_end
270   \XINT_microrevsep_end\XINT_microrevsep_end
271   \XINT_microrevsep_end\XINT_microrevsep_end\Z
272   1\Z!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
273 }%
274 \def\XINT_revdigits_a {\XINT_revdigits_b {}}%
275 \def\XINT_revdigits_b #11#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
276 {%
277   \xint_gob_til_R #9\XINT_revdigits_end\R
278   \XINT_revdigits_b {#9#8#7#6#5#4#3#2#1}%
279 }%
280 \edef\XINT_revdigits_end\R\XINT_revdigits_b #1#2\W
281 {\noexpand\expandafter\space\noexpand\xint_gob_til_Z #1}%

```

3.9 \xintSgn, \xintiisgn, \XINT_Sgn, \XINT_cntSgn

Changed in 1.05. Earlier code was unnecessarily strange. 1.09a with \xintnum

1.09i defines \XINT_Sgn and \XINT_cntSgn (was \XINT__Sgn in 1.09i) for reasons of internal optimizations.

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

```

282 \def\xintiisgn {\romannumeral0\xintiisgn }%
283 \def\xintiisgn #1%
284 {%
285   \expandafter\XINT_sgn \romannumeral`&&@#1\Z%
286 }%

```

```

287 \def\xintSgn {\romannumeral0\xintsgn }%
288 \def\xintsgn #1%
289 {%
290   \expandafter\XINT_sgn \romannumeral0\xintnum{#1}\Z%
291 }%
292 \def\XINT_sgn #1#2\Z
293 {%
294   \xint_UDzerominusfork
295   #1-{ 0}%
296   0#1{ -1}%
297   0-{ 1}%
298   \krof
299 }%
300 \def\XINT_Sgn #1#2\Z
301 {%
302   \xint_UDzerominusfork
303   #1-{0}%
304   0#1{-1}%
305   0-{1}%
306   \krof
307 }%
308 \def\XINT_cntSgn #1#2\Z
309 {%
310   \xint_UDzerominusfork
311   #1-\xint_c_
312   0#1\xint_c_mone
313   0-\xint_c_i
314   \krof
315 }%

```

3.10 \xintiOpp, \xintiiOpp

```

316 \def\xintiiOpp {\romannumeral0\xintiiopp }%
317 \def\xintiiopp #1%
318 {%
319   \expandafter\XINT_opp \romannumeral`&&@#1%
320 }%
321 \def\xintiOpp {\romannumeral0\xintiopp }%
322 \def\xintiopp #1%
323 {%
324   \expandafter\XINT_opp \romannumeral0\xintnum{#1}%
325 }%
326 \def\XINT_Opp #1{\romannumeral0\XINT_opp #1}%
327 \def\XINT_opp #1%
328 {%
329   \xint_UDzerominusfork
330   #1-{ 0}%      zero
331   0#1{ }%      negative
332   0-{ -#1}%    positive
333   \krof
334 }%

```

3.11 `\xintiAbs`, `\xintiiAbs`

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

```

335 \def\xintiiAbs {\romannumeral0\xintiiabs }%
336 \def\xintiiabs #1%
337 {%
338   \expandafter\XINT_abs \romannumeral`&&@#1%
339 }%
340 \def\xintiAbs {\romannumeral0\xintiabs }%
341 \def\xintiabs #1%
342 {%
343   \expandafter\XINT_abs \romannumeral0\xintnum{#1}%
344 }%
345 \def\XINT_Abs #1{\romannumeral0\XINT_abs #1}%
346 \def\XINT_abs #1%
347 {%
348   \xint_UDsignfork
349   #1{ }%
350   -{ #1}%
351   \krof
352 }%

```

3.12 `\xintFDg`, `\xintiiFDg`

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

```

353 \def\xintiiFDg {\romannumeral0\xintiifdg }%
354 \def\xintiifdg #1%
355 {%
356   \expandafter\XINT_fdg \romannumeral`&&@#1\W\Z
357 }%
358 \def\xintFDg {\romannumeral0\xintfdg }%
359 \def\xintfdg #1%
360 {%
361   \expandafter\XINT_fdg \romannumeral0\xintnum{#1}\W\Z
362 }%
363 \def\XINT_FDg #1{\romannumeral0\XINT_fdg #1\W\Z }%
364 \def\XINT_fdg #1#2#3\Z
365 {%
366   \xint_UDzerominusfork
367   #1-{ 0}%   zero
368   0#1{ #2}%  negative
369   0-{ #1}%   positive
370   \krof
371 }%

```

3.13 `\xintLDg`, `\xintiLDg`

LAST DIGIT. Simplified in 1.05. And prepared for extension by `xintfrac` to parse through `\xintNum`. Release 1.09a adds the `\xintnum` already here, and this propagates to `\xintOdd`, etc... 1.09e The `\xintiLDg` is for defining `\xintiiOdd` which is used once (currently) elsewhere .

bug fix (1.1b): `\xintiLDg` is needed by the division macros next, thus it needs to be in the `xintcore.sty`.

Rewritten for 1.2.

```

372 \def\xintLDg {\romannumeral0\xintldg }%
373 \def\xintldg #1{\xintiildg {\xintNum{#1}}}%
374 \def\xintiilDg {\romannumeral0\xintiildg }%
375 \def\xintiildg #1%
376 {%
377   \expandafter\XINT_ldg_done\romannumeral0%
378   \expandafter\XINT_revdigits_a\the\numexpr\expandafter\XINT_microrevsep
379   \romannumeral0\expandafter\XINT_abs
380   \romannumeral`&&@#1{\XINT_microrevsep_end\W}\XINT_microrevsep_end
381   \XINT_microrevsep_end\XINT_microrevsep_end
382   \XINT_microrevsep_end\XINT_microrevsep_end
383   \XINT_microrevsep_end\XINT_microrevsep_end\Z
384   1\Z!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
385   \Z
386 }%
387 \def\XINT_ldg_done #1#2\Z { #1}%

```

3.14 `\xintDouble`

v1.08. Rewritten for v1.2.

```

388 \def\xintDouble {\romannumeral0\xintdouble }%
389 \def\xintdouble #1%
390 {%
391   \expandafter\XINT_dbl\romannumeral`&&@#1\Z
392 }%
393 \def\XINT_dbl #1%
394 {%
395   \xint_UDzerominusfork
396   #1-\XINT_dbl_zero
397   0#1\XINT_dbl_neg
398   0-{\XINT_dbl_pos #1}%
399   \krof
400 }%
401 \def\XINT_dbl_zero #1\Z { 0}%
402 \def\XINT_dbl_neg
403   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_dbl_pos }%
404 \def\XINT_dbl_pos #1\Z
405 {%
406   \expandafter\XINT_dbl_pos_aa
407   \romannumeral0\expandafter\XINT_sepandrev
408   \romannumeral0\XINT_zeroes_forviii #1\R\R\R\R\R\R\R\{10}0000001\W
409   #1\XINT_rsepbyviii_end_A 2345678%
410   \XINT_rsepbyviii_end_B 2345678\relax XX%

```

```

411 \R.\R.\R.\R.\R.\R.\R.\R.\W 1\Z!%
412 1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
413 }%
414 \def\XINT_dbl_pos_aa
415 {%
416 \expandafter\XINT_mul_out\the\numexpr\XINT_verysmallmul 0.2!%
417 }%

```

3.15 \xintHalf

v1.08. Rewritten for v1.2.

```

418 \def\xintHalf {\romannumeral0\xinthalf }%
419 \def\xinthalf #1%
420 {%
421 \expandafter\XINT_half\romannumeral`&&@#1\Z
422 }%
423 \def\XINT_half #1%
424 {%
425 \xint_UDzerominusfork
426 #1-\XINT_half_zero
427 0#1\XINT_half_neg
428 0-{\XINT_half_pos #1}%
429 \krof
430 }%
431 \def\XINT_half_zero #1\Z { 0}%
432 \def\XINT_half_neg {\expandafter\XINT_opp\romannumeral0\XINT_half_pos }%
433 \def\XINT_half_pos #1\Z
434 {%
435 \expandafter\XINT_half_pos_a
436 \romannumeral0\expandafter\XINT_sepandrev
437 \romannumeral0\XINT_zeroes_forviii #1\R\R\R\R\R\R\R\{10}0000001\W
438 #1\XINT_rsepbyviii_end_A 2345678%
439 \XINT_rsepbyviii_end_B 2345678\relax XX%
440 \R.\R.\R.\R.\R.\R.\R.\R.\W
441 1\Z!%
442 1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
443 }%
444 \def\XINT_half_pos_a
445 {\expandafter\XINT_half_pos_b\the\numexpr\XINT_verysmallmul 0.5!}%
446 \def\XINT_half_pos_b 1#1#2#3#4#5#6#7#8!1#9%
447 {%
448 \xint_gob_til_Z #9\XINT_half_small \Z
449 \XINT_mul_out 1#1#2#3#4#5#6#7!1#9%
450 }%
451 \edef\XINT_half_small \Z\XINT_mul_out 1#1!#2\W
452 {%
453 \noexpand\expandafter\space\noexpand\the\numexpr #1\relax
454 }%

```

3.16 \xintDec

v1.08. Rewritten for v1.2.

```

455 \def\xintDec {\romannumeral0\xintdec }%
456 \def\xintdec #1%
457 {%
458   \expandafter\XINT_dec\romannumeral`&&@#1\Z
459 }%
460 \def\XINT_dec #1%
461 {%
462   \xint_UDzerominusfork
463   #1-\XINT_dec_zero
464   0#1\XINT_dec_neg
465   0-{\XINT_dec_pos #1}%
466   \krof
467 }%
468 \def\XINT_dec_zero #1\Z { -1}%
469 \def\XINT_dec_neg
470   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_inc_pos }%
471 \def\XINT_dec_pos #1\Z
472 {%
473   \expandafter\XINT_dec_pos_aa
474   \romannumeral0\expandafter\XINT_sepandrev
475   \romannumeral0\XINT_zeroes_forviii #1\R\R\R\R\R\R\R\{10}0000001\W
476   #1\XINT_rsepbyviii_end_A 2345678%
477   \XINT_rsepbyviii_end_B 2345678\relax XX%
478   \R.\R.\R.\R.\R.\R.\R.\R.\W
479   \Z!\Z!\Z!\Z!\W
480 }%
481 \def\XINT_dec_pos_aa {\XINT_sub_aa 100000001!\Z!\Z!\Z!\Z!\W }%

```

3.17 \xintInc

v1.08. Rewritten for v1.2.

```

482 \def\xintInc {\romannumeral0\xintinc }%
483 \def\xintinc #1%
484 {%
485   \expandafter\XINT_inc\romannumeral`&&@#1\Z
486 }%
487 \def\XINT_inc #1%
488 {%
489   \xint_UDzerominusfork
490   #1-\XINT_inc_zero
491   0#1\XINT_inc_neg
492   0-{\XINT_inc_pos #1}%
493   \krof
494 }%
495 \def\XINT_inc_zero #1\Z { 1}%
496 \def\XINT_inc_neg {\expandafter\XINT_opp\romannumeral0\XINT_dec_pos }%

```

1.2d interface to addition has changed.

```

497 \def\XINT_inc_pos #1\Z
498 {%
499   \expandafter\XINT_inc_pos_aa

```



```

500 \romannumeral0\expandafter\XINT_sepandrev
501 \romannumeral0\XINT_zeroes_forviii #1\R\R\R\R\R\R\R\{10}0000001\W
502 #1\XINT_rsepbyviii_end_A 2345678%
503 \XINT_rsepbyviii_end_B 2345678\relax XX%
504 \R.\R.\R.\R.\R.\R.\R.\R.\W
505 1\Z!1\Z!1\Z!1\Z!\W
506 1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
507 }%
508 \def\XINT_inc_pos_aa {\XINT_add_aa 100000001!1\Z!1\Z!1\Z!1\Z!\W }%

```

3.18 Core arithmetic

The four operations have been rewritten entirely for release v1.2. 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.

3.19 `\xintiAdd`, `\xintiiAdd`

```

509 \def\xintiAdd {\romannumeral0\xintiadd }%
510 \def\xintiadd #1{\expandafter\XINT_iadd\romannumeral0\xintnum{#1}\Z }%
511 \def\xintiiAdd {\romannumeral0\xintiiadd }%
512 \def\xintiiadd #1{\expandafter\XINT_iiadd\romannumeral`&&@#1\Z }%
513 \def\XINT_iiadd #1#2\Z #3%
514 {%
515 \expandafter\XINT_add_nfork\expandafter #1\romannumeral`&&@#3\Z #2\Z
516 }%
517 \def\XINT_iadd #1#2\Z #3%
518 {%
519 \expandafter\XINT_add_nfork\expandafter #1\romannumeral0\xintnum{#3}\Z #2\Z
520 }%
521 \def\XINT_add_fork #1#2\Z #3\Z {\XINT_add_nfork #1#3\Z #2\Z}%
522 \def\XINT_add_nfork #1#2%
523 {%
524 \xint_UDzerofork
525 #1\XINT_add_firstiszero
526 #2\XINT_add_secondiszero
527 0{}}%
528 \krof
529 \xint_UDsignsfork
530 #1#2\XINT_add_minusminus
531 #1-\XINT_add_minusplus
532 #2-\XINT_add_plusminus
533 --\XINT_add_plusplus
534 \krof #1#2%
535 }%
536 \def\XINT_add_firstiszero #1\krof 0#2#3\Z #4\Z { #2#3}%

```

3 Package *xintcore* implementation

```

537 \def\XINT_add_secondiszero #1\krof #2#3\Z #4\Z { #2#4}%
538 \def\XINT_add_minusminus #1#2%
539   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_add_pp_a {}}}%
540 \def\XINT_add_minusplus #1#2{\XINT_sub_mmm_a {}}#2}%
541 \def\XINT_add_plusminus #1#2%
542   {\expandafter\XINT_opp\romannumeral0\XINT_sub_mmm_a #1{}}}%
543 \def\XINT_add_pp_a #1#2#3\Z
544 {%
545   \expandafter\XINT_add_pp_b
546     \romannumeral0\expandafter\XINT_sepandrev_andcount
547     \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
548     #2#3\XINT_rsepbyviii_end_A 2345678%
549     \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_iii
550     \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
551     \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
552   \X #1%
553 }%
554 \let\XINT_add_plusplus \XINT_add_pp_a

```

I have been annoyed since the preparation of 1.2 release that addition sometimes had the (pre-reverse) output with a final 1! (now 1\Z!) sometimes not. It didn't matter for addition itself if it executes the final reverse as the 1! was then be swallowed. But if one wants to call addition repeatedly or from another routine such as `\XINT_mul_loop`, keeping reverse format, this is annoying. Finally for 1.2c I decide (2015/11/14) to impose always the ending 1! (or rather 1\Z!, which thus does not need to be put in the pattern for `_unrevbyviii`). I take this opportunity to move the ending pattern needed by `\XINT_add_out` to `\XINT_add_pp_b`, thus replacing a final fetch of the complete output to clean up the `\Z`'s and `\W` at the end of the input. This was also needed to make `\XINT_mul_loop` callable directly independently of whether the first argument is only one 10^8 digit long.

Impacted callers: `\XINT_mul_loop` (and through it square and pow) and `\XINT_inc_pos_` the latter must insert the pattern previously found in `\XINT_add_out` as it calls `\XINT_add_aa` directly.

I also modify addition to use `1\Z!1\Z!1\Z!1\Z!\W` as input delimiter (earlier version had `\Z!\Z!\Z!\Z!\Z!\W` but four are enough and we now have 1's). The rationale is that multiplication and now addition always set the output (before reversal) to be followed by `1\Z!`, thus it makes sense for `1\Z!` to also serve as (part of) delimiting inputs. Earlier, addition had `\Z!` for input, but this can not be put on output by a `\numexpr`, hence it used `1!` on output, but this is not a good delimiter as the `1!` may and will arise in number part, thus one had to use `!1!` or `1!\W` etc... to use it. With `1\Z!` things are more unified and facilitate doing repeated additions and multiplications maintaining things reversed.

```

555 \def\XINT_add_pp_b #1.#2\X #3\Z
556 {%
557   \expandafter\XINT_add_checklengths
558   \the\numexpr #1\expandafter.%
559   \romannumeral0\expandafter\XINT_sepandrev_andcount
560   \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
561   #3\XINT_rsepbyviii_end_A 2345678%
562   \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_iii
563   \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
564   \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
565   1\Z!1\Z!1\Z!1\Z!\W #21\Z!1\Z!1\Z!1\Z!\W
566   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
567 }%

```

3 Package *xintcore* implementation

I keep #1.#2. to check if at most 6 + 6 base 10⁸ digits which can be treated faster for final reverse. But is this overhead at all useful ?

```
568 \def\XINT_add_checklengths #1.#2.%
569 {%
570   \ifnum #2>#1
571     \expandafter\XINT_add_exchange
572   \else
573     \expandafter\XINT_add_A
574   \fi
575   #1.#2.%
576 }%
577 \def\XINT_add_exchange #1.#2.#3\W #4\W
578 {%
579   \XINT_add_A #2.#1.#4\W #3\W
580 }%
581 \def\XINT_add_A #1.#2.%
582 {%
583   \ifnum #1>\xint_c_vi
584     \expandafter\XINT_add_aa
585   \else \expandafter\XINT_add_aa_small
586   \fi
587 }%
588 \def\XINT_add_aa {\expandafter\XINT_add_out\the\numexpr\XINT_add_a \xint_c_ii}%
589 \def\XINT_add_out{\expandafter\XINT_cuz_small\romannumeral0\XINT_unrevbyviii {}}%
590 \def\XINT_add_aa_small
591   {\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.

```
592 \def\XINT_add_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
593 {%
594   \XINT_add_b #1!#6!#2!#7!#3!#8!#4!#9!#5\W
595 }%
596 \def\XINT_add_b #1!#2!#3!#4!%
597 {%
598   \xint_gob_til_Z #2\XINT_add_bi \Z
599   \expandafter\XINT_add_c\the\numexpr#1+1#2#3+#4-\xint_c_ii.%
600 }%
601 \def\XINT_add_bi\Z\expandafter\XINT_add_c
602   \the\numexpr#1+#2+#3-\xint_c_ii.#4!#5!#6!#7!#8!#9!\W
603 {%
604   \XINT_add_k #1#3!#5!#7!#9!%
605 }%
606 \def\XINT_add_c #1#2.%
607 {%
608   1#2\expandafter!\the\numexpr\XINT_add_d #1%
609 }%
610 \def\XINT_add_d #1!#2!#3!#4!%
611 {%
612   \xint_gob_til_Z #2\XINT_add_di \Z
613   \expandafter\XINT_add_e\the\numexpr#1+1#2#3+#4-\xint_c_ii.%
614 }%
```

3 Package *xintcore* implementation

```

615 \def\XINT_add_di\Z\expandafter\XINT_add_e
616   \the\numexpr#1+#2+#3-\xint_c_ii.#4!#5!#6!#7!#8\W
617 {%
618   \XINT_add_k #1#3!#5!#7!%
619 }%
620 \def\XINT_add_e #1#2.%
621 {%
622   1#2\expandafter!\the\numexpr\XINT_add_f #1%
623 }%
624 \def\XINT_add_f #1#2#3!#4!%
625 {%
626   \xint_gob_til_Z #2\XINT_add_fi \Z
627   \expandafter\XINT_add_g\the\numexpr#1+1#2#3+#4-\xint_c_ii.%
628 }%
629 \def\XINT_add_fi\Z\expandafter\XINT_add_g
630   \the\numexpr#1+#2+#3-\xint_c_ii.#4!#5!#6\W
631 {%
632   \XINT_add_k #1#3!#5!%
633 }%
634 \def\XINT_add_g #1#2.%
635 {%
636   1#2\expandafter!\the\numexpr\XINT_add_h #1%
637 }%
638 \def\XINT_add_h #1#2#3!#4!%
639 {%
640   \xint_gob_til_Z #2\XINT_add_hi \Z
641   \expandafter\XINT_add_i\the\numexpr#1+1#2#3+#4-\xint_c_ii.%
642 }%
643 \def\XINT_add_hi\Z
644   \expandafter\XINT_add_i\the\numexpr#1+#2+#3-\xint_c_ii.#4\W
645 {%
646   \XINT_add_k #1#3!%
647 }%
648 \def\XINT_add_i #1#2.%
649 {%
650   1#2\expandafter!\the\numexpr\XINT_add_a #1%
651 }%

```

These ending routines modified in 1.2c in order to clean up here (and not via \XINT_add_out) the tokens up to the final \W, and to always have a final 1\Z! (1.2 version had a final 1! not 1\Z!, and only under certain circumstances): when the two operands have the same length and the addition creates no carry or more generally when we had a carry propagating to the last block but the final addition created no carry, we end up in \XINT_add_ke with an empty #1 and a \numexpr to stop. This is why we put 1\Z! (1.2 had 1!, but 1\Z! is also used by multiplication on output) in that case, and now with 1.2c for all other cases as well.

```

652 \def\XINT_add_k #1{\if #12\expandafter\XINT_add_ke\else\expandafter\XINT_add_l \fi}%
653 \def\XINT_add_ke #11\Z #2\W {\XINT_add_kf #11\Z!}%
654 \def\XINT_add_kf 1{1\relax }%
655 \def\XINT_add_l 1#1#2{\xint_gob_til_Z #1\XINT_add_lf \Z \XINT_add_m 1#1#2}%
656 \def\XINT_add_lf #1\W {1\relax 00000001!1\Z!}%
657 \def\XINT_add_m #1!{\expandafter\XINT_add_n\the\numexpr\xint_c_i+#1.}%
658 \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>.)

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

3.20 \xintiSub, \xintiiSub

Entirely rewritten for v1.2.

```
660 \def\xintiiSub {\romannumeral0\xintiisub }%
661 \def\xintiisub #1{\expandafter\XINT_iisub\romannumeral`&&@#1\Z }%
662 \def\XINT_iisub #1#2\Z #3%
663 {%
664   \expandafter\XINT_sub_nfork\expandafter #1\romannumeral`&&@#3\Z #2\Z
665 }%
666 \def\xintiSub {\romannumeral0\xintisub }%
667 \def\xintisub #1{\expandafter\XINT_isub\romannumeral0\xintnum{#1}\Z }%
668 \def\XINT_isub #1#2\Z #3%
669 {%
670   \expandafter\XINT_sub_nfork\expandafter #1\romannumeral0\xintnum{#3}\Z #2\Z
671 }%
672 \def\XINT_sub_nfork #1#2%
673 {%
674   \xint_UDzerofork
675   #1\XINT_sub_firstiszero
676   #2\XINT_sub_secondiszero
677   0{}}%
678   \krof
679   \xint_UDsignsfork
680   #1#2\XINT_sub_minusminus
681   #1-\XINT_sub_minusplus
682   #2-\XINT_sub_plusminus
683   --\XINT_sub_plusplus
684   \krof #1#2%
685 }%
686 \def\XINT_sub_firstiszero #1\krof 0#2#3\Z #4\Z {\XINT_opp #2#3}%
687 \def\XINT_sub_secondiszero #1\krof #20#3\Z #4\Z { #2#4}%
688 \def\XINT_sub_plusminus #1#2{\XINT_add_pp_a #1{}}%
689 \def\XINT_sub_plusplus #1#2%
690   {\expandafter\XINT_opp\romannumeral0\XINT_sub_mm_a #1#2}%
691 \def\XINT_sub_minusplus #1#2%
692   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_add_pp_a { }#2}%
693 \def\XINT_sub_minusminus #1#2{\XINT_sub_mm_a { }{ }}%
694 \def\XINT_sub_mm_a #1#2#3\Z
695 {%
696   \expandafter\XINT_sub_mm_b
697   \romannumeral0\expandafter\XINT_sepandrev_andcount
698   \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
699   #2#3\XINT_rsepbyviii_end_A 2345678%
700   \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_iii
701   \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
702   \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
703   \X #1%
704 }%
705 \def\XINT_sub_mm_b #1.#2\X #3\Z
```

3 Package *xintcore* implementation

```

706 {%
707   \expandafter\XINT_sub_checklengths
708   \the\numexpr #1\expandafter.%
709   \romannumeral0\expandafter\XINT_sepandrev_andcount
710   \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\{10}0000001\W
711   #3\XINT_rsepbyviii_end_A 2345678%
712   \XINT_rsepbyviii_end_B 2345678\relax \xint_c_ii\xint_c_iii
713   \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
714   \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
715   \Z!\Z!\Z!\Z!\W #2\Z!\Z!\Z!\Z!\W
716 }%
717 \def\XINT_sub_checklengths #1.#2.%
718 {%
719   \ifnum #2>#1
720     \expandafter\XINT_sub_exchange
721   \else
722     \expandafter\XINT_sub_aa
723   \fi
724 }%
725 \def\XINT_sub_exchange #1\W #2\W
726 {%
727   \expandafter\XINT_opp\romannumeral0\XINT_sub_aa #2\W #1\W
728 }%
729 \def\XINT_sub_aa {\expandafter\XINT_sub_out\the\numexpr\XINT_sub_a \xint_c_i }%
730 \def\XINT_sub_out #1\Z #2#3\W
731 {%
732   \if-#2\expandafter\XINT_sub_startrescue\fi
733   \expandafter\XINT_cuz_small
734   \romannumeral0\XINT_unrevbyviii {}#11\Z!1\R!1\R!1\R!1\R!1\R!1\R!1\W
735 }%

```

The routine starting with `\XINT_sub_a` requires the first argument to be at most as long as second argument.

```

736 \def\XINT_sub_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
737 {%
738   \XINT_sub_b #1!#6!#2!#7!#3!#8!#4!#9!#5\W
739 }%
740 \def\XINT_sub_b #1#2#3!#4!%
741 {%
742   \xint_gob_til_Z #2\XINT_sub_bi \Z
743   \expandafter\XINT_sub_c\the\numexpr#1+1#4-#3-\xint_c_i.%
744 }%
745 \def\XINT_sub_c 1#1#2.%
746 {%
747   1#2\expandafter!\the\numexpr\XINT_sub_d #1%
748 }%
749 \def\XINT_sub_d #1#2#3!#4!%
750 {%
751   \xint_gob_til_Z #2\XINT_sub_di \Z
752   \expandafter\XINT_sub_e\the\numexpr#1+1#4-#3-\xint_c_i.%
753 }%
754 \def\XINT_sub_e 1#1#2.%

```

3 Package *xintcore* implementation

```
755 {%
756     1#2\expandafter!\the\numexpr\XINT_sub_f #1%
757 }%
758 \def\XINT_sub_f #1#2#3!#4!%
759 {%
760     \xint_gob_til_Z #2\XINT_sub_fi \Z
761     \expandafter\XINT_sub_g\the\numexpr#1+1#4-#3-\xint_c_i.%
762 }%
763 \def\XINT_sub_g 1#1#2.%
764 {%
765     1#2\expandafter!\the\numexpr\XINT_sub_h #1%
766 }%
767 \def\XINT_sub_h #1#2#3!#4!%
768 {%
769     \xint_gob_til_Z #2\XINT_sub_hi \Z
770     \expandafter\XINT_sub_i\the\numexpr#1+1#4-#3-\xint_c_i.%
771 }%
772 \def\XINT_sub_i 1#1#2.%
773 {%
774     1#2\expandafter!\the\numexpr\XINT_sub_a #1%
775 }%
776 \def\XINT_sub_bi\Z
777     \expandafter\XINT_sub_c\the\numexpr#1+1#2-#3.#4!#5!#6!#7!#8!#9!\W
778 {%
779     \XINT_sub_k #1#2!#5!#7!#9!%
780 }%
781 \def\XINT_sub_di\Z
782     \expandafter\XINT_sub_e\the\numexpr#1+1#2-#3.#4!#5!#6!#7!#8\W
783 {%
784     \XINT_sub_k #1#2!#5!#7!%
785 }%
786 \def\XINT_sub_fi\Z
787     \expandafter\XINT_sub_g\the\numexpr#1+1#2-#3.#4!#5!#6\W
788 {%
789     \XINT_sub_k #1#2!#5!%
790 }%
791 \def\XINT_sub_hi\Z
792     \expandafter\XINT_sub_i\the\numexpr#1+1#2-#3.#4\W
793 {%
794     \XINT_sub_k #1#2!%
795 }%
```

First input terminated. Have we reached the end of second (necessarily at least as long) input? If not, then we are certain that even if there is carry it will not propagate beyond the end of second input. But it may propagate along chains of 00000000. And if its goes to the final block which is just $1 < 00000001 > !$, we will have at least those eight zeros to clean up. But not more than those eight followed by the leading zeroes of next to last block (which will be leading block of final output). On the other hand if we have also reached the end of the second input, then if first input was smaller there might be arbitrarily many zeroes to clean up, if it was larger, we will have to rescue the whole thing.

```
796 \def\XINT_sub_k #1#2%
797 {%
```

3 Package *xintcore* implementation

```
798 \xint_gob_til_Z #2\XINT_sub_p\Z \XINT_sub_l #1#2%
799 }%
```

Here second input was longer. The carry if there is one will be extinguished before the end. 1.2c wants subtraction to output before final reversal the blocks with the same 1\Z! terminator as addition and multiplication. CANCELED FOR THE TIME BEING.

2015/11/15. I discover with shame that Release 1.2 of 10/10 had a bad bad bad bad bug in case of long stretches of zeroes, for example with \xintiiSub {10000000112345678}{12345679} which returned 99999999 sorry.

I was rewriting inner entry to subtraction to look a bit more for input/output as addition and multiplication but I will now rather quickly leave everything standing and issue a bugfix release asap.

```
800 \def\XINT_sub_l #1{\xint_UDzerofork #1\XINT_sub_l_carry 0\XINT_sub_l_nocarry\krof}%
801 \def\XINT_sub_l_nocarry 1{1\relax }%
802 \def\XINT_sub_l_carry #1!\{expandafter\XINT_sub_m\the\numexpr 1#1-\xint_c_i!}%
803 \def\XINT_sub_m 1#1{\xint_UDzerofork #1\XINT_sub_n_carry 0\XINT_sub_n_nocarry\krof}%
804 \def\XINT_sub_n_carry #1!\{1#1\expandafter!\the\numexpr\XINT_sub_l_carry }%
805 \def\XINT_sub_n_nocarry #1!#2#3!%
806 {%
807 \xint_gob_til_Z #2\xint_gob_til_eightzeroes #1\XINT_sub_n_zero
808 00000000\xint_gob_til_Z\Z 1\relax #1!#2#3!%
809 }%
810 \def\XINT_sub_n_zero 00000000\xint_gob_til_Z\Z 1\relax 00000000!{1!}%

```

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

```
811 \def\XINT_sub_p\Z\XINT_sub_l #1#2\W
812 {%
813 \xint_UDzerofork
814 #1{-1\relax\Z -\W}%
815 0{1\relax \XINT_cuz_byviii!\Z 0\W\R }%
816 \krof
817 }%
818 \def\XINT_sub_startrescue\expandafter\XINT_cuz_small
819 \romannumeral0\XINT_unrevbyviii #1#2\Z!#3\W
820 {%
821 \expandafter\XINT_sub_rescue_finish
822 \the\numexpr\XINT_sub_rescue_a #2!%
823 1\Z!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W \R
824 }%
825 \def\XINT_sub_rescue_finish
826 {\expandafter-\romannumeral0\expandafter\XINT_cuz\romannumeral0\XINT_unrevbyviii }%
827 \def\XINT_sub_rescue_a #1!%
828 {%
829 \expandafter\XINT_sub_rescue_c\the\numexpr \xint_c_xii_e_viii-#1.%
830 }%
831 \def\XINT_sub_rescue_c 1#1#2.%
832 {%
833 1#2\expandafter!\the\numexpr\XINT_sub_rescue_d #1%
834 }%

```



```

835 \def\XINT_sub_rescue_d #1#2#3!%
836 {%
837     \xint_gob_til_minus #2\XINT_sub_rescue_z -%
838     \expandafter\XINT_sub_rescue_c\the\numexpr \xint_c_xii_e_viii_mone-#2#3+#1.%
839 }%
840 \def\XINT_sub_rescue_z #1.{1!}%

```

3.21 \xintiMul, \xintiiMul

Completely rewritten for v1.2.

```

841 \def\xintiMul {\romannumeral0\xintimul }%
842 \def\xintimul #1%
843 {%
844     \expandafter\XINT_imul\romannumeral0\xintnum{#1}\Z
845 }%
846 \def\XINT_imul #1#2\Z #3%
847 {%
848     \expandafter\XINT_mul_nfork\expandafter #1\romannumeral0\xintnum{#3}\Z #2\Z
849 }%
850 \def\xintiiMul {\romannumeral0\xintiimul }%
851 \def\xintiimul #1%
852 {%
853     \expandafter\XINT_iimul\romannumeral`&&@#1\Z
854 }%
855 \def\XINT_iimul #1#2\Z #3%
856 {%
857     \expandafter\XINT_mul_nfork\expandafter #1\romannumeral`&&@#3\Z #2\Z
858 }%

```

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

```

859 \def\XINT_mul_fork #1#2\Z #3\Z{\XINT_mul_nfork #1#3\Z #2\Z}%
860 \def\XINT_mul_nfork #1#2%
861 {%
862     \xint_UDzerofork
863     #1\XINT_mul_zero
864     #2\XINT_mul_zero
865     0}%
866 \krof
867 \xint_UDsignsfork
868     #1#2\XINT_mul_minusminus
869     #1-\XINT_mul_minusplus
870     #2-\XINT_mul_plusminus
871     --\XINT_mul_plusplus
872 \krof #1#2%
873 }%
874 \def\XINT_mul_zero #1\krof #2#3\Z #4\Z { 0}%
875 \def\XINT_mul_minusminus #1#2{\XINT_mul_plusplus {}{}}%
876 \def\XINT_mul_minusplus #1#2%
877     {\expandafter\xint_minus_thenstop\romannumeral0\XINT_mul_plusplus {}#2}%
878 \def\XINT_mul_plusminus #1#2%
879     {\expandafter\xint_minus_thenstop\romannumeral0\XINT_mul_plusplus #1{}}%

```

3 Package *xintcore* implementation

```

880 \def\XINT_mul_plusplus #1#2#3\Z
881 {%
882   \expandafter\XINT_mul_pre_b
883     \romannumeral0\expandafter\XINT_sepandrev_andcount
884     \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
885     #2#3\XINT_rsepbyviii_end_A 2345678%
886     \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_iii
887     \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
888     \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
889   \W #1%
890 }%
891 \def\XINT_mul_pre_b #1.#2\W #3\Z
892 {%
893   \expandafter\XINT_mul_checklengths
894   \the\numexpr #1\expandafter.%
895   \romannumeral0\expandafter\XINT_sepandrev_andcount
896   \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
897   #3\XINT_rsepbyviii_end_A 2345678%
898   \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_iii
899   \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
900   \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
901   1\Z!\W #21\Z!%
902   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
903 }%

```

Cooking recipe, 2015/10/05.

```

904 \def\XINT_mul_checklengths #1.#2.%
905 {%
906   \ifnum #2=\xint_c_i\expandafter\XINT_mul_smallbyfirst\fi
907   \ifnum #1=\xint_c_i\expandafter\XINT_mul_smallbysecond\fi
908   \ifnum #2<#1
909     \ifnum \numexpr (#2-\xint_c_i)*(#1-#2)<383
910       \XINT_mul_exchange
911     \fi
912   \else
913     \ifnum \numexpr (#1-\xint_c_i)*(#2-#1)>383
914       \XINT_mul_exchange
915     \fi
916   \fi
917   \XINT_mul_start
918 }%
919 \def\XINT_mul_smallbyfirst #1\XINT_mul_start 1#2!1\Z!\W
920 {%
921   \ifnum#2=\xint_c_i\expandafter\XINT_mul_oneisone\fi
922   \ifnum#2<\xint_c_xxii\expandafter\XINT_mul_verysmall\fi
923   \expandafter\XINT_mul_out\the\numexpr\XINT_smallmul 1#2!%
924 }%
925 \def\XINT_mul_smallbysecond #1\XINT_mul_start #2\W 1#3!1\Z!%
926 {%
927   \ifnum#3=\xint_c_i\expandafter\XINT_mul_oneisone\fi
928   \ifnum#3<\xint_c_xxii\expandafter\XINT_mul_verysmall\fi
929   \expandafter\XINT_mul_out\the\numexpr\XINT_smallmul 1#3!#2%

```

3 Package *xintcore* implementation

```

930 }%
931 \def\XINT_mul_oneisone #1!\XINT_mul_out }%
932 \def\XINT_mul_verysmall\expandafter\XINT_mul_out
933         \the\numexpr\XINT_smallmul 1#1!%
934     {\expandafter\XINT_mul_out\the\numexpr\XINT_verysmallmul 0.#1!}%
935 \def\XINT_mul_exchange #1\XINT_mul_start #2\W #31\Z!%
936     {\fi\fi\XINT_mul_start #31\Z!\W #2}%

```

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

```

937 \def\XINT_mul_start
938     {\expandafter\XINT_mul_out\the\numexpr\XINT_mul_loop 100000000!1\Z!\W}%
939 \def\XINT_mul_out
940     {\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).

```

941 \def\XINT_mul_loop #1\W #2\W 1#3!%
942 {%
943     \xint_gob_til_Z #3\XINT_mul_e \Z
944     \expandafter\XINT_mul_a\the\numexpr \XINT_smallmul 1#3!#2\W
945     #1\W #2\W
946 }%

```

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

```

947 \def\XINT_mul_a #1\W #2\W
948 {%
949     \expandafter\XINT_mul_b\the\numexpr
950     \XINT_add_a \xint_c_ii #21\Z!1\Z!1\Z!\W #11\Z!1\Z!1\Z!\W\W
951 }%
952 \def\XINT_mul_b 1#1!{1#1\expandafter!\the\numexpr\XINT_mul_loop }%
953 \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.

```

954 \def\XINT_minimulwc_a 1#1.#2.#3!#4#5#6#7#8.%

```

3 Package *xintcore* implementation

```

955 {%
956   \expandafter\XINT_minimulwc_b
957   \the\numexpr \xint_c_x^ix+#1+#3*#8.#3*#4#5#6#7+#2*#8.#2*#4#5#6#7.%
958 }%
959 \def\XINT_minimulwc_b 1#1#2#3#4#5#6.#7.%
960 {%
961   \expandafter\XINT_minimulwc_c
962   \the\numexpr \xint_c_x^ix+#1#2#3#4#5+#7.#6.%
963 }%
964 \def\XINT_minimulwc_c 1#1#2#3#4#5#6.#7.#8.%
965 {%
966   1#6#7\expandafter!%
967   \the\numexpr\expandafter\XINT_smallmul_a
968   \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8.%
969 }%
970 \def\XINT_smallmul 1#1#2#3#4#5!\{\XINT_smallmul_a 100000000.#1#2#3#4.#5!}%
971 \def\XINT_smallmul_a #1.#2.#3!1#4!%
972 {%
973   \xint_gob_til_Z #4\XINT_smallmul_e\Z
974   \XINT_minimulwc_a #1.#2.#3!#4.#2.#3!%
975 }%
976 \def\XINT_smallmul_e\Z\XINT_minimulwc_a 1#1.#2\Z #3!%
977   {\xint_gob_til_eightzeroes #1\XINT_smallmul_f 000000001\relax #1!1\Z!}%
978 \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.

```

979 \def\XINT_verysmallmul #1.#2!1#3!%
980 {%
981   \xint_gob_til_Z #3\XINT_verysmallmul_e\Z
982   \expandafter\XINT_verysmallmul_a
983   \the\numexpr #2*#3+#1.#2!%
984 }%
985 \def\XINT_verysmallmul_e\Z\expandafter\XINT_verysmallmul_a\the\numexpr
986   #1+#2#3.#4!%
987 {\xint_gob_til_zero #2\XINT_verysmallmul_f 0\xint_c_x^viii+#2#3!1\Z!}%
988 \def\XINT_verysmallmul_f #1!1{1\relax}%
989 \def\XINT_verysmallmul_a #1#2.%
990 {%
991   \unless\ifnum #1#2<\xint_c_x^ix
992     \expandafter\XINT_verysmallmul_bi\else
993     \expandafter\XINT_verysmallmul_bj\fi
994     \the\numexpr \xint_c_x^ix+#1#2.%
995 }%
996 \def\XINT_verysmallmul_bj{\expandafter\XINT_verysmallmul_cj }%
997 \def\XINT_verysmallmul_cj 1#1#2.%
998   {1#2\expandafter!\the\numexpr\XINT_verysmallmul #1.}%
999 \def\XINT_verysmallmul_bi\the\numexpr\xint_c_x^ix+#1#2#3.%
1000   {1#3\expandafter!\the\numexpr\XINT_verysmallmul #1#2.}%

```

Used by division and by squaring, not by multiplication itself. Attention, returns least significant 1<8digits> first.

```

1001 \def\XINT_minimul_a #1.#2!#3#4#5#6#7!%
1002 {%
1003   \expandafter\XINT_minimul_b
1004   \the\numexpr \xint_c_x^viii+#2*#7.#2*#3#4#5#6+#1*#7.#1*#3#4#5#6.%
1005 }%
1006 \def\XINT_minimul_b 1#1#2#3#4#5.#6.%
1007 {%
1008   \expandafter\XINT_minimul_c
1009   \the\numexpr \xint_c_x^ix+#1#2#3#4+#6.#5.%
1010 }%
1011 \def\XINT_minimul_c 1#1#2#3#4#5#6.#7.#8.%
1012 {%
1013   1#6#7\expandafter!\the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8!%
1014 }%

```

3.22 \xintiSqr, \xintiiSqr

Rewritten for v1.2.

```

1015 \def\xintiiSqr {\romannumeral0\xintiisqr }%
1016 \def\xintiisqr #1%
1017 {%
1018   \expandafter\XINT_sqr\romannumeral0\xintiabs{#1}\Z
1019 }%
1020 \def\xintiSqr {\romannumeral0\xintisqr }%
1021 \def\xintisqr #1%
1022 {%
1023   \expandafter\XINT_sqr\romannumeral0\xintiabs{#1}\Z
1024 }%
1025 \def\XINT_sqr #1\Z
1026 {%
1027   \expandafter\XINT_sqr_a
1028   \romannumeral0\expandafter\XINT_sepandrev_andcount
1029   \romannumeral0\XINT_zeroes_forviii #1\R\R\R\R\R\R\R\R{10}0000001\W
1030   #1\XINT_rsepbyviii_end_A 2345678%
1031   \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_iii
1032   \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
1033   \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
1034   \Z
1035 }%

```

1.2c \XINT_mul_loop can be called directly even with small arguments, thus the following is not anymore a necessity. The 1!\R in \XINT_sqr_start is to obey the new calling pattern of \XINT_mul_loop.

```

1036 \def\XINT_sqr_a #1.%
1037 {%
1038   \ifnum #1=\xint_c_i \expandafter\XINT_sqr_small
1039   \else\expandafter\XINT_sqr_start\fi
1040 }%
1041 \def\XINT_sqr_small 1#1#2#3#4#5!\Z
1042 {%
1043   \ifnum #1#2#3#4#5<46341 \expandafter\XINT_sqr_verysmall\fi

```

```

1044 \expandafter\XINT_sqr_small_out
1045 \the\numexpr\XINT_minimul_a #1#2#3#4.#5!#1#2#3#4#5!%
1046 }%
1047 \edef\XINT_sqr_verysmall
1048 \expandafter\XINT_sqr_small_out\the\numexpr\XINT_minimul_a #1!#2!%
1049 {\noexpand\expandafter\space\noexpand\the\numexpr #2*#2\relax}%
1050 \def\XINT_sqr_small_out 1#1!1#2!%
1051 {%
1052 \XINT_cuz #2#1\R
1053 }%
1054 \def\XINT_sqr_start #1\Z
1055 {%
1056 \expandafter\XINT_mul_out
1057 \the\numexpr\XINT_mul_loop 100000000!1\Z!\W #11\Z!\W #11\Z!%
1058 1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1059 }%

```

3.23 \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 $N = 2^{17} = 131072$.

```

1060 \def\xintiiPow {\romannumeral0\xintiipow }%
1061 \def\xintiipow #1%
1062 {%
1063 \expandafter\xint_pow\romannumeral`&&@#1\Z%
1064 }%
1065 \def\xintiPow {\romannumeral0\xintipow }%
1066 \def\xintipow #1%
1067 {%
1068 \expandafter\xint_pow\romannumeral0\xintnum{#1}\Z%
1069 }%
1070 \def\xint_pow #1#2\Z
1071 {%
1072 \xint_UDsignfork
1073 #1\XINT_pow_Aneg
1074 -\XINT_pow_Anonneg
1075 \krof
1076 #1{#2}%
1077 }%
1078 \def\XINT_pow_Aneg #1#2#3%
1079 {%
1080 \expandafter\XINT_pow_Aneg_\expandafter{\the\numexpr #3}#2\Z
1081 }%
1082 \def\XINT_pow_Aneg_ #1%
1083 {%
1084 \ifodd #1
1085 \expandafter\XINT_pow_Aneg_Bodd
1086 \fi
1087 \XINT_pow_Anonneg_ {#1}%
1088 }%
1089 \def\XINT_pow_Aneg_Bodd #1%
1090 {%

```

3 Package *xintcore* implementation

```
1091 \expandafter\XINT_opp\romannumeral0\XINT_pow_Anonneg_
1092 }%
```

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

```
1093 \def\XINT_pow_Anonneg #1#2#3%
1094 {%
1095 \expandafter\XINT_pow_Anonneg_\expandafter {\the\numexpr #3}#1#2\Z
1096 }%
```

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

```
1097 \def\XINT_pow_Anonneg_ #1#2#3\Z
1098 {%
1099 \if\relax #3\relax\xint_dothis
1100 {\ifcase #2 \expandafter\XINT_pow_AisZero
1101 \or\expandafter\XINT_pow_AisOne
1102 \else\expandafter\XINT_pow_AatleastTwo
1103 \fi }\fi
1104 \xint_orthat \XINT_pow_AatleastTwo {#1}{#2#3}%
1105 }%
1106 \def\XINT_pow_AisOne #1#2{ 1}%
```

#1 = B

```
1107 \def\XINT_pow_AisZero #1#2%
1108 {%
1109 \ifcase\XINT_cntSgn #1\Z
1110 \xint_afterfi { 1}%
1111 \or
1112 \xint_afterfi { 0}%
1113 \else
1114 \xint_afterfi {\xintError:DivisionByZero\space 0}%
1115 \fi
1116 }%
1117 \def\XINT_pow_AatleastTwo #1%
1118 {%
1119 \ifcase\XINT_cntSgn #1\Z
1120 \expandafter\XINT_pow_BisZero
1121 \or
1122 \expandafter\XINT_pow_I_in
1123 \else
1124 \expandafter\XINT_pow_BisNegative
1125 \fi
1126 {#1}%
1127 }%
1128 \edef\XINT_pow_BisNegative #1#2%
1129 {\noexpand\xintError:FractionRoundedToZero\space 0}%
1130 \def\XINT_pow_BisZero #1#2{ 1}%
```

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

3 Package *xintcore* implementation

```

1131 \def\XINT_pow_I_in #1#2%
1132 {%
1133   \expandafter\XINT_pow_I_loop
1134   \the\numexpr #1\expandafter.%
1135   \romannumeral0\expandafter\XINT_sepandrev
1136   \romannumeral0\XINT_zeroes_forviii #2\R\R\R\R\R\R\R\R{10}0000001\W
1137   #2\XINT_rsepbyviii_end_A 2345678%
1138   \XINT_rsepbyviii_end_B 2345678\relax XX%
1139   \R.\R.\R.\R.\R.\R.\R.\R.\W 1\Z!\W
1140   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1141 }%
1142 \def\XINT_pow_I_loop #1.%
1143 {%
1144   \ifnum #1 = \xint_c_i\expandafter\XINT_pow_I_exit\fi
1145   \ifodd #1
1146     \expandafter\XINT_pow_II_in
1147   \else
1148     \expandafter\XINT_pow_I_squareit
1149   \fi #1.%
1150 }%
1151 \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.

```

1152 \def\XINT_pow_I_squareit #1.#2\W%
1153 {%
1154   \expandafter\XINT_pow_I_loop
1155   \the\numexpr #1/\xint_c_ii\expandafter.%
1156   \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W
1157 }%
1158 \def\XINT_pow_mulbutcheckifsmall #1!1#2%
1159 {%
1160   \xint_gob_til_Z #2\XINT_pow_mul_small\Z
1161   \XINT_mul_loop 100000000!1\Z!\W #1!1#2%
1162 }%
1163 \def\XINT_pow_mul_small\Z \XINT_mul_loop 100000000!1\Z!\W 1#1!1\Z!\W
1164 {%
1165   \XINT_smallmul 1#1!%
1166 }%
1167 \def\XINT_pow_II_in #1.#2\W
1168 {%
1169   \expandafter\XINT_pow_II_loop
1170   \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter.%
1171   \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W #2\W
1172 }%
1173 \def\XINT_pow_II_loop #1.%
1174 {%
1175   \ifnum #1 = \xint_c_i\expandafter\XINT_pow_II_exit\fi
1176   \ifodd #1
1177     \expandafter\XINT_pow_II_odda
1178   \else

```



```

1179     \expandafter\XINT_pow_II_even
1180     \fi #1.%
1181 }%
1182 \def\XINT_pow_II_exit\ifodd #1\fi #2.#3\W #4\W
1183 {%
1184     \expandafter\XINT_mul_out
1185     \the\numexpr\XINT_pow_mulbutcheckifsmall #4\W #3%
1186 }%
1187 \def\XINT_pow_II_even #1.#2\W
1188 {%
1189     \expandafter\XINT_pow_II_loop
1190     \the\numexpr #1/\xint_c_ii\expandafter.%
1191     \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W
1192 }%
1193 \def\XINT_pow_II_odda #1.#2\W #3\W
1194 {%
1195     \expandafter\XINT_pow_II_oddb
1196     \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter.%
1197     \the\numexpr\XINT_pow_mulbutcheckifsmall #3\W #2\W #2\W
1198 }%
1199 \def\XINT_pow_II_oddb #1.#2\W #3\W
1200 {%
1201     \expandafter\XINT_pow_II_loop
1202     \the\numexpr #1\expandafter.%
1203     \the\numexpr\XINT_pow_mulbutcheckifsmall #3\W #3\W #2\W
1204 }%

```

3.24 `\xintiFac`, `\xintiiFac`

Moved to `xintcore.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. Anyhow with current default settings of the etex memory and the current 1.2 routine (last commit: eadab1), 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 $n/2$ "small" multiplications of numbers of size $k \log(k)$ with k along a step 2 arithmetic sequence up to n . Each small multiplication should have a linear cost $O(k \log(k))$ (as we maintain the reversed representation) hence a total cost of $O(n^2 \log(n))$; and this seems to be confirmed experimentally, or rather 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 the big variability of `\pdfelapsedtime` allows to draw any conclusion -- I did not repeat the computations at least 100 times as I should have. 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 five lines of code implementation of binary splitting approach with recursive uses of `\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

3 Package *xintcore* implementation

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).

```
1205 \def\xintiFac {\romannumeral0\xintifac }%
1206 \def\xintifac #1%
1207 {%
1208   \expandafter\XINT_fac_fork\expandafter {\the\numexpr#1}%
1209 }%
1210 \def\xintiiFac {\romannumeral0\xintiifac }%
1211 \def\xintiifac #1%
1212 {%
1213   \expandafter\XINT_fac_fork\expandafter {\romannumeral`&&@#1}%
1214 }%
1215 \let\xintFac\xintiFac \let\xintfac\xintifac
1216 \def\XINT_fac_fork #1%
1217 {%
1218   \ifcase\XINT_cntSgn #1\Z
1219     \xint_afterfi{\expandafter\space\expandafter 1\xint_gobble_i }%
1220   \or
1221     \expandafter\XINT_fac_checksize
1222   \else
1223     \xint_afterfi{\expandafter\xintError:FactorialOfNegativeNumber
1224       \expandafter\space\expandafter 1\xint_gobble_i }%
1225   \fi
1226   {#1}%
1227 }%
1228 \def\XINT_fac_checksize #1%
1229 {%
1230   \ifnum #1>9999
1231     \xint_dothis{\expandafter\xintError:FactorialOfTooBigNumber
1232       \expandafter\space\expandafter 1\xint_gob_til_W }\fi
1233   \ifnum #1>465 \xint_dothis{\XINT_fac_bigloop_a #1.}\fi
1234   \ifnum #1>101 \xint_dothis{\XINT_fac_medloop_a #1.\XINT_mul_out}\fi
1235     \xint_orthat{\XINT_fac_smallloop_a #1.\XINT_mul_out}%
1236   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1237 }%
1238 \def\XINT_fac_bigloop_a #1.%
1239 {%
1240   \expandafter\XINT_fac_bigloop_b \the\numexpr
1241   #1-\xint_c_i-\xint_c_ii*((#1-464)/\xint_c_ii).#1.%
1242 }%
1243 \def\XINT_fac_bigloop_b #1.#2.%
1244 {%
1245   \expandafter\XINT_fac_medloop_a
1246   \the\numexpr #1-\xint_c_i.{\XINT_fac_bigloop_loop #1.#2.}%
1247 }%
1248 \def\XINT_fac_bigloop_loop #1.#2.%
1249 {%
1250   \ifnum #1>#2 \expandafter\XINT_fac_bigloop_exit\fi
1251   \expandafter\XINT_fac_bigloop_loop
1252   \the\numexpr #1+\xint_c_ii\expandafter.%
1253   \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_bigloop_mul #1!%
```

3 Package *xintcore* implementation

```

1254 }%
1255 \def\XINT_fac_bigloop_exit #1!\XINT_mul_out}%
1256 \def\XINT_fac_bigloop_mul #1!%
1257 {%
1258   \expandafter\XINT_smallmul
1259   \the\numexpr \xint_c_x^viii+1*(#1+\xint_c_i)!%
1260 }%
1261 \def\XINT_fac_medloop_a #1.%
1262 {%
1263   \expandafter\XINT_fac_medloop_b
1264   \the\numexpr #1+\xint_c_i-\xint_c_iii*((#1-100)/\xint_c_iii).#1.%
1265 }%
1266 \def\XINT_fac_medloop_b #1.#2.%
1267 {%
1268   \expandafter\XINT_fac_smallloop_a
1269   \the\numexpr #1-\xint_c_i.{\XINT_fac_medloop_loop #1.#2.}%
1270 }%
1271 \def\XINT_fac_medloop_loop #1.#2.%
1272 {%
1273   \ifnum #1>#2 \expandafter\XINT_fac_loop_exit\fi
1274   \expandafter\XINT_fac_medloop_loop
1275   \the\numexpr #1+\xint_c_iii\expandafter.%
1276   \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_medloop_mul #1!%
1277 }%
1278 \def\XINT_fac_medloop_mul #1!%
1279 {%
1280   \expandafter\XINT_smallmul
1281   \the\numexpr
1282     \xint_c_x^viii+1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1283 }%
1284 \def\XINT_fac_smallloop_a #1.%
1285 {%
1286   \csname
1287     XINT_fac_smallloop_\the\numexpr #1-\xint_c_iv*(#1/\xint_c_iv)\relax
1288   \endcsname #1.%
1289 }%
1290 \expandafter\def\csname XINT_fac_smallloop_1\endcsname #1.%
1291 {%
1292   \XINT_fac_smallloop_loop 2.#1.100000001!1\Z!%
1293 }%
1294 \expandafter\def\csname XINT_fac_smallloop_-2\endcsname #1.%
1295 {%
1296   \XINT_fac_smallloop_loop 3.#1.100000002!1\Z!%
1297 }%
1298 \expandafter\def\csname XINT_fac_smallloop_-1\endcsname #1.%
1299 {%
1300   \XINT_fac_smallloop_loop 4.#1.100000006!1\Z!%
1301 }%
1302 \expandafter\def\csname XINT_fac_smallloop_0\endcsname #1.%
1303 {%
1304   \XINT_fac_smallloop_loop 5.#1.1000000024!1\Z!%
1305 }%

```

3 Package *xintcore* implementation

```

1306 \def\XINT_fac_smallloop_loop #1.#2.%
1307 {%
1308     \ifnum #1>#2 \expandafter\XINT_fac_loop_exit\fi
1309     \expandafter\XINT_fac_smallloop_loop
1310     \the\numexpr #1+\xint_c_iv\expandafter.%
1311     \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_smallloop_mul #1!%
1312 }%
1313 \def\XINT_fac_smallloop_mul #1!%
1314 {%
1315     \expandafter\XINT_smallmul
1316     \the\numexpr
1317         \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1318 }%
1319 \def\XINT_fac_loop_exit #1!#2\Z!#3{#3#2\Z!}%

```

3.25 *\xintiDivision*, *\xintiQuo*, *\xintiRem*, *\xintiiDivision*, *\xintiiQuo*, *\xintiiRem*

Completely rewritten for v1.2.

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 v1.2, although the situation has changed. It was fine-tuned in xint v1.0/v1.1 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 v1.0/v1.1 fetched up to 9 parameters now need handle less such parameters. Thus, some rationale for the way the code was structured has disappeared.

v1.2 2015/10/15 had a bad bug which got corrected in v1.2b 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.

```

1320 \def\xintiiQuo {\romannumeral0\xintiiquo }%
1321 \def\xintiiRem {\romannumeral0\xintiiirem }%
1322 \def\xintiiquo {\expandafter\xint_firstoftwo_thenstop\romannumeral0\xintiidivision }%
1323 \def\xintiirem {\expandafter\xint_secondoftwo_thenstop\romannumeral0\xintiidivision }%
1324 \def\xintiQuo {\romannumeral0\xintiquo }%
1325 \def\xintiRem {\romannumeral0\xintirem }%
1326 \def\xintiquo {\expandafter\xint_firstoftwo_thenstop\romannumeral0\xintidivision }%
1327 \def\xintirem {\expandafter\xint_secondoftwo_thenstop\romannumeral0\xintidivision }%
1328 \let\xintQuo\xintiQuo\let\xintquo\xintiquo % deprecated
1329 \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: $A=BQ+R$, $0 \leq R < |B|$.

```

1330 \def\xintiDivision {\romannumeral0\xintidivision }%
1331 \def\xintidivision #1{\expandafter\XINT_idivision\romannumeral0\xintnum{#1}\Z }%
1332 \def\XINT_idivision #1#2\Z #3{\expandafter\XINT_iidivision_a\expandafter #1%
1333     \romannumeral0\xintnum{#3}\Z #2\Z }%

```

3 Package *xintcore* implementation

```

1334 \def\xintiDivision    {\romannumeral0\xintiDivision }%
1335 \def\xintiDivision    #1{\expandafter\XINT_iidivision \romannumeral`&&@#1\Z }%
1336 \def\XINT_iidivision #1#2\Z #3{\expandafter\XINT_iidivision_a\expandafter #1%
1337     \romannumeral`&&@#3\Z #2\Z }%
1338 \def\XINT_iidivision_a #1#2% #1 de A, #2 de B.
1339 {%
1340     \if0#2\xint_dothis\XINT_iidivision_divbyzero\fi
1341     \if0#1\xint_dothis\XINT_iidivision_aiszero\fi
1342     \if-#2\xint_dothis{\expandafter\XINT_iidivision_bneg
1343         \romannumeral0\XINT_iidivision_bpos #1}\fi
1344     \xint_orthat{\XINT_iidivision_bpos #1#2}%
1345 }%
1346 \def\XINT_iidivision_divbyzero #1\Z #2\Z {\xintError:DivisionByZero{0}{0}}%
1347 \def\XINT_iidivision_aiszero #1\Z #2\Z {{0}{0}}%
1348 \def\XINT_iidivision_bneg #1% q->-q, r unchanged
1349     {\expandafter{\romannumeral0\XINT_opp #1}}%
1350 \def\XINT_iidivision_bpos #1%
1351 {%
1352     \xint_UDsignfork
1353         #1\XINT_iidivision_aneg
1354         -{\XINT_iidivision_apos #1}%
1355     \krof
1356 }%
1357 \def\XINT_iidivision_apos #1#2\Z #3\Z{\XINT_div_prepare {#2}{#1#3}}%
1358 \def\XINT_iidivision_aneg #1\Z #2\Z
1359     {\expandafter
1360         \XINT_iidivision_aneg_b\romannumeral0\XINT_div_prepare {#1}{#2}{#1}}%
1361 \def\XINT_iidivision_aneg_b #1#2{\if0\XINT_Sgn #2\Z
1362     \expandafter\XINT_iidivision_aneg_rzero
1363     \else
1364     \expandafter\XINT_iidivision_aneg_rpos
1365     \fi {#1}{#2}}%
1366 \def\XINT_iidivision_aneg_rzero #1#2#3{{-#1}{0}}% necessarily q was >0
1367 \def\XINT_iidivision_aneg_rpos #1%
1368 {%
1369     \expandafter\XINT_iidivision_aneg_end\expandafter
1370     {\expandafter-\romannumeral0\xintinc {#1}}% q-> -(1+q)
1371 }%
1372 \def\XINT_iidivision_aneg_end #1#2#3%
1373 {%
1374     \expandafter\xint_exchangetwo_keepbraces
1375     \expandafter{\romannumeral0\XINT_sub_mm_a {}{}#3\Z #2\Z}{#1}% r-> b-r
1376 }%
1377 %%%%%%%%%%%%%%
1378 \def\XINT_div_prepare #1%
1379 {%
1380     \XINT_div_prepare_a #1\R\R\R\R\R\R\R {10}0000001\W !{#1}%
1381 }%
1382 \def\XINT_div_prepare_a #1#2#3#4#5#6#7#8#9%
1383 {%
1384     \xint_gob_til_R #9\XINT_div_prepare_small\R
1385     \XINT_div_prepare_b #9%

```

3 Package *xintcore* implementation

```

1386 }%
1387 %%%%%%%%%%
1388 \def\XINT_div_prepare_small\R #1!#2%
1389 {%
1390     \ifcase #2
1391     \or\expandafter\XINT_div_BisOne
1392     \or\expandafter\XINT_div_BisTwo
1393     \else\expandafter\XINT_div_small_a
1394     \fi {#2}%
1395 }%
1396 \def\XINT_div_BisOne #1#2{{#2}{0}}%
1397 \def\XINT_div_BisTwo #1#2%
1398 {%
1399     \expandafter\expandafter\expandafter\XINT_div_BisTwo_a
1400     \ifodd\xintLDg{#2} \expandafter1\else \expandafter0\fi {#2}%
1401 }%
1402 \def\XINT_div_BisTwo_a #1#2%
1403 {%
1404     \expandafter{\romannumeral0\xinthalft {#2}}{#1}%
1405 }%
1406 \def\XINT_div_small_a #1#2%
1407 {%
1408     \expandafter\XINT_div_small_b
1409     \the\numexpr #1/\xint_c_ii\expandafter
1410     .\the\numexpr \xint_c_x^viii+#1\expandafter!%
1411     \romannumeral0%
1412     \XINT_div_small_ba #2\R\R\R\R\R\R\R{10}0000001\W
1413     #2\XINT_sepbyviii_Z_end 2345678\relax
1414 }%
1415 \def\XINT_div_small_b #1!#2{#2#1!}%
1416 \def\XINT_div_small_ba #1#2#3#4#5#6#7#8#9%
1417 {%
1418     \xint_gob_til_R #9\XINT_div_smallsmall\R
1419     \expandafter\XINT_div_dosmallldiv
1420     \the\numexpr\expandafter\XINT_sepbyviii_Z
1421     \romannumeral0\XINT_zeroes_forviii
1422     #1#2#3#4#5#6#7#8#9%
1423 }%
1424 \def\XINT_div_smallsmall\R
1425     \expandafter\XINT_div_dosmallldiv
1426     \the\numexpr\expandafter\XINT_sepbyviii_Z
1427     \romannumeral0\XINT_zeroes_forviii #1\R #2\relax
1428     {{\XINT_div_dosmallsmall}{#1}}%
1429 \def\XINT_div_dosmallsmall #1.1#2!#3%
1430 {%
1431     \expandafter\XINT_div_smallsmallend
1432     \the\numexpr (#3+#1)/#2-\xint_c_i.#2.#3.%
1433 }%
1434 \def\XINT_div_smallsmallend #1.#2.#3.{\expandafter
1435     {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #3-#1*#2}}%
1436 \def\XINT_div_dosmallldiv
1437     {{\expandafter\XINT_sdiv_out\the\numexpr\XINT_smallldivx_a}}%

```

3 Package *xintcore* implementation

```

1438 %%%%%%%%%%
1439 \def\XINT_div_prepare_b
1440   {\expandafter\XINT_div_prepare_c\romannumeral0\XINT_zeroes_forviii }%
1441 \def\XINT_div_prepare_c #1!%
1442 {%
1443   \XINT_div_prepare_d #1.00000000!{#1}%
1444 }%
1445 \def\XINT_div_prepare_d #1#2#3#4#5#6#7#8#9%
1446 {%
1447   \expandafter\XINT_div_prepare_e\xint_gob_til_dot #1#2#3#4#5#6#7#8#9!%
1448 }%
1449 \def\XINT_div_prepare_e #1!#2!#3#4%
1450 {%
1451   \XINT_div_prepare_f #4#3\X {#1}{#3}%
1452 }%

```

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 v1.2. C'est le problème lorsqu'au lieu de tout refaire à partir de zéro on recycle d'anciennes routines qui avaient un contexte différent.

```

1453 \def\XINT_div_prepare_f #1#2#3#4#5#6#7#8#9\X
1454 {%
1455   \expandafter\XINT_div_prepare_g
1456   \the\numexpr #1#2#3#4#5#6#7#8+\xint_c_i\expandafter
1457   .\the\numexpr (#1#2#3#4#5#6#7#8+\xint_c_i)/\xint_c_ii\expandafter
1458   .\the\numexpr #1#2#3#4#5#6#7#8\expandafter
1459   .\romannumeral0\XINT_sepandrev_andcount
1460   #1#2#3#4#5#6#7#8#9\XINT_rsepybviii_end_A 2345678%
1461   \XINT_rsepybviii_end_B 2345678%
1462   \relax\xint_c_ii\xint_c_iii
1463   \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
1464   \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
1465   \X
1466 }%
1467 \def\XINT_div_prepare_g #1.#2.#3.#4.#5\X #6#7#8%
1468 {%
1469   \expandafter\XINT_div_prepare_h
1470   \the\numexpr\expandafter\XINT_sepbyviii_andcount
1471   \romannumeral0\XINT_zeroes_forviii #8#7\R\R\R\R\R\R\R\{10}0000001\W
1472   #8#7\XINT_sepbyviii_end 2345678\relax
1473   \xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!%
1474   \xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_\W
1475   {#1}{#2}{#3}{#4}{#5}{#6}%
1476 }%
1477 \def\XINT_div_prepare_h #11.#2.#3#4#5#6#7#8%
1478 {%
1479   \XINT_div_start_a {#2}{#6}{#1}{#3}{#4}{#5}{#7}{#8}%
1480 }%

```

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

3 Package *xintcore* implementation

```

1481 \def\XINT_div_start_a #1#2%
1482 {%
1483   \ifnum #1 < #2
1484     \expandafter\XINT_div_zeroQ
1485   \else
1486     \expandafter\XINT_div_start_b
1487   \fi
1488   {#1}{#2}%
1489 }%
1490 \def\XINT_div_zeroQ #1#2#3#4#5#6#7%
1491 {%
1492   \expandafter\XINT_div_zeroQ_end
1493   \romannumeral0\XINT_unsep_cuzsmall
1494   #31\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W .%
1495 }%
1496 \def\XINT_div_zeroQ_end #1.#2%
1497   {\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»

```

1498 \def\XINT_div_start_b #1#2#3#4#5#6%
1499 {%
1500   \expandafter\XINT_div_finish\the\numexpr
1501   \XINT_div_start_c {#2}.#3.{#6}{#1}{#2}{#4}{#5}{#6}%
1502 }%
1503 \def\XINT_div_finish
1504 {%
1505   \expandafter\XINT_div_finish_a \romannumeral`&&\XINT_div_unsepQ
1506 }%
1507 \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».

```

1508 \def\XINT_div_finish_b #1%
1509 {%
1510   \if0#1%
1511     \expandafter\XINT_div_finish_bRzero
1512   \else
1513     \expandafter\XINT_div_finish_bRpos
1514   \fi
1515   #1%
1516 }%
1517 \def\XINT_div_finish_bRzero 0.#1#2{{#1}{0}}%
1518 \def\XINT_div_finish_bRpos #1.#2#3%
1519 {%
1520   \expandafter\xint_exchangetwo_keepbraces\XINT_div_cleanR #1#3.{#2}%
1521 }%
1522 \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.

```

1523 \def\XINT_div_start_c #1%
1524 {%

```


3 Package *xintcore* implementation

```

1525 \ifnum #1>\xint_c_vi
1526 \expandafter\XINT_div_start_ca
1527 \else
1528 \expandafter\XINT_div_start_cb
1529 \fi {#1}%
1530 }%
1531 \def\XINT_div_start_ca #1#2.#3!#4!#5!#6!#7!#8!#9!%
1532 {%
1533 \expandafter\XINT_div_start_c\expandafter
1534 {\the\numexpr #1-\xint_c_vii}#2#3!#4!#5!#6!#7!#8!#9!.%
1535 }%
1536 \def\XINT_div_start_cb #1%
1537 {\csname XINT_div_start_c_\romannumeral\numexpr#1\endcsname}%
1538 \def\XINT_div_start_c_i #1.#2!%
1539 {\XINT_div_start_c_ #1#2!.%}%
1540 \def\XINT_div_start_c_ii #1.#2!#3!%
1541 {\XINT_div_start_c_ #1#2!#3!.%}%
1542 \def\XINT_div_start_c_iii #1.#2!#3!#4!%
1543 {\XINT_div_start_c_ #1#2!#3!#4!.%}%
1544 \def\XINT_div_start_c_iv #1.#2!#3!#4!#5!%
1545 {\XINT_div_start_c_ #1#2!#3!#4!#5!.%}%
1546 \def\XINT_div_start_c_v #1.#2!#3!#4!#5!#6!%
1547 {\XINT_div_start_c_ #1#2!#3!#4!#5!#6!.%}%
1548 \def\XINT_div_start_c_vi #1.#2!#3!#4!#5!#6!#7!%
1549 {\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».

```

1550 \def\XINT_div_start_c_ 1#1!#2.#3.#4#5#6%
1551 {%
1552 \XINT_div_I_a {#1}{#4}{1#1!#2}{#6}{00000000}#5{#3}{#6}%
1553 }%

```

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

```

1554 \def\XINT_div_I_a #1#2%
1555 {%
1556 \expandafter\XINT_div_I_b\the\numexpr #1/#2.{#1}{#2}%
1557 }%
1558 \def\XINT_div_I_b #1%
1559 {%
1560 \xint_gob_til_zero #1\XINT_div_I_czero 0\XINT_div_I_c #1%
1561 }%

```

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».

```

1562 \def\XINT_div_I_czero 0\XINT_div_I_c 0.#1#2#3#4#5{1#5\XINT_div_I_g {#3}}%
1563 \def\XINT_div_I_c #1.#2#3%
1564 {%
1565 \expandafter\XINT_div_I_da\the\numexpr #2-#1*#3.#1.{#2}{#3}%
1566 }%

```

3 Package *xintcore* implementation

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

```

1567 \def\XINT_div_I_da #1.%
1568 {%
1569     \ifnum #1>\xint_c_ix
1570         \expandafter\XINT_div_I_dP
1571     \else
1572         \ifnum #1<\xint_c_
1573             \expandafter\expandafter\expandafter\XINT_div_I_dN
1574         \else
1575             \expandafter\expandafter\expandafter\XINT_div_I_db
1576         \fi
1577     \fi
1578 }%
```

attention très mauvaises notations avec _b et _db.

```

1579 \def\XINT_div_I_dN #1.%
1580 {%
1581     \expandafter\XINT_div_I_b\the\numexpr #1-\xint_c_i.%
1582 }%
1583 \def\XINT_div_I_db #1.#2#3#4#5%
1584 {%
1585     \expandafter\XINT_div_I_dc\expandafter #1%
1586     \romannumeral0\expandafter\XINT_div_sub\expandafter
1587     {\romannumeral0\XINT_rev_nounsep }{#4\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1588     {\the\numexpr\XINT_div_verysmallmul #1!#51\Z!}%
1589     \Z {#4}{#5}%
1590 }%
```

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

```

1591 \def\XINT_div_I_dc #1#2%
1592 {%
1593     \if-#2\expandafter\XINT_div_I_dd\else\expandafter\XINT_div_I_de\fi
1594     #1#2%
1595 }%
1596 \def\XINT_div_I_dd #1-\Z
1597 {%
1598     \if #11\expandafter\XINT_div_I_dz\fi
1599     \expandafter\XINT_div_I_dP\the\numexpr #1-\xint_c_i.XX%
1600 }%
1601 \def\XINT_div_I_dz #1XX#2#3#4%
1602 {%
1603     1#4\XINT_div_I_g {#2}%
1604 }%
1605 \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»

```

1606 \def\XINT_div_I_dP #1.#2#3#4#5#6%
1607 {%
```

3 Package *xintcore* implementation

```

1608 1#6+1\expandafter\XINT_div_I_g\expandafter
1609 {\romannumeral0\expandafter\XINT_div_sub\expandafter
1610   {\romannumeral0\XINT_rev_nounsep {}#4\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1611   {\the\numexpr\XINT_div_verysmallmul #1!#51\Z!}%
1612 }%
1613 }%

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»

1614 \def\XINT_div_I_g #1#2#3#4#5#6#7%
1615 {%
1616   \expandafter !\the\numexpr
1617   \ifnum#2=#3
1618     \expandafter\XINT_div_exittofinish
1619   \else
1620     \expandafter\XINT_div_I_h
1621   \fi
1622   {#4}#1.#6.{#4}{#5}{#3}{#2}}{#7}%
1623 }%

{x'y}alpha.alpha'.{\x'y}xKL}B«c» -> Attention retour à l'expéditeur 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»

1624 \def\XINT_div_exittofinish #1#2.#3.#4#5%
1625 {%
1626   1\expandafter\expandafter\expandafter!\expandafter\XINT_unsep_delim
1627   \romannumeral0\XINT_div_unsepR #2#31\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W.%
1628 }%

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

1629 \def\XINT_div_I_h #1.#2!#3.#4#5%
1630 {%
1631   \XINT_div_II_b #1#2!.{#5}{#4}{#3}{#5}%
1632 }%

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

1633 \def\XINT_div_II_b #1#2!#3!%
1634 {%
1635   \xint_gob_til_eightzeroes #2\XINT_div_II_skipc 00000000%
1636   \XINT_div_II_c #1{1#2}{#3}%
1637 }%

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 dimin-
uer de 4), {alpha sur K}B{q1=00000000}{alpha'}B,«c»

1638 \def\XINT_div_II_skipc 00000000\XINT_div_II_c #1#2#3#4#5.#6#7%
1639 {%
1640   \XINT_div_II_k #7{#4!#5}{#6}{00000000}%
1641 }%

```

3 Package *xintcore* implementation

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

ATTENTION!

2015/10/29 : j'avais introduit un bug ici dans v1.2 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.

```
1642 \def\XINT_div_II_c #1#2#3#4%
1643 {%
1644     \expandafter\XINT_div_II_d\the\numexpr\XINT_div_xmini
1645     #1.#2!#3!#4!\{#1\}\{#2\}\{#3\}\{#4\}%
1646 }%
1647 \def\XINT_div_xmini #1%
1648 {%
1649     \xint_gob_til_one #1\XINT_div_xmini_a 1\XINT_div_mini #1%
1650 }%
1651 \def\XINT_div_xmini_a 1\XINT_div_mini 1#1%
1652 {%
1653     \xint_gob_til_zero #1\XINT_div_xmini_b 0\XINT_div_mini 1#1%
1654 }%
1655 \def\XINT_div_xmini_b 0\XINT_div_mini 10#1#2#3#4#5#6#7%
1656 {%
1657     \xint_gob_til_zero #7\XINT_div_xmini_c 0\XINT_div_mini 10#1#2#3#4#5#6#7%
1658 }%
```

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

```
1659 \def\XINT_div_xmini_c 0\XINT_div_mini 1000000000.500000000!#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»

```
1660 \def\XINT_div_II_d 1#1#2#3#4#5!#6#7#8.#9%
1661 {%
1662     \expandafter\XINT_div_II_e
1663     \romannumeral0\expandafter\XINT_div_sub\expandafter
1664     {\romannumeral0\XINT_rev_nounsep {\#8\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1665     {\the\numexpr\XINT_div_smallmul_a 1000000000.#1#2#3#4.#5!#91\Z!}%
1666     .{\#6\}\{\#7\}\{\#9\}\{#1#2#3#4#5\}%
1667 }%
```

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

```
1668 \def\XINT_div_II_e 1#1!%
1669 {%
1670     \xint_gob_til_eightzeroes #1\XINT_div_II_skipf 000000000%
1671     \XINT_div_II_f 1#1!%
1672 }%
```

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

3 Package *xintcore* implementation

```

1673 \def\XINT_div_II_skipf 00000000\XINT_div_II_f 100000000!#1.#2#3#4#5#6%
1674 {%
1675   \XINT_div_II_k #6{#1}{#4}{#5}%
1676 }%

```

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.

```

1677 \def\XINT_div_II_f #1!#2!#3.%
1678 {%
1679   \XINT_div_II_fa {#1!#2!}{#1!#2!#3}%
1680 }%
1681 \def\XINT_div_II_fa #1#2#3#4%
1682 {%
1683   \expandafter\XINT_div_II_g \the\numexpr\XINT_div_xmini #3.#4!#1{#2}%
1684 }%

```

#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>

```

1685 \def\XINT_div_II_g 1#1#2#3#4#5!#6#7#8%
1686 {%
1687   \expandafter \XINT_div_II_h
1688   \the\numexpr 1#1#2#3#4#5+#8\expandafter\expandafter\expandafter
1689   .\expandafter\expandafter\expandafter
1690   {\expandafter\xint_gob_til_exclam
1691    \romannumeral0\expandafter\XINT_div_sub\expandafter
1692    {\romannumeral0\XINT_rev_nounsep }#6\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1693    {\the\numexpr\XINT_div_smallmul_a 100000000.#1#2#3#4.#5!#71\Z!}%
1694    {#7}%
1695 }%

```

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>

```

1696 \def\XINT_div_II_h 1#1.#2#3#4%
1697 {%
1698   \XINT_div_II_k #4{#2}{#3}{#1}%
1699 }%

```

{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>

```

1700 \def\XINT_div_II_k #1#2#3#4#5%
1701 {%
1702   \expandafter\XINT_div_II_l \the\numexpr #4-\xint_c_i.{#3}#1{#2}#5.%
1703 }%
1704 \def\XINT_div_II_l #1.#2#3#4#51#6!%
1705 {%
1706   \XINT_div_II_m {{#1}{#2}{{#3}{#4}{{#5}{{#5}{{#6}1#6!%
1707 }%

```

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

3 Package *xintcore* implementation

```
1708 \def\XINT_div_II_m #1#2#3#4.#5#6%
1709 {%
1710     \XINT_div_I_a {#3}{#2}{#4}{#5}{#6}#1%
1711 }%
```

This multiplication is exactly like `\XINT_smallmul`, but it always keeps the ending carry. For optimization I duplicated the whole code.

```
1712 \def\XINT_div_minimulwc_a 1#1.#2.#3!#4#5#6#7#8.%
1713 {%
1714     \expandafter\XINT_div_minimulwc_b
1715     \the\numexpr \xint_c_x^ix+#1+#3*#8.#3*#4#5#6#7+#2*#8.#2*#4#5#6#7.%
1716 }%
1717 \def\XINT_div_minimulwc_b 1#1#2#3#4#5#6.#7.%
1718 {%
1719     \expandafter\XINT_div_minimulwc_c
1720     \the\numexpr \xint_c_x^ix+#1#2#3#4#5+#7.#6.%
1721 }%
1722 \def\XINT_div_minimulwc_c 1#1#2#3#4#5#6.#7.#8.%
1723 {%
1724     1#6#7\expandafter!%
1725     \the\numexpr\expandafter\XINT_div_smallmul_a
1726     \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8.%
1727 }%
1728 \def\XINT_div_smallmul_a #1.#2.#3!1#4!%
1729 {%
1730     \xint_gob_til_Z #4\XINT_div_smallmul_e\Z
1731     \XINT_div_minimulwc_a #1.#2.#3!#4.#2.#3!%
1732 }%
1733 \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 `#1=1` let's not forget to add the 100000000! at the end.

```
1734 \def\XINT_div_verysmallmul #1%
1735     {\xint_gob_til_one #1\XINT_div_verysmallisone 1\XINT_div_verysmallmul_a 0.#1}%
1736 \def\XINT_div_verysmallisone 1\XINT_div_verysmallmul_a 0.1!1#11\Z!%
1737     {1\relax #1100000000!}%
1738 \def\XINT_div_verysmallmul_a #1.#2!1#3!%
1739 {%
1740     \xint_gob_til_Z #3\XINT_div_verysmallmul_e\Z
1741     \expandafter\XINT_div_verysmallmul_b
1742     \the\numexpr \xint_c_x^ix+#2*#3+#1.#2!%
1743 }%
1744 \def\XINT_div_verysmallmul_b 1#1#2.%
1745     {1#2\expandafter!\the\numexpr\XINT_div_verysmallmul_a #1.%}
1746 \def\XINT_div_verysmallmul_e\Z #1\Z +#2#3!{1\relax 0000000#2!}%

```

Special subtraction for division purposes.

```
1747 \def\XINT_div_sub #1#2%
1748 {%
1749     \expandafter\XINT_div_sub_clean

```

3 Package *xintcore* implementation

```
1750 \the\numexpr\expandafter\XINT_div_sub_a\expandafter
1751 1#2\Z!\Z!\Z!\Z!\Z!\W #1\Z!\Z!\Z!\Z!\Z!\W
1752 }%
1753 \def\XINT_div_sub_clean #1-#2#3\W
1754 {%
1755 \if1#2\expandafter\XINT_rev_nounsep\else\expandafter\XINT_div_sub_neg\fi
1756 {}#1\R!\R!\R!\R!\R!\R!\R!\R!\W
1757 }%
1758 \def\XINT_div_sub_neg #1\W { -}%
1759 \def\XINT_div_sub_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
1760 {%
1761 \XINT_div_sub_b #1!#6!#2!#7!#3!#8!#4!#9!#5\W
1762 }%
1763 \def\XINT_div_sub_b #1#2#3!#4!%
1764 {%
1765 \xint_gob_til_Z #4\XINT_div_sub_bi \Z
1766 \expandafter\XINT_div_sub_c\the\numexpr#1-#3+1#4-\xint_c_i.%
1767 }%
1768 \def\XINT_div_sub_c 1#1#2.%
1769 {%
1770 1#2\expandafter!\the\numexpr\XINT_div_sub_d #1%
1771 }%
1772 \def\XINT_div_sub_d #1#2#3!#4!%
1773 {%
1774 \xint_gob_til_Z #4\XINT_div_sub_di \Z
1775 \expandafter\XINT_div_sub_e\the\numexpr#1-#3+1#4-\xint_c_i.%
1776 }%
1777 \def\XINT_div_sub_e 1#1#2.%
1778 {%
1779 1#2\expandafter!\the\numexpr\XINT_div_sub_f #1%
1780 }%
1781 \def\XINT_div_sub_f #1#2#3!#4!%
1782 {%
1783 \xint_gob_til_Z #4\XINT_div_sub_fi \Z
1784 \expandafter\XINT_div_sub_g\the\numexpr#1-#3+1#4-\xint_c_i.%
1785 }%
1786 \def\XINT_div_sub_g 1#1#2.%
1787 {%
1788 1#2\expandafter!\the\numexpr\XINT_div_sub_h #1%
1789 }%
1790 \def\XINT_div_sub_h #1#2#3!#4!%
1791 {%
1792 \xint_gob_til_Z #4\XINT_div_sub_hi \Z
1793 \expandafter\XINT_div_sub_i\the\numexpr#1-#3+1#4-\xint_c_i.%
1794 }%
1795 \def\XINT_div_sub_i 1#1#2.%
1796 {%
1797 1#2\expandafter!\the\numexpr\XINT_div_sub_a #1%
1798 }%
1799 \def\XINT_div_sub_bi\Z
1800 \expandafter\XINT_div_sub_c\the\numexpr#1-#2+#3.#4!#5!#6!#7!#8!#9!\Z !\W
1801 }
```

3 Package `xintcore` implementation

```

1802 \XINT_div_sub_l #1#2!#5!#7!#9!%
1803 }%
1804 \def\XINT_div_sub_di\Z
1805 \expandafter\XINT_div_sub_e\the\numexpr#1-#2+#3.#4!#5!#6!#7!#8\W
1806 {%
1807 \XINT_div_sub_l #1#2!#5!#7!%
1808 }%
1809 \def\XINT_div_sub_fi\Z
1810 \expandafter\XINT_div_sub_g\the\numexpr#1-#2+#3.#4!#5!#6\W
1811 {%
1812 \XINT_div_sub_l #1#2!#5!%
1813 }%
1814 \def\XINT_div_sub_hi\Z
1815 \expandafter\XINT_div_sub_i\the\numexpr#1-#2+#3.#4\W
1816 {%
1817 \XINT_div_sub_l #1#2!%
1818 }%
1819 \def\XINT_div_sub_l #1%
1820 {%
1821 \xint_UDzerofork
1822 #1{-2\relax}%
1823 0\XINT_div_sub_r
1824 \krof
1825 }%
1826 \def\XINT_div_sub_r #1!%
1827 {%
1828 -\ifnum 0#1=\xint_c_ 1\else2\fi\relax
1829 }%
1830 %%%%%%%%%%
1831 \def\XINT_sdiv_out #1\Z #2\W%
1832 {\expandafter
1833 {\romannumeral0\XINT_unsep_cuzsmall#11\R!1\R!1\R!1\R!1\R!1\R!1\R!1\W}%
1834 {#2}}%
1835 \def\XINT_smallldivx_a #1.1#2!1#3!%
1836 {%
1837 \expandafter\XINT_smallldivx_b
1838 \the\numexpr (#3+#1)/#2-\xint_c_i!#1.#2!#3!%
1839 }%
1840 \def\XINT_smallldivx_b #1!%
1841 {%
1842 \if0#1\else
1843 \xint_c_x^viii+#1\xint_afterfi{\expandafter!\the\numexpr}\fi
1844 \XINT_smallldiv_c #1!%
1845 }%
1846 \def\XINT_smallldiv_c #1!#2.#3!#4!%
1847 {%
1848 \expandafter\XINT_smallldiv_d\the\numexpr #4-#1*#3!#2.#3!%
1849 }%
1850 \def\XINT_smallldiv_d #1!#2!#3#4!%
1851 {%
1852 \xint_gob_til_Z #4\XINT_smallldiv_end \Z
1853 \XINT_smallldiv_e #1!#2!#3#4!%

```


3 Package *xintcore* implementation

```

1854 }%
1855 \def\XINT_smallldiv_end\Z\XINT_smallldiv_e #1!#2!1\Z!{1!\Z #1\W }%
1856 \def\XINT_smallldiv_e #1!#2.#3!%
1857 {%
1858     \expandafter\XINT_smallldiv_f\the\numexpr
1859     \xint_c_xi_e_viii_mone+#1*\xint_c_x^viii/#3!#2.#3!#1!%
1860 }%
1861 \def\XINT_smallldiv_f 1#1#2#3#4#5#6!#7.#8!%
1862 {%
1863     \xint_gob_til_zero #1\XINT_smallldiv_fz 0%
1864     \expandafter\XINT_smallldiv_g
1865     \the\numexpr\XINT_minimul_a #2#3#4#5.#6!#8!#2#3#4#5#6!#7.#8!%
1866 }%
1867 \def\XINT_smallldiv_fz 0%
1868     \expandafter\XINT_smallldiv_g\the\numexpr\XINT_minimul_a
1869     9999.9999!#1!99999999!#2!0!1#3!%
1870 {%
1871     \XINT_smallldiv_i .#3!\xint_c_!#2!%
1872 }%
1873 \def\XINT_smallldiv_g 1#1!1#2!#3!#4!#5!#6!%
1874 {%
1875     \expandafter\XINT_smallldiv_h
1876     \the\numexpr 1#6-#1.#2!#5!#3!#4!%
1877 }%
1878 \def\XINT_smallldiv_h 1#1#2.#3!#4!%
1879 {%
1880     \expandafter\XINT_smallldiv_i
1881     \the\numexpr #4-#3+#1-\xint_c_i.#2!%
1882 }%
1883 \def\XINT_smallldiv_i #1.#2!#3!#4.#5!%
1884 {%
1885     \expandafter\XINT_smallldiv_j
1886     \the\numexpr (#1#2+#4)/#5-\xint_c_i!#3!#1#2!#4.#5!%
1887 }%
1888 \def\XINT_smallldiv_j #1!#2!%
1889 {%
1890     \xint_c_x^viii+#1+#2\expandafter!\the\numexpr\XINT_smallldiv_k
1891     #1!%
1892 }%
1893 \def\XINT_smallldiv_k #1!#2!#3.#4!%
1894 {%
1895     \expandafter\XINT_smallldiv_d\the\numexpr #2-#1*#4!#3.#4!%
1896 }%

```

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 (ARRRRRRGGGGHHHH) fait par `/`. Le nombre divisé $XY = X \cdot 10^8 + Y$ se présente sous la forme `1<8chiffres>!1<8chiffres>!` avec plus significatif en premier.

ATTENTION UNIQUEMENT UTILISÉ POUR DES SITUATIONS OÙ IL EST GARANTI QUE $X < C$!! le quotient euclidien de $X \cdot 10^8 + Y$ par C sera donc $< 10^8$. Il sera renvoyé sous la forme `1<8chiffres>`.

```

1897 \def\XINT_div_mini #1.#2!1#3!%
1898 {%

```

3 Package *xintcore* implementation

```
1899 \expandafter\XINT_div_mini_a\the\numexpr
1900 \xint_c_xi_e_viii_mone+#3*\xint_c_x^viii/#1!#1.#2!#3!%
1901 }%
```

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.

```
1902 \def\XINT_div_mini_a 1#1#2#3#4#5#6!#7.#8!%
1903 {%
1904 \xint_gob_til_zero #1\XINT_div_mini_w 0%
1905 \expandafter\XINT_div_mini_b
1906 \the\numexpr\XINT_minimul_a #2#3#4#5.#6!#7!#2#3#4#5#6!#7.#8!%
1907 }%
1908 \def\XINT_div_mini_w 0%
1909 \expandafter\XINT_div_mini_b\the\numexpr\XINT_minimul_a
1910 9999.9999!#1!99999999!#2.#3!00000000!#4!%
1911 {%
1912 \xint_c_x^viii_mone+(#4+#3)/#2!%
1913 }%
1914 \def\XINT_div_mini_b 1#1!1#2!#3!#4!#5!#6!%
1915 {%
1916 \expandafter\XINT_div_mini_c
1917 \the\numexpr 1#6-#1.#2!#5!#3!#4!%
1918 }%
1919 \def\XINT_div_mini_c 1#1#2.#3!#4!%
1920 {%
1921 \expandafter\XINT_div_mini_d
1922 \the\numexpr #4-#3+#1-\xint_c_i.#2!%
1923 }%
1924 \def\XINT_div_mini_d 1.#2!#3!#4.#5!%
1925 {%
1926 \xint_c_x^viii_mone+#3+(#1#2+#5)/#4!%
1927 }%
```

3.26 `\xintiDivRound`, `\xintiiDivRound`

v1.1, transferred from first release of `bnumexpr`. Rewritten for v1.2.

```
1928 \def\xintiDivRound {\romannumeral0\xintidivround }%
1929 \def\xintidivround #1%
1930 {\expandafter\XINT_idivround\romannumeral0\xintnum{#1}\Z }%
1931 \def\xintiiDivRound {\romannumeral0\xintiidivround }%
1932 \def\xintiidivround #1{\expandafter\XINT_iidivround \romannumeral`&&@#1\Z }%
1933 \def\XINT_idivround #1#2\Z #3%
1934 {\expandafter\XINT_iidivround_a\expandafter #1%
1935 \romannumeral0\xintnum{#3}\Z #2\Z }%
1936 \def\XINT_iidivround #1#2\Z #3%
1937 {\expandafter\XINT_iidivround_a\expandafter #1\romannumeral`&&@#3\Z #2\Z }%
1938 \def\XINT_iidivround_a #1#2# #1 de A, #2 de B.
1939 {%
1940 \if0#2\xint_dothis\XINT_iidivround_divbyzero\fi
1941 \if0#1\xint_dothis\XINT_iidivround_aiszero\fi
1942 \if-#2\xint_dothis{\XINT_iidivround_bneg #1}\fi
```

3 Package *xintcore* implementation

```

1943      \xint_orthat{\XINT_iidivround_bpos #1#2}%
1944 }%
1945 \def\XINT_iidivround_divbyzero #1\Z #2\Z {\xintError:DivisionByZero\space 0}%
1946 \def\XINT_iidivround_aiszero #1\Z #2\Z { 0}%
1947 \def\XINT_iidivround_bpos #1%
1948 {%
1949   \xint_UDsignfork
1950     #1{\xintiiopp\XINT_iidivround_pos {}}%
1951     -{\XINT_iidivround_pos #1}%
1952   \krof
1953 }%
1954 \def\XINT_iidivround_bneg #1%
1955 {%
1956   \xint_UDsignfork
1957     #1{\XINT_iidivround_pos {}}%
1958     -{\xintiiopp\XINT_iidivround_pos #1}%
1959   \krof
1960 }%
1961 \def\XINT_iidivround_pos #1#2\Z #3\Z
1962 {%
1963   \expandafter\XINT_iidivround_pos_a
1964   \romannumeral0\XINT_div_prepare {#2}{#1#30}%
1965 }%

```

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.

```

1966 \def\XINT_iidivround_pos_a #1#2%
1967 {%
1968   \expandafter\XINT_iidivround_pos_b
1969   \romannumeral0\expandafter\XINT_sepandrev
1970   \romannumeral0\XINT_zeroes_forviii #1\R\R\R\R\R\R\R\R{10}0000001\W
1971   #1\XINT_rsepbyviii_end_A 2345678\XINT_rsepbyviii_end_B 2345678\relax XX%
1972   \R.\R.\R.\R.\R.\R.\R.\R.\W
1973   1\Z!1\Z!1\Z!1\Z!\W\R
1974 }%
1975 \def\XINT_iidivround_pos_b 1#1#2#3#4#5#6#7#8!1#9%
1976 {%
1977   \xint_gob_til_Z #9\XINT_iidivround_small\Z
1978   \ifnum #8>\xint_c_iv
1979     \expandafter\XINT_iidivround_pos_up
1980   \else \expandafter\XINT_iidivround_pos_finish
1981   \fi
1982   1#1#2#3#4#5#6#70!1#9%
1983 }%
1984 \def\XINT_iidivround_pos_up
1985 {%
1986   \expandafter\XINT_iidivround_pos_finish
1987   \the\numexpr\XINT_add_a\xint_c_ii 100000010!1\Z!1\Z!1\Z!1\Z!\W
1988 }%

```

2015/11/17 Damn'ed. I added a bug in \XINT_iidivround_pos_finish when I was preparing 1.2c and was still doing modifications to the format for calling \XINT_add_a, and then I did a hurried release because I had found a bug in the subtraction from release 1.2. The 1.2c pattern #2\R was only ok

if addition had been executed else obviously we have many extra 1\Z!'s. And we have to inject an explicit 1\Z! for unrevbyviii. Sadly xint.pdf itself obviously did not have a single example of an integer division not rounded up ... Anyway, 1.2d then.

Why wasn't the 1.2c typo detected ? mainly because my main test file compared with usage of big-intcalc which has no macro for rounded division, hence \xintiiDivision was tested but not \xintiDivRound. And my newer generic test files somehow only tested \xintiiDivision also. Of course I have separate test files for \xintiDivRound, but they were used at the time of first creation of this macro. In the future I must have a test suite which will have entries for all macros and will be integrated to the last pre-build before release... the description of the macros in xint.pdf usually contain at least one example, but there are lacunae.

```

1989 \def\xINT_iidivround_pos_finish #10!#21\Z!#3\R
1990 {%
1991   \expandafter\xINT_cuz_small\romannumeral0\xINT_unrevbyviii {%
1992     #1!#21\Z!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1993 }%
1994 \def\xINT_iidivround_small\Z\ifnum #1>#2\fi 1#30!#4\W\R
1995 {%
1996   \ifnum #1>\xint_c_iv
1997     \expandafter\xINT_iidivround_small_up
1998   \else \expandafter\xINT_iidivround_small_trunc
1999   \fi {%3}%
2000 }%
2001 \edef\xINT_iidivround_small_up #1%
2002   {\noexpand\expandafter\space\noexpand\the\numexpr #1+\xint_c_i\relax }%
2003 \edef\xINT_iidivround_small_trunc #1%
2004   {\noexpand\expandafter\space\noexpand\the\numexpr #1\relax }%

```

3.27 \xintiDivTrunc, \xintiiDivTrunc

```

2005 \def\xintiDivTrunc {\romannumeral0\xintidivtrunc }%
2006 \def\xintidivtrunc #1{\expandafter\xINT_iidivtrunc\romannumeral0\xintnum{#1}\Z }%
2007 \def\xintiiDivTrunc {\romannumeral0\xintiidivtrunc }%
2008 \def\xintiidivtrunc #1{\expandafter\xINT_iidivtrunc \romannumeral`&&@#1\Z }%
2009 \def\xINT_iidivtrunc #1#2\Z #3{\expandafter\xINT_iidivtrunc_a\expandafter #1%
2010   \romannumeral`&&@#3\Z #2\Z }%
2011 \def\xINT_iidivtrunc_a #1#2% #1 de A, #2 de B.
2012 {%
2013   \if0#2\xint_dothis\xINT_iidivround_divbyzero\fi
2014   \if0#1\xint_dothis\xINT_iidivround_aiszero\fi
2015   \if-#2\xint_dothis{\XINT_iidivtrunc_bneg #1}\fi
2016   \xint_orthat{\XINT_iidivtrunc_bpos #1#2}%
2017 }%
2018 \def\xINT_iidivtrunc_bpos #1%
2019 {%
2020   \xint_UDsignfork
2021     #1{\xintiiopp\xINT_iidivtrunc_pos {}}%
2022   -{\XINT_iidivtrunc_pos #1}%
2023   \krof
2024 }%
2025 \def\xINT_iidivtrunc_bneg #1%
2026 {%
2027   \xint_UDsignfork

```

```

2028      #1{\XINT_iidivtrunc_pos {}}%
2029      -{\xintiiopp\XINT_iidivtrunc_pos #1}%
2030      \krof
2031 }%
2032 \def\XINT_iidivtrunc_pos #1#2\Z #3\Z%
2033     {\expandafter\xint_firstoftwo_thenstop
2034     \romannumeral0\XINT_div_prepare {#2}{#1#3}}%

```

3.28 `\xintiMod`, `\xintiiMod`

```

2035 \def\xintiMod      {\romannumeral0\xintimod }%
2036 \def\xintimod      #1{\expandafter\XINT_iimod\romannumeral0\xintnum{#1}\Z }%
2037 \def\xintiiMod      {\romannumeral0\xintiiimod }%
2038 \def\xintiiimod      #1{\expandafter\XINT_iimod \romannumeral`&&@#1\Z }%
2039 \def\XINT_iimod      #1#2\Z #3{\expandafter\XINT_iimod_a\expandafter #1%
2040                                \romannumeral`&&@#3\Z #2\Z }%
2041 \def\XINT_iimod_a      #1#2\Z #1 de A, #2 de B.
2042 {%
2043     \if0#2\xint_dothis\XINT_iidivround_divbyzero\fi
2044     \if0#1\xint_dothis\XINT_iidivround_aiszero\fi
2045     \if-#2\xint_dothis{\XINT_iimod_bneg #1}\fi
2046     \xint_orthat{\XINT_iimod_bpos #1#2}%
2047 }%
2048 \def\XINT_iimod_bpos #1%
2049 {%
2050     \xint_UDsignfork
2051     #1{\xintiiopp\XINT_iimod_pos {}}%
2052     -{\XINT_iimod_pos #1}%
2053     \krof
2054 }%
2055 \def\XINT_iimod_bneg #1%
2056 {%
2057     \xint_UDsignfork
2058     #1{\xintiiopp\XINT_iimod_pos {}}%
2059     -{\XINT_iimod_pos #1}%
2060     \krof
2061 }%
2062 \def\XINT_iimod_pos #1#2\Z #3\Z%
2063     {\expandafter\xint_secondoftwo_thenstop\romannumeral0\XINT_div_prepare
2064     {#2}{#1#3}}%

```

3.29 “Load *xintfrac*” macros

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

```

2065 \catcode`! 11
2066 \def\xintAbs {\Did_you_mean_iiAbs?or_load_xintfrac!}%
2067 \def\xintOpp {\Did_you_mean_iiOpp?or_load_xintfrac!}%
2068 \def\xintAdd {\Did_you_mean_iiAdd?or_load_xintfrac!}%
2069 \def\xintSub {\Did_you_mean_iiSub?or_load_xintfrac!}%
2070 \def\xintMul {\Did_you_mean_iiMul?or_load_xintfrac!}%
2071 \def\xintPow {\Did_you_mean_iiPow?or_load_xintfrac!}%
2072 \def\xintSqr {\Did_you_mean_iiSqr?or_load_xintfrac!}%
2073 \XINT_restorecatcodes_endinput%

```

4 Package `xint` implementation

.1	Package identification	87	.24	<code>\xintAND</code> , <code>\xintOR</code> , <code>\xintXOR</code>	97
.2	More token management	87	.25	<code>\xintANDof</code>	98
.3	<code>\xintSgnFork</code>	87	.26	<code>\xintORof</code>	98
.4	<code>\xintIsOne</code> , <code>\xintiiIsOne</code>	88	.27	<code>\xintXORof</code>	98
.5	<code>\xintRev</code>	88	.28	<code>\xintGeq</code> , <code>\xintiiGeq</code>	98
.6	<code>\xintLen</code>	88	.29	<code>\xintiMax</code> , <code>\xintiiMax</code>	101
.7	<code>\xintBool</code> , <code>\xintToggle</code>	89	.30	<code>\xintiMaxof</code> , <code>\xintiiMaxof</code>	103
.8	<code>\xintifSgn</code> , <code>\xintiiifSgn</code>	89	.31	<code>\xintiMin</code> , <code>\xintiiMin</code>	103
.9	<code>\xintifZero</code> , <code>\xintifNotZero</code> , <code>\xintii-</code> <code>ifZero</code> , <code>\xintiiifNotZero</code>	89	.32	<code>\xintiMinof</code> , <code>\xintiiMinof</code>	105
.10	<code>\xintifOne</code> , <code>\xintiiifOne</code>	90	.33	<code>\xintiiSum</code>	105
.11	<code>\xintifTrueAelseB</code> , <code>\xintifFalseAelseB</code>	91	.34	<code>\xintiiPrd</code>	106
.12	<code>\xintifCmp</code> , <code>\xintiiifCmp</code>	91	.35	<code>\xintMON</code> , <code>\xintMMON</code> , <code>\xintiiMON</code> , <code>\xinti-</code> <code>imMON</code>	106
.13	<code>\xintifEq</code> , <code>\xintiiifEq</code>	91	.36	<code>\xintOdd</code> , <code>\xintiiOdd</code> , <code>\xintEven</code> , <code>\xinti-</code> <code>iEven</code>	107
.14	<code>\xintifGt</code> , <code>\xintiiifGt</code>	92	.37	<code>\xintDSL</code>	108
.15	<code>\xintifLt</code> , <code>\xintiiifLt</code>	92	.38	<code>\xintDSR</code>	108
.16	<code>\xintifOdd</code> , <code>\xintiiifOdd</code>	92	.39	<code>\xintDSH</code> , <code>\xintDSHr</code>	109
.17	<code>\xintCmp</code> , <code>\xintiiCmp</code>	93	.40	<code>\xintDSx</code>	110
.18	<code>\xintEq</code> , <code>\xintGt</code> , <code>\xintLt</code>	96	.41	<code>\xintDecSplit</code> , <code>\xintDecSplitL</code> , <code>\xint-</code> <code>DecSplitR</code>	112
.19	<code>\xintNeq</code> , <code>\xintGtorEq</code> , <code>\xintLtorEq</code>	96	.42	<code>\xintiiSqrt</code> , <code>\xintiiSqrtR</code> , <code>\xinti-</code> <code>iSquareRoot</code>	116
.20	<code>\xintiiEq</code> , <code>\xintiiGt</code> , <code>\xintiiLt</code>	96	.43	<code>\xintiiE</code>	119
.21	<code>\xintiiNeq</code> , <code>\xintiiGtorEq</code> , <code>\xintiiLtorEq</code>	97	.44	“Load <code>xintfrac</code> ” macros	120
.22	<code>\xintIsZero</code> , <code>\xintIsNotZero</code> , <code>\xintii-</code> <code>IsZero</code> , <code>\xintiiIsNotZero</code>	97			
.23	<code>\xintIsTrue</code> , <code>\xintNot</code> , <code>\xintIsFalse</code>	97			

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 [2015/11/22 v1.2e 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 }%

```

New fork of 1.2 makes it less convenient here for \XINT_cmp_pre and \XINT_Cmp, which just avoided the \romannumeral-`0. Nanosecond loss ? I vaguely recalled that for \xintNewExpr things, I did need another name such as \XINT_cmp for \xintiiCmp.

```

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}%

```

4 Package *xint* implementation

```

298      0-{ -1}%
299      \krof
300 }%
301 \def\XINT_cmp_secondiszero #1\krof #20#3\Z #4\Z
302 {%
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

```

4 Package *xint* implementation

```

350     \expandafter\xint_secondoftwo
351   \fi
352   { -1}{ 1}%
353 }%
354 %%%%%%%%%%
355 \def\xint_cmp_aa {\expandafter\xint_cmp_w\the\numexpr\xint_cmp_a \xint_c_i }%
356 %%%%%%%%%%
357 \def\xint_cmp_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
358 {%
359   \xint_cmp_b #1!#6!#2!#7!#3!#8!#4!#9!#5\W
360 }%
361 \def\xint_cmp_b #1#2#3!#4!%
362 {%
363   \xint_gob_til_Z #2\xint_cmp_bi \Z
364   \expandafter\xint_cmp_c\the\numexpr#1+1#4-#3-\xint_c_i.%
365 }%
366 \def\xint_cmp_c 1#1#2.%
367 {%
368   1#2\expandafter!\the\numexpr\xint_cmp_d #1%
369 }%
370 \def\xint_cmp_d #1#2#3!#4!%
371 {%
372   \xint_gob_til_Z #2\xint_cmp_di \Z
373   \expandafter\xint_cmp_e\the\numexpr#1+1#4-#3-\xint_c_i.%
374 }%
375 \def\xint_cmp_e 1#1#2.%
376 {%
377   1#2\expandafter!\the\numexpr\xint_cmp_f #1%
378 }%
379 \def\xint_cmp_f #1#2#3!#4!%
380 {%
381   \xint_gob_til_Z #2\xint_cmp_fi \Z
382   \expandafter\xint_cmp_g\the\numexpr#1+1#4-#3-\xint_c_i.%
383 }%
384 \def\xint_cmp_g 1#1#2.%
385 {%
386   1#2\expandafter!\the\numexpr\xint_cmp_h #1%
387 }%
388 \def\xint_cmp_h #1#2#3!#4!%
389 {%
390   \xint_gob_til_Z #2\xint_cmp_hi \Z
391   \expandafter\xint_cmp_i\the\numexpr#1+1#4-#3-\xint_c_i.%
392 }%
393 \def\xint_cmp_i 1#1#2.%
394 {%
395   1#2\expandafter!\the\numexpr\xint_cmp_a #1%
396 }%
397 \def\xint_cmp_bi\Z
398   \expandafter\xint_cmp_c\the\numexpr#1+1#2-#3.#4!#5!#6!#7!#8!#9!\Z !\W
399 {%
400   \xint_cmp_k #1#2!#5!#7!#9!%
401 }%

```

4 Package `xint` implementation

```

402 \def\XINT_cmp_di\Z
403   \expandafter\XINT_cmp_e\the\numexpr#1+1#2-#3.#4!#5!#6!#7!#8\W
404 {%
405   \XINT_cmp_k #1#2!#5!#7!%
406 }%
407 \def\XINT_cmp_fi\Z
408   \expandafter\XINT_cmp_g\the\numexpr#1+1#2-#3.#4!#5!#6\W
409 {%
410   \XINT_cmp_k #1#2!#5!%
411 }%
412 \def\XINT_cmp_hi\Z
413   \expandafter\XINT_cmp_i\the\numexpr#1+1#2-#3.#4\W
414 {%
415   \XINT_cmp_k #1#2!%
416 }%
417 %%%%%%%%%%%
418 \def\XINT_cmp_k #1#2\W
419 {%
420   \xint_UDzerofork
421     #1{-1\relax \XINT_cmp_greater}%
422     0{-1\relax \XINT_cmp_lessequal}%
423   \krof
424 }%
425 \def\XINT_cmp_w #1-1#2{#2#11\Z!\W}%
426 \def\XINT_cmp_greater #1\Z!\W{ 1}%
427 \def\XINT_cmp_lessequal 1#1!%
428   {\xint_gob_til_Z #1\XINT_cmp_equal\Z
429     \xint_gob_til_eightzeroes #1\XINT_cmp_continue 00000000%
430     \XINT_cmp_less }%
431 \def\XINT_cmp_less #1\W { -1}%
432 \def\XINT_cmp_continue 00000000\XINT_cmp_less {\XINT_cmp_lessequal }%
433 \def\XINT_cmp_equal\Z\xint_gob_til_eightzeroes\Z\XINT_cmp_continue
434   00000000\XINT_cmp_less\W { 0}%

```

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

1.09a.

```

435 \def\xintEq {\romannumeral0\xinteq }\def\xinteq #1#2{\xintifeq{#1}{#2}{1}{0}}%
436 \def\xintGt {\romannumeral0\xintgt }\def\xintgt #1#2{\xintifgt{#1}{#2}{1}{0}}%
437 \def\xintLt {\romannumeral0\xintlts }\def\xintlts #1#2{\xintiflt{#1}{#2}{1}{0}}%

```

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

1.1. Pour `xintexpr`. No lowercase macros

```

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

```

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

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


```

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

```

4.21 \xintiiNeq, \xintiiGtorEq, \xintiiLtorEq

1.1a. Pour \xintiiexpr. No lowercase macros.

```

444 \def\xintiiLtorEq #1#2{\romannumeral0\xintiiifgt {#1}{#2}{0}{1}}%
445 \def\xintiiGtorEq #1#2{\romannumeral0\xintiiiflt {#1}{#2}{0}{1}}%
446 \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

```

447 \def\xintIsZero {\romannumeral0\xintiszero }%
448 \def\xintiszero #1{\if0\xintSgn{#1}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%
449 \def\xintIsNotZero {\romannumeral0\xintisnotzero }%
450 \def\xintisnotzero
451 #1{\if0\xintSgn{#1}\xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi}%
452 \def\xintiiIsZero {\romannumeral0\xintiiiszero }%
453 \def\xintiiiszero #1{\if0\xintiiSgn{#1}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%
454 \def\xintiiIsNotZero {\romannumeral0\xintiiisnotzero }%
455 \def\xintiiisnotzero
456 #1{\if0\xintiiSgn{#1}\xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi}%

```

4.23 \xintIsTrue, \xintNot, \xintIsFalse

1.09c

```

457 \let\xintIsTrue\xintIsNotZero
458 \let\xintNot\xintIsZero
459 \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

```

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

```

4.25 `\xintANDof`

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

```

471 \def\xintANDof      {\romannumeral0\xintandof }%
472 \def\xintandof      #1{\expandafter\XINT_andof_a\romannumeral`&&@#1\relax }%
473 \def\XINT_andof_a #1{\expandafter\XINT_andof_b\romannumeral`&&@#1\Z }%
474 \def\XINT_andof_b #1%
475     {\xint_gob_til_relax #1\XINT_andof_e\relax\XINT_andof_c #1}%
476 \def\XINT_andof_c #1\Z
477     {\xintifTrueAelseB {#1}{\XINT_andof_a}{\XINT_andof_no}}%
478 \def\XINT_andof_no #1\relax { 0}%
479 \def\XINT_andof_e #1\Z { 1}%

```

4.26 `\xintORof`

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

```

480 \def\xintORof      {\romannumeral0\xintorof }%
481 \def\xintorof      #1{\expandafter\XINT_orof_a\romannumeral`&&@#1\relax }%
482 \def\XINT_orof_a #1{\expandafter\XINT_orof_b\romannumeral`&&@#1\Z }%
483 \def\XINT_orof_b #1%
484     {\xint_gob_til_relax #1\XINT_orof_e\relax\XINT_orof_c #1}%
485 \def\XINT_orof_c #1\Z
486     {\xintifTrueAelseB {#1}{\XINT_orof_yes}{\XINT_orof_a}}%
487 \def\XINT_orof_yes #1\relax { 1}%
488 \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

```

489 \def\xintXORof      {\romannumeral0\xintxorof }%
490 \def\xintxorof      #1{\expandafter\XINT_xorof_a\expandafter
491     0\romannumeral`&&@#1\relax }%
492 \def\XINT_xorof_a #1#2{\expandafter\XINT_xorof_b\romannumeral`&&@#2\Z #1}%
493 \def\XINT_xorof_b #1%
494     {\xint_gob_til_relax #1\XINT_xorof_e\relax\XINT_xorof_c #1}%
495 \def\XINT_xorof_c #1\Z #2%
496     {\xintifTrueAelseB {#1}{\if #20\xint_afterfi{\XINT_xorof_a 1}%
497     \else\xint_afterfi{\XINT_xorof_a 0}\fi}%
498     {\XINT_xorof_a #2}%
499     }%
500 \def\XINT_xorof_e #1\Z #2{ #2}%

```

4.28 `\xintGeq`, `\xintiGeq`

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

```

501 \def\xintGeq      {\romannumeral0\xintgeq }%
502 \def\xintgeq      #1{\expandafter\XINT_geq\romannumeral0\xintnum{#1}\Z }%
503 \def\xintiGeq      {\romannumeral0\xintiigeq }%
504 \def\xintiigeq #1{\expandafter\XINT_iigeq\romannumeral`&&@#1\Z }%

```

4 Package `xint` implementation

```

505 \def\XINT_iigeq #1#2\Z #3%
506 {%
507   \expandafter\XINT_geq_fork\expandafter #1\romannumeral`&&@#3\Z #2\Z
508 }%
509 \let\XINT_geq_pre \xintiigeq % TEMPORAIRE
510 \let\XINT_Geq \xintGeq % TEMPORAIRE ATTENTION FAIT xintNum
511 \def\XINT_geq #1#2\Z #3%
512 {%
513   \expandafter\XINT_geq_fork\expandafter #1\romannumeral0\xintnum{#3}\Z #2\Z
514 }%
515 \def\XINT_geq_fork #1#2%
516 {%
517   \xint_UDzerofork
518   #1\XINT_geq_firstiszero
519   #2\XINT_geq_secondiszero
520   0{}}%
521   \krof
522   \xint_UDsignsfork
523   #1#2\XINT_geq_minusminus
524   #1-\XINT_geq_minusplus
525   #2-\XINT_geq_plusminus
526   --\XINT_geq_plusplus
527   \krof #1#2%
528 }%
529 \def\XINT_geq_firstiszero #1\krof 0#2#3\Z #4\Z
530   {\xint_UDzerofork #2{ 1}0{ 0}\krof }%
531 \def\XINT_geq_secondiszero #1\krof #20#3\Z #4\Z { 1}%
532 \def\XINT_geq_plusminus #1-{\XINT_geq_plusplus #1{}}%
533 \def\XINT_geq_minusplus -#1{\XINT_geq_plusplus {}}#1}%
534 \def\XINT_geq_minusminus --{\XINT_geq_plusplus {}}{}%
535 \def\XINT_geq_plusplus #1#2#3\Z #4\Z {\XINT_geq_pp #1#4\Z #2#3\Z }%
536 \def\XINT_geq_pp #1\Z
537 {%
538   \expandafter\XINT_geq_pp_a
539   \romannumeral0\expandafter\XINT_sepandrev_andcount
540   \romannumeral0\XINT_zeroes_forviii #1\R\R\R\R\R\R\R\R{10}0000001\W
541   #1\XINT_rsepybviii_end_A 2345678%
542   \XINT_rsepybviii_end_B 2345678\relax\xint_c_ii\xint_c_iii
543   \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
544   \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
545   \X
546 }%
547 \def\XINT_geq_pp_a #1.#2\X #3\Z
548 {%
549   \expandafter\XINT_geq_checklengths
550   \the\numexpr #1\expandafter.%
551   \romannumeral0\expandafter\XINT_sepandrev_andcount
552   \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
553   #3\XINT_rsepybviii_end_A 2345678%
554   \XINT_rsepybviii_end_B 2345678\relax \xint_c_ii\xint_c_iii
555   \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
556   \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W

```

4 Package *xint* implementation

```

557 \Z!\Z!\Z!\Z!\Z!\W #2\Z!\Z!\Z!\Z!\Z!\W
558 }%
559 \def\XINT_geq_checklengths #1.#2.%
560 {%
561 \ifnum #1=#2
562 \expandafter\xint_firstoftwo
563 \else
564 \expandafter\xint_secondoftwo
565 \fi
566 \XINT_geq_aa {\XINT_geq_distinctlengths {#1}{#2}}
567 }%
568 \def\XINT_geq_distinctlengths #1#2#3\W #4\W
569 {%
570 \ifnum #1>#2
571 \expandafter\xint_firstoftwo
572 \else
573 \expandafter\xint_secondoftwo
574 \fi
575 { 1}{ 0}%
576 }%
577 %%%%%%%%%%
578 \def\XINT_geq_aa {\expandafter\XINT_geq_w\the\numexpr\XINT_geq_a \xint_c_i }%
579 %%%%%%%%%%
580 \def\XINT_geq_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
581 {%
582 \XINT_geq_b #1!#6!#2!#7!#3!#8!#4!#9!#5\W
583 }%
584 \def\XINT_geq_b #1#2#3!#4!%
585 {%
586 \xint_gob_til_Z #2\XINT_geq_bi \Z
587 \expandafter\XINT_geq_c\the\numexpr#1+1#4-#3-\xint_c_i.%
588 }%
589 \def\XINT_geq_c 1#1#2.%
590 {%
591 1#2\expandafter!\the\numexpr\XINT_geq_d #1%
592 }%
593 \def\XINT_geq_d #1#2#3!#4!%
594 {%
595 \xint_gob_til_Z #2\XINT_geq_di \Z
596 \expandafter\XINT_geq_e\the\numexpr#1+1#4-#3-\xint_c_i.%
597 }%
598 \def\XINT_geq_e 1#1#2.%
599 {%
600 1#2\expandafter!\the\numexpr\XINT_geq_f #1%
601 }%
602 \def\XINT_geq_f #1#2#3!#4!%
603 {%
604 \xint_gob_til_Z #2\XINT_geq_fi \Z
605 \expandafter\XINT_geq_g\the\numexpr#1+1#4-#3-\xint_c_i.%
606 }%
607 \def\XINT_geq_g 1#1#2.%
608 {%

```

```

609 1#2\expandafter!\the\numexpr\XINT_geq_h #1%
610}%
611\def\XINT_geq_h #1#2#3!#4!%
612{%
613 \xint_gob_til_Z #2\XINT_geq_hi \Z
614 \expandafter\XINT_geq_i\the\numexpr#1+1#4-#3-\xint_c_i.%
615}%
616\def\XINT_geq_i #1#2.%
617{%
618 1#2\expandafter!\the\numexpr\XINT_geq_a #1%
619}%
620\def\XINT_geq_bi\Z
621 \expandafter\XINT_geq_c\the\numexpr#1+1#2-#3.#4!#5!#6!#7!#8!#9!\Z !\W
622{%
623 \XINT_geq_k #1#2!#5!#7!#9!%
624}%
625\def\XINT_geq_di\Z
626 \expandafter\XINT_geq_e\the\numexpr#1+1#2-#3.#4!#5!#6!#7!#8\W
627{%
628 \XINT_geq_k #1#2!#5!#7!%
629}%
630\def\XINT_geq_fi\Z
631 \expandafter\XINT_geq_g\the\numexpr#1+1#2-#3.#4!#5!#6\W
632{%
633 \XINT_geq_k #1#2!#5!%
634}%
635\def\XINT_geq_hi\Z
636 \expandafter\XINT_geq_i\the\numexpr#1+1#2-#3.#4\W
637{%
638 \XINT_geq_k #1#2!%
639}%
640%%%%%%%%%%
641\def\XINT_geq_k #1#2\W
642{%
643 \xint_UDzerofork
644 #1{-1\relax { 0}}%
645 0{-1\relax { 1}}%
646 \krof
647}%
648\def\XINT_geq_w #1-1#2{#2}%

```

4.29 \xintiMax, \xintiiMax

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

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

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

```

649\def\xintiMax {\romannumeral0\xintimax}%
650\def\xintimax #1%
651{%

```

4 Package *xint* implementation

```

652 \expandafter\xint_max\expandafter {\romannumeral0\xintnum{#1}}%
653 }%
654 \def\xint_max #1#2%
655 {%
656 \expandafter\XINT_max_pre\expandafter {\romannumeral0\xintnum{#2}}{#1}%
657 }%
658 \def\xintiMax {\romannumeral0\xintiimax }%
659 \def\xintiimax #1%
660 {%
661 \expandafter\xint_iimax\expandafter {\romannumeral`&&@#1}%
662 }%
663 \def\xint_iimax #1#2%
664 {%
665 \expandafter\XINT_max_pre\expandafter {\romannumeral`&&@#2}{#1}%
666 }%
667 \def\XINT_max_pre #1#2{\XINT_max_fork #1\Z #2\Z {#2}{#1}}%
668 \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*

669 \def\XINT_max_fork #1#2\Z #3#4\Z
670 {%
671 \xint_UDsignsfork
672 #1#3\XINT_max_minusminus % A < 0, B < 0
673 #1-\XINT_max_minusplus % B < 0, A >= 0
674 #3-\XINT_max_plusminus % A < 0, B >= 0
675 --{\xint_UDzerosfork
676 #1#3\XINT_max_zerozero % A = B = 0
677 #10\XINT_max_zeroplus % B = 0, A > 0
678 #30\XINT_max_pluszero % A = 0, B > 0
679 00\XINT_max_plusplus % A, B > 0
680 \krof }%
681 \krof
682 {#2}{#4}#1#3%
683 }%

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

684 \def\XINT_max_zerozero #1#2#3#4{\xint_firstoftwo_thenstop }%
685 \def\XINT_max_zeroplus #1#2#3#4{\xint_firstoftwo_thenstop }%
686 \def\XINT_max_pluszero #1#2#3#4{\xint_secondoftwo_thenstop }%
687 \def\XINT_max_minusplus #1#2#3#4{\xint_firstoftwo_thenstop }%
688 \def\XINT_max_plusminus #1#2#3#4{\xint_secondoftwo_thenstop }%
689 \def\XINT_max_plusplus #1#2#3#4%
690 {%
691 \ifodd\XINT_Geq {#4#2}{#3#1}
692 \expandafter\xint_firstoftwo_thenstop
693 \else
694 \expandafter\xint_secondoftwo_thenstop
695 \fi
696 }%

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

```

```

697 \def\XINT_max_minusminus #1#2#3#4%
698 {%
699   \ifodd\XINT_Geq {#1}{#2}
700   \expandafter\xint_firstoftwo_thenstop
701   \else
702   \expandafter\xint_secondoftwo_thenstop
703   \fi
704 }%

```

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.

```

705 \def\xintiMaxof      {\romannumeral0\xintimaxof }%
706 \def\xintimaxof      #1{\expandafter\XINT_imaxof_a\romannumeral`&&@#1\relax }%
707 \def\XINT_imaxof_a   #1{\expandafter\XINT_imaxof_b\romannumeral0\xintnum{#1}\Z }%
708 \def\XINT_imaxof_b   #1\Z #2%
709   {\expandafter\XINT_imaxof_c\romannumeral`&&@#2\Z {#1}\Z}%
710 \def\XINT_imaxof_c   #1%
711   {\xint_gob_til_relax #1\XINT_imaxof_e\relax\XINT_imaxof_d #1}%
712 \def\XINT_imaxof_d   #1\Z
713   {\expandafter\XINT_imaxof_b\romannumeral0\xintiimax {#1}}%
714 \def\XINT_imaxof_e   #1\Z #2\Z { #2}%
715 \def\xintiiMaxof     {\romannumeral0\xintiimaxof }%
716 \def\xintiimaxof     #1{\expandafter\XINT_iimaxof_a\romannumeral`&&@#1\relax }%
717 \def\XINT_iimaxof_a  #1{\expandafter\XINT_iimaxof_b\romannumeral`&&@#1\Z }%
718 \def\XINT_iimaxof_b  #1\Z #2%
719   {\expandafter\XINT_iimaxof_c\romannumeral`&&@#2\Z {#1}\Z}%
720 \def\XINT_iimaxof_c  #1%
721   {\xint_gob_til_relax #1\XINT_iimaxof_e\relax\XINT_iimaxof_d #1}%
722 \def\XINT_iimaxof_d  #1\Z
723   {\expandafter\XINT_iimaxof_b\romannumeral0\xintiimax {#1}}%
724 \def\XINT_iimaxof_e  #1\Z #2\Z { #2}%

```

4.31 `\xintiMin`, `\xintiiMin`

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

```

725 \def\xintiMin {\romannumeral0\xintimin }%
726 \def\xintimin #1%
727 {%
728   \expandafter\xint_min\expandafter {\romannumeral0\xintnum{#1}}%
729 }%
730 \def\xint_min #1#2%
731 {%
732   \expandafter\XINT_min_pre\expandafter {\romannumeral0\xintnum{#2}}{#1}%
733 }%
734 \def\xintiiMin {\romannumeral0\xintiimin }%
735 \def\xintiimin #1%
736 {%

```

4 Package *xint* implementation

```

737 \expandafter\xint_iimin\expandafter {\romannumeral`&&@#1}%
738 }%
739 \def\xint_iimin #1#2%
740 {%
741 \expandafter\xint_min_pre\expandafter {\romannumeral`&&@#2}{#1}%
742 }%
743 \def\xint_min_pre #1#2{\xint_min_fork #1\Z #2\Z {#2}{#1}}%
744 \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*

745 \def\xint_min_fork #1#2\Z #3#4\Z
746 {%
747 \xint_UDsignsfork
748 #1#3\xint_min_minusminus % A < 0, B < 0
749 #1-\xint_min_minusplus % B < 0, A >= 0
750 #3-\xint_min_plusminus % A < 0, B >= 0
751 --{\xint_UDzerosfork
752 #1#3\xint_min_zerozero % A = B = 0
753 #10\xint_min_zeroplus % B = 0, A > 0
754 #30\xint_min_pluszero % A = 0, B > 0
755 00\xint_min_plusplus % A, B > 0
756 \krof }%
757 \krof
758 {#2}{#4}#1#3%
759 }%

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

760 \def\xint_min_zerozero #1#2#3#4{\xint_firstoftwo_thenstop }%
761 \def\xint_min_zeroplus #1#2#3#4{\xint_secondoftwo_thenstop }%
762 \def\xint_min_pluszero #1#2#3#4{\xint_firstoftwo_thenstop }%
763 \def\xint_min_minusplus #1#2#3#4{\xint_secondoftwo_thenstop }%
764 \def\xint_min_plusminus #1#2#3#4{\xint_firstoftwo_thenstop }%
765 \def\xint_min_plusplus #1#2#3#4%
766 {%
767 \ifodd\xint_Geq {#4#2}{#3#1}
768 \expandafter\xint_secondoftwo_thenstop
769 \else
770 \expandafter\xint_firstoftwo_thenstop
771 \fi
772 }%

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

773 \def\xint_min_minusminus #1#2#3#4%
774 {%
775 \ifodd\xint_Geq {#1}{#2}
776 \expandafter\xint_secondoftwo_thenstop
777 \else
778 \expandafter\xint_firstoftwo_thenstop
779 \fi
780 }%

```


4.32 `\xintiMinof`, `\xintiiMinof`

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

```

781 \def\xintiMinof      {\romannumeral0\xintiminof }%
782 \def\xintiminof      #1{\expandafter\XINT_iminof_a\romannumeral`&&@#1\relax }%
783 \def\XINT_iminof_a   #1{\expandafter\XINT_iminof_b\romannumeral0\xintnum{#1}\Z }%
784 \def\XINT_iminof_b   #1\Z #2%
785       {\expandafter\XINT_iminof_c\romannumeral`&&@#2\Z {#1}\Z}%
786 \def\XINT_iminof_c   #1%
787       {\xint_gob_til_relax #1\XINT_iminof_e\relax\XINT_iminof_d #1}%
788 \def\XINT_iminof_d   #1\Z
789       {\expandafter\XINT_iminof_b\romannumeral0\xintimin {#1}}%
790 \def\XINT_iminof_e   #1\Z #2\Z { #2}%
791 \def\xintiiMinof      {\romannumeral0\xintiiminof }%
792 \def\xintiiminof      #1{\expandafter\XINT_iiminof_a\romannumeral`&&@#1\relax }%
793 \def\XINT_iiminof_a   #1{\expandafter\XINT_iiminof_b\romannumeral`&&@#1\Z }%
794 \def\XINT_iiminof_b   #1\Z #2%
795       {\expandafter\XINT_iiminof_c\romannumeral`&&@#2\Z {#1}\Z}%
796 \def\XINT_iiminof_c   #1%
797       {\xint_gob_til_relax #1\XINT_iiminof_e\relax\XINT_iiminof_d #1}%
798 \def\XINT_iiminof_d   #1\Z
799       {\expandafter\XINT_iiminof_b\romannumeral0\xintiimin {#1}}%
800 \def\XINT_iiminof_e   #1\Z #2\Z { #2}%

```

4.33 `\xintiiSum`

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

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

1.03 (drastically) simplifies and makes the routines more efficient (for big computations). Also the way `\xintSum` and `\xintSumExpr ... \relax` are related. has been modified. Now `\xintSumExpr \z \relax` is accepted input when `\z` expands to a list of braced terms (prior only `\xintSum {z}` or `\xintSum \z` was possible).

1.09a does NOT add the `\xintnum` overhead. 1.09h renames `\xintiSum` to `\xintiiSum` to correctly reflect this.

The `xint 1.0x` routine could benefit from the fact that addition and subtraction did not check the lengths of the arguments and were able to do their job independently of the order (but not at equal speed). Thus it was possible to add separately positive and negative summands and do one big subtraction at the end, keeping during all that time the intermediate result in reverse order suitable for both addition and subtraction. The lazy programmer being a bit tired after the 95% rewrite of `xintcore` has not tried to do the same with the new model. Thus we just do stupidly repeated additions. The code is thus much shorter... and in fact I just copied the routine for products and changed products to sums.

```

801 \def\xintiiSum {\romannumeral0\xintiisum }%
802 \def\xintiisum #1{\xintiisumexpr #1\relax }%
803 \def\xintiiSumExpr {\romannumeral0\xintiisumexpr }%
804 \def\xintiisumexpr {\expandafter\XINT_sumexpr\romannumeral`&&@}%
805 \def\XINT_sumexpr {\XINT_sum_loop_a 0\Z }%
806 \def\XINT_sum_loop_a #1\Z #2%
807       {\expandafter\XINT_sum_loop_b \romannumeral`&&@#2\Z #1\Z \Z}%
808 \def\XINT_sum_loop_b #1%
809       {\xint_gob_til_relax #1\XINT_sum_finished\relax\XINT_sum_loop_c #1}%

```

```

810 \def\XINT_sum_loop_c
811   {\expandafter\XINT_sum_loop_a\romannumeral0\XINT_add_fork }%
812 \def\XINT_sum_finished #1\Z #2\Z \Z { #2}%

```

4.34 `\xintiiPrd`

```

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

```

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

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

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

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

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

```

813 \def\xintiiPrd {\romannumeral0\xintiiprd }%
814 \def\xintiiprd #1{\xintiiprdexpr #1\relax }%
815 \def\xintiiPrdExpr {\romannumeral0\xintiiprdexpr }%
816 \def\xintiiprdexpr {\expandafter\XINT_prdexpr\romannumeral`&&@}%
817 \def\XINT_prdexpr {\XINT_prod_loop_a 1\Z }%
818 \def\XINT_prod_loop_a #1\Z #2%
819   {\expandafter\XINT_prod_loop_b \romannumeral`&&@#2\Z #1\Z \Z}%
820 \def\XINT_prod_loop_b #1%
821   {\xint_gob_til_relax #1\XINT_prod_finished\relax\XINT_prod_loop_c #1}%
822 \def\XINT_prod_loop_c
823   {\expandafter\XINT_prod_loop_a\romannumeral0\XINT_mul_fork }%
824 \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 `\xintMON`, `\xintMMON`, `\xintiiMON`, `\xintiiMMON`

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

```

825 \def\xintiiMON {\romannumeral0\xintiimon }%
826 \def\xintiimon #1%
827 {%
828   \ifodd\xintiilDg {#1}
829     \xint_afterfi{ -1}%
830   \else

```

```

831     \xint_afterfi{ 1}%
832   \fi
833 }%
834 \def\xintiMMON {\romannumeral0\xintiimmon }%
835 \def\xintiimmon #1%
836 {%
837   \ifodd\xintiiLDg {#1}
838     \xint_afterfi{ 1}%
839   \else
840     \xint_afterfi{ -1}%
841   \fi
842 }%
843 \def\xintMON {\romannumeral0\xintmon }%
844 \def\xintmon #1%
845 {%
846   \ifodd\xintLDg {#1}
847     \xint_afterfi{ -1}%
848   \else
849     \xint_afterfi{ 1}%
850   \fi
851 }%
852 \def\xintMMON {\romannumeral0\xintmmon }%
853 \def\xintmmon #1%
854 {%
855   \ifodd\xintLDg {#1}
856     \xint_afterfi{ 1}%
857   \else
858     \xint_afterfi{ -1}%
859   \fi
860 }%

```

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

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

```

861 \def\xintiiOdd {\romannumeral0\xintiiodd }%
862 \def\xintiiodd #1%
863 {%
864   \ifodd\xintiiLDg{#1}
865     \xint_afterfi{ 1}%
866   \else
867     \xint_afterfi{ 0}%
868   \fi
869 }%
870 \def\xintiiEven {\romannumeral0\xintiieven }%
871 \def\xintiieven #1%
872 {%
873   \ifodd\xintiiLDg{#1}
874     \xint_afterfi{ 0}%
875   \else

```

```

876      \xint_afterfi{ 1}%
877    \fi
878 }%
879 \def\xintOdd {\romannumeral0\xintodd }%
880 \def\xintodd #1%
881 {%
882   \ifodd\xintLDg{#1}
883     \xint_afterfi{ 1}%
884   \else
885     \xint_afterfi{ 0}%
886   \fi
887 }%
888 \def\xintEven {\romannumeral0\xinteven }%
889 \def\xinteven #1%
890 {%
891   \ifodd\xintLDg{#1}
892     \xint_afterfi{ 0}%
893   \else
894     \xint_afterfi{ 1}%
895   \fi
896 }%

```

4.37 `\xintDSL`

DECIMAL SHIFT LEFT (=MULTIPLICATION PAR 10)

```

897 \def\xintDSL {\romannumeral0\xintdsl }%
898 \def\xintdsl #1%
899 {%
900   \expandafter\XINT_dsl \romannumeral`&&@#1\Z
901 }%
902 \def\XINT_DSL #1{\romannumeral0\XINT_dsl #1\Z }%
903 \def\XINT_dsl #1%
904 {%
905   \xint_gob_til_zero #1\xint_dsl_zero 0\XINT_dsl_ #1%
906 }%
907 \def\xint_dsl_zero 0\XINT_dsl_ 0#1\Z { 0}%
908 \def\XINT_dsl_ #1\Z { #10}%

```

4.38 `\xintDSR`

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

```

909 \def\xintDSR {\romannumeral0\xintdsr }%
910 \def\xintdsr #1%
911 {%
912   \expandafter\XINT_dsr_a\expandafter {\romannumeral`&&@#1}\W\Z
913 }%
914 \def\XINT_DSR #1{\romannumeral0\XINT_dsr_a {#1}\W\Z }%
915 \def\XINT_dsr_a
916 {%
917   \expandafter\XINT_dsr_b\romannumeral0\xintreverseorder

```

```

918 }%
919 \def\XINT_dsr_b #1#2#3\Z
920 {%
921   \xint_gob_til_W #2\xint_dsr_onedigit\W
922   \xint_gob_til_minus #2\xint_dsr_onedigit-%
923   \expandafter\XINT_dsr_removev
924   \romannumeral0\xintreverseorder {#2#3}%
925 }%
926 \def\xint_dsr_onedigit #1\xintreverseorder #2{ 0}%
927 \def\XINT_dsr_removev #1\W { }%

```

4.39 \xintDSH, \xintDSHr

DECIMAL SHIFTS \xintDSH {x}{A}
 si $x \leq 0$, fait $A \rightarrow A \cdot 10^{|x|}$. v1.03 corrige l'oversight pour $A=0$.
 si $x > 0$, et $A \geq 0$, fait $A \rightarrow \text{quo}(A, 10^x)$
 si $x > 0$, et $A < 0$, fait $A \rightarrow -\text{quo}(-A, 10^x)$
 (donc pour $x > 0$ c'est comme DSR itéré x fois)
 \xintDSHr donne le 'reste' (si $x \leq 0$ donne zéro).
 Release 1.06 now feeds x to a \numexpr first. I will have to revise this code at some point.

```

928 \def\xintDSHr {\romannumeral0\xintdshr }%
929 \def\xintdshr #1%
930 {%
931   \expandafter\XINT_dshr_checkxpositive \the\numexpr #1\relax\Z
932 }%
933 \def\XINT_dshr_checkxpositive #1%
934 {%
935   \xint_UDzerominusfork
936   0#1\XINT_dshr_xzeroorneg
937   #1-\XINT_dshr_xzeroorneg
938   0-\XINT_dshr_xpositive
939   \krof #1%
940 }%
941 \def\XINT_dshr_xzeroorneg #1\Z #2{ 0}%
942 \def\XINT_dshr_xpositive #1\Z
943 {%
944   \expandafter\xint_secondoftwo_thenstop\romannumeral0\xintdsx {#1}%
945 }%
946 \def\xintDSH {\romannumeral0\xintdsh }%
947 \def\xintdsh #1#2%
948 {%
949   \expandafter\xint_dsh\expandafter {\romannumeral`&&@#2}{#1}%
950 }%
951 \def\xint_dsh #1#2%
952 {%
953   \expandafter\XINT_dsh_checksignx \the\numexpr #2\relax\Z {#1}%
954 }%
955 \def\XINT_dsh_checksignx #1%
956 {%
957   \xint_UDzerominusfork
958   #1-\XINT_dsh_xiszero
959   0#1\XINT_dsx_xisNeg_checkA      % on passe direct dans DSx

```

```

960      0-{\XINT_dsh_xisPos #1}%
961      \krof
962 }%
963 \def\XINT_dsh_xiszero #1\Z #2{ #2}%
964 \def\XINT_dsh_xisPos #1\Z #2%
965 {%
966      \expandafter\xint_firstoftwo_thenstop
967      \romannumeral0\XINT_dsx_checksiga #2\Z {#1}% via DSx
968 }%

```

4.40 \xintDSx

Je fais cette routine pour la version 1.01, après modification de \xintDecSplit. Dorénavant \xintDSx fera appel à \xintDecSplit et de même \xintDSH fera appel à \xintDSx. J'ai donc supprimé entièrement l'ancien code de \xintDSH et re-écrit entièrement celui de \xintDecSplit pour *x* positif.

--> Attention le cas $x=0$ est traité dans la même catégorie que $x > 0$ <--
 si $x < 0$, fait $A \rightarrow A \cdot 10^{|x|}$
 si $x \geq 0$, et $A \geq 0$, fait $A \rightarrow \{ \text{quo}(A, 10^x) \} \{ \text{rem}(A, 10^x) \}$
 si $x \geq 0$, et $A < 0$, d'abord on calcule $\{ \text{quo}(-A, 10^x) \} \{ \text{rem}(-A, 10^x) \}$
 puis, si le premier n'est pas nul on lui donne le signe -
 si le premier est nul on donne le signe - au second.

On peut donc toujours reconstituer l'original A par $10^x Q \pm R$ où il faut prendre le signe plus si Q est positif ou nul et le signe moins si Q est strictement négatif.

Release 1.06 has a faster and more compactly coded \XINT_dsx_zeroloop. Also, *x* is now given to a \numexpr. The earlier code should be then simplified, but I leave as is for the time being.

Release 1.07 modified the coding of \XINT_dsx_zeroloop, to avoid impacting the input stack. Indeed the truncating, rounding, and conversion to float routines all use internally \XINT_dsx_zeroloop (via \XINT_dsx_addzerosnofuss), and they were thus roughly limited to generating $N = 8$ times the input save stack size digits. On TL2012 and TL2013, this means $40000 = 8 \times 5000$ digits. Although generating more than 40000 digits is more like a one shot thing, I wanted to open the possibility of outputting tens of thousands of digits to fail, thus I re-organized \XINT_dsx_zeroloop.

January 5, 2014: but it is only with the new division implementation of 1.09j and also with its special \xintXTrunc routine that the possibility mentioned in the last paragraph has become a concrete one in terms of computation time.

```

969 \def\xintDSx {\romannumeral0\xintdsx }%
970 \def\xintdsx #1#2%
971 {%
972      \expandafter\xint_dsx\expandafter {\romannumeral`&&#2}{#1}%
973 }%
974 \def\xint_dsx #1#2%
975 {%
976      \expandafter\XINT_dsx_checksiga \the\numexpr #2\relax\Z {#1}%
977 }%
978 \def\XINT_DSx #1#2{\romannumeral0\XINT_dsx_checksiga #1\Z {#2}}%
979 \def\XINT_dsx #1#2{\XINT_dsx_checksiga #1\Z {#2}}%
980 \def\XINT_dsx_checksiga #1%
981 {%
982      \xint_UDzerominusfork
983      #1-\XINT_dsx_xisZero
984      0#1\XINT_dsx_xisNeg_checkA
985      0-{\XINT_dsx_xisPos #1}%

```

4 Package *xint* implementation

```

986 \krof
987 }%
988 \def\XINT_dsx_xisZero #1\Z #2{ {#2}{0}}% attention comme x > 0
989 \def\XINT_dsx_xisNeg_checkA #1\Z #2%
990 {%
991 \XINT_dsx_xisNeg_checkA_ #2\Z {#1}%
992 }%
993 \def\XINT_dsx_xisNeg_checkA_ #1#2\Z #3%
994 {%
995 \xint_gob_til_zero #1\XINT_dsx_xisNeg_Azero 0%
996 \XINT_dsx_xisNeg_checkx {#3}{#3}{ }\Z {#1#2}%
997 }%
998 \def\XINT_dsx_xisNeg_Azero #1\Z #2{ 0}%
999 \def\XINT_dsx_xisNeg_checkx #1%
1000 {%
1001 \ifnum #1>1000000
1002 \xint_afterfi
1003 {\xintError:TooBigDecimalShift
1004 \expandafter\space\expandafter 0\xint_gobble_iv }%
1005 \else
1006 \expandafter \XINT_dsx_zeroloop
1007 \fi
1008 }%
1009 \def\XINT_dsx_addzerosnofuss #1{\XINT_dsx_zeroloop {#1}{ }\Z }%
1010 \def\XINT_dsx_zeroloop #1#2%
1011 {%
1012 \ifnum #1<\xint_c_ix \XINT_dsx_exita\fi
1013 \expandafter\XINT_dsx_zeroloop\expandafter
1014 {\the\numexpr #1-\xint_c_viii}{#200000000}%
1015 }%
1016 \def\XINT_dsx_exita\fi\expandafter\XINT_dsx_zeroloop
1017 {%
1018 \fi\expandafter\XINT_dsx_exitb
1019 }%
1020 \def\XINT_dsx_exitb #1#2%
1021 {%
1022 \expandafter\expandafter\expandafter
1023 \XINT_dsx_addzeros\csname xint_gobble_\romannumeral -#1\endcsname #2%
1024 }%
1025 \def\XINT_dsx_addzeros #1\Z #2{ #2#1}%
1026 \def\XINT_dsx_xisPos #1\Z #2%
1027 {%
1028 \XINT_dsx_checksiga #2\Z {#1}%
1029 }%
1030 \def\XINT_dsx_checksiga #1%
1031 {%
1032 \xint_UDzerominusfork
1033 #1-\XINT_dsx_AisZero
1034 0#1\XINT_dsx_AisNeg
1035 0-{\XINT_dsx_AisPos #1}%
1036 \krof
1037 }%

```

```

1038 \def\XINT_dsx_AisZero #1\Z #2{ {0}{0}}%
1039 \def\XINT_dsx_AisNeg #1\Z #2%
1040 {%
1041   \expandafter\XINT_dsx_AisNeg_dosplit_andcheckfirst
1042   \romannumeral0\XINT_split_checksizex {#2}{#1}%
1043 }%
1044 \def\XINT_dsx_AisNeg_dosplit_andcheckfirst #1%
1045 {%
1046   \XINT_dsx_AisNeg_checkiffirstempty #1\Z
1047 }%
1048 \def\XINT_dsx_AisNeg_checkiffirstempty #1%
1049 {%
1050   \xint_gob_til_Z #1\XINT_dsx_AisNeg_finish_zero\Z
1051   \XINT_dsx_AisNeg_finish_notzero #1%
1052 }%
1053 \def\XINT_dsx_AisNeg_finish_zero\Z
1054   \XINT_dsx_AisNeg_finish_notzero\Z #1%
1055 {%
1056   \expandafter\XINT_dsx_end
1057   \expandafter {\romannumeral0\XINT_num {-#1}}{0}%
1058 }%
1059 \def\XINT_dsx_AisNeg_finish_notzero #1\Z #2%
1060 {%
1061   \expandafter\XINT_dsx_end
1062   \expandafter {\romannumeral0\XINT_num {#2}}{-#1}%
1063 }%
1064 \def\XINT_dsx_AisPos #1\Z #2%
1065 {%
1066   \expandafter\XINT_dsx_AisPos_finish
1067   \romannumeral0\XINT_split_checksizex {#2}{#1}%
1068 }%
1069 \def\XINT_dsx_AisPos_finish #1#2%
1070 {%
1071   \expandafter\XINT_dsx_end
1072   \expandafter {\romannumeral0\XINT_num {#2}}%
1073   {\romannumeral0\XINT_num {#1}}%
1074 }%
1075 \edef\XINT_dsx_end #1#2%
1076 {%
1077   \noexpand\expandafter\space\noexpand\expandafter{#2}{#1}%
1078 }%

```

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 guar-

4 Package *xint* implementation

anteed to remain the same.

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

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

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

```
1079 \def\xintDecSplitL {\romannumeral0\xintdecsplitl }%
1080 \def\xintDecSplitR {\romannumeral0\xintdecsplitr }%
1081 \def\xintdecsplitl
1082 {%
1083   \expandafter\xint_firstoftwo_thenstop
1084   \romannumeral0\xintdecsplit
1085 }%
1086 \def\xintdecsplitr
1087 {%
1088   \expandafter\xint_secondoftwo_thenstop
1089   \romannumeral0\xintdecsplit
1090 }%
1091 \def\xintDecSplit {\romannumeral0\xintdecsplit }%
1092 \def\xintdecsplit #1#2%
1093 {%
1094   \expandafter \xint_split \expandafter
1095   {\romannumeral0\xintiabs {#2}}{#1}%   fait expansion de A
1096 }%
1097 \def\xint_split #1#2%
1098 {%
1099   \expandafter\XINT_split_checksizex\expandafter{\the\numexpr #2}{#1}%
1100 }%
1101 \def\XINT_split_checksizex #1% 999999999 is anyhow very big, could be reduced
1102 {%
1103   \ifnum\numexpr\XINT_Abs{#1}>999999999
1104     \xint_afterfi {\xintError:TooBigDecimalSplit\XINT_split_bigx }%
1105   \else
1106     \expandafter\XINT_split_xfork
1107   \fi
1108   #1\Z
1109 }%
1110 \def\XINT_split_bigx  #1\Z #2%
1111 {%
1112   \ifcase\XINT_cntSgn #1\Z
1113   \or \xint_afterfi { }{#2}% positive big x
1114   \else
1115     \xint_afterfi { }{#2}% negative big x
1116   \fi
1117 }%
1118 \def\XINT_split_xfork #1%
1119 {%
1120   \xint_UDzerominusfork
1121   #1-\XINT_split_zerosplit
1122   0#1\XINT_split_fromleft
```

4 Package `xint` implementation

```

1123      0-{\XINT_split_fromright #1}%
1124      \krof
1125 }%
1126 \def\XINT_split_zerosplit #1\Z #2{ {#2}{}}%
1127 \def\XINT_split_fromleft #1\Z #2%
1128 {%
1129      \XINT_split_fromleft_loop {#1}{#2\W\W\W\W\W\W\W\Z
1130 }%
1131 \def\XINT_split_fromleft_loop #1%
1132 {%
1133      \ifnum #1<\xint_c_viii\XINT_split_fromleft_exita\fi
1134      \expandafter\XINT_split_fromleft_loop_perhaps\expandafter
1135      {\the\numexpr #1-\xint_c_viii\expandafter}\XINT_split_fromleft_eight
1136 }%
1137 \def\XINT_split_fromleft_eight #1#2#3#4#5#6#7#8#9{#9{#1#2#3#4#5#6#7#8#9}}%
1138 \def\XINT_split_fromleft_loop_perhaps #1#2%
1139 {%
1140      \xint_gob_til_W #2\XINT_split_fromleft_toofar\W
1141      \XINT_split_fromleft_loop {#1}%
1142 }%
1143 \def\XINT_split_fromleft_toofar\W\XINT_split_fromleft_loop #1#2#3\Z
1144 {%
1145      \XINT_split_fromleft_toofar_b #2\Z
1146 }%
1147 \def\XINT_split_fromleft_toofar_b #1\W #2\Z { {#1}{}}%
1148 \def\XINT_split_fromleft_exita\fi
1149      \expandafter\XINT_split_fromleft_loop_perhaps\expandafter #1#2%
1150      {\fi \XINT_split_fromleft_exitb #1}%
1151 \def\XINT_split_fromleft_exitb\the\numexpr #1-\xint_c_viii\expandafter
1152 {%
1153      \csname XINT_split_fromleft_endsplit_\romannumeral #1\endcsname
1154 }%
1155 \def\XINT_split_fromleft_endsplit_ #1#2\W #3\Z { {#1}{#2}}%
1156 \def\XINT_split_fromleft_endsplit_i #1#2%
1157      {\XINT_split_fromleft_checkiftoofar #2{#1#2}}%
1158 \def\XINT_split_fromleft_endsplit_ii #1#2#3%
1159      {\XINT_split_fromleft_checkiftoofar #3{#1#2#3}}%
1160 \def\XINT_split_fromleft_endsplit_iii #1#2#3#4%
1161      {\XINT_split_fromleft_checkiftoofar #4{#1#2#3#4}}%
1162 \def\XINT_split_fromleft_endsplit_iv #1#2#3#4#5%
1163      {\XINT_split_fromleft_checkiftoofar #5{#1#2#3#4#5}}%
1164 \def\XINT_split_fromleft_endsplit_v #1#2#3#4#5#6%
1165      {\XINT_split_fromleft_checkiftoofar #6{#1#2#3#4#5#6}}%
1166 \def\XINT_split_fromleft_endsplit_vi #1#2#3#4#5#6#7%
1167      {\XINT_split_fromleft_checkiftoofar #7{#1#2#3#4#5#6#7}}%
1168 \def\XINT_split_fromleft_endsplit_vii #1#2#3#4#5#6#7#8%
1169      {\XINT_split_fromleft_checkiftoofar #8{#1#2#3#4#5#6#7#8}}%
1170 \def\XINT_split_fromleft_checkiftoofar #1#2#3\W #4\Z
1171 {%
1172      \xint_gob_til_W #1\XINT_split_fromleft_wenttoofar\W
1173      \space {#2}{#3}%
1174 }%

```

4 Package *xint* implementation

```

1175 \def\XINT_split_fromleft_wenttoofar\W\space #1%
1176 {%
1177   \XINT_split_fromleft_wenttoofar_b #1\Z
1178 }%
1179 \def\XINT_split_fromleft_wenttoofar_b #1\W #2\Z { {#1}}%
1180 \def\XINT_split_fromright #1\Z #2%
1181 {%
1182   \expandafter \XINT_split_fromright_a \expandafter
1183   {\romannumeral0\xintreverseorder {#2}}{#1}{#2}%
1184 }%
1185 \def\XINT_split_fromright_a #1#2%
1186 {%
1187   \XINT_split_fromright_loop {#2}{#1\W\W\W\W\W\W\W\W\Z
1188 }%
1189 \def\XINT_split_fromright_loop #1%
1190 {%
1191   \ifnum #1<\xint_c_viii\XINT_split_fromright_exita\fi
1192   \expandafter\XINT_split_fromright_loop_perhaps\expandafter
1193   {\the\numexpr #1-\xint_c_viii\expandafter }\XINT_split_fromright_eight
1194 }%
1195 \def\XINT_split_fromright_eight #1#2#3#4#5#6#7#8#9{#9{#9#8#7#6#5#4#3#2#1}}%
1196 \def\XINT_split_fromright_loop_perhaps #1#2%
1197 {%
1198   \xint_gob_til_W #2\XINT_split_fromright_toofar\W
1199   \XINT_split_fromright_loop {#1}%
1200 }%
1201 \def\XINT_split_fromright_toofar\W\XINT_split_fromright_loop #1#2#3\Z { {}}%
1202 \def\XINT_split_fromright_exita\fi
1203   \expandafter\XINT_split_fromright_loop_perhaps\expandafter #1#2%
1204   {\fi \XINT_split_fromright_exitb #1}%
1205 \def\XINT_split_fromright_exitb\the\numexpr #1-\xint_c_viii\expandafter
1206 {%
1207   \csname XINT_split_fromright_endsplit_\romannumeral #1\endcsname
1208 }%
1209 \edef\XINT_split_fromright_endsplit_ #1#2\W #3\Z #4%
1210 {%
1211   \noexpand\expandafter\space\noexpand\expandafter
1212   {\noexpand\romannumeral0\noexpand\xintreverseorder {#2}}{#1}%
1213 }%
1214 \def\XINT_split_fromright_endsplit_i #1#2%
1215   {\XINT_split_fromright_checkiftoofar #2{#2#1}}%
1216 \def\XINT_split_fromright_endsplit_ii #1#2#3%
1217   {\XINT_split_fromright_checkiftoofar #3{#3#2#1}}%
1218 \def\XINT_split_fromright_endsplit_iii #1#2#3#4%
1219   {\XINT_split_fromright_checkiftoofar #4{#4#3#2#1}}%
1220 \def\XINT_split_fromright_endsplit_iv #1#2#3#4#5%
1221   {\XINT_split_fromright_checkiftoofar #5{#5#4#3#2#1}}%
1222 \def\XINT_split_fromright_endsplit_v #1#2#3#4#5#6%
1223   {\XINT_split_fromright_checkiftoofar #6{#6#5#4#3#2#1}}%
1224 \def\XINT_split_fromright_endsplit_vi #1#2#3#4#5#6#7%
1225   {\XINT_split_fromright_checkiftoofar #7{#7#6#5#4#3#2#1}}%
1226 \def\XINT_split_fromright_endsplit_vii #1#2#3#4#5#6#7#8%

```

```

1227      {\XINT_split_fromright_checkiftoofar #8{#8#7#6#5#4#3#2#1}}%
1228 \def\XINT_split_fromright_checkiftoofar #1%
1229 {%
1230   \xint_gob_til_W #1\XINT_split_fromright_wenttoofar\W
1231   \XINT_split_fromright_endsplit_
1232 }%
1233 \def\XINT_split_fromright_wenttoofar\W\XINT_split_fromright_endsplit_ #1\Z #2%
1234 { {#{#2}}}%

```

4.42 \xintiiSqrt, \xintiiSqrtR, \xintiiSquareRoot

v1.08. 1.09a uses \xintnum.

Some overhead was added inadvertently in 1.09a to inner routines when \xintiquo and \xintidivision were also promoted to use \xintnum; release 1.09f thus uses \xintiiquo and \xintiidivision which avoid this \xintnum overhead.

1.09j replaced the previous long \ifcase from \XINT_sqrt_c by some nested \ifnum's.

1.1 Ajout de \xintiiSqrt et \xintiiSquareRoot.

1.1a ajoute \xintiiSqrtR, which provides the rounded, not truncated square root.

```

1235 \def\xintiiSqrt {\romannumeral0\xintiisqrt }%
1236 \def\xintiiSqrtR {\romannumeral0\xintiisqrtr }%
1237 \def\xintiiSquareRoot {\romannumeral0\xintiisquareroot }%
1238 \def\xintiSqrt {\romannumeral0\xintisqrt }%
1239 \def\xintiSquareRoot {\romannumeral0\xintisquareroot }%
1240 \def\xintisqrt {\expandafter\XINT_sqrt_post\romannumeral0\xintisquareroot }%
1241 \def\xintiisqrt {\expandafter\XINT_sqrt_post\romannumeral0\xintiisquareroot }%
1242 \def\xintiisqrtr {\expandafter\XINT_sqrtr_post\romannumeral0\xintiisquareroot }%
1243 \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 $\#2 \geq \#1$, $(\#1 - .5)^2 \geq N + .25 > N$, donc rounded- $\rightarrow \#1 - 1$.

```

1244 \def\XINT_sqrtr_post #1#2{\xintiiiflt {#2}{#1}{ #1}{\XINT_dec_pos #1\Z}}%
1245 \def\xintisquareroot #1%
1246   {\expandafter\XINT_sqrt_checkin\romannumeral0\xintnum{#1}\Z }%
1247 \def\xintiisquareroot #1{\expandafter\XINT_sqrt_checkin\romannumeral`&&@#1\Z }%
1248 \def\XINT_sqrt_checkin #1%
1249 {%
1250   \xint_UDzerominusfork
1251   #1-\XINT_sqrt_iszero
1252   0#1\XINT_sqrt_isneg
1253   0-{\XINT_sqrt #1}%
1254   \krof
1255 }%
1256 \def\XINT_sqrt_iszero #1\Z { 11}%
1257 \edef\XINT_sqrt_isneg #1\Z {\noexpand\xintError:RootOfNegative\space 11}%
1258 \def\XINT_sqrt #1\Z
1259 {%
1260   \expandafter\XINT_sqrt_start\expandafter {\romannumeral0\xintlenght {#1}}{#1}%
1261 }%
1262 \def\XINT_sqrt_start #1%
1263 {%

```

4 Package *xint* implementation

```

1264 \ifnum #1<\xint_c_x
1265 \expandafter\xint_sqrt_small_a
1266 \else
1267 \expandafter\xint_sqrt_big_a
1268 \fi
1269 {#1}%
1270 }%
1271 \def\xint_sqrt_small_a #1{\xint_sqrt_a {#1}\xint_sqrt_small_d }%
1272 \def\xint_sqrt_big_a #1{\xint_sqrt_a {#1}\xint_sqrt_big_d }%
1273 \def\xint_sqrt_a #1%
1274 {%
1275 \ifodd #1
1276 \expandafter\xint_sqrt_bB
1277 \else
1278 \expandafter\xint_sqrt_bA
1279 \fi
1280 {#1}%
1281 }%
1282 \def\xint_sqrt_bA #1#2#3%
1283 {%
1284 \xint_sqrt_bA_b #3\Z #2{#1}{#3}%
1285 }%
1286 \def\xint_sqrt_bA_b #1#2#3\Z
1287 {%
1288 \xint_sqrt_c {#1#2}%
1289 }%
1290 \def\xint_sqrt_bB #1#2#3%
1291 {%
1292 \xint_sqrt_bB_b #3\Z #2{#1}{#3}%
1293 }%
1294 \def\xint_sqrt_bB_b #1#2\Z
1295 {%
1296 \xint_sqrt_c #1%
1297 }%
1298 \def\xint_sqrt_c #1#2%
1299 {%
1300 \expandafter #2\expandafter
1301 {\the\numexpr\ifnum #1>\xint_c_iii
1302 \ifnum #1>\xint_c_viii
1303 \ifnum #1>15 \ifnum #1>24 \ifnum #1>35
1304 \ifnum #1>48 \ifnum #1>63 \ifnum #1>80
1305 10\else 9\fi \else 8\fi \else 7\fi \else 6\fi
1306 \else 5\fi \else 4\fi \else 3\fi \else 2\fi \relax }%
1307 }%
1308 \def\xint_sqrt_small_d #1#2%
1309 {%
1310 \expandafter\xint_sqrt_small_e\expandafter
1311 {\the\numexpr #1\ifcase \numexpr #2/\xint_c_ii-\xint_c_i\relax
1312 \or 0\or 00\or 000\or 0000\fi }%
1313 }%
1314 \def\xint_sqrt_small_e #1#2%
1315 {%

```

4 Package `xint` implementation

```

1316 \expandafter\XINT_sqrt_small_f\expandafter {\the\numexpr #1*#1-#2}{#1}%
1317 }%
1318 \def\XINT_sqrt_small_f #1#2%
1319 {%
1320 \expandafter\XINT_sqrt_small_g\expandafter
1321 {\the\numexpr ((#1+#2)/(\xint_c_ii*#2))-\xint_c_i}{#1}{#2}%
1322 }%
1323 \def\XINT_sqrt_small_g #1%
1324 {%
1325 \ifnum #1>\xint_c_
1326 \expandafter\XINT_sqrt_small_h
1327 \else
1328 \expandafter\XINT_sqrt_small_end
1329 \fi
1330 {#1}%
1331 }%
1332 \def\XINT_sqrt_small_h #1#2#3%
1333 {%
1334 \expandafter\XINT_sqrt_small_f\expandafter
1335 {\the\numexpr #2-\xint_c_ii*#1*#3+#1*#1\expandafter}\expandafter
1336 {\the\numexpr #3-#1}%
1337 }%
1338 \def\XINT_sqrt_small_end #1#2#3{ {#3}{#2}}%
1339 \def\XINT_sqrt_big_d #1#2%
1340 {%
1341 \ifodd #2
1342 \expandafter\expandafter\expandafter\XINT_sqrt_big_eB
1343 \else
1344 \expandafter\expandafter\expandafter\XINT_sqrt_big_eA
1345 \fi
1346 \expandafter {\the\numexpr #2/\xint_c_ii }{#1}%
1347 }%
1348 \def\XINT_sqrt_big_eA #1#2#3%
1349 {%
1350 \XINT_sqrt_big_eA_a #3\Z {#2}{#1}{#3}%
1351 }%
1352 \def\XINT_sqrt_big_eA_a #1#2#3#4#5#6#7#8#9\Z
1353 {%
1354 \XINT_sqrt_big_eA_b {#1#2#3#4#5#6#7#8}%
1355 }%
1356 \def\XINT_sqrt_big_eA_b #1#2%
1357 {%
1358 \expandafter\XINT_sqrt_big_f
1359 \romannumeral0\XINT_sqrt_small_e {#2000}{#1}{#1}%
1360 }%
1361 \def\XINT_sqrt_big_eB #1#2#3%
1362 {%
1363 \XINT_sqrt_big_eB_a #3\Z {#2}{#1}{#3}%
1364 }%
1365 \def\XINT_sqrt_big_eB_a #1#2#3#4#5#6#7#8#9%
1366 {%
1367 \XINT_sqrt_big_eB_b {#1#2#3#4#5#6#7#8#9}%

```

```

1368 }%
1369 \def\XINT_sqrt_big_eB_b #1#2\Z #3%
1370 {%
1371     \expandafter\XINT_sqrt_big_f
1372     \romannumeral0\XINT_sqrt_small_e {#30000}{#1}{#1}%
1373 }%
1374 \def\XINT_sqrt_big_f #1#2#3#4%
1375 {%
1376     \expandafter\XINT_sqrt_big_f_a\expandafter
1377     {\the\numexpr #2+#3\expandafter}\expandafter
1378     {\romannumeral0\XINT_dsx_addzerosnofuss
1379         {\numexpr #4-\xint_c_iv\relax}{#1}}{#4}%
1380 }%
1381 \def\XINT_sqrt_big_f_a #1#2#3#4%
1382 {%
1383     \expandafter\XINT_sqrt_big_g\expandafter
1384     {\romannumeral0\xintiisub
1385         {\XINT_dsx_addzerosnofuss
1386             {\numexpr \xint_c_ii*#3-\xint_c_viii\relax}{#1}}{#4}}%
1387     {#2}{#3}%
1388 }%
1389 \def\XINT_sqrt_big_g #1#2%
1390 {%
1391     \expandafter\XINT_sqrt_big_j
1392     \romannumeral0\xintiidivision{#1}{\romannumeral0\XINT_dbl_pos #2\Z}{#2}%
1393 }%
1394 \def\XINT_sqrt_big_j #1%
1395 {%
1396     \if0\XINT_Sgn #1\Z
1397         \expandafter \XINT_sqrt_big_end
1398     \else \expandafter \XINT_sqrt_big_k
1399     \fi {#1}%
1400 }%
1401 \def\XINT_sqrt_big_k #1#2#3%
1402 {%
1403     \expandafter\XINT_sqrt_big_l\expandafter
1404     {\romannumeral0\xintiisub {#3}{#1}}%
1405     {\romannumeral0\xintiiaadd {#2}{\xintiiSqr {#1}}}%
1406 }%
1407 \def\XINT_sqrt_big_l #1#2%
1408 {%
1409     \expandafter\XINT_sqrt_big_g\expandafter
1410     {#2}{#1}%
1411 }%
1412 \def\XINT_sqrt_big_end #1#2#3#4{ {#3}{#2}}%

```

4.43 `\xintiiE`

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

```

1413 \def\xintiiE {\romannumeral0\xintiiE }% used in \xintMod.
1414 \def\xintiiE #1#2%
1415     {\expandafter\XINT_iiE\the\numexpr #2\expandafter.\expandafter{\romannumeral`&&@#1}}%

```

5 Package `xintbinhex` implementation

```
1416 \def\XINT_ii #1.#2{\ifnum#1>\xint_c_ \xint_dothis{\xint_dsh {#2}{-#1}}\fi
1417 \xint_orthat{ #2}}%
```

4.44 “Load `xintfrac`” macros

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

```
1418 \catcode`! 11
1419 \def\xintMax {\Did_you_mean_iiMax?or_load_xintfrac!}%
1420 \def\xintMin {\Did_you_mean_iiMin?or_load_xintfrac!}%
1421 \def\xintMaxof {\Did_you_mean_iMaxof?or_load_xintfrac!}%
1422 \def\xintMinof {\Did_you_mean_iMinof?or_load_xintfrac!}%
1423 \def\xintSum {\Did_you_mean_iiSum?or_load_xintfrac!}%
1424 \def\xintPrd {\Did_you_mean_iiPrd?or_load_xintfrac!}%
1425 \def\xintPrdExpr {\Did_you_mean_iiPrdExpr?or_load_xintfrac!}%
1426 \def\xintSumExpr {\Did_you_mean_iiSumExpr?or_load_xintfrac!}%
1427 \XINT_restorecatcodes_endinput%
```

5 Package `xintbinhex` implementation

.1	Catcodes, ε -TeX and reload detection . . .	120	.6	<code>\xintHexToDec</code>	127
.2	Package identification	121	.7	<code>\xintBinToDec</code>	128
.3	Constants, etc...	121	.8	<code>\xintBinToHex</code>	130
.4	<code>\XINT_OQ</code>	123	.9	<code>\xintHexToBin</code>	131
.5	<code>\xintDecToHex</code> , <code>\xintDecToBin</code>	124	.10	<code>\xintCHexToBin</code>	132

The commenting is currently (2015/11/22) very sparse.

5.1 Catcodes, ε -TeX and reload detection

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

```
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 [2015/11/22 v1.2e Expandable binary and hexadecimal conversions (jfb)]%

```

5.3 Constants, etc...

v1.08

```

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}%

```

5 Package *xintbinhex* implementation

```

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
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}%

```

5 Package *xintbinhex* implementation

```
119 \def\XINT_shthb_D {1101}%
120 \def\XINT_shthb_E {1110}%
121 \def\XINT_shthb_F {1111}%
122 \def\XINT_shthb_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_sdtb_#1\expandafter\expandafter\expandafter\endcsname
131     \csname XINT_sdtb_\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}%
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#900000000}%
```

```

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 \Z #1#2#3#4#5#6#7#8\Z { #1#2#3#4#5#6#7#80}%

```

5.5 \xintDecToHex, \xintDecToBin

v1.08

```

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
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}}%

```

5 Package *xintbinhex* implementation

```

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
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 {%

```

5 Package *xintbinhex* implementation

```

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 }%
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

v1.08

```

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
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

```

5 Package *xintbinhex* implementation

```
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#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*

v1.08

```
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 {0000000}{#1}}%
412   \krof
413 }%
```


5 Package *xintbinhex* implementation

```

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
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 #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\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\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\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 }}%

```

5 Package *xintbinhex* implementation

```
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#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+(\XINT_btd_II_d {#2#3}{#1})\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#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#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*

v1.08

```
509 \def\xintBinToHex {\romannumeral0\xintbinto hex }%
510 \def\xintbinto hex #1%
511 {%
512   \expandafter\XINT_bth_checkin
513     \romannumeral0\expandafter\XINT_num_loop
```

5 Package *xintbinhex* implementation

```
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\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*

v1.08

```
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*

v1.08

```

588 \def\xintCHexToBin {\romannumeral0\xintchextobin }%
589 \def\xintchextobin #1%
590 {%
591   \expandafter\XINT_cho_b_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_cho_b_checkin #1%
595 {%
596   \xint_UDsignfork
597   #1\XINT_cho_b_N
598   -{\XINT_cho_b_P #1}%
599   \krof
600 }%
601 \def\XINT_cho_b_N {\expandafter\xint_minus_thenstop\romannumeral0\XINT_cho_b_P }%
602 \def\XINT_cho_b_P {\expandafter\XINT_cho_b_I\expandafter{\expandafter}%
603   \romannumeral0\XINT_OQ {}}%
604 \def\XINT_cho_b_I #1#2#3#4#5#6#7#8#9%
605 {%
606   \xint_gob_til_W #9\XINT_cho_b_end_a\W
607   \expandafter\expandafter\expandafter
608   \XINT_cho_b_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 }%
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, ε -T _E X and reload detection . . .	133	.7	<code>\xintBezoutAlgorithm</code>	141
.2	Package identification	134	.8	<code>\xintGCDof</code>	143
.3	<code>\xintGCD</code> , <code>\xintiiGCD</code>	134	.9	<code>\xintLCMof</code>	143
.4	<code>\xintLCM</code> , <code>\xintiiLCM</code>	135	.10	<code>\xintTypesetEuclideanAlgorithm</code>	144
.5	<code>\xintBezout</code>	136	.11	<code>\xintTypesetBezoutAlgorithm</code>	144
.6	<code>\xintEuclideanAlgorithm</code>	140			

The commenting is currently (2015/11/22) 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 T_EX or L^AT_EX'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, ε -T_EX and reload detection

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

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

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2   \catcode13=5      % ^^M
3   \endlinechar=13 %
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 [2015/11/22 v1.2e Euclidean algorithm with xint package (jfb)]%

```

6.3 *\xintGCD*, *\xintiGCD*

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

```

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\xintiiGCD {\romannumeral0\xintiigcd }%
57 \def\xintiigcd #1%
58 {%
59   \expandafter\XINT_iigcd\expandafter{\romannumeral0\xintiabs {#1}}%
60 }%
61 \def\XINT_iigcd #1#2%
62 {%
63   \expandafter\XINT_gcd_fork\romannumeral0\xintiabs {#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\xint_gcd_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

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

```

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

1.09a inserts use of \xintnum

```

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

#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\xintiadd{\XINT_mul_fork #5\Z #2\Z}{#7}}%
227 {\romannumeral0\xintiadd{\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}}%

```

$\{n\}\{rn\}\{an\}\{qn\}\{rn\}}\dots\{A\}\{B\}}\}\Z$
 $a(n) = r(n-1)$. 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

$\{N\}{A}\{0\}\{1\}{D=r(n)}\{B\}\{1\}\{0\}\{q1\}{r1}\{\alpha1=q1\}\{\beta1=1\}$
 $\{q2\}{r2}\{\alpha2\}\{\beta2\}...\{qN\}{rN=0}\{\alphaN=A/D\}\{\betaN=B/D\}$
 $\alpha0=1, \beta0=0, \alpha(-1)=0, \beta(-1)=1$

```

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)}{al
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)}...

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_end\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

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

```

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, ε -TeX and reload detection . . .	146	.9	<code>\XINT_factortens</code> , <code>\XINT_cuz_cnt</code> . . .	152
.2	Package identification	147	.10	<code>\XINT_addm_A</code>	154
.3	<code>\XINT_cntSgnFork</code>	147	.11	<code>\xintRaw</code>	156
.4	<code>\xintLen</code>	147	.12	<code>\xintPRaw</code>	156
.5	<code>\XINT_lenrord_loop</code>	148	.13	<code>\xintRawWithZeros</code>	156
.6	<code>\XINT_outfrac</code>	148	.14	<code>\xintFloor</code> , <code>\xintiFloor</code>	157
.7	<code>\XINT_inFrac</code>	149	.15	<code>\xintCeil</code> , <code>\xintiCeil</code>	157
.8	<code>\XINT_frac_gen</code>	150	.16	<code>\xintNumerator</code>	157

.17	<code>\xintDenominator</code>	158	.44	<code>\xintFac</code>	186
.18	<code>\xintFrac</code>	158	.45	<code>\xintPrd</code>	186
.19	<code>\xintSignedFrac</code>	159	.46	<code>\xintDiv</code>	186
.20	<code>\xintFwOver</code>	159	.47	<code>\xintDivFloor</code>	187
.21	<code>\xintSignedFwOver</code>	160	.48	<code>\xintDivTrunc</code>	187
.22	<code>\xintREZ</code>	160	.49	<code>\xintDivRound</code>	187
.23	<code>\xintE, \xintFloatE, \XINTinFloatE</code>	161	.50	<code>\xintMod</code>	187
.24	<code>\xintIrr</code>	162	.51	<code>\XINTinFloatMod</code>	188
.25	<code>\xintifInt</code>	163	.52	<code>\xintIsOne</code>	188
.26	<code>\xintJrr</code>	163	.53	<code>\xintGeq</code>	189
.27	<code>\xintTFrac</code>	165	.54	<code>\xintMax</code>	190
.28	<code>\XINTinFloatFracdigits</code>	165	.55	<code>\xintMaxof</code>	190
.29	<code>\xintTrunc, \xintiTrunc</code>	165	.56	<code>\xintMin</code>	191
.30	<code>\xintTTrunc</code>	167	.57	<code>\xintMinof</code>	191
.31	<code>\xintNum</code>	168	.58	<code>\xintCmp</code>	192
.32	<code>\xintRound, \xintiRound</code>	168	.59	<code>\xintAbs</code>	193
.33	<code>\xintXTrunc</code>	169	.60	<code>\xintOpp</code>	193
.34	<code>\xintDigits</code>	175	.61	<code>\xintSgn</code>	193
.35	<code>\xintFloat</code>	175	.62	Floating point macros	193
.36	<code>\xintPFloat</code>	178	.63	<code>\xintFloatAdd, \XINTinFloatAdd</code>	194
.37	<code>\XINTinFloat</code>	180	.64	<code>\xintFloatSub, \XINTinFloatSub</code>	195
.38	<code>\xintAdd</code>	182	.65	<code>\xintFloatMul, \XINTinFloatMul</code>	195
.39	<code>\xintSub</code>	183	.66	<code>\xintFloatDiv, \XINTinFloatDiv</code>	196
.40	<code>\xintSum</code>	184	.67	<code>\xintFloatPow, \XINTinFloatPow</code>	196
.41	<code>\xintMul</code>	184	.68	<code>\xintFloatPower, \XINTinFloatPower</code>	199
.42	<code>\xintSqr</code>	185	.69	<code>\xintFloatFac, \XINTFloatFac</code>	202
.43	<code>\xintPow</code>	185	.70	<code>\xintFloatSqrt, \XINTinFloatSqrt</code>	206

The commenting is currently (2015/11/22) very sparse.

7.1 Catcodes, ε - \TeX and reload detection

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

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

```

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\cename 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 [2015/11/22 v1.2e Expandable operations on fractions (jfb)]%

```

7.3 *\XINT_cntSgnFork*

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

```

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*

```

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`

```

64 \def\XINT_lenrord_loop #1#2#3#4#5#6#7#8#9%
65 {% faire \romannumeral`&&\XINT_lenrord_loop 0{#1}\Z\W\W\W\W\W\W\W\Z
66 \xint_gob_til_W #9\XINT_lenrord_W\W
67 \expandafter\XINT_lenrord_loop\expandafter
68 {\the\numexpr #1+7}{#9#8#7#6#5#4#3#2}%
69 }%
70 \def\XINT_lenrord_W\W\expandafter\XINT_lenrord_loop\expandafter #1#2#3\Z
71 {%
72 \expandafter\XINT_lenrord_X\expandafter {#1}#2\Z
73 }%
74 \def\XINT_lenrord_X #1#2\Z
75 {%
76 \XINT_lenrord_Y #2\R\R\R\R\R\R\T {#1}%
77 }%
78 \def\XINT_lenrord_Y #1#2#3#4#5#6#7#8\T
79 {%
80 \xint_gob_til_W
81 #7\XINT_lenrord_Z \xint_c_viii
82 #6\XINT_lenrord_Z \xint_c_vii
83 #5\XINT_lenrord_Z \xint_c_vi
84 #4\XINT_lenrord_Z \xint_c_v
85 #3\XINT_lenrord_Z \xint_c_iv
86 #2\XINT_lenrord_Z \xint_c_iii
87 \W\XINT_lenrord_Z \xint_c_ii \Z
88 }%
89 \def\XINT_lenrord_Z #1#2\Z #3% retourne: {longueur}renverse\Z
90 {%
91 \expandafter{\the\numexpr #3-#1\relax}%
92 }%

```

7.6 `\XINT_outfrac`

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

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

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

```

93 \def\XINT_outfrac #1#2#3%
94 {%
95 \ifcase\XINT_cntSgn #3\Z
96 \expandafter \XINT_outfrac_divisionbyzero
97 \or
98 \expandafter \XINT_outfrac_P

```

```

99   \else
100   \expandafter \XINT_outfrac_N
101   \fi
102   {#2}{#3}[#1]%
103 }%
104 \def\XINT_outfrac_divisionbyzero #1#2{\xintError:DivisionByZero\space #1/0}%
105 \edef\XINT_outfrac_P #1#2%
106 {%
107   \noexpand\if0\noexpand\XINT_Sgn #1\noexpand\Z
108   \noexpand\expandafter\noexpand\XINT_outfrac_Zero
109   \noexpand\fi
110   \space #1/#2%
111 }%
112 \def\XINT_outfrac_Zero #1[#2]{ 0/1[0]}%
113 \def\XINT_outfrac_N #1#2%
114 {%
115   \expandafter\XINT_outfrac_N_a\expandafter
116   {\romannumeral0\XINT_opp #2}{\romannumeral0\XINT_opp #1}%
117 }%
118 \def\XINT_outfrac_N_a #1#2%
119 {%
120   \expandafter\XINT_outfrac_P\expandafter {#2}{#1}%
121 }%

```

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 early days (1.03, 2013/04/14) has gotten a lifting for release 1.2. There were two kinds of issues:

- 1) use of `\W`, `\Z`, `\T` delimiters was very poor choice as this could clash with user input,
- 2) the new `\XINT_frac_gen` handles macros (possibly empty) in the input as general as `\A.\Be\C/\D.\Ee\F`. The earlier version would not have expanded the `\B` for example (only `\A`, `\D`, `\C`, `\F`).

I wanted to make stricter the restricted `A/B[N]` case, doing no expansion of `B`, but this clashed with some established uses in the documentation like `1/\xintiiSqr{...}[0]` for example. Thus I maintained it despite overhead of having to go over `A` one more time. Careful also here about potential brace removals if one does stuff like `#1/#2#3[#4]` regarding the `#3`. And while I was at it I added `\numexpr` parsing of the `N`, which earlier was restricted to be only explicit digits, and I even allowed `[]` with an empty `N`.

This little event makes me think I should read again other remaining portions my early code, as I was still learning TeX coding at that time.

```

122 \def\XINT_inFrac {\romannumeral0\XINT_infrac }%
123 \def\XINT_infrac #1%
124 {%
125   \expandafter\XINT_infrac_fork\romannumeral`&&@#1/\XINT_W[\XINT_W\XINT_T
126 }%
127 \def\XINT_infrac_fork #1[#2%
128 {%
129   \xint_UDXINTWfork
130   #2\XINT_frac_gen

```

```

131 \XINT_W\XINT_infrac_res_a % strict A[N] or A/B[N] input
132 \krof
133 #1/#2%
134 }%
135 \def\XINT_infrac_res_a #1%
136 {%
137 \xint_gob_til_zero #1\XINT_infrac_res_zero 0\XINT_infrac_res_b #1%
138 }%
139 \def\XINT_infrac_res_zero 0\XINT_infrac_res_b #1\XINT_T {{0}{0}{1}}%
140 \def\XINT_infrac_res_b #1/#2%
141 {%
142 \xint_UDXINTWfork
143 #2\XINT_infrac_res_ca
144 \XINT_W\XINT_infrac_res_cb
145 \krof
146 #1/#2%
147 }%
148 \def\XINT_infrac_res_ca #1[#2]/\XINT_W[\XINT_W\XINT_T
149 {\expandafter{\the\numexpr 0#2}{#1}{1}}%
150 \def\XINT_infrac_res_cb #1/#2[%
151 {\expandafter\XINT_infrac_res_cc\romannumeral`&&@#2~#1[%
152 \def\XINT_infrac_res_cc #1~#2[#3]/\XINT_W[\XINT_W\XINT_T
153 {\expandafter{\the\numexpr 0#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 `\xint-expr.. \relax`

Completely rewritten for 1.2 2015/10/10. It now is able to 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`.

```

154 \def\XINT_frac_gen #1/#2%
155 {%
156 \xint_UDXINTWfork
157 #2\XINT_frac_gen_A
158 \XINT_W\XINT_frac_gen_B
159 \krof
160 #1/#2%
161 }%
162 \def\XINT_frac_gen_A #1/\XINT_W [\XINT_W {\XINT_frac_gen_C 0~1!#1ee.\XINT_W }%
163 \def\XINT_frac_gen_B #1/#2/\XINT_W[%\XINT_W
164 {%
165 \expandafter\XINT_frac_gen_Ba
166 \romannumeral`&&@#2ee.\XINT_W\XINT_Z #1ee.%\XINT_W
167 }%
168 \def\XINT_frac_gen_Ba #1.#2%
169 {%
170 \xint_UDXINTWfork
171 #2\XINT_frac_gen_Bb
172 \XINT_W\XINT_frac_gen_Bc
173 \krof
174 #1.#2%

```

7 Package *xintfrac* implementation

```

175 }%
176 \def\XINT_frac_gen_Bb #1e#2e#3\XINT_Z
177         {\expandafter\XINT_frac_gen_C\the\numexpr 0#2~#1!}%
178 \def\XINT_frac_gen_Bc #1.#2e%
179 {%
180     \expandafter\XINT_frac_gen_Bd\romannumeral`&&@#2.#1e%
181 }%
182 \def\XINT_frac_gen_Bd #1.#2e#3e#4\XINT_Z
183 {%
184     \expandafter\XINT_frac_gen_C\the\numexpr 0#3-\romannumeral0\expandafter
185     \XINT_length_loop
186     0.#1\xint_relax\xint_relax\xint_relax\xint_relax
187     \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye~#2#1!%
188 }%
189 \def\XINT_frac_gen_C #1!#2.#3%
190 {%
191     \xint_UDXINTWfork
192     #3\XINT_frac_gen_Ca
193     \XINT_W\XINT_frac_gen_Cb
194     \krof
195     #1!#2.#3%
196 }%
197 \def\XINT_frac_gen_Ca #1~#2!#3e#4e#5\XINT_T
198 {%
199     \expandafter\XINT_frac_gen_F\the\numexpr #4-#1\expandafter
200     ~\romannumeral0\XINT_num_loop
201     #2\xint_relax\xint_relax\xint_relax\xint_relax
202     \xint_relax\xint_relax\xint_relax\xint_relax\Z~#3~%
203 }%
204 \def\XINT_frac_gen_Cb #1.#2e%
205 {%
206     \expandafter\XINT_frac_gen_Cc\romannumeral`&&@#2.#1e%
207 }%
208 \def\XINT_frac_gen_Cc #1.#2~#3!#4e#5e#6\XINT_T
209 {%
210     \expandafter\XINT_frac_gen_F\the\numexpr #5-#2-%
211     \romannumeral0\XINT_length_loop
212     0.#1\xint_relax\xint_relax\xint_relax\xint_relax
213     \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye\expandafter
214     ~\romannumeral0\XINT_num_loop
215     #3\xint_relax\xint_relax\xint_relax\xint_relax
216     \xint_relax\xint_relax\xint_relax\xint_relax\Z
217     ~#4#1~%
218 }%
219 \def\XINT_frac_gen_F #1~#2%
220 {%
221     \xint_UDzerominusfork
222     #2-\XINT_frac_gen_Gdivbyzero
223     0#2{\XINT_frac_gen_G -{}}%
224     0-{\XINT_frac_gen_G {}#2}%
225     \krof #1~%
226 }%

```

```

227 \def\XINT_frac_gen_Gdivbyzero #1~#2~%
228 {%
229   \expandafter\XINT_frac_gen_Gdivbyzero_a
230   \romannumeral0\XINT_num_loop
231   #2\xint_relax\xint_relax\xint_relax\xint_relax
232   \xint_relax\xint_relax\xint_relax\xint_relax\Z~#1~%
233 }%
234 \def\XINT_frac_gen_Gdivbyzero_a #1~#2~%
235 {%
236   \xintError:DivisionByZero {#2}{#1}{0}%
237 }%
238 \def\XINT_frac_gen_G #1#2#3~#4~#5~%
239 {%
240   \expandafter\XINT_frac_gen_Ga
241   \romannumeral0\XINT_num_loop
242   #1#5\xint_relax\xint_relax\xint_relax\xint_relax
243   \xint_relax\xint_relax\xint_relax\xint_relax\Z~#3~{#2#4}%
244 }%
245 \def\XINT_frac_gen_Ga #1#2~#3~%
246 {%
247   \xint_gob_til_zero #1\XINT_frac_gen_zero 0%
248   {#3}{#1#2}%
249 }%
250 \def\XINT_frac_gen_zero 0#1#2#3{{0}{0}{1}}%

```

7.9 \XINT_factortens, \XINT_cuz_cnt

```

251 \def\XINT_factortens #1%
252 {%
253   \expandafter\XINT_cuz_cnt_loop\expandafter
254   {\expandafter}\romannumeral0\XINT_rord_main {#1%
255     \xint_relax
256     \xint_bye\xint_bye\xint_bye\xint_bye
257     \xint_bye\xint_bye\xint_bye\xint_bye
258     \xint_relax
259     \R\R\R\R\R\R\R\R\Z
260 }%
261 \def\XINT_cuz_cnt #1%
262 {%
263   \XINT_cuz_cnt_loop {#1\R\R\R\R\R\R\R\R\Z
264 }%
265 \def\XINT_cuz_cnt_loop #1#2#3#4#5#6#7#8#9%
266 {%
267   \xint_gob_til_R #9\XINT_cuz_cnt_toofara \R
268   \expandafter\XINT_cuz_cnt_checka\expandafter
269   {\the\numexpr #1+8\relax}{#2#3#4#5#6#7#8#9}%
270 }%
271 \def\XINT_cuz_cnt_toofara\R
272   \expandafter\XINT_cuz_cnt_checka\expandafter #1#2%
273 {%
274   \XINT_cuz_cnt_toofarb {#1}#2%
275 }%
276 \def\XINT_cuz_cnt_toofarb #1#2\Z {\XINT_cuz_cnt_toofarc #2\Z {#1}}%

```



```

277 \def\XINT_cuz_cnt_toofarc #1#2#3#4#5#6#7#8%
278 {%
279   \xint_gob_til_R #2\XINT_cuz_cnt_toofard 7%
280     #3\XINT_cuz_cnt_toofard 6%
281     #4\XINT_cuz_cnt_toofard 5%
282     #5\XINT_cuz_cnt_toofard 4%
283     #6\XINT_cuz_cnt_toofard 3%
284     #7\XINT_cuz_cnt_toofard 2%
285     #8\XINT_cuz_cnt_toofard 1%
286     \Z #1#2#3#4#5#6#7#8%
287 }%
288 \def\XINT_cuz_cnt_toofard #1#2\Z #3\R #4\Z #5%
289 {%
290   \expandafter\XINT_cuz_cnt_toofare
291   \the\numexpr #3\relax \R\R\R\R\R\R\R\R\Z
292   {\the\numexpr #5-#1\relax}\R\Z
293 }%
294 \def\XINT_cuz_cnt_toofare #1#2#3#4#5#6#7#8%
295 {%
296   \xint_gob_til_R #2\XINT_cuz_cnt_stopc 1%
297     #3\XINT_cuz_cnt_stopc 2%
298     #4\XINT_cuz_cnt_stopc 3%
299     #5\XINT_cuz_cnt_stopc 4%
300     #6\XINT_cuz_cnt_stopc 5%
301     #7\XINT_cuz_cnt_stopc 6%
302     #8\XINT_cuz_cnt_stopc 7%
303     \Z #1#2#3#4#5#6#7#8%
304 }%
305 \def\XINT_cuz_cnt_checka #1#2%
306 {%
307   \expandafter\XINT_cuz_cnt_checkb\the\numexpr #2\relax \Z {#1}%
308 }%
309 \def\XINT_cuz_cnt_checkb #1%
310 {%
311   \xint_gob_til_zero #1\expandafter\XINT_cuz_cnt_loop\xint_gob_til_Z
312   0\XINT_cuz_cnt_stopa #1%
313 }%
314 \def\XINT_cuz_cnt_stopa #1\Z
315 {%
316   \XINT_cuz_cnt_stopb #1\R\R\R\R\R\R\R\R\Z %
317 }%
318 \def\XINT_cuz_cnt_stopb #1#2#3#4#5#6#7#8#9%
319 {%
320   \xint_gob_til_R #2\XINT_cuz_cnt_stopc 1%
321     #3\XINT_cuz_cnt_stopc 2%
322     #4\XINT_cuz_cnt_stopc 3%
323     #5\XINT_cuz_cnt_stopc 4%
324     #6\XINT_cuz_cnt_stopc 5%
325     #7\XINT_cuz_cnt_stopc 6%
326     #8\XINT_cuz_cnt_stopc 7%
327     #9\XINT_cuz_cnt_stopc 8%
328     \Z #1#2#3#4#5#6#7#8#9%

```

```

329 }%
330 \def\XINT_cuz_cnt_stopc #1#2\Z #3\R #4\Z #5%
331 {%
332   \expandafter\XINT_cuz_cnt_stopd\expandafter
333   {\the\numexpr #5-#1}\#3%
334 }%
335 \def\XINT_cuz_cnt_stopd #1#2\R #3\Z
336 {%
337   \expandafter\space\expandafter
338   {\romannumeral0\XINT_rord_main {}}\#2%
339   \xint_relax
340   \xint_bye\xint_bye\xint_bye\xint_bye
341   \xint_bye\xint_bye\xint_bye\xint_bye
342   \xint_relax }\#1}%
343 }%

```

7.10 \XINT_addm_A

This is a routine from *xintcore* 1.0x, which is needed by *\xintFloat*, *\XINTinFloat* and *\xintRound*, for the time being. I should moved it here, now that *xintcore* has been entirely rewritten with release 1.2.

```

344 \def\XINT_addm_A #1#2#3#4#5#6%
345 {%
346   \xint_gob_til_W #3\xint_addm_az\W
347   \XINT_addm_AB #1{\#3#4#5#6}\#2}%
348 }%
349 \def\xint_addm_az\W\XINT_addm_AB #1#2%
350 {%
351   \XINT_addm_AC_checkcarry #1%
352 }%
353 \def\XINT_addm_AB #1#2#3#4\W\X\Y\Z #5#6#7#8%
354 {%
355   \XINT_addm_ABE #1#2{\#8#7#6#5}\#3}\#4\W\X\Y\Z
356 }%
357 \def\XINT_addm_ABE #1#2#3#4#5#6%
358 {%
359   \expandafter\XINT_addm_ABEA\the\numexpr #1+10#5#4#3#2+#6.%
360 }%
361 \def\XINT_addm_ABEA #1#2#3.#4%
362 {%
363   \XINT_addm_A #2{\#3#4}%
364 }%
365 \def\XINT_addm_AC_checkcarry #1%
366 {%
367   \xint_gob_til_zero #1\xint_addm_AC_nocarry 0\XINT_addm_C
368 }%
369 \def\xint_addm_AC_nocarry 0\XINT_addm_C #1#2\W\X\Y\Z
370 {%
371   \expandafter
372   \xint_cleanupzeros_andstop
373   \romannumeral0%
374   \XINT_rord_main {}}\#2%
375   \xint_relax

```

```

376      \xint_bye\xint_bye\xint_bye\xint_bye
377      \xint_bye\xint_bye\xint_bye\xint_bye
378      \xint_relax
379      #1%
380 }%
381 \def\xint_addm_C #1#2#3#4#5%
382 {%
383     \xint_gob_til_W
384     #5\xint_addm_cw
385     #4\xint_addm_cx
386     #3\xint_addm_cy
387     #2\xint_addm_cz
388     \W\xint_addm_CD {#5#4#3#2}{#1}%
389 }%
390 \def\xint_addm_CD #1%
391 {%
392     \expandafter\xint_addm_CD\the\numexpr 1+10#1.%
393 }%
394 \def\xint_addm_CC #1#2#3.#4%
395 {%
396     \XINT_addm_AC_checkcarry #2{#3#4}%
397 }%
398 \def\xint_addm_cw
399     #1\xint_addm_cx
400     #2\xint_addm_cy
401     #3\xint_addm_cz
402     \W\xint_addm_CD
403 {%
404     \expandafter\xint_addm_CDw\the\numexpr 1+#1#2#3.%
405 }%
406 \def\xint_addm_CDw #1.#2#3\X\Y\Z
407 {%
408     \XINT_addm_end #1#3%
409 }%
410 \def\xint_addm_cx
411     #1\xint_addm_cy
412     #2\xint_addm_cz
413     \W\xint_addm_CD
414 {%
415     \expandafter\xint_addm_CDx\the\numexpr 1+#1#2.%
416 }%
417 \def\xint_addm_CDx #1.#2#3\Y\Z
418 {%
419     \XINT_addm_end #1#3%
420 }%
421 \def\xint_addm_cy
422     #1\xint_addm_cz
423     \W\xint_addm_CD
424 {%
425     \expandafter\xint_addm_CDy\the\numexpr 1+#1.%
426 }%
427 \def\xint_addm_CDy #1.#2#3\Z

```

```

428 {%
429   \XINT_addm_end #1#3%
430 }%
431 \def\xint_addm_cz\W\XINT_addm_CD #1#2#3{\XINT_addm_end #1#3}%
432 \edef\XINT_addm_end #1#2#3#4#5%
433   {\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.

```

434 \def\xintRaw {\romannumeral0\xintraw }%
435 \def\xintraw
436 {%
437   \expandafter\XINT_raw\romannumeral0\XINT_infrac
438 }%
439 \def\XINT_raw #1#2#3{ #2/#3[#1]}%

```

7.12 `\xintPRaw`

1.09b

```

440 \def\xintPRaw {\romannumeral0\xintpraw }%
441 \def\xintpraw
442 {%
443   \expandafter\XINT_praw\romannumeral0\XINT_infrac
444 }%
445 \def\XINT_praw #1%
446 {%
447   \ifnum #1=\xint_c_ \expandafter\XINT_praw_a\fi \XINT_praw_A {#1}%
448 }%
449 \def\XINT_praw_A #1#2#3%
450 {%
451   \if\XINT_isOne{#3}1\expandafter\xint_firstoftwo
452     \else\expandafter\xint_secondoftwo
453   \fi { #2[#1]}{ #2/#3[#1]}%
454 }%
455 \def\XINT_praw_a\XINT_praw_A #1#2#3%
456 {%
457   \if\XINT_isOne{#3}1\expandafter\xint_firstoftwo
458     \else\expandafter\xint_secondoftwo
459   \fi { #2}{ #2/#3}%
460 }%

```

7.13 `\xintRawWithZeros`

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

```

461 \def\xintRawWithZeros {\romannumeral0\xintrawwithzeros }%
462 \def\xintrawwithzeros
463 {%
464   \expandafter\XINT_rawz\romannumeral0\XINT_infrac

```

```

465 }%
466 \def\XINT_rawz #1%
467 {%
468   \ifcase\XINT_cntSgn #1\Z
469     \expandafter\XINT_rawz_Ba
470   \or
471     \expandafter\XINT_rawz_A
472   \else
473     \expandafter\XINT_rawz_Ba
474   \fi
475   {#1}%
476 }%
477 \def\XINT_rawz_A #1#2#3{\xint_dsh {#2}{-#1}/#3}%
478 \def\XINT_rawz_Ba #1#2#3{\expandafter\XINT_rawz_Bb
479   \expandafter{\romannumeral0\xint_dsh {#3}{#1}}{#2}}%
480 \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.

```

481 \def\xintFloor {\romannumeral0\xintfloor }%
482 \def\xintfloor #1% devrais-je faire \xintREZ?
483   {\expandafter\XINT_ifloor \romannumeral0\xintraawwithzeros {#1}./1[0]}%
484 \def\xintiFloor {\romannumeral0\xintifloor }%
485 \def\xintifloor #1%
486   {\expandafter\XINT_ifloor \romannumeral0\xintraawwithzeros {#1}.}%
487 \def\XINT_ifloor #1/#2.{\xintiipro {#1}{#2}}%

```

7.15 \xintCeil, \xintiCeil

1.09a

```

488 \def\xintCeil {\romannumeral0\xintceil }%
489 \def\xintceil #1{\xintiioopp {\xintFloor {\xintOpp{#1}}}}%
490 \def\xintiCeil {\romannumeral0\xinticeil }%
491 \def\xinticeil #1{\xintiioopp {\xintiFloor {\xintOpp{#1}}}}%

```

7.16 \xintNumerator

```

492 \def\xintNumerator {\romannumeral0\xintnumerator }%
493 \def\xintnumerator
494 {%
495   \expandafter\XINT_numer\romannumeral0\XINT_infrac
496 }%
497 \def\XINT_numer #1%
498 {%
499   \ifcase\XINT_cntSgn #1\Z
500     \expandafter\XINT_numer_B
501   \or
502     \expandafter\XINT_numer_A
503   \else

```

```

504     \expandafter\XINT_numer_B
505     \fi
506     {#1}%
507 }%
508 \def\XINT_numer_A #1#2#3{\xint_dsh {#2}{-#1}}%
509 \def\XINT_numer_B #1#2#3{ #2}%

```

7.17 \xintDenominator

```

510 \def\xintDenominator {\romannumeral0\xintdenominator }%
511 \def\xintdenominator
512 {%
513     \expandafter\XINT_denom\romannumeral0\XINT_infrac
514 }%
515 \def\XINT_denom #1%
516 {%
517     \ifcase\XINT_cntSgn #1\Z
518         \expandafter\XINT_denom_B
519     \or
520         \expandafter\XINT_denom_A
521     \else
522         \expandafter\XINT_denom_B
523     \fi
524     {#1}%
525 }%
526 \def\XINT_denom_A #1#2#3{ #3}%
527 \def\XINT_denom_B #1#2#3{\xint_dsh {#3}{#1}}%

```

7.18 \xintFrac

```

528 \def\xintFrac {\romannumeral0\xintfrac }%
529 \def\xintfrac #1%
530 {%
531     \expandafter\XINT_fracfrac_A\romannumeral0\XINT_infrac {#1}%
532 }%
533 \def\XINT_fracfrac_A #1{\XINT_fracfrac_B #1\Z }%
534 \catcode`^=7
535 \def\XINT_fracfrac_B #1#2\Z
536 {%
537     \xint_gob_til_zero #1\XINT_fracfrac_C 0\XINT_fracfrac_D {10^{#1#2}}%
538 }%
539 \def\XINT_fracfrac_C 0\XINT_fracfrac_D #1#2#3%
540 {%
541     \if1\XINT_isOne {#3}%
542         \xint_afterfi {\expandafter\xint_firstoftwo_thenstop\xint_gobble_ii }%
543     \fi
544     \space
545     \frac {#2}{#3}%
546 }%
547 \def\XINT_fracfrac_D #1#2#3%
548 {%
549     \if1\XINT_isOne {#3}\XINT_fracfrac_E\fi
550     \space
551     \frac {#2}{#3}#1%
552 }%

```

```
553 \def\XINT_fracfrac_E \fi\space\frac #1#2{\fi \space #1\cdot }%
```

7.19 \xintSignedFrac

```
554 \def\xintSignedFrac {\romannumeral0\xintsignedfrac }%
555 \def\xintsignedfrac #1%
556 {%
557   \expandafter\XINT_sgnfrac_a\romannumeral0\XINT_infrac {#1}%
558 }%
559 \def\XINT_sgnfrac_a #1#2%
560 {%
561   \XINT_sgnfrac_b #2\Z {#1}%
562 }%
563 \def\XINT_sgnfrac_b #1%
564 {%
565   \xint_UDsignfork
566   #1\XINT_sgnfrac_N
567   -{\XINT_sgnfrac_P #1}%
568   \krof
569 }%
570 \def\XINT_sgnfrac_P #1\Z #2%
571 {%
572   \XINT_fracfrac_A {#2}{#1}%
573 }%
574 \def\XINT_sgnfrac_N
575 {%
576   \expandafter\xint_minus_thenstop\romannumeral0\XINT_sgnfrac_P
577 }%
```

7.20 \xintFwOver

```
578 \def\xintFwOver {\romannumeral0\xintfwover }%
579 \def\xintfwover #1%
580 {%
581   \expandafter\XINT_fwover_A\romannumeral0\XINT_infrac {#1}%
582 }%
583 \def\XINT_fwover_A #1{\XINT_fwover_B #1\Z }%
584 \def\XINT_fwover_B #1#2\Z
585 {%
586   \xint_gob_til_zero #1\XINT_fwover_C 0\XINT_fwover_D {10^{#1#2}}%
587 }%
588 \catcode\^=11
589 \def\XINT_fwover_C #1#2#3#4#5%
590 {%
591   \if0\XINT_isOne {#5}\xint_afterfi { {#4\over #5}}%
592   \else\xint_afterfi { #4}%
593   \fi
594 }%
595 \def\XINT_fwover_D #1#2#3%
596 {%
597   \if0\XINT_isOne {#3}\xint_afterfi { {#2\over #3}}%
598   \else\xint_afterfi { #2\cdot }%
599   \fi
600   #1%
601 }%
```

7.21 `\xintSignedFwOver`

```

602 \def\xintSignedFwOver {\romannumeral0\xintsignedfwover }%
603 \def\xintsignedfwover #1%
604 {%
605   \expandafter\XINT_sgnfwover_a\romannumeral0\XINT_infrac {#1}%
606 }%
607 \def\XINT_sgnfwover_a #1#2%
608 {%
609   \XINT_sgnfwover_b #2\Z {#1}%
610 }%
611 \def\XINT_sgnfwover_b #1%
612 {%
613   \xint_UDsignfork
614     #1\XINT_sgnfwover_N
615     -{\XINT_sgnfwover_P #1}%
616   \krof
617 }%
618 \def\XINT_sgnfwover_P #1\Z #2%
619 {%
620   \XINT_fwover_A {#2}{#1}%
621 }%
622 \def\XINT_sgnfwover_N
623 {%
624   \expandafter\xint_minus_thenstop\romannumeral0\XINT_sgnfwover_P
625 }%

```

7.22 `\xintREZ`

```

626 \def\xintREZ {\romannumeral0\xintrez }%
627 \def\xintrez
628 {%
629   \expandafter\XINT_rez_A\romannumeral0\XINT_infrac
630 }%
631 \def\XINT_rez_A #1#2%
632 {%
633   \XINT_rez_AB #2\Z {#1}%
634 }%
635 \def\XINT_rez_AB #1%
636 {%
637   \xint_UDzerominusfork
638     #1-\XINT_rez_zero
639     0#1\XINT_rez_neg
640     0-{\XINT_rez_B #1}%
641   \krof
642 }%
643 \def\XINT_rez_zero #1\Z #2#3{ 0/1[0]}%
644 \def\XINT_rez_neg {\expandafter\xint_minus_thenstop\romannumeral0\XINT_rez_B }%
645 \def\XINT_rez_B #1\Z
646 {%
647   \expandafter\XINT_rez_C\romannumeral0\XINT_factortens {#1}%
648 }%
649 \def\XINT_rez_C #1#2#3#4%
650 {%

```



```

651 \expandafter\XINT_rez_D\romannumeral0\XINT_factortens {#4}{#3}{#2}{#1}%
652 }%
653 \def\XINT_rez_D #1#2#3#4#5%
654 {%
655 \expandafter\XINT_rez_E\expandafter
656 {\the\numexpr #3+#4-#2}{#1}{#5}%
657 }%
658 \def\XINT_rez_E #1#2#3{ #3/#2[#1]}%

```

7.23 \xintE, \xintFloatE, \XINTinFloatE

1.07: The fraction is the first argument contrarily to \xintTrunc and \xintRound.

\xintfE (1.07) and \xintiE (1.09i) are for \xintexpr and cousins. It is quite annoying that \numexpr does not know how to deal correctly with a minus sign - as prefix: \numexpr -(1)\relax is illegal! (one can do \numexpr 0-(1)\relax).

the 1.07 \xintE puts directly its second argument in a \numexpr. The \xintfE first uses \xintNum on it, this is necessary for use in \xintexpr. (but one cannot use directly infix notation in the second argument of \xintfE)

1.09i also adds \xintFloatE and modifies \XINTinFloatfE, although currently the latter is only used from \xintfloatexpr hence always with \XINTdigits, it comes equipped with its first argument within brackets as the other \XINTinFloat... macros.

1.09m ceases here and elsewhere, also in \xintcfracname, to use \Z as delimiter in the code for the optional argument, as this is unsafe (it makes impossible to the user to employ \Z as argument to the macro). Replaced by \xint_relax. 1.09e had already done that in \xintSeq, but this should have been systematic.

1.1 modifies and moves \xintiiE to xint.sty, and cleans up some unneeded stuff, now that expressions implement scientific notation directly at the number parsing level.

```

659 \def\xintE {\romannumeral0\xinte }%
660 \def\xinte #1%
661 {%
662 \expandafter\XINT_e \romannumeral0\XINT_infrac {#1}%
663 }%
664 \def\XINT_e #1#2#3#4%
665 {%
666 \expandafter\XINT_e_end\expandafter{\the\numexpr #1+#4}{#2}{#3}%
667 }%
668 \def\XINT_e_end #1#2#3{ #2/#3[#1]}%
669 \def\xintFloatE {\romannumeral0\xintfloate }%
670 \def\xintfloate #1{\XINT_floate_chkopt #1\xint_relax }%
671 \def\XINT_floate_chkopt #1%
672 {%
673 \ifx [#1\expandafter\XINT_floate_opt
674 \else\expandafter\XINT_floate_noopt
675 \fi #1%
676 }%
677 \def\XINT_floate_noopt #1\xint_relax
678 {%
679 \expandafter\XINT_floate_a\expandafter\XINTdigits
680 \romannumeral0\XINT_infrac {#1}%
681 }%
682 \def\XINT_floate_opt [\xint_relax #1]#2%
683 {%

```

```

684 \expandafter\XINT_floate_a\expandafter
685 {\the\numexpr #1\expandafter}\romannumeral0\XINT_infrac {#2}%
686 }%
687 \def\XINT_floate_a #1#2#3#4#5%
688 {%
689 \expandafter\expandafter\expandafter\XINT_float_a
690 \expandafter\xint_exchangetwo_keepbraces\expandafter
691 {\the\numexpr #2+#5}{#1}{#3}{#4}\XINT_float_Q
692 }%
693 \def\XINTinFloatE {\romannumeral0\XINTinfloate }%
694 \def\XINTinfloate {\expandafter\XINT_infloate\romannumeral0\XINTinfloat [\XINTdigits]}%
695 \def\XINT_infloate #1[#2]#3%
696 {\expandafter\XINT_infloate_end\expandafter {\the\numexpr #3+#2}{#1}}%
697 \def\XINT_infloate_end #1#2{ #2[#1]}%

```

7.24 \xintIrr

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

```

698 \def\xintIrr {\romannumeral0\xintirr }%
699 \def\xintirr #1%
700 {%
701 \expandafter\XINT_irr_start\romannumeral0\xintrawwithzeros {#1}\Z
702 }%
703 \def\XINT_irr_start #1#2/#3\Z
704 {%
705 \if0\XINT_isOne {#3}%
706 \xint_afterfi
707 {\xint_UDsignfork
708 #1\XINT_irr_negative
709 -{\XINT_irr_nonneg #1}%
710 \krof}%
711 \else
712 \xint_afterfi{\XINT_irr_denomisone #1}%
713 \fi
714 #2\Z {#3}%
715 }%
716 \def\XINT_irr_denomisone #1\Z #2{ #1/1}% changed in 1.08
717 \def\XINT_irr_negative #1\Z #2{\XINT_irr_D #1\Z #2\Z \xint_minus_thenstop}%
718 \def\XINT_irr_nonneg #1\Z #2{\XINT_irr_D #1\Z #2\Z \space}%
719 \def\XINT_irr_D #1#2\Z #3#4\Z
720 {%
721 \xint_UDzerosfork
722 #3#1\XINT_irr_indeterminate
723 #30\XINT_irr_divisionbyzero
724 #10\XINT_irr_zero
725 00\XINT_irr_loop_a
726 \krof
727 {#3#4}{#1#2}{#3#4}{#1#2}%
728 }%
729 \def\XINT_irr_indeterminate #1#2#3#4#5{\xintError:NaN\space 0/0}%

```

7 Package *xintfrac* implementation

```
730 \def\XINT_irr_divisionbyzero #1#2#3#4#5{\xintError:DivisionByZero #5#2/0}%
731 \def\XINT_irr_zero #1#2#3#4#5{ 0/1}% changed in 1.08
732 \def\XINT_irr_loop_a #1#2%
733 {%
734   \expandafter\XINT_irr_loop_d
735   \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
736 }%
737 \def\XINT_irr_loop_d #1#2%
738 {%
739   \XINT_irr_loop_e #2\Z
740 }%
741 \def\XINT_irr_loop_e #1#2\Z
742 {%
743   \xint_gob_til_zero #1\xint_irr_loop_exit0\XINT_irr_loop_a {#1#2}%
744 }%
745 \def\xint_irr_loop_exit0\XINT_irr_loop_a #1#2#3#4%
746 {%
747   \expandafter\XINT_irr_loop_exitb\expandafter
748   {\romannumeral0\xintiique {#3}{#2}}%
749   {\romannumeral0\xintiique {#4}{#2}}%
750 }%
751 \def\XINT_irr_loop_exitb #1#2%
752 {%
753   \expandafter\XINT_irr_finish\expandafter {#2}{#1}%
754 }%
755 \def\XINT_irr_finish #1#2#3{#3#1/#2}% changed in 1.08
```

7.25 *\xintifInt*

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

```
756 \def\xintifInt {\romannumeral0\xintifint}%
757 \def\xintifint #1{\expandafter\XINT_ifint\romannumeral0\xintrawwithzeros {#1}.}%
758 \def\XINT_ifint #1/#2.%
759 {%
760   \if 0\xintiiRem {#1}{#2}%
761   \expandafter\xint_firstoftwo_thenstop
762   \else
763   \expandafter\xint_secondoftwo_thenstop
764   \fi
765 }%
```

7.26 *\xintJrr*

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

```
766 \def\xintJrr {\romannumeral0\xintjrr}%
767 \def\xintjrr #1%
768 {%
769   \expandafter\XINT_jrr_start\romannumeral0\xintrawwithzeros {#1}\Z
770 }%
771 \def\XINT_jrr_start #1#2/#3\Z
```

```

772 {%
773   \if0\XINT_isOne {#3}\xint_afterfi
774     {\xint_UDsignfork
775       #1\XINT_jrr_negative
776       -{\XINT_jrr_nonneg #1}%
777       \krof}%
778   \else
779     \xint_afterfi{\XINT_jrr_denomisine #1}%
780   \fi
781   #2\Z {#3}%
782 }%
783 \def\XINT_jrr_denomisine #1\Z #2{ #1/1}% changed in 1.08
784 \def\XINT_jrr_negative #1\Z #2{\XINT_jrr_D #1\Z #2\Z \xint_minus_thenstop }%
785 \def\XINT_jrr_nonneg #1\Z #2{\XINT_jrr_D #1\Z #2\Z \space}%
786 \def\XINT_jrr_D #1#2\Z #3#4\Z
787 {%
788   \xint_UDzerosfork
789     #3#1\XINT_jrr_indeterminate
790     #30\XINT_jrr_divisionbyzero
791     #10\XINT_jrr_zero
792     00\XINT_jrr_loop_a
793   \krof
794   {#3#4}{#1#2}1001%
795 }%
796 \def\XINT_jrr_indeterminate #1#2#3#4#5#6#7{\xintError:NaN\space 0/0}%
797 \def\XINT_jrr_divisionbyzero #1#2#3#4#5#6#7{\xintError:DivisionByZero #7#2/0}%
798 \def\XINT_jrr_zero #1#2#3#4#5#6#7{ 0/1}% changed in 1.08
799 \def\XINT_jrr_loop_a #1#2%
800 {%
801   \expandafter\XINT_jrr_loop_b
802   \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
803 }%
804 \def\XINT_jrr_loop_b #1#2#3#4#5#6#7%
805 {%
806   \expandafter \XINT_jrr_loop_c \expandafter
807     {\romannumeral0\xintiiadd{\XINT_mul_fork #4\Z #1\Z}{#6}}%
808     {\romannumeral0\xintiiadd{\XINT_mul_fork #5\Z #1\Z}{#7}}%
809     {#2}{#3}{#4}{#5}%
810 }%
811 \def\XINT_jrr_loop_c #1#2%
812 {%
813   \expandafter \XINT_jrr_loop_d \expandafter{#2}{#1}%
814 }%
815 \def\XINT_jrr_loop_d #1#2#3#4%
816 {%
817   \XINT_jrr_loop_e #3\Z {#4}{#2}{#1}%
818 }%
819 \def\XINT_jrr_loop_e #1#2\Z
820 {%
821   \xint_gob_til_zero #1\xint_jrr_loop_exit0\XINT_jrr_loop_a {#1#2}%
822 }%
823 \def\xint_jrr_loop_exit0\XINT_jrr_loop_a #1#2#3#4#5#6%

```

```

824 {%
825   \XINT_irr_finish {#3}{#4}%
826 }%

```

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.

```

827 \def\xintTFrac {\romannumeral0\xinttfrac }%
828 \def\xinttfrac #1{\expandafter\XINT_tfrac_fork\romannumeral0\xintrawwithzeros {#1}\Z }%
829 \def\XINT_tfrac_fork #1%
830 {%
831   \xint_UDzerominusfork
832   #1-\XINT_tfrac_zero
833   0#1{\xintiopp\XINT_tfrac_P }%
834   0-{\XINT_tfrac_P #1}%
835   \krof
836 }%
837 \def\XINT_tfrac_zero #1\Z { 0/1[0]}%
838 \def\XINT_tfrac_P #1/#2\Z {\expandafter\XINT_rez_AB
839   \romannumeral0\xintiirem{#1}{#2}\Z {0}{#2}}%

```

7.28 \XINTinFloatFracdigits

1.09i, for frac in \xintfloatexpr. This version computes exactly from the input the fractional part and then only converts it into a float with the asked-for number of digits. I will have to think it again some day, certainly.

1.1 removes optional argument for which there was anyhow no interface, for technical reasons having to do with \xintNewExpr.

1.1a renames the macro as \XINTinFloatFracdigits (from \XINTinFloatFrac) to be synchronous with the \XINTinFloatSqrt and \XINTinFloat habits related to \xintNewExpr problems.

Note to myself: I still have to rethink the whole thing about what is the best to do, the initial way of going through \xinttfrac was just a first implementation.

```

840 \def\XINTinFloatFracdigits {\romannumeral0\XINTinfloatfracdigits }%
841 \def\XINTinfloatfracdigits #1%
842 {%
843   \expandafter\XINT_infloatfracdg_a\expandafter {\romannumeral0\xinttfrac{#1}}%
844 }%
845 \def\XINT_infloatfracdg_a {\XINTinfloat [\XINTdigits]}%

```

7.29 \xintTrunc, \xintiTrunc

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

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

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

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

```

846 \def\xintTrunc {\romannumeral0\xinttrunc}%
847 \def\xintiTrunc {\romannumeral0\xintitrunc}%
848 \def\xinttrunc #1%
849 {%
850   \expandafter\XINT_trunc\expandafter {\the\numexpr #1}%
851 }%
852 \def\XINT_trunc #1#2%
853 {%
854   \expandafter\XINT_trunc_G
855   \romannumeral0\expandafter\XINT_trunc_A
856   \romannumeral0\XINT_infrac {#2}{#1}{#1}%
857 }%
858 \def\xintitrunc #1%
859 {%
860   \expandafter\XINT_itrunc\expandafter {\the\numexpr #1}%
861 }%
862 \def\XINT_itrunc #1#2%
863 {%
864   \expandafter\XINT_itrunc_G
865   \romannumeral0\expandafter\XINT_trunc_A
866   \romannumeral0\XINT_infrac {#2}{#1}{#1}%
867 }%
868 \def\XINT_trunc_A #1#2#3#4%
869 {%
870   \expandafter\XINT_trunc_checkifzero
871   \expandafter{\the\numexpr #1+#4}#2\Z {#3}%
872 }%
873 \def\XINT_trunc_checkifzero #1#2#3\Z
874 {%
875   \xint_gob_til_zero #2\XINT_trunc_iszero0\XINT_trunc_B {#1}{#2#3}%
876 }%
877 \def\XINT_trunc_iszero0\XINT_trunc_B #1#2#3{ 0\Z 0}%
878 \def\XINT_trunc_B #1%
879 {%
880   \ifcase\XINT_cntSgn #1\Z
881     \expandafter\XINT_trunc_D
882   \or
883     \expandafter\XINT_trunc_D
884   \else
885     \expandafter\XINT_trunc_C
886   \fi
887   {#1}%
888 }%
889 \def\XINT_trunc_C #1#2#3%
890 {%
891   \expandafter\XINT_trunc_CE\expandafter
892   {\romannumeral0\XINT_dsx_zero loop {-#1}{ }\Z {#3}}{#2}%
893 }%
894 \def\XINT_trunc_CE #1#2{\XINT_trunc_E #2.{#1}}%
895 \def\XINT_trunc_D #1#2%
896 {%

```

```

897 \expandafter\XINT_trunc_E
898 \romannumeral0\XINT_dsx_zeroloop {#1}{}\Z {#2}.%
899 }%
900 \def\XINT_trunc_E #1%
901 {%
902 \xint_UDsignfork
903 #1\XINT_trunc_Fneg
904 -{\XINT_trunc_Fpos #1}%
905 \krof
906 }%
907 \def\XINT_trunc_Fneg #1.#2{\expandafter\xint_firstoftwo_thenstop
908 \romannumeral0\XINT_div_prepare {#2}{#1}\Z \xint_minus_thenstop}%
909 \def\XINT_trunc_Fpos #1.#2{\expandafter\xint_firstoftwo_thenstop
910 \romannumeral0\XINT_div_prepare {#2}{#1}\Z \space }%
911 \def\XINT_itrunc_G #1#2\Z #3#4%
912 {%
913 \xint_gob_til_zero #1\XINT_trunc_zero 0#3#1#2%
914 }%
915 \def\XINT_trunc_zero 0#1#20{ 0}%
916 \def\XINT_trunc_G #1\Z #2#3%
917 {%
918 \xint_gob_til_zero #2\XINT_trunc_zero 0%
919 \expandafter\XINT_trunc_H\expandafter
920 {\the\numexpr\romannumeral0\xintlength {#1}-#3}{#3}{#1}#2%
921 }%
922 \def\XINT_trunc_H #1#2%
923 {%
924 \ifnum #1 > \xint_c_
925 \xint_afterfi {\XINT_trunc_Ha {#2}}%
926 \else
927 \xint_afterfi {\XINT_trunc_Hb {-#1}}% -0,--1,--2, ....
928 \fi
929 }%
930 \def\XINT_trunc_Ha
931 {%
932 \expandafter\XINT_trunc_Haa\romannumeral0\xintdecsplit
933 }%
934 \def\XINT_trunc_Haa #1#2#3%
935 {%
936 #3#1.#2%
937 }%
938 \def\XINT_trunc_Hb #1#2#3%
939 {%
940 \expandafter #3\expandafter0\expandafter.%
941 \romannumeral0\XINT_dsx_zeroloop {#1}{}\Z {}#2% #1=-0 autoris'e !
942 }%

```

7.30 \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.

```

943 \def\xintTTrunc {\romannumeral0\xintttrunc }%

```

```

944 \def\xintttrunc #1%
945 {%
946   \expandafter\XINT_itrunc_G
947   \romannumeral0\expandafter\XINT_ttrunc_A
948   \romannumeral0\XINT_infrac {#1}0% this last 0 to let \XINT_itrunc_G be happy
949 }%
950 \def\XINT_ttrunc_A #1#2#3{\XINT_trunc_checkifzero {#1}#2\Z {#3}}%

```

7.31 `\xintNum`

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

22 june 2014 (dev 1.1) I just don't understand what was the point of going through `\xintIrr` if to raise an error afterwards... and raising errors is silly, so let's do it sanely at last. In between I added `\xintiFloor`, thus, let's just let it to it.

24 october 2014 (final 1.1) (I left it taking dust since June...), I did `\xintTTrunc`, and will thus map `\xintNum` to it

```

951 \let\xintNum \xintTTrunc
952 \let\xintnum \xintttrunc

```

7.32 `\xintRound`, `\xintiRound`

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

```

953 \def\xintRound {\romannumeral0\xintround }%
954 \def\xintiRound {\romannumeral0\xintiround }%
955 \def\xintround #1%
956 {%
957   \expandafter\XINT_round\expandafter {\the\numexpr #1}%
958 }%
959 \def\XINT_round
960 {%
961   \expandafter\XINT_trunc_G\romannumeral0\XINT_round_A
962 }%
963 \def\xintiround #1%
964 {%
965   \expandafter\XINT_iround\expandafter {\the\numexpr #1}%
966 }%
967 \def\XINT_iround
968 {%
969   \expandafter\XINT_itrunc_G\romannumeral0\XINT_round_A
970 }%
971 \def\XINT_round_A #1#2%
972 {%
973   \expandafter\XINT_round_B
974   \romannumeral0\expandafter\XINT_trunc_A
975   \romannumeral0\XINT_infrac {#2}{#1+\xint_c_i}{#1}%
976 }%
977 \def\XINT_round_B #1\Z

```



```

978 {%
979   \expandafter\XINT_round_C
980   \romannumeral0\XINT_rord_main {}#1%
981   \xint_relax
982   \xint_bye\xint_bye\xint_bye\xint_bye
983   \xint_bye\xint_bye\xint_bye\xint_bye
984   \xint_relax
985   \Z
986 }%
987 \def\XINT_round_C #1%
988 {%
989   \ifnum #1<\xint_c_v
990     \expandafter\XINT_round_Daa
991   \else
992     \expandafter\XINT_round_Dba
993   \fi
994 }%
995 \def\XINT_round_Daa #1%
996 {%
997   \xint_gob_til_Z #1\XINT_round_Daz\Z \XINT_round_Da #1%
998 }%
999 \def\XINT_round_Daz\Z \XINT_round_Da \Z { 0\Z }%
1000 \def\XINT_round_Da #1\Z
1001 {%
1002   \XINT_rord_main {}#1%
1003   \xint_relax
1004   \xint_bye\xint_bye\xint_bye\xint_bye
1005   \xint_bye\xint_bye\xint_bye\xint_bye
1006   \xint_relax \Z
1007 }%
1008 \def\XINT_round_Dba #1%
1009 {%
1010   \xint_gob_til_Z #1\XINT_round_Dbz\Z \XINT_round_Db #1%
1011 }%
1012 \def\XINT_round_Dbz\Z \XINT_round_Db \Z { 1\Z }%
1013 \def\XINT_round_Db #1\Z
1014 {%
1015   \XINT_addm_A 0{}1000\W\X\Y\Z #1000\W\X\Y\Z \Z
1016 }%

```

7.33 \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`.

```

1017 \def\xintXTrunc #1#2%
1018 {%
1019   \expandafter\XINT_xtrunc_a\expandafter
1020   {\the\numexpr #1\expandafter}\romannumeral0\xintra #2}%
1021 }%
1022 \def\XINT_xtrunc_a #1%
1023 {%
1024   \expandafter\XINT_xtrunc_b\expandafter
1025   {\the\numexpr\ifnum#1<\xint_c_i \xint_c_i-\fi #1}%
1026 }%
1027 \def\XINT_xtrunc_b #1%
1028 {%
1029   \expandafter\XINT_xtrunc_c\expandafter
1030   {\the\numexpr (#1+\xint_c_ii^v)/\xint_c_ii^vi-\xint_c_i}\{#1}%
1031 }%
1032 \def\XINT_xtrunc_c #1#2%
1033 {%
1034   \expandafter\XINT_xtrunc_d\expandafter
1035   {\the\numexpr #2-\xint_c_ii^vi*#1}\{#1}\{#2}%
1036 }%
1037 \def\XINT_xtrunc_d #1#2#3#4/#5[#6]%
1038 {%
1039   \XINT_xtrunc_e #4.\{#6}\{#5}\{#3}\{#2}\{#1}%
1040 }%

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

1041 \def\XINT_xtrunc_e #1%
1042 {%
1043   \xint_UDzerominusfork
1044   #1-\XINT_xtrunc_zero

```

7 Package `xintfrac` implementation

[illegible]

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

7 Package `xintfrac` implementation

[illegible]

7 Package `xintfrac` implementation

```

1195 }%
1196 \def\XINT_xtrunc_prepare_d #1#2#3#4#5#6#7#8#9%
1197 {%
1198     \expandafter\XINT_xtrunc_prepare_e\xint_gob_til_dot #1#2#3#4#5#6#7#8#9!%
1199 }%
1200 \def\XINT_xtrunc_prepare_e #1!#2!#3#4%
1201 {%
1202     \XINT_xtrunc_prepare_f #4#3\X {#1}{#3}%
1203 }%
1204 \def\XINT_xtrunc_prepare_f #1#2#3#4#5#6#7#8#9\X
1205 {%
1206     \expandafter\square\expandafter\XINT_div_prepare_g
1207         \the\numexpr #1#2#3#4#5#6#7#8+\xint_c_i\expandafter
1208 .\the\numexpr (#1#2#3#4#5#6#7#8+\xint_c_i)\xint_c_ii\expandafter
1209 .\the\numexpr #1#2#3#4#5#6#7#8\expandafter
1210 .\romannumeral0\XINT_sepandrev_andcount
1211 #1#2#3#4#5#6#7#8#9\XINT_rsepybviii_end_A 2345678%
1212     \XINT_rsepybviii_end_B 2345678%
1213 \relax\xint_c_ii\xint_c_iii
1214     \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
1215     \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_W
1216 \X
1217 }%
1218 %%%%%%%%%%%
1219 \def\XINT_xtrunc_Pa #1#2%
1220 {%
1221     \expandafter\XINT_xtrunc_Pb\romannumeral0#1{#2}{#1}%
1222 }%
1223 \def\XINT_xtrunc_Pb #1#2#3#4{#1.\XINT_xtrunc_A {#4}{#2}{#3}}%
1224 \def\XINT_xtrunc_A #1%
1225 {%
1226     \unless\ifnum #1>\xint_c \XINT_xtrunc_transition\fi
1227     \expandafter\XINT_xtrunc_B\expandafter{\the\numexpr #1-\xint_c_i}%
1228 }%
1229 \def\XINT_xtrunc_B #1#2#3%
1230 {%
1231     \expandafter\XINT_xtrunc_D\romannumeral0#3%
1232     #{20000000000000000000000000000000000000000000000000000000000000000000000000000000}
1233     {#1}{#3}%
1234 }%
1235 \def\XINT_xtrunc_D #1#2#3%
1236 {%
1237     \romannumeral0\expandafter\XINT_dsx_zero loop\expandafter
1238         {\the\numexpr \xint_c_ii^vi-\xintLength{#1}}{}Z {}#1%
1239     \XINT_xtrunc_A {#3}{#2}%
1240 }%
1241 \def\XINT_xtrunc_transition\fi
1242     \expandafter\XINT_xtrunc_B\expandafter #1#2#3#4%
1243 {%
1244     \fi
1245     \ifnum #4=\xint_c \XINT_xtrunc_abort\fi
1246     \expandafter\XINT_xtrunc_x\expandafter

```

```

1247     {\romannumeral0\XINT_dsx_zeroloop {#4}{}\Z {#2}}{#3}{#4}%
1248 }%
1249 \def\XINT_xtrunc_x #1#2%
1250 {%
1251     \expandafter\XINT_xtrunc_y\romannumeral0#2{#1}%
1252 }%
1253 \def\XINT_xtrunc_y #1#2#3%
1254 {%
1255     \romannumeral0\expandafter\XINT_dsx_zeroloop\expandafter
1256         {\the\numexpr #3-\xintLength{#1}}{ }\Z {#1}%
1257 }%
1258 \def\XINT_xtrunc_abort\fi\expandafter\XINT_xtrunc_x\expandafter #1#2#3{\fi}%

```

7.34 `\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.

```

1259 \mathchardef\XINTdigits 16
1260 \def\xintDigits #1#2%
1261     {\afterassignment \xint_gobble_i \mathchardef\XINTdigits=}%
1262 \def\xinttheDigits {\number\XINTdigits }%

```

7.35 `\xintFloat`

1.07. Completely re-written in 1.08a, with spectacular speed gains. The earlier version was seriously silly when dealing with inputs having a big power of ten. Again some modifications in 1.08b for a better treatment of cases with long explicit numerators or denominators.

```

1263 \def\xintFloat     {\romannumeral0\xintfloat }%
1264 \def\xintfloat #1{\XINT_float_chkopt #1\xint_relax }%
1265 \def\XINT_float_chkopt #1%
1266 {%
1267     \ifx [#1\expandafter\XINT_float_opt
1268         \else\expandafter\XINT_float_noopt
1269     \fi #1%
1270 }%
1271 \def\XINT_float_noopt #1\xint_relax
1272 {%
1273     \expandafter\XINT_float_a\expandafter\XINTdigits
1274     \romannumeral0\XINT_infrac {#1}\XINT_float_Q
1275 }%
1276 \def\XINT_float_opt [\xint_relax #1]#2%
1277 {%
1278     \expandafter\XINT_float_a\expandafter
1279     {\the\numexpr #1\expandafter}%
1280     \romannumeral0\XINT_infrac {#2}\XINT_float_Q
1281 }%
1282 \def\XINT_float_a #1#2#3% #1=P, #2=n, #3=A, #4=B
1283 {%
1284     \XINT_float_fork #3\Z {#1}{#2}% #1 = precision, #2=n
1285 }%
1286 \def\XINT_float_fork #1%

```

```

1287 {%
1288   \xint_UDzerominusfork
1289   #1-\XINT_float_zero
1290   0#1\XINT_float_J
1291   0-\XINT_float_K #1}%
1292   \krof
1293 }%
1294 \def\XINT_float_zero #1\Z #2#3#4#5{ 0.e0}%
1295 \def\XINT_float_J {\expandafter\xint_minus_thenstop\romannumeral0\XINT_float_K }%
1296 \def\XINT_float_K #1\Z #2% #1=A, #2=P, #3=n, #4=B
1297 {%
1298   \expandafter\XINT_float_L\expandafter
1299   {\the\numexpr\xintLength{#1}\expandafter}\expandafter
1300   {\the\numexpr #2+\xint_c_ii}{#1}{#2}%
1301 }%
1302 \def\XINT_float_L #1#2%
1303 {%
1304   \ifnum #1>#2
1305     \expandafter\XINT_float_Ma
1306   \else
1307     \expandafter\XINT_float_Mc
1308   \fi {#1}{#2}%
1309 }%
1310 \def\XINT_float_Ma #1#2#3%
1311 {%
1312   \expandafter\XINT_float_Mb\expandafter
1313   {\the\numexpr #1-#2\expandafter\expandafter\expandafter}%
1314   \expandafter\expandafter\expandafter
1315   {\expandafter\xint_firstoftwo
1316    \romannumeral0\XINT_split_fromleft_loop {#2}{}#3\W\W\W\W\W\W\W\Z
1317   }{#2}%
1318 }%
1319 \def\XINT_float_Mb #1#2#3#4#5#6% #2=A', #3=P+2, #4=P, #5=n, #6=B
1320 {%
1321   \expandafter\XINT_float_N\expandafter
1322   {\the\numexpr\xintLength{#6}\expandafter}\expandafter
1323   {\the\numexpr #3\expandafter}\expandafter
1324   {\the\numexpr #1+#5}%
1325   {#6}{#3}{#2}{#4}%
1326 }% long de B, P+2, n', B, |A'|=P+2, A', P
1327 \def\XINT_float_Mc #1#2#3#4#5#6%
1328 {%
1329   \expandafter\XINT_float_N\expandafter
1330   {\romannumeral0\xintlength{#6}}{#2}{#5}{#6}{#1}{#3}{#4}%
1331 }% long de B, P+2, n, B, |A|, A, P
1332 \def\XINT_float_N #1#2%
1333 {%
1334   \ifnum #1>#2
1335     \expandafter\XINT_float_0
1336   \else
1337     \expandafter\XINT_float_P
1338   \fi {#1}{#2}%

```



```

1339 }%
1340 \def\XINT_float_0 #1#2#3#4%
1341 {%
1342   \expandafter\XINT_float_P\expandafter
1343   {\the\numexpr #2\expandafter}\expandafter
1344   {\the\numexpr #2\expandafter}\expandafter
1345   {\the\numexpr #3-#1+#2\expandafter\expandafter\expandafter}%
1346   \expandafter\expandafter\expandafter
1347   {\expandafter\xint_firstoftwo
1348     \romannumeral0\XINT_split_fromleft_loop {#2}{#4\W\W\W\W\W\W\W\W\Z
1349     }%
1350 }% |B|,P+2,n,B,|A|,A,P
1351 \def\XINT_float_P #1#2#3#4#5#6#7#8%
1352 {%
1353   \expandafter #8\expandafter {\the\numexpr #1-#5+#2-\xint_c_i}%
1354   {#6}{#4}{#7}{#3}%
1355 }% |B|-|A|+P+1,A,B,P,n
1356 \def\XINT_float_Q #1%
1357 {%
1358   \ifnum #1<\xint_c_
1359     \expandafter\XINT_float_Ri
1360   \else
1361     \expandafter\XINT_float_Rii
1362   \fi {#1}%
1363 }%
1364 \def\XINT_float_Ri #1#2#3%
1365 {%
1366   \expandafter\XINT_float_Sa
1367   \romannumeral0\xintiique {#2}%
1368   {\XINT_dsx_addzerosnofuss {-#1}{#3}}\Z {#1}%
1369 }%
1370 \def\XINT_float_Rii #1#2#3%
1371 {%
1372   \expandafter\XINT_float_Sa
1373   \romannumeral0\xintiique
1374   {\XINT_dsx_addzerosnofuss {#1}{#2}}{#3}\Z {#1}%
1375 }%
1376 \def\XINT_float_Sa #1%
1377 {%
1378   \if #19%
1379     \xint_afterfi {\XINT_float_Sb\XINT_float_Wb }%
1380   \else
1381     \xint_afterfi {\XINT_float_Sb\XINT_float_Wa }%
1382   \fi #1%
1383 }%
1384 \def\XINT_float_Sb #1#2\Z #3#4%
1385 {%
1386   \expandafter\XINT_float_T\expandafter
1387   {\the\numexpr #4+\xint_c_i\expandafter}%
1388   \romannumeral &&@\XINT_lenrord_loop 0{#2\Z\W\W\W\W\W\W\W\Z #1{#3}{#4}%
1389 }%
1390 \def\XINT_float_T #1#2#3%

```

```

1391 {%
1392   \ifnum #2>#1
1393     \xint_afterfi{\XINT_float_U\XINT_float_Xb}%
1394   \else
1395     \xint_afterfi{\XINT_float_U\XINT_float_Xa #3}%
1396   \fi
1397 }%
1398 \def\XINT_float_U #1#2%
1399 {%
1400   \ifnum #2<\xint_c_v
1401     \expandafter\XINT_float_Va
1402   \else
1403     \expandafter\XINT_float_Vb
1404   \fi #1%
1405 }%
1406 \def\XINT_float_Va #1#2\Z #3%
1407 {%
1408   \expandafter#1%
1409   \romannumeral0\expandafter\XINT_float_Wa
1410   \romannumeral0\XINT_rord_main {}#2%
1411   \xint_relax
1412   \xint_bye\xint_bye\xint_bye\xint_bye
1413   \xint_bye\xint_bye\xint_bye\xint_bye
1414   \xint_relax \Z
1415 }%
1416 \def\XINT_float_Vb #1#2\Z #3%
1417 {%
1418   \expandafter #1%
1419   \romannumeral0\expandafter #3%
1420   \romannumeral0\XINT_addm_A 0{}1000\W\X\Y\Z #2000\W\X\Y\Z \Z
1421 }%
1422 \def\XINT_float_Wa #1{ #1.}%
1423 \def\XINT_float_Wb #1#2%
1424   {\if #11\xint_afterfi{ 10.}\else\xint_afterfi{ #1.#2}\fi }%
1425 \def\XINT_float_Xa #1\Z #2#3#4%
1426 {%
1427   \expandafter\XINT_float_Y\expandafter
1428   {\the\numexpr #3+#4-#2}{#1}%
1429 }%
1430 \def\XINT_float_Xb #1\Z #2#3#4%
1431 {%
1432   \expandafter\XINT_float_Y\expandafter
1433   {\the\numexpr #3+#4+\xint_c_i-#2}{#1}%
1434 }%
1435 \def\XINT_float_Y #1#2{ #2e#1}%

```

7.36 \xintPFloat

1.1

```

1436 \def\xintPFloat {\romannumeral0\xintpfloat }%
1437 \def\xintpfloat #1{\XINT_pfloat_chkopt #1\xint_relax }%
1438 \def\XINT_pfloat_chkopt #1%

```

```

1439 {%
1440   \ifx [#1\expandafter\XINT_pfloat_opt
1441     \else\expandafter\XINT_pfloat_noopt
1442   \fi #1%
1443 }%
1444 \def\XINT_pfloat_noopt #1\xint_relax
1445 {%
1446   \expandafter\XINT_pfloat_a\expandafter\XINTdigits
1447   \romannumeral0\XINTinfloat [\XINTdigits]{#1}%
1448 }%
1449 \def\XINT_pfloat_opt [\xint_relax #1]#2%
1450 {%
1451   \expandafter\XINT_pfloat_a\expandafter {\the\numexpr #1\expandafter}%
1452   \romannumeral0\XINTinfloat [\numexpr #1\relax]#2}%
1453 }%
1454 \def\XINT_pfloat_a #1#2%
1455 {%
1456   \xint_UDzerominusfork
1457                                     #2-\XINT_pfloat_zero
1458                                     0#2\XINT_pfloat_neg
1459                                     0-{\XINT_pfloat_pos #2}%
1460   \krof {#1}%
1461 }%
1462 \def\XINT_pfloat_zero #1[#2]{ 0}%
1463 \def\XINT_pfloat_neg
1464   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_pfloat_pos {}}%
1465 \def\XINT_pfloat_pos #1#2#3[#4]%
1466 {%
1467   \ifnum#4>0 \xint_dothis\XINT_pfloat_no\fi
1468   \ifnum#4>\numexpr-#2\relax \xint_dothis\XINT_pfloat_b\fi
1469   \ifnum#4>\numexpr-#2-\xint_c_v\relax \xint_dothis\XINT_pfloat_B\fi
1470   \xint_orthat\XINT_pfloat_no {#2}{#4}{#1#3}%
1471 }%
1472 \def\XINT_pfloat_no #1#2%
1473 {%
1474   \expandafter\XINT_pfloat_no_b\expandafter{\the\numexpr #2+#1-\xint_c_i\relax}%
1475 }%
1476 \def\XINT_pfloat_no_b #1#2{\XINT_pfloat_no_c #2e#1}%
1477 \def\XINT_pfloat_no_c #1{ #1.}%
1478 \def\XINT_pfloat_b #1#2#3%
1479   {\expandafter\XINT_pfloat_c
1480    \romannumeral0\expandafter\XINT_split_fromleft_loop
1481    \expandafter {\the\numexpr #1+#2-\xint_c_i}#3\W\W\W\W\W\W\W\W\Z }%
1482 \def\XINT_pfloat_c #1#2{ #1.#2}% #2 peut \^etre vide
1483 \def\XINT_pfloat_B #1#2#3%
1484   {\expandafter\XINT_pfloat_C
1485    \romannumeral0\XINT_dsx_zero loop {\numexpr -#1-#2}{\Z {}}#3}%
1486 \def\XINT_pfloat_C { 0.}%

```

7.37 \XINTinFloat

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

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

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

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

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

```

1487 \def\XINTinFloat {\romannumeral0\XINTinfloat }%
1488 \def\XINTinfloat [#1]#2%
1489 {%
1490   \expandafter\XINT_infloat_a\expandafter
1491   {\the\numexpr #1\expandafter}%
1492   \romannumeral0\XINT_infrac {#2}\XINT_infloat_Q
1493 }%
1494 \def\XINT_infloat_a #1#2#3% #1=P, #2=n, #3=A, #4=B
1495 {%
1496   \XINT_infloat_fork #3\Z {#1}{#2}% #1 = precision, #2=n
1497 }%
1498 \def\XINT_infloat_fork #1%
1499 {%
1500   \xint_UDzerominusfork
1501   #1-\XINT_infloat_zero
1502   0#1\XINT_infloat_J
1503   0-\XINT_float_K #1}%
1504   \krof
1505 }%
1506 \def\XINT_infloat_zero #1\Z #2#3#4#5{ 0[0]}%
```

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

```

1507 \def\XINT_infloat_J {\expandafter-\romannumeral0\XINT_float_K }%
1508 \def\XINT_infloat_Q #1%
1509 {%
1510   \ifnum #1<\xint_c_
1511     \expandafter\XINT_infloat_Ri
1512   \else
1513     \expandafter\XINT_infloat_Rii
1514   \fi {#1}%
1515 }%
1516 \def\XINT_infloat_Ri #1#2#3%
1517 {%
1518   \expandafter\XINT_infloat_S\expandafter
1519   {\romannumeral0\xintiigo {#2}%
1520    {\XINT_dsx_addzerosnofuss {-#1}{#3}}}{#1}%
1521 }%
```

```

1522 \def\XINT_infloat_Rii #1#2#3%
1523 {%
1524   \expandafter\XINT_infloat_S\expandafter
1525   {\romannumeral0\xintiiquo
1526     {\XINT_dsx_addzerosnofuss {#1}{#2}}{#3}}{#1}%
1527 }%
1528 \def\XINT_infloat_S #1#2#3%
1529 {%
1530   \expandafter\XINT_infloat_T\expandafter
1531   {\the\numexpr #3+\xint_c_i\expandafter}%
1532   \romannumeral`&&\XINT_lenrord_loop 0{#1\Z\W\W\W\W\W\W\W\Z
1533     {#2}%
1534 }%
1535 \def\XINT_infloat_T #1#2#3%
1536 {%
1537   \ifnum #2>#1
1538     \xint_afterfi{\XINT_infloat_U\XINT_infloat_Wb}%
1539   \else
1540     \xint_afterfi{\XINT_infloat_U\XINT_infloat_Wa #3}%
1541   \fi
1542 }%
1543 \def\XINT_infloat_U #1#2%
1544 {%
1545   \ifnum #2<\xint_c_v
1546     \expandafter\XINT_infloat_Va
1547   \else
1548     \expandafter\XINT_infloat_Vb
1549   \fi #1%
1550 }%
1551 \def\XINT_infloat_Va #1#2\Z
1552 {%
1553   \expandafter#1%
1554   \romannumeral0\XINT_rord_main {#2}%
1555   \xint_relax
1556   \xint_bye\xint_bye\xint_bye\xint_bye
1557   \xint_bye\xint_bye\xint_bye\xint_bye
1558   \xint_relax \Z
1559 }%
1560 \def\XINT_infloat_Vb #1#2\Z
1561 {%
1562   \expandafter #1%
1563   \romannumeral0\XINT_addm_A 0{1000\W\X\Y\Z #2000\W\X\Y\Z \Z
1564 }%
1565 \def\XINT_infloat_Wa #1\Z #2#3%
1566 {%
1567   \expandafter\XINT_infloat_X\expandafter
1568   {\the\numexpr #3+\xint_c_i-#2}{#1}%
1569 }%
1570 \def\XINT_infloat_Wb #1\Z #2#3%
1571 {%
1572   \expandafter\XINT_infloat_X\expandafter
1573   {\the\numexpr #3+\xint_c_ii-#2}{#1}%

```

```

1574 }%
1575 \def\XINT_infloat_X #1#2{ #2[#1]}%

```

7.38 \xintAdd

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

```

1576 \def\xintAdd {\romannumeral0\xintadd }%
1577 \def\xintadd #1{\expandafter\xint_fadd\romannumeral0\xintra {#1}}%
1578 \def\xint_fadd #1{\xint_gob_til_zero #1\XINT_fadd_Azero 0\XINT_fadd_a #1}%
1579 \def\XINT_fadd_Azero #1{\xintra }%
1580 \def\XINT_fadd_a #1/#2[#3]#4%
1581   {\expandafter\XINT_fadd_b\romannumeral0\xintra {#4}{#3}{#1}{#2}}%
1582 \def\XINT_fadd_b #1{\xint_gob_til_zero #1\XINT_fadd_Bzero 0\XINT_fadd_c #1}%
1583 \def\XINT_fadd_Bzero #1]#2#3#4{ #3/#4[#2]}%
1584 \def\XINT_fadd_c #1/#2[#3]#4%
1585 {%
1586   \expandafter\XINT_fadd_Aa\expandafter{\the\numexpr #4-#3}{#3}{#4}{#1}{#2}%
1587 }%
1588 \def\XINT_fadd_Aa #1%
1589 {%
1590   \ifcase\XINT_cntSgn #1\Z
1591     \expandafter\XINT_fadd_B
1592   \or
1593     \expandafter \XINT_fadd_Ba
1594   \else
1595     \expandafter \XINT_fadd_Bb
1596   \fi {#1}%
1597 }%
1598 \def\XINT_fadd_B #1#2#3#4#5#6#7{\XINT_fadd_C {#4}{#5}{#7}{#6}[#3]}%
1599 \def\XINT_fadd_Ba #1#2#3#4#5#6#7%
1600 {%
1601   \expandafter\XINT_fadd_C\expandafter
1602     {\romannumeral0\XINT_dsx_zeroloop {#1}{}\Z {#6}}%
1603   {#7}{#5}{#4}[#2]}%
1604 }%
1605 \def\XINT_fadd_Bb #1#2#3#4#5#6#7%
1606 {%
1607   \expandafter\XINT_fadd_C\expandafter
1608     {\romannumeral0\XINT_dsx_zeroloop {-#1}{}\Z {#4}}%
1609   {#5}{#7}{#6}[#3]}%
1610 }%
1611 \def\XINT_fadd_C #1#2#3%
1612 {%
1613   \ifcase\romannumeral0\xintiicomp {#2}{#3} %<- intentional space here.
1614     \expandafter\XINT_fadd_eq
1615   \or\expandafter\XINT_fadd_D
1616   \else\expandafter\XINT_fadd_Da
1617   \fi {#2}{#3}{#1}%
1618 }%
1619 \def\XINT_fadd_eq #1#2#3#4#5%
1620 {%
1621   \expandafter\XINT_fadd_G

```

```

1622 \romannumeral0\xintiiadd {#3}{#4}/#1#[#5]%
1623 }%
1624 \def\xINT_fadd_D #1#2%
1625 {%
1626 \expandafter\xINT_fadd_E\romannumeral0\xINT_div_prepare {#2}{#1}{#1}{#2}%
1627 }%
1628 \def\xINT_fadd_E #1#2%
1629 {%
1630 \if0\xINT_Sgn #2\Z
1631 \expandafter\xINT_fadd_F
1632 \else\expandafter\xINT_fadd_K
1633 \fi {#1}%
1634 }%
1635 \def\xINT_fadd_F #1#2#3#4#5#6%
1636 {%
1637 \expandafter\xINT_fadd_G
1638 \romannumeral0\xintiiadd {\xintiiMul {#5}{#1}}{#4}/#2#[#6]%
1639 }%
1640 \def\xINT_fadd_Da #1#2%
1641 {%
1642 \expandafter\xINT_fadd_Ea\romannumeral0\xINT_div_prepare {#1}{#2}{#1}{#2}%
1643 }%
1644 \def\xINT_fadd_Ea #1#2%
1645 {%
1646 \if0\xINT_Sgn #2\Z
1647 \expandafter\xINT_fadd_Fa
1648 \else\expandafter\xINT_fadd_K
1649 \fi {#1}%
1650 }%
1651 \def\xINT_fadd_Fa #1#2#3#4#5#6%
1652 {%
1653 \expandafter\xINT_fadd_G
1654 \romannumeral0\xintiiadd {\xintiiMul {#4}{#1}}{#5}/#3#[#6]%
1655 }%
1656 \def\xINT_fadd_G #1{\if0#1\xINT_fadd_iszero\fi\space #1}%
1657 \def\xINT_fadd_K #1#2#3#4#5%
1658 {%
1659 \expandafter\xINT_fadd_L
1660 \romannumeral0\xintiiadd {\xintiiMul {#2}{#5}}{\xintiiMul {#3}{#4}}.%
1661 {{#2}{#3}}}%
1662 }%
1663 \def\xINT_fadd_L #1{\if0#1\xINT_fadd_iszero\fi \xINT_fadd_M #1}%
1664 \def\xINT_fadd_M #1.#2{\expandafter\xINT_fadd_N \expandafter
1665 {\romannumeral0\xintiimul #2}{#1}}%
1666 \def\xINT_fadd_N #1#2{ #2/#1}%
1667 \edef\xINT_fadd_iszero\fi #1[#2]{\noexpand\fi\space 0/1[0]}% ou [#2] originel?

```

7.39 \xintSub

refait dans 1.1 pour vérifier si summands nuls.

```

1668 \def\xintSub {\romannumeral0\xintsub }%
1669 \def\xintsub #1{\expandafter\xint_fsub\romannumeral0\xintraw {#1}}%

```

```

1670 \def\xint_fsub #1{\xint_gob_til_zero #1\XINT_fsub_Azero 0\XINT_fsub_a #1}%
1671 \def\XINT_fsub_Azero #1]{\xintopp }%
1672 \def\XINT_fsub_a #1/#2[#3]#4%
1673   {\expandafter\XINT_fsub_b\romannumeral0\xintra {#4}{#3}{#1}{#2}}%
1674 \def\XINT_fsub_b #1{\xint_UDzerominusfork
1675   #1-\XINT_fadd_Bzero
1676   0#1\XINT_fadd_c
1677   0-\XINT_fadd_c -#1}%
1678   \krof }%

```

7.40 \xintSum

```

1679 \def\xintSum {\romannumeral0\xintsum }%
1680 \def\xintsum #1{\xintsumexpr #1\relax }%
1681 \def\xintSumExpr {\romannumeral0\xintsumexpr }%
1682 \def\xintsumexpr {\expandafter\XINT_fsumexpr\romannumeral`&&@}%
1683 \def\XINT_fsumexpr {\XINT_fsum_loop_a {0/1[0]}}%
1684 \def\XINT_fsum_loop_a #1#2%
1685 {%
1686   \expandafter\XINT_fsum_loop_b \romannumeral`&&@#2\Z {#1}%
1687 }%
1688 \def\XINT_fsum_loop_b #1%
1689 {%
1690   \xint_gob_til_relax #1\XINT_fsum_finished\relax
1691   \XINT_fsum_loop_c #1%
1692 }%
1693 \def\XINT_fsum_loop_c #1\Z #2%
1694 {%
1695   \expandafter\XINT_fsum_loop_a\expandafter{\romannumeral0\xintadd {#2}{#1}}%
1696 }%
1697 \def\XINT_fsum_finished #1\Z #2{ #2}%

```

7.41 \xintMul

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

```

1698 \def\xintMul {\romannumeral0\xintmul }%
1699 \def\xintmul #1{\expandafter\xint_fmulo\romannumeral0\xintra {#1}.}%
1700 \def\xint_fmulo #1{\xint_gob_til_zero #1\XINT_fmulo_zero 0\XINT_fmulo_a #1}%
1701 \def\XINT_fmulo_a #1[#2].#3%
1702   {\expandafter\XINT_fmulo_b\romannumeral0\xintra {#3}#1[#2.]}%
1703 \def\XINT_fmulo_b #1{\xint_gob_til_zero #1\XINT_fmulo_zero 0\XINT_fmulo_c #1}%
1704 \def\XINT_fmulo_c #1/#2[#3]#4/#5[#6.]%
1705 {%
1706   \expandafter\XINT_fmulo_d
1707   \expandafter{\the\numexpr #3+#6\expandafter}%
1708   \expandafter{\romannumeral0\xintiimulo {#5}{#2}}%
1709   {\romannumeral0\xintiimulo {#4}{#1}}%
1710 }%
1711 \def\XINT_fmulo_d #1#2#3%
1712 {%
1713   \expandafter \XINT_fmulo_e \expandafter{#3}{#1}{#2}%
1714 }%
1715 \def\XINT_fmulo_e #1#2{\XINT_outfrac {#2}{#1}}%

```



```
1716 \def\XINT_fmulo_zero #1.#2{ 0/1[0]}%
```

7.42 \xintSqr

1.1 modifs comme xintMul

```
1717 \def\xintSqr {\romannumeral0\xintsqr }%
1718 \def\xintsqr #1{\expandafter\xint_fsqr\romannumeral0\xintra {#1}}%
1719 \def\xint_fsqr #1{\xint_gob_til_zero #1\XINT_fsqr_zero 0\XINT_fsqr_a #1}%
1720 \def\xint_fsqr_a #1/#2[#3]%
1721 {%
1722   \expandafter\XINT_fsqr_b
1723   \expandafter{\the\numexpr #3+#3\expandafter}%
1724   \expandafter{\romannumeral0\xintiisqr {#2}}%
1725   {\romannumeral0\xintiisqr {#1}}%
1726 }%
1727 \def\XINT_fsqr_b #1#2#3{\expandafter \XINT_fmulo_e \expandafter{#3}{#1}{#2}}%
1728 \def\XINT_fsqr_zero #1{\ 0/1[0]}%
```

7.43 \xintPow

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

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

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

```
1729 \def\xintPow {\romannumeral0\xintpow }%
1730 \def\xintpow #1%
1731 {%
1732   \expandafter\xint_fpow\expandafter {\romannumeral0\XINT_infrac {#1}}%
1733 }%
1734 \def\xint_fpow #1#2%
1735 {%
1736   \expandafter\XINT_fpow_fork\the\numexpr \xintNum{#2}\relax\Z #1%
1737 }%
1738 \def\XINT_fpow_fork #1#2\Z
1739 {%
1740   \xint_UDzerominusfork
1741   #1-\XINT_fpow_zero
1742   0#1\XINT_fpow_neg
1743   0-{\XINT_fpow_pos #1}%
1744   \krof
1745   {#2}%
1746 }%
1747 \def\XINT_fpow_zero #1#2#3#4{ 1/1[0]}%
1748 \def\XINT_fpow_pos #1#2#3#4#5%
1749 {%
1750   \expandafter\XINT_fpow_pos_A\expandafter
1751   {\the\numexpr #1#2*#3\expandafter}\expandafter
1752   {\romannumeral0\xintiipow {#5}{#1#2}}%
```

```

1753   {\romannumeral0\xintiipow {#4}{#1#2}}%
1754 }%
1755 \def\XINT_fpow_neg #1#2#3#4%
1756 {%
1757   \expandafter\XINT_fpow_pos_A\expandafter
1758   {\the\numexpr -#1*#2\expandafter}\expandafter
1759   {\romannumeral0\xintiipow {#3}{#1}}%
1760   {\romannumeral0\xintiipow {#4}{#1}}%
1761 }%
1762 \def\XINT_fpow_pos_A #1#2#3%
1763 {%
1764   \expandafter\XINT_fpow_pos_B\expandafter {#3}{#1}{#2}%
1765 }%
1766 \def\XINT_fpow_pos_B #1#2{\XINT_outfrac {#2}{#1}}%

```

7.44 *\xintFac*

1.07: to be used by the *\xintexpr* scanner which needs to be able to apply *\xintFac* to a fraction which is an integer in disguise; so we use *\xintNum* and not only *\numexpr*. Je modifie cela dans 1.08b, au lieu d'avoir un *\xintfFac* spécialement pour *\xintexpr*, tout simplement j'étends *\xintFac* comme les autres macros, pour qu'elle utilise *\xintNum*. Remarque: bien sûr si on n'utilisait pas *\xintNum* ici, il faudrait que la macro s'appelât *\xintiiFac*. Qui est celle définit dans *xintcore.sty*. Donc ce n'est pas seulement le problème de l'utilisation dans *\xintexpr*..*\relax*.

```

1767 \def\xintFac {\romannumeral0\xintfac }%
1768 \def\xintfac #1%
1769 {%
1770   \expandafter\XINT_fac_fork\expandafter{\the\numexpr \xintNum{#1}}%
1771 }%

```

7.45 *\xintPrd*

```

1772 \def\xintPrd {\romannumeral0\xintprd }%
1773 \def\xintprd #1{\xintprdexpr #1\relax }%
1774 \def\xintPrdExpr {\romannumeral0\xintprdexpr }%
1775 \def\xintprdexpr {\expandafter\XINT_fprdexpr \romannumeral`&&@}%
1776 \def\XINT_fprdexpr {\XINT_fprod_loop_a {1/1[0]}}%
1777 \def\XINT_fprod_loop_a #1#2%
1778 {%
1779   \expandafter\XINT_fprod_loop_b \romannumeral`&&@#2\Z {#1}%
1780 }%
1781 \def\XINT_fprod_loop_b #1%
1782 {%
1783   \xint_gob_til_relax #1\XINT_fprod_finished\relax
1784   \XINT_fprod_loop_c #1%
1785 }%
1786 \def\XINT_fprod_loop_c #1\Z #2%
1787 {%
1788   \expandafter\XINT_fprod_loop_a\expandafter{\romannumeral0\xintmul {#1}{#2}}%
1789 }%
1790 \def\XINT_fprod_finished #1\Z #2{ #2}%

```

7.46 *\xintDiv*

```

1791 \def\xintDiv {\romannumeral0\xintdiv }%
1792 \def\xintdiv #1%
1793 {%
1794   \expandafter\xint_fdiv\expandafter {\romannumeral0\XINT_infrac {#1}}%
1795 }%
1796 \def\xint_fdiv #1#2%
1797   {\expandafter\XINT_fdiv_A\romannumeral0\XINT_infrac {#2}#1}%
1798 \def\XINT_fdiv_A #1#2#3#4#5#6%
1799 {%
1800   \expandafter\XINT_fdiv_B
1801   \expandafter{\the\numexpr #4-#1\expandafter}%
1802   \expandafter{\romannumeral0\xintiimul {#2}{#6}}%
1803   {\romannumeral0\xintiimul {#3}{#5}}%
1804 }%
1805 \def\XINT_fdiv_B #1#2#3%
1806 {%
1807   \expandafter\XINT_fdiv_C
1808   \expandafter{#3}{#1}{#2}%
1809 }%
1810 \def\XINT_fdiv_C #1#2{\XINT_outfrac {#2}{#1}}%

```

7.47 \xintDivFloor

1.1

```

1811 \def\xintDivFloor {\romannumeral0\xintdivfloor }%
1812 \def\xintdivfloor #1#2{\xintfloor{\xintDiv {#1}{#2}}}%

```

7.48 \xintDivTrunc

1.1. \xintttrunc rather than \xintitrunc0 in 1.1a

```

1813 \def\xintDivTrunc {\romannumeral0\xintdivtrunc }%
1814 \def\xintdivtrunc #1#2{\xintttrunc {\xintDiv {#1}{#2}}}%

```

7.49 \xintDivRound

1.1

```

1815 \def\xintDivRound {\romannumeral0\xintdivround }%
1816 \def\xintdivround #1#2{\xintiround 0{\xintDiv {#1}{#2}}}%

```

7.50 \xintMod

1.1. \xintMod {q1}{q2} computes $q_2 * t(q_1/q_2)$ with $t(q_1/q_2)$ equal to the truncated division of two arbitrary fractions q_1 and q_2 . We put some efforts into minimizing the amount of computations.

```

1817 \def\xintMod {\romannumeral0\xintmod }%
1818 \def\xintmod #1{\expandafter\XINT_mod_a\romannumeral0\xintraw{#1}.}%
1819 \def\XINT_mod_a #1#2.#3%
1820   {\expandafter\XINT_mod_b\expandafter #1\romannumeral0\xintraw{#3}#2.}%
1821 \def\XINT_mod_b #1#2% #1 de A, #2 de B.
1822 {%

```

```

1823 \if0#2\xint_dothis\XINT_mod_divbyzero\fi
1824 \if0#1\xint_dothis\XINT_mod_aiszero\fi
1825 \if-#2\xint_dothis{\XINT_mod_bneg #1}\fi
1826 \xint_orthat{\XINT_mod_bpos #1#2}%
1827 }%
1828 \def\XINT_mod_bpos #1%
1829 {%
1830 \xint_UDsignfork
1831 #1{\xintiiopt\XINT_mod_pos {}}%
1832 -{\XINT_mod_pos #1}%
1833 \krof
1834 }%
1835 \def\XINT_mod_bneg #1%
1836 {%
1837 \xint_UDsignfork
1838 #1{\xintiiopt\XINT_mod_pos {}}%
1839 -{\XINT_mod_pos #1}%
1840 \krof
1841 }%
1842 \def\XINT_mod_divbyzero #1.{\xintError:DivisionByZero\space 0/1[0]}%
1843 \def\XINT_mod_aiszero #1.{ 0/1[0]}%
1844 \def\XINT_mod_pos #1#2/#3[#4]#5/#6[#7].%
1845 {%
1846 \expandafter\XINT_mod_pos_a
1847 \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.\expandafter
1848 {\romannumeral0\xintiimul {#6}{#3}}% n fois u
1849 {\xintiiE{\xintiiMul {#1#5}{#3}}{#7-#4}}% m fois u
1850 {\xintiiE{\xintiiMul {#2}{#6}}{#4-#7}}% t fois n
1851 }%
1852 \def\XINT_mod_pos_a #1.#2#3#4{\xintiirem {#3}{#4}/#2[#1]}%

```

7.51 \XINTinFloatMod

Pour emploi dans *xintexpr* 1.1

```

1853 \def\XINTinFloatMod {\romannumeral0\XINTinfloatmod [\XINTdigits]}%
1854 \def\XINTinfloatmod [#1]#2#3{\expandafter\XINT_infloatmod\expandafter
1855 {\romannumeral0\XINTinfloat[#1]{#2}}%
1856 {\romannumeral0\XINTinfloat[#1]{#3}}{#1}}%
1857 \def\XINT_infloatmod #1#2{\expandafter\XINT_infloatmod_a\expandafter {#2}{#1}}%
1858 \def\XINT_infloatmod_a #1#2#3{\XINTinfloat [#3]{\xintMod {#2}{#1}}}%

```

7.52 \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.

```

1859 \def\xintIsOne {\romannumeral0\xintisone }%
1860 \def\xintisone #1{\expandafter\XINT_fracisone
1861 \romannumeral0\xintraawithzeros{#1}\Z }%
1862 \def\XINT_fracisone #1/#2\Z
1863 {\if0\XINT_Cmp {#1}{#2}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%

```

7.53 `\xintGeq`

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

```

1864 \def\xintGeq {\romannumeral0\xintgeq }%
1865 \def\xintgeq #1%
1866 {%
1867   \expandafter\xint_fgeq\expandafter {\romannumeral0\xintabs {#1}}%
1868 }%
1869 \def\xint_fgeq #1#2%
1870 {%
1871   \expandafter\XINT_fgeq_A \romannumeral0\xintabs {#2}#1%
1872 }%
1873 \def\XINT_fgeq_A #1%
1874 {%
1875   \xint_gob_til_zero #1\XINT_fgeq_Zii 0%
1876   \XINT_fgeq_B #1%
1877 }%
1878 \def\XINT_fgeq_Zii 0\XINT_fgeq_B #1[#2]#3[#4]{ 1}%
1879 \def\XINT_fgeq_B #1/#2[#3]#4#5/#6[#7]%
1880 {%
1881   \xint_gob_til_zero #4\XINT_fgeq_Zi 0%
1882   \expandafter\XINT_fgeq_C\expandafter
1883   {\the\numexpr #7-#3\expandafter}\expandafter
1884   {\romannumeral0\xintiimul {#4#5}{#2}}%
1885   {\romannumeral0\xintiimul {#6}{#1}}%
1886 }%
1887 \def\XINT_fgeq_Zi 0#1#2#3#4#5#6#7{ 0}%
1888 \def\XINT_fgeq_C #1#2#3%
1889 {%
1890   \expandafter\XINT_fgeq_D\expandafter
1891   {#3}{#1}{#2}%
1892 }%
1893 \def\XINT_fgeq_D #1#2#3%
1894 {%
1895   \expandafter\XINT_cntSgnFork\romannumeral`&&\expandafter\XINT_cntSgn
1896   \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\Z
1897   { 0}{\XINT_fgeq_E #2\Z {#3}{#1}}{ 1}%
1898 }%
1899 \def\XINT_fgeq_E #1%
1900 {%
1901   \xint_UDsignfork
1902   #1\XINT_fgeq_Fd
1903   -{\XINT_fgeq_Fn #1}%
1904   \krof
1905 }%
1906 \def\XINT_fgeq_Fd #1\Z #2#3%
1907 {%
1908   \expandafter\XINT_fgeq_Fe\expandafter
1909   {\romannumeral0\XINT_dsx_addzerosnofuss {#1}{#3}}{#2}%
1910 }%
1911 \def\XINT_fgeq_Fe #1#2{\XINT_geq_pre {#2}{#1}}%
1912 \def\XINT_fgeq_Fn #1\Z #2#3%

```

```

1913 {%
1914   \expandafter\XINT_geq_pre\expandafter
1915   {\romannumeral0\XINT_dsx_addzerosnofuss {#1}{#2}}{#3}%
1916 }%

```

7.54 \xintMax

Rewritten completely in 1.08a.

```

1917 \def\xintMax {\romannumeral0\xintmax }%
1918 \def\xintmax #1%
1919 {%
1920   \expandafter\xint_fmax\expandafter {\romannumeral0\xintraw {#1}}%
1921 }%
1922 \def\xint_fmax #1#2%
1923 {%
1924   \expandafter\XINT_fmax_A\romannumeral0\xintraw {#2}#1%
1925 }%
1926 \def\XINT_fmax_A #1#2/#3[#4]#5#6/#7[#8]%
1927 {%
1928   \xint_UDsignsfork
1929     #1#5\XINT_fmax_minusminus
1930     -#5\XINT_fmax_firstneg
1931     #1-\XINT_fmax_secondneg
1932     --\XINT_fmax_nonneg_a
1933   \krof
1934   #1#5{#2/#3[#4]}{#6/#7[#8]}%
1935 }%
1936 \def\XINT_fmax_minusminus --%
1937   {\expandafter\xint_minus_thenstop\romannumeral0\XINT_fmin_nonneg_b }%
1938 \def\XINT_fmax_firstneg #1-#2#3{ #1#2}%
1939 \def\XINT_fmax_secondneg -#1#2#3{ #1#3}%
1940 \def\XINT_fmax_nonneg_a #1#2#3#4%
1941 {%
1942   \XINT_fmax_nonneg_b {#1#3}{#2#4}%
1943 }%
1944 \def\XINT_fmax_nonneg_b #1#2%
1945 {%
1946   \if0\romannumeral0\XINT_fgeq_A #1#2%
1947     \xint_afterfi{ #1}%
1948   \else \xint_afterfi{ #2}%
1949   \fi
1950 }%

```

7.55 \xintMaxof

```

1951 \def\xintMaxof {\romannumeral0\xintmaxof }%
1952 \def\xintmaxof #1{\expandafter\XINT_maxof_a\romannumeral`&&@#1\relax }%
1953 \def\XINT_maxof_a #1{\expandafter\XINT_maxof_b\romannumeral0\xintraw{#1}\Z }%
1954 \def\XINT_maxof_b #1\Z #2%
1955   {\expandafter\XINT_maxof_c\romannumeral`&&@#2\Z {#1}\Z}%
1956 \def\XINT_maxof_c #1%
1957   {\xint_gob_til_relax #1\XINT_maxof_e\relax\XINT_maxof_d #1}%

```

```

1958 \def\XINT_maxof_d #1\Z
1959      {\expandafter\XINT_maxof_b\romannumeral0\xintmax {#1}}%
1960 \def\XINT_maxof_e #1\Z #2\Z { #2}%

```

7.56 \xintMin

Rewritten completely in 1.08a.

```

1961 \def\xintMin {\romannumeral0\xintmin }%
1962 \def\xintmin #1%
1963 {%
1964     \expandafter\xint_fmin\expandafter {\romannumeral0\xintra {#1}}%
1965 }%
1966 \def\xint_fmin #1#2%
1967 {%
1968     \expandafter\XINT_fmin_A\romannumeral0\xintra {#2}#1%
1969 }%
1970 \def\XINT_fmin_A #1#2/#3[#4]#5#6/#7[#8]%
1971 {%
1972     \xint_UDsignsfork
1973     #1#5\XINT_fmin_minusminus
1974     -#5\XINT_fmin_firstneg
1975     #1-\XINT_fmin_secondneg
1976     --\XINT_fmin_nonneg_a
1977     \krof
1978     #1#5{#2/#3[#4]}{#6/#7[#8]}%
1979 }%
1980 \def\XINT_fmin_minusminus --%
1981     {\expandafter\xint_minus_thenstop\romannumeral0\XINT_fmax_nonneg_b }%
1982 \def\XINT_fmin_firstneg #1-#2#3{ -#3}%
1983 \def\XINT_fmin_secondneg -#1#2#3{ -#2}%
1984 \def\XINT_fmin_nonneg_a #1#2#3#4%
1985 {%
1986     \XINT_fmin_nonneg_b {#1#3}{#2#4}%
1987 }%
1988 \def\XINT_fmin_nonneg_b #1#2%
1989 {%
1990     \if0\romannumeral0\XINT_fgeq_A #1#2%
1991         \xint_afterfi{ #2}%
1992     \else \xint_afterfi{ #1}%
1993     \fi
1994 }%

```

7.57 \xintMinof

```

1995 \def\xintMinof {\romannumeral0\xintminof }%
1996 \def\xintminof #1{\expandafter\XINT_minof_a\romannumeral`&&@#1\relax }%
1997 \def\XINT_minof_a #1{\expandafter\XINT_minof_b\romannumeral0\xintra{#1}\Z }%
1998 \def\XINT_minof_b #1\Z #2%
1999     {\expandafter\XINT_minof_c\romannumeral`&&@#2\Z {#1}\Z}%
2000 \def\XINT_minof_c #1%
2001     {\xint_gob_til_relax #1\XINT_minof_e\relax\XINT_minof_d #1}%
2002 \def\XINT_minof_d #1\Z
2003     {\expandafter\XINT_minof_b\romannumeral0\xintmin {#1}}%

```

2004 \def\XINT_minof_e #1\Z #2\Z { #2}%

7.58 \xintCmp

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

```

2005 %\def\xintCmp {\romannumeral0\xintcmp }%
2006 \def\xintcmp #1%
2007 {%
2008   \expandafter\xint_fcmp\expandafter {\romannumeral0\xintraw {#1}}%
2009 }%
2010 \def\xint_fcmp #1#2%
2011 {%
2012   \expandafter\XINT_fcmp_A\romannumeral0\xintraw {#2}#1%
2013 }%
2014 \def\XINT_fcmp_A #1#2/#3[#4]#5#6/#7[#8]%
2015 {%
2016   \xint_UDsignsfork
2017     #1#5\XINT_fcmp_minusminus
2018     -#5\XINT_fcmp_firstneg
2019     #1-\XINT_fcmp_secondneg
2020     --\XINT_fcmp_nonneg_a
2021   \krof
2022   #1#5{#2/#3[#4]}{#6/#7[#8]}%
2023 }%
2024 \def\XINT_fcmp_minusminus --#1#2{\XINT_fcmp_B #2#1}%
2025 \def\XINT_fcmp_firstneg #1-#2#3{ -1}%
2026 \def\XINT_fcmp_secondneg -#1#2#3{ 1}%
2027 \def\XINT_fcmp_nonneg_a #1#2%
2028 {%
2029   \xint_UDzerosfork
2030     #1#2\XINT_fcmp_zerozero
2031     0#2\XINT_fcmp_firstzero
2032     #10\XINT_fcmp_secondzero
2033     00\XINT_fcmp_pos
2034   \krof
2035   #1#2%
2036 }%
2037 \def\XINT_fcmp_zerozero #1#2#3#4{ 0}% 1.08b had some [ and ] here!!!
2038 \def\XINT_fcmp_firstzero #1#2#3#4{ -1}% incredibly I never saw that until
2039 \def\XINT_fcmp_secondzero #1#2#3#4{ 1}% preparing 1.09a.
2040 \def\XINT_fcmp_pos #1#2#3#4%
2041 {%
2042   \XINT_fcmp_B #1#3#2#4%
2043 }%
2044 \def\XINT_fcmp_B #1/#2[#3]#4/#5[#6]%
2045 {%
2046   \expandafter\XINT_fcmp_C\expandafter
2047   {\the\numexpr #6-#3\expandafter}\expandafter
2048   {\romannumeral0\xintiimul {#4}{#2}}%
2049   {\romannumeral0\xintiimul {#5}{#1}}%
2050 }%
2051 \def\XINT_fcmp_C #1#2#3%
2052 {%

```



```

2053 \expandafter\XINT_fcmp_D\expandafter
2054 {#3}{#1}{#2}%
2055 }%
2056 \def\XINT_fcmp_D #1#2#3%
2057 {%
2058 \expandafter\XINT_cntSgnFork\romannumeral`&&\expandafter\XINT_cntSgn
2059 \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\Z
2060 { -1}{\XINT_fcmp_E #2\Z {#3}{#1}}{ 1}%
2061 }%
2062 \def\XINT_fcmp_E #1%
2063 {%
2064 \xint_UDsignfork
2065 #1\XINT_fcmp_Fd
2066 -{\XINT_fcmp_Fn #1}%
2067 \krof
2068 }%
2069 \def\XINT_fcmp_Fd #1\Z #2#3%
2070 {%
2071 \expandafter\XINT_fcmp_Fe\expandafter
2072 {\romannumeral0\XINT_dsx_addzerosnofuss {#1}{#3}}{#2}%
2073 }%
2074 \def\XINT_fcmp_Fe #1#2{\xintiicomp {#2}{#1}}%
2075 \def\XINT_fcmp_Fn #1\Z #2#3%
2076 {%
2077 \expandafter\xintiicomp\expandafter
2078 {\romannumeral0\XINT_dsx_addzerosnofuss {#1}{#2}}{#3}%
2079 }%

```

7.59 *\xintAbs*

```

2080 \def\xintAbs {\romannumeral0\xintabs }%
2081 \def\xintabs #1{\expandafter\XINT_abs\romannumeral0\xintraw {#1}}%

```

7.60 *\xintOpp*

```

2082 \def\xintOpp {\romannumeral0\xintopp }%
2083 \def\xintopp #1{\expandafter\XINT_opp\romannumeral0\xintraw {#1}}%

```

7.61 *\xintSgn*

```

2084 \def\xintSgn {\romannumeral0\xintsgn }%
2085 \def\xintsgn #1{\expandafter\XINT_sgn\romannumeral0\xintraw {#1}\Z }%

```

7.62 Floating point macros

1.2 release has not touched the floating point routines apart from adding the new *\xintFloatFac*.(*) The others should be revised for some optimizations related to the underlying model of the new core routines. This is particularly the case for *\xintFloatPow* and *\xintFloatPower* which should keep intermediate results in a suitable format, like *\xintiiPow* does.

The switch to 1.2 was smooth (apart from the writing up of the new *\xintFloatFac*), as I didn't have to change a single line of code anywhere here !

(*) note 2015/11/19: the 1.2 *\xintFloatFac* had the bug not to normalize its input via *\xintNum*, using only *\numexpr*. Fixed in 1.2e.

7.63 `\xintFloatAdd`, `\XINTinFloatAdd`

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

```

2086 \def\xintFloatAdd      {\romannumeral0\xintfloatadd }%
2087 \def\xintfloatadd      #1{\XINT_fladd_chkopt \xintfloat #1\xint_relax }%
2088 \def\XINTinFloatAdd     {\romannumeral0\XINTinfloatadd }%
2089 \def\XINTinfloatadd     #1{\XINT_fladd_chkopt \XINTinfloat #1\xint_relax }%
2090 \def\XINT_fladd_chkopt #1#2%
2091 {%
2092   \ifx [#2\expandafter\XINT_fladd_opt
2093     \else\expandafter\XINT_fladd_noopt
2094   \fi #1#2%
2095 }%
2096 \def\XINT_fladd_noopt #1#2\xint_relax #3%
2097 {%
2098   #1[\XINTdigits]{\XINT_FL_Add {\XINTdigits+\xint_c_ii}{#2}{#3}}%
2099 }%
2100 \def\XINT_fladd_opt #1[\xint_relax #2]#3#4%
2101 {%
2102   #1[#2]{\XINT_FL_Add {#2+\xint_c_ii}{#3}{#4}}%
2103 }%
2104 \def\XINT_FL_Add #1#2%
2105 {%
2106   \expandafter\XINT_FL_Add_a\expandafter{\the\numexpr #1\expandafter}%
2107   \expandafter{\romannumeral0\XINTinfloat [#1]{#2}}%
2108 }%
2109 \def\XINT_FL_Add_a #1#2#3%
2110 {%
2111   \expandafter\XINT_FL_Add_b\romannumeral0\XINTinfloat [#1]{#3}#2{#1}%
2112 }%
2113 \def\XINT_FL_Add_b #1%
2114 {%
2115   \xint_gob_til_zero #1\XINT_FL_Add_zero 0\XINT_FL_Add_c #1%
2116 }%
2117 \def\XINT_FL_Add_c #1[#2]#3%
2118 {%
2119   \xint_gob_til_zero #3\XINT_FL_Add_zerobis 0\XINT_FL_Add_d #1[#2]#3%
2120 }%
2121 \def\XINT_FL_Add_d #1[#2]#3[#4]#5%
2122 {%
2123   \ifnum \numexpr #2-#4-#5>\xint_c_i
2124     \expandafter \xint_secondofthree_thenstop
2125   \else
2126     \ifnum \numexpr #4-#2-#5>\xint_c_i
2127       \expandafter\expandafter\expandafter\xint_thirdofthree_thenstop
2128     \fi
2129   \fi
2130   \xintadd {#1[#2]}{#3[#4]}%
2131 }%
2132 \def\XINT_FL_Add_zero 0\XINT_FL_Add_c 0[0]#1[#2]#3{#1[#2]}%
2133 \def\XINT_FL_Add_zerobis 0\XINT_FL_Add_d #1[#2]0[0]#3{#1[#2]}%

```

7.64 \backslash xintFloatSub, \backslash XINTinFloatSub

1.07

```

2134 \def\xintFloatSub {\romannumeral0\xintfloatsub }%
2135 \def\xintfloatsub #1{\XINT_flsb_chkopt \xintfloat #1\xint_relax }%
2136 \def\XINTinFloatSub {\romannumeral0\XINTinfloatsub }%
2137 \def\XINTinfloatsub #1{\XINT_flsb_chkopt \XINTinfloat #1\xint_relax }%
2138 \def\XINT_flsb_chkopt #1#2%
2139 {%
2140   \ifx [#2\expandafter\XINT_flsb_opt
2141     \else\expandafter\XINT_flsb_noopt
2142   \fi #1#2%
2143 }%
2144 \def\XINT_flsb_noopt #1#2\xint_relax #3%
2145 {%
2146   #1[\XINTdigits]{\XINT_FL_Add {\XINTdigits+\xint_c_ii}{#2}{\xintOpp{#3}}}%
2147 }%
2148 \def\XINT_flsb_opt #1[\xint_relax #2]#3#4%
2149 {%
2150   #1[#2]{\XINT_FL_Add {#2+\xint_c_ii}{#3}{\xintOpp{#4}}}%
2151 }%

```

7.65 \backslash xintFloatMul, \backslash 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

```

2152 \def\xintFloatMul {\romannumeral0\xintfloatmul}%
2153 \def\xintfloatmul #1{\XINT_flmul_chkopt \xintfloat #1\xint_relax }%
2154 \def\XINTinFloatMul {\romannumeral0\XINTinfloatmul }%
2155 \def\XINTinfloatmul #1{\XINT_flmul_chkopt \XINTinfloat #1\xint_relax }%
2156 \def\XINT_flmul_chkopt #1#2%
2157 {%
2158   \ifx [#2\expandafter\XINT_flmul_opt
2159     \else\expandafter\XINT_flmul_noopt
2160   \fi #1#2%
2161 }%
2162 \def\XINT_flmul_noopt #1#2\xint_relax #3%
2163 {%
2164   #1[\XINTdigits]{\XINT_FL_Mul {\XINTdigits+\xint_c_ii}{#2}{#3}}%
2165 }%
2166 \def\XINT_flmul_opt #1[\xint_relax #2]#3#4%
2167 {%
2168   #1[#2]{\XINT_FL_Mul {#2+\xint_c_ii}{#3}{#4}}%
2169 }%
2170 \def\XINT_FL_Mul #1#2%
2171 {%
2172   \expandafter\XINT_FL_Mul_a\expandafter{\the\numexpr #1\expandafter}%
2173   \expandafter{\romannumeral0\XINTinfloat [#1]{#2}}%

```

```

2174 }%
2175 \def\XINT_FL_Mul_a #1#2#3%
2176 {%
2177   \expandafter\XINT_FL_Mul_b\romannumeral0\XINTinfloat [#1]{#3}#2%
2178 }%
2179 \def\XINT_FL_Mul_b #1[#2]#3[#4]{\xintE{\xintiiMul {#1}{#3}}{#2+#4}}%

```

7.66 \xintFloatDiv, \XINTinFloatDiv

1.07

```

2180 \def\xintFloatDiv {\romannumeral0\xintfloatdiv}%
2181 \def\xintfloatdiv #1{\XINT_fldiv_chkopt \xintfloat #1\xint_relax }%
2182 \def\XINTinFloatDiv {\romannumeral0\XINTinfloatdiv }%
2183 \def\XINTinfloatdiv #1{\XINT_fldiv_chkopt \XINTinfloat #1\xint_relax }%
2184 \def\XINT_fldiv_chkopt #1#2%
2185 {%
2186   \ifx [#2]\expandafter\XINT_fldiv_opt
2187   \else\expandafter\XINT_fldiv_noopt
2188   \fi #1#2%
2189 }%
2190 \def\XINT_fldiv_noopt #1#2\xint_relax #3%
2191 {%
2192   #1[\XINTdigits]{\XINT_FL_Div {\XINTdigits+\xint_c_ii}{#2}{#3}}%
2193 }%
2194 \def\XINT_fldiv_opt #1[\xint_relax #2]#3#4%
2195 {%
2196   #1[#2]{\XINT_FL_Div {#2+\xint_c_ii}{#3}{#4}}%
2197 }%
2198 \def\XINT_FL_Div #1#2%
2199 {%
2200   \expandafter\XINT_FL_Div_a\expandafter{\the\numexpr #1\expandafter}%
2201   \expandafter{\romannumeral0\XINTinfloat [#1]{#2}}%
2202 }%
2203 \def\XINT_FL_Div_a #1#2#3%
2204 {%
2205   \expandafter\XINT_FL_Div_b\romannumeral0\XINTinfloat [#1]{#3}#2%
2206 }%
2207 \def\XINT_FL_Div_b #1[#2]#3[#4]{\xintE{#3/#1}{#4-#2}}%

```

7.67 \xintFloatPow, \XINTinFloatPow

This definitely should be revised to better take into account the new multiplication to maintain through intermediate states a suitable internal format, optimized for calls to `\XINT_mul_loop`.

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

```

2208 \def\xintFloatPow {\romannumeral0\xintfloatpow}%
2209 \def\xintfloatpow #1{\XINT_flpow_chkopt \xintfloat #1\xint_relax }%

```

```

2210 \def\XINTinFloatPow {\romannumeral0\XINTinfloatpow }%
2211 \def\XINTinfloatpow #1{\XINT_flpow_chkopt \XINTinfloat #1\xint_relax }%
2212 \def\XINT_flpow_chkopt #1#2%
2213 {%
2214   \ifx [#2\expandafter\XINT_flpow_opt
2215     \else\expandafter\XINT_flpow_noopt
2216   \fi
2217   #1#2%
2218 }%
2219 \def\XINT_flpow_noopt #1#2\xint_relax #3%
2220 {%
2221   \expandafter\XINT_flpow_checkB_start\expandafter
2222     {\the\numexpr #3\expandafter}\expandafter
2223     {\the\numexpr \XINTdigits}{#2}{#1[\XINTdigits]]}%
2224 }%
2225 \def\XINT_flpow_opt #1[\xint_relax #2]#3#4%
2226 {%
2227   \expandafter\XINT_flpow_checkB_start\expandafter
2228     {\the\numexpr #4\expandafter}\expandafter
2229     {\the\numexpr #2}{#3}{#1[#2]}%
2230 }%
2231 \def\XINT_flpow_checkB_start #1{\XINT_flpow_checkB_a #1\Z }%
2232 \def\XINT_flpow_checkB_a #1%
2233 {%
2234   \xint_UDzerominusfork
2235   #1-\XINT_flpow_BisZero
2236   0#1{\XINT_flpow_checkB_b 1}%
2237   0-{\XINT_flpow_checkB_b 0#1}%
2238   \krof
2239 }%
2240 \def\XINT_flpow_BisZero \Z #1#2#3{#3{1/1[0]}}%
2241 \def\XINT_flpow_checkB_b #1#2\Z #3%
2242 {%
2243   \expandafter\XINT_flpow_checkB_c \expandafter
2244   {\romannumeral0\xintlength{#2}}{#3}{#2}#1%
2245 }%
2246 \def\XINT_flpow_checkB_c #1#2%
2247 {%
2248   \expandafter\XINT_flpow_checkB_d \expandafter
2249   {\the\numexpr \expandafter\xintLength\expandafter
2250     {\the\numexpr #1*20/\xint_c_iii }+#1+#2+\xint_c_i }%
2251 }%
2252 \def\XINT_flpow_checkB_d #1#2#3#4%
2253 {%
2254   \expandafter \XINT_flpow_a
2255   \romannumeral0\XINTinfloat [#1]{#4}{#1}{#2}#3%
2256 }%
2257 \def\XINT_flpow_a #1%
2258 {%
2259   \xint_UDzerominusfork
2260   #1-\XINT_flpow_zero
2261   0#1{\XINT_flpow_b 1}%

```

7 Package *xintfrac* implementation

```

2262      0-{\XINT_flpow_b 0#1}%
2263      \krof
2264 }%
2265 \def\XINT_flpow_b #1#2[#3]#4#5%
2266 {%
2267      \XINT_flpow_loopI {#5}{#2[#3]}\romannumeral0\XINTinfloatmul [#4]}%
2268      {#1*\ifodd #5 1\else 0\fi}%
2269 }%
2270 \def\XINT_flpow_zero [#1]#2#3#4#5%

```

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

```

2271 {%
2272      \if #41\xint_afterfi {\xintError:DivisionByZero #5{1[2147483648]}}}%
2273      \else \xint_afterfi {#5{0[0]}}\fi
2274 }%
2275 \def\XINT_flpow_loopI #1%
2276 {%
2277      \ifnum #1=\xint_c_i\XINT_flpow_ItoIII\fi
2278      \ifodd #1
2279          \expandafter\XINT_flpow_loopI_odd
2280      \else
2281          \expandafter\XINT_flpow_loopI_even
2282      \fi
2283      {#1}%
2284 }%
2285 \def\XINT_flpow_ItoIII\fi #1\fi #2#3#4#5%
2286 {%
2287      \fi\expandafter\XINT_flpow_III\the\numexpr #5\relax #3%
2288 }%
2289 \def\XINT_flpow_loopI_even #1#2#3%
2290 {%
2291      \expandafter\XINT_flpow_loopI\expandafter
2292      {\the\numexpr #1/\xint_c_ii\expandafter}\expandafter
2293      {#3{#2}{#2}}{#3}%
2294 }%
2295 \def\XINT_flpow_loopI_odd #1#2#3%
2296 {%
2297      \expandafter\XINT_flpow_loopII\expandafter
2298      {\the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter}\expandafter
2299      {#3{#2}{#2}}{#3}{#2}%
2300 }%
2301 \def\XINT_flpow_loopII #1%
2302 {%
2303      \ifnum #1 = \xint_c_i\XINT_flpow_ItoIII\fi
2304      \ifodd #1
2305          \expandafter\XINT_flpow_loopII_odd
2306      \else
2307          \expandafter\XINT_flpow_loopII_even
2308      \fi
2309      {#1}%
2310 }%

```

```

2311 \def\XINT_flpow_loopII_even #1#2#3%
2312 {%
2313   \expandafter\XINT_flpow_loopII\expandafter
2314   {\the\numexpr #1/\xint_c_ii\expandafter}\expandafter
2315   {#3{#2}{#2}}{#3}%
2316 }%
2317 \def\XINT_flpow_loopII_odd #1#2#3#4%
2318 {%
2319   \expandafter\XINT_flpow_loopII_odda\expandafter
2320   {#3{#2}{#4}}{#1}{#2}{#3}%
2321 }%
2322 \def\XINT_flpow_loopII_odda #1#2#3#4%
2323 {%
2324   \expandafter\XINT_flpow_loopII\expandafter
2325   {\the\numexpr #2/\xint_c_ii-\xint_c_i\expandafter}\expandafter
2326   {#4{#3}{#3}}{#4}{#1}%
2327 }%
2328 \def\XINT_flpow_IItoIII\fi #1\fi #2#3#4#5#6%
2329 {%
2330   \fi\expandafter\XINT_flpow_III\the\numexpr #6\expandafter\relax
2331   #4{#3}{#5}%
2332 }%
2333 \def\XINT_flpow_III #1#2[#3]#4%
2334 {%
2335   \expandafter\XINT_flpow_IIIend\expandafter
2336   {\the\numexpr\if #41-\fi#3\expandafter}%
2337   \xint_UDzerofork
2338   #4{{#2}}%
2339   0{{1/#2}}%
2340   \krof #1%
2341 }%
2342 \def\XINT_flpow_IIIend #1#2#3#4%
2343 {%
2344   \xint_UDzerofork
2345   #3{#4{#2[#1]}}%
2346   0{#4{-#2[#1]}}%
2347   \krof
2348 }%

```

7.68 \xintFloatPower, \XINTinFloatPower

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

```

2349 \def\xintFloatPower {\romannumeral0\xintfloatpower}%
2350 \def\xintfloatpower #1{\XINT_flpower_chkopt \xintfloat #1\xint_relax}%
2351 \def\XINTinFloatPower {\romannumeral0\XINTinfloatpower}%
2352 \def\XINTinfloatpower #1{\XINT_flpower_chkopt \XINTinfloat #1\xint_relax}%
2353 \def\XINT_flpower_chkopt #1#2%
2354 {%
2355   \ifx [#2\expandafter\XINT_flpower_opt
2356   \else\expandafter\XINT_flpower_noopt
2357   \fi
2358   #1#2%

```

7 Package *xintfrac* implementation

```

2359 }%
2360 \def\XINT_flpower_noopt #1#2\xint_relax #3%
2361 {%
2362   \expandafter\XINT_flpower_checkB_start\expandafter
2363     {\the\numexpr \XINTdigits\expandafter}\expandafter
2364     {\romannumeral0\xintnum{#3}}{#2}{#1[\XINTdigits]}%
2365 }%
2366 \def\XINT_flpower_opt #1[\xint_relax #2]#3#4%
2367 {%
2368   \expandafter\XINT_flpower_checkB_start\expandafter
2369     {\the\numexpr #2\expandafter}\expandafter
2370     {\romannumeral0\xintnum{#4}}{#3}{#1[#2]}%
2371 }%
2372 \def\XINT_flpower_checkB_start #1#2{\XINT_flpower_checkB_a #2\Z {#1}}%
2373 \def\XINT_flpower_checkB_a #1%
2374 {%
2375   \xint_UDzerominusfork
2376   #1-\XINT_flpower_BisZero
2377   0#1{\XINT_flpower_checkB_b 1}%
2378   0-{\XINT_flpower_checkB_b 0#1}%
2379   \krof
2380 }%
2381 \def\XINT_flpower_BisZero \Z #1#2#3{#3{1/1[0]}}%
2382 \def\XINT_flpower_checkB_b #1#2\Z #3%
2383 {%
2384   \expandafter\XINT_flpower_checkB_c \expandafter
2385   {\romannumeral0\xintlenth{#2}}{#3}{#2}#1%
2386 }%
2387 \def\XINT_flpower_checkB_c #1#2%
2388 {%
2389   \expandafter\XINT_flpower_checkB_d \expandafter
2390   {\the\numexpr \expandafter\xintLength\expandafter
2391     {\the\numexpr #1*20/\xint_c_iii }+#1+#2+\xint_c_i }%
2392 }%
2393 \def\XINT_flpower_checkB_d #1#2#3#4%
2394 {%
2395   \expandafter \XINT_flpower_a
2396   \romannumeral0\XINTinfloat [#1]{#4}{#1}{#2}#3%
2397 }%
2398 \def\XINT_flpower_a #1%
2399 {%
2400   \xint_UDzerominusfork
2401   #1-\XINT_flpow_zero
2402   0#1{\XINT_flpower_b 1}%
2403   0-{\XINT_flpower_b 0#1}%
2404   \krof
2405 }%
2406 \def\XINT_flpower_b #1#2[#3]#4#5%
2407 {%
2408   \XINT_flpower_loopI {#5}{#2[#3]}\romannumeral0\XINTinfloatmul [#4]}%
2409   {#1*\xintiiOdd {#5}}%
2410 }%

```


7 Package *xintfrac* implementation

```

2411 \def\XINT_flpower_loopI #1%
2412 {%
2413   \if1\XINT_isOne {#1}\XINT_flpower_ItoIII\fi
2414   \if1\xintiiOdd{#1}%
2415     \expandafter\expandafter\expandafter\XINT_flpower_loopI_odd
2416   \else
2417     \expandafter\expandafter\expandafter\XINT_flpower_loopI_even
2418   \fi
2419   \expandafter {\romannumeral0\xinthalft{#1}}%
2420 }%
2421 \def\XINT_flpower_ItoIII\fi #1\fi\expandafter #2#3#4#5%
2422 {%
2423   \fi\expandafter\XINT_flpow_III \the\numexpr #5\relax #3%
2424 }%
2425 \def\XINT_flpower_loopI_even #1#2#3%
2426 {%
2427   \expandafter\XINT_flpower_toI\expandafter {#3{#2}{#2}}{#1}{#3}%
2428 }%
2429 \def\XINT_flpower_loopI_odd #1#2#3%
2430 {%
2431   \expandafter\XINT_flpower_toII\expandafter {#3{#2}{#2}}{#1}{#3}{#2}%
2432 }%
2433 \def\XINT_flpower_toI #1#2{\XINT_flpower_loopI {#2}{#1}}%
2434 \def\XINT_flpower_toII #1#2{\XINT_flpower_loopII {#2}{#1}}%
2435 \def\XINT_flpower_loopII #1%
2436 {%
2437   \if1\XINT_isOne {#1}\XINT_flpower_IItoIII\fi
2438   \if1\xintiiOdd{#1}%
2439     \expandafter\expandafter\expandafter\XINT_flpower_loopII_odd
2440   \else
2441     \expandafter\expandafter\expandafter\XINT_flpower_loopII_even
2442   \fi
2443   \expandafter {\romannumeral0\xinthalft{#1}}%
2444 }%
2445 \def\XINT_flpower_loopII_even #1#2#3%
2446 {%
2447   \expandafter\XINT_flpower_toII\expandafter
2448   {#3{#2}{#2}}{#1}{#3}%
2449 }%
2450 \def\XINT_flpower_loopII_odd #1#2#3#4%
2451 {%
2452   \expandafter\XINT_flpower_loopII_odda\expandafter
2453   {#3{#2}{#4}}{#2}{#3}{#1}%
2454 }%
2455 \def\XINT_flpower_loopII_odda #1#2#3#4%
2456 {%
2457   \expandafter\XINT_flpower_toII\expandafter
2458   {#3{#2}{#2}}{#4}{#3}{#1}%
2459 }%
2460 \def\XINT_flpower_IItoIII\fi #1\fi\expandafter #2#3#4#5#6%
2461 {%
2462   \fi\expandafter\XINT_flpow_III\the\numexpr #6\expandafter\relax

```

```
2463     #4{#3}{#5}%
2464 }%
```

7.69 \xintFloatFac, \XINTFloatFac

1.2. Je dois documenter le raisonnement sur la précision à imposer pour les calculs par blocs de huit faits en sous-main. Par ailleurs j'ai été amené à une routine `smallmul` spéciale.

Comment 2015/11/13: at least I do have a file which privately document my choice of precision (I could reduce by one unit here). I hesitated with doing a divide and conquer approach, but last time I thought about it I did not see an obvious advantage in this context. But I should implement and compare.

2015/11/19: there was a bug here that the input was only subjected to `\numexpr`. It should have used `\xintNum`. Fixed in 1.2e

```
2465 \def\xintFloatFac      {\romannumeral0\xintfloatfac}%
2466 \def\xintfloatfac      #1{\XINT_flfac_chkopt \xintfloat #1\xint_relax }%
2467 \def\XINTinFloatFac    {\romannumeral0\XINTinfloatfac }%
2468 \def\XINTinfloatfac    #1{\XINT_flfac_chkopt \XINTinfloat #1\xint_relax }%
2469 \def\XINT_flfac_chkopt #1#2%
2470 {%
2471   \ifx [#2\expandafter\XINT_flfac_opt
2472     \else\expandafter\XINT_flfac_noopt
2473   \fi
2474   #1#2%
2475 }%
2476 \def\XINT_flfac_noopt  #1#2\xint_relax
2477 {%
2478   \expandafter\XINT_FL_fac_start\expandafter
2479   {\the\numexpr \xintNum{#2}}{\XINTdigits}{#1[\XINTdigits]}%
2480 }%
```

`#2` should arguably be subjected to `\numexpr` only once. Here it will again be during final `\xintfloat`, or `\XINTinfloat`.

```
2481 \def\XINT_flfac_opt #1[\xint_relax #2]#3%
2482 {%
2483   \expandafter\XINT_FL_fac_start\expandafter
2484   {\the\numexpr \xintNum{#3}\expandafter}\expandafter{\the\numexpr#2}{#1[#2]}%
2485 }%
2486 \def\XINT_FL_fac_start #1%
2487 {%
2488   \ifcase\XINT_cntSgn #1\Z
2489     \expandafter\XINT_FL_fac_iszero
2490   \or
2491     \expandafter\XINT_FL_fac_increaseP
2492   \else
2493     \expandafter\XINT_FL_fac_isneg
2494   \fi {#1}%
2495 }%
2496 \def\XINT_FL_fac_iszero #1#2#3{#3{1/1[0]}}%
2497 \def\XINT_FL_fac_isneg  #1#2#3%
2498   {\expandafter\xintError:FactorialOfNegativeNumber #3{1/1[0]}}%
2499 \def\XINT_FL_fac_increaseP #1#2%
```

```

2500 {%
2501   \expandafter\XINT_FL_fac_fork
2502   \the\numexpr \xint_c_viii*%
2503   ((\xint_c_v+#2+\XINT_FL_fac_extradigits #187654321\Z)/\xint_c_viii).%
2504   #1.%
2505 }%
2506 \def\XINT_FL_fac_extradigits #1#2#3#4#5#6#7#8{\XINT_FL_fac_extra_a }%
2507 \def\XINT_FL_fac_extra_a #1#2\Z {#1}%
2508 \def\XINT_FL_fac_fork #1.#2.#3%
2509 {%
2510   \ifnum #2>999999999 \xint_dothis{\XINT_FL_fac_toobig } \fi
2511   \ifnum #2>9999 \xint_dothis{\XINT_FL_fac_vbigloop_a } \fi
2512   \ifnum #2>465 \xint_dothis{\XINT_FL_fac_bigloop_a } \fi
2513   \ifnum #2>101 \xint_dothis{\XINT_FL_fac_medloop_a } \fi
2514   \xint_orthat{\XINT_FL_fac_smallloop_a }%
2515   #2.#1.{\XINT_FL_fac_out}{#3}%
2516 }%
2517 \def\XINT_FL_fac_toobig #1.#2.#3#4%
2518   {\expandafter\xintError:FactorialOfTooBigNumber #4{1/1[0]}}%
2519 \def\XINT_FL_fac_out #1\Z![#2]#3{#3{\romannumeral0\XINT_mul_out
2520   #1\Z!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W [#2]}}%
2521 \def\XINT_FL_fac_vbigloop_a #1.#2.%
2522 {%
2523   \XINT_FL_fac_bigloop_a 9999.#2.%
2524   {\expandafter\XINT_FL_fac_vbigloop_loop\the\numexpr 100010000\expandafter.%
2525   \the\numexpr \xint_c_x^viii+#1.}%
2526 }%
2527 \def\XINT_FL_fac_vbigloop_loop #1.#2.%
2528 {%
2529   \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2530   \expandafter\XINT_FL_fac_vbigloop_loop
2531   \the\numexpr #1+\xint_c_i\expandafter.%
2532   \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_mul #1!%
2533 }%
2534 \def\XINT_FL_fac_bigloop_a #1.%
2535 {%
2536   \expandafter\XINT_FL_fac_bigloop_b \the\numexpr
2537   #1+\xint_c_i-\xint_c_ii*((#1-464)/\xint_c_ii).#1.%
2538 }%
2539 \def\XINT_FL_fac_bigloop_b #1.#2.#3.%
2540 {%
2541   \expandafter\XINT_FL_fac_medloop_a
2542   \the\numexpr #1-\xint_c_i.#3.{\XINT_FL_fac_bigloop_loop #1.#2.}%
2543 }%
2544 \def\XINT_FL_fac_bigloop_loop #1.#2.%
2545 {%
2546   \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2547   \expandafter\XINT_FL_fac_bigloop_loop
2548   \the\numexpr #1+\xint_c_ii\expandafter.%
2549   \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_bigloop_mul #1!%
2550 }%
2551 \def\XINT_FL_fac_bigloop_mul #1!%

```

7 Package *xintfrac* implementation

```

2552 {%
2553   \expandafter\XINT_FL_fac_mul
2554   \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2555 }%
2556 \def\XINT_FL_fac_medloop_a #1.%
2557 {%
2558   \expandafter\XINT_FL_fac_medloop_b
2559   \the\numexpr #1+\xint_c_i-\xint_c_iii*((#1-100)/\xint_c_iii).#1.%
2560 }%
2561 \def\XINT_FL_fac_medloop_b #1.#2.#3.%
2562 {%
2563   \expandafter\XINT_FL_fac_smallloop_a
2564   \the\numexpr #1-\xint_c_i.#3.{\XINT_FL_fac_medloop_loop #1.#2.}%
2565 }%
2566 \def\XINT_FL_fac_medloop_loop #1.#2.%
2567 {%
2568   \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2569   \expandafter\XINT_FL_fac_medloop_loop
2570   \the\numexpr #1+\xint_c_iii\expandafter.%
2571   \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_medloop_mul #1!%
2572 }%
2573 \def\XINT_FL_fac_medloop_mul #1!%
2574 {%
2575   \expandafter\XINT_FL_fac_mul
2576   \the\numexpr
2577     \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2578 }%
2579 \def\XINT_FL_fac_smallloop_a #1.%
2580 {%
2581   \csname
2582     XINT_FL_fac_smallloop_\the\numexpr #1-\xint_c_iv*(#1/\xint_c_iv)\relax
2583   \endcsname #1.%
2584 }%
2585 \expandafter\def\csname XINT_FL_fac_smallloop_1\endcsname #1.#2.%
2586 {%
2587   \XINT_FL_fac_addzeros #2.100000001!.{2.#1.}{#2}%
2588 }%
2589 \expandafter\def\csname XINT_FL_fac_smallloop_2\endcsname #1.#2.%
2590 {%
2591   \XINT_FL_fac_addzeros #2.100000002!.{3.#1.}{#2}%
2592 }%
2593 \expandafter\def\csname XINT_FL_fac_smallloop_3\endcsname #1.#2.%
2594 {%
2595   \XINT_FL_fac_addzeros #2.100000003!.{4.#1.}{#2}%
2596 }%
2597 \expandafter\def\csname XINT_FL_fac_smallloop_4\endcsname #1.#2.%
2598 {%
2599   \XINT_FL_fac_addzeros #2.100000004!.{5.#1.}{#2}%
2600 }%
2601 \def\XINT_FL_fac_addzeros #1.%
2602 {%
2603   \ifnum #1=\xint_c_viii \expandafter\XINT_FL_fac_addzeros_exit\fi

```

7 Package *xintfrac* implementation

```

2604 \expandafter\XINT_FL_fac_addzeros\the\numexpr #1-\xint_c_viii.100000000!%
2605 }%
2606 \def\XINT_FL_fac_addzeros_exit #1.#2.#3#4%
2607 {\XINT_FL_fac_smallloop_loop #3#21\Z![-#4]}%
2608 \def\XINT_FL_fac_smallloop_loop #1.#2.%
2609 {%
2610 \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2611 \expandafter\XINT_FL_fac_smallloop_loop
2612 \the\numexpr #1+\xint_c_iv\expandafter.%
2613 \the\numexpr #2\expandafter.\romannumeral0\XINT_FL_fac_smallloop_mul #1!%
2614 }%
2615 \def\XINT_FL_fac_smallloop_mul #1!%
2616 {%
2617 \expandafter\XINT_FL_fac_mul
2618 \the\numexpr
2619 \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2620 }%[[
2621 \def\XINT_FL_fac_loop_exit #1!#2]#3{#3#2]}%
2622 \def\XINT_FL_fac_mul 1#1!%
2623 {\expandafter\XINT_FL_fac_mul_a\the\numexpr\XINT_FL_fac_smallmul 10!{#1}}%
2624 \def\XINT_FL_fac_mul_a #1-#2%
2625 {%
2626 \if#21\xint_afterfi{\expandafter\space\xint_gob_til_exclam}\else
2627 \expandafter\space\fi #11\Z!%
2628 }%
2629 \def\XINT_FL_fac_minimulwc_a #1#2#3#4#5!#6#7#8#9%
2630 {%
2631 \XINT_FL_fac_minimulwc_b {#1#2#3#4}{#5}{#6#7#8#9}%
2632 }%
2633 \def\XINT_FL_fac_minimulwc_b #1#2#3#4!#5%
2634 {%
2635 \expandafter\XINT_FL_fac_minimulwc_c
2636 \the\numexpr \xint_c_x^ix+#5+#2*#4.{#1}{#2}{#3}{#4}%
2637 }%
2638 \def\XINT_FL_fac_minimulwc_c 1#1#2#3#4#5#6.#7%
2639 {%
2640 \expandafter\XINT_FL_fac_minimulwc_d {#1#2#3#4#5}#7{#6}%
2641 }%
2642 \def\XINT_FL_fac_minimulwc_d #1#2#3#4#5%
2643 {%
2644 \expandafter\XINT_FL_fac_minimulwc_e
2645 \the\numexpr \xint_c_x^ix+#1+#2*#5+#3*#4.{#2}{#4}%
2646 }%
2647 \def\XINT_FL_fac_minimulwc_e 1#1#2#3#4#5#6.#7#8#9%
2648 {%
2649 1#6#9\expandafter!%
2650 \the\numexpr\expandafter\XINT_FL_fac_smallmul
2651 \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#7*#8!%
2652 }%
2653 \def\XINT_FL_fac_smallmul 1#1!#21#3!%
2654 {%
2655 \xint_gob_til_Z #3\XINT_FL_fac_smallmul_end\Z

```

```

2656 \XINT_FL_fac_minimulwc_a #2!#3!{#1}{#2}%
2657 }%
2658 \def\XINT_FL_fac_smallmul_end\Z\XINT_FL_fac_minimulwc_a #1!\Z!#2#3[#4]%
2659 {%
2660 \ifnum #2=\xint_c_
2661 \expandafter\xint_firstoftwo\else
2662 \expandafter\xint_secondoftwo
2663 \fi
2664 {-2\relax[#4]}%
2665 {1#2\expandafter!\expandafter-\expandafter1\expandafter
2666 [\the\numexpr #4+\xint_c_viii]}%
2667 }%

```

7.70 \xintFloatSqrt, \XINTinFloatSqrt

1.08. Note 2015/11/16: I absolutely must document what's happening here, I have a file with comments from June 2013 which however is not completely explicit (as I did the rounding integer square root \xintiiSqrtR more than one year later, I am not 100% sure the one here does correct rounding, and I don't have time to check now.)

```

2668 \def\xintFloatSqrt {\romannumeral0\xintfloatsqr }%
2669 \def\xintfloatsqr #1{\XINT_flsqrt_chkopt \xintfloat #1\xint_relax }%
2670 \def\XINTinFloatSqrt {\romannumeral0\XINTinfloatsqr }%
2671 \def\XINTinfloatsqr #1{\XINT_flsqrt_chkopt \XINTinfloat #1\xint_relax }%
2672 \def\XINT_flsqrt_chkopt #1#2%
2673 {%
2674 \ifx [#2\expandafter\XINT_flsqrt_opt
2675 \else\expandafter\XINT_flsqrt_noopt
2676 \fi #1#2%
2677 }%
2678 \def\XINT_flsqrt_noopt #1#2\xint_relax
2679 {%
2680 #1[\XINTdigits]{\XINT_FL_sqrt \XINTdigits {#2}}%
2681 }%
2682 \def\XINT_flsqrt_opt #1[\xint_relax #2]#3%
2683 {%
2684 #1[#2]{\XINT_FL_sqrt {#2}{#3}}%
2685 }%
2686 \def\XINT_FL_sqrt #1%
2687 {%
2688 \ifnum\numexpr #1<\xint_c_xviii
2689 \xint_afterfi {\XINT_FL_sqrt_a\xint_c_xviii}%
2690 \else
2691 \xint_afterfi {\XINT_FL_sqrt_a {#1+\xint_c_i}}%
2692 \fi
2693 }%
2694 \def\XINT_FL_sqrt_a #1#2%
2695 {%
2696 \expandafter\XINT_FL_sqrt_checkifzeroorneg
2697 \romannumeral0\XINTinfloat [#1]{#2}%
2698 }%
2699 \def\XINT_FL_sqrt_checkifzeroorneg #1%
2700 {%

```

```

2701 \xint_UDzerominusfork
2702 #1-\XINT_FL_sqrt_iszero
2703 0#1\XINT_FL_sqrt_isneg
2704 0-\XINT_FL_sqrt_b #1}%
2705 \krof
2706 }%
2707 \def\XINT_FL_sqrt_iszero #1[#2]{0[0]}%
2708 \def\XINT_FL_sqrt_isneg #1[#2]{\xintError:RootOfNegative 0[0]}%
2709 \def\XINT_FL_sqrt_b #1[#2]%
2710 {%
2711 \ifodd #2
2712 \xint_afterfi{\XINT_FL_sqrt_c 01}%
2713 \else
2714 \xint_afterfi{\XINT_FL_sqrt_c {}0}%
2715 \fi
2716 {#1}{#2}%
2717 }%
2718 \def\XINT_FL_sqrt_c #1#2#3#4%
2719 {%
2720 \expandafter\XINT_flsqrt\expandafter {\the\numexpr #4-#2}{#3#1}%
2721 }%
2722 \def\XINT_flsqrt #1#2%
2723 {%
2724 \expandafter\XINT_sqrt_a
2725 \expandafter{\romannumeral0\xintlength {#2}}\XINT_flsqrt_big_d {#2}{#1}%
2726 }%
2727 \def\XINT_flsqrt_big_d #1#2%
2728 {%
2729 \ifodd #2
2730 \expandafter\expandafter\expandafter\XINT_flsqrt_big_eB
2731 \else
2732 \expandafter\expandafter\expandafter\XINT_flsqrt_big_eA
2733 \fi
2734 \expandafter {\the\numexpr (#2-\xint_c_i)/\xint_c_ii }{#1}%
2735 }%
2736 \def\XINT_flsqrt_big_eA #1#2#3%
2737 {%
2738 \XINT_flsqrt_big_eA_a #3\Z {#2}{#1}{#3}%
2739 }%
2740 \def\XINT_flsqrt_big_eA_a #1#2#3#4#5#6#7#8#9\Z
2741 {%
2742 \XINT_flsqrt_big_eA_b {#1#2#3#4#5#6#7#8}%
2743 }%
2744 \def\XINT_flsqrt_big_eA_b #1#2%
2745 {%
2746 \expandafter\XINT_flsqrt_big_f
2747 \romannumeral0\XINT_flsqrt_small_e {#2001}{#1}%
2748 }%
2749 \def\XINT_flsqrt_big_eB #1#2#3%
2750 {%
2751 \XINT_flsqrt_big_eB_a #3\Z {#2}{#1}{#3}%
2752 }%

```

7 Package *xintfrac* implementation

```

2753 \def\XINT_flsqrt_big_eB_a #1#2#3#4#5#6#7#8#9%
2754 {%
2755     \XINT_flsqrt_big_eB_b {#1#2#3#4#5#6#7#8#9}%
2756 }%
2757 \def\XINT_flsqrt_big_eB_b #1#2\Z #3%
2758 {%
2759     \expandafter\XINT_flsqrt_big_f
2760     \romannumeral0\XINT_flsqrt_small_e {#30001}{#1}%
2761 }%
2762 \def\XINT_flsqrt_small_e #1#2%
2763 {%
2764     \expandafter\XINT_flsqrt_small_f\expandafter
2765     {\the\numexpr #1*#1-#2-\xint_c_i}{#1}%
2766 }%
2767 \def\XINT_flsqrt_small_f #1#2%
2768 {%
2769     \expandafter\XINT_flsqrt_small_g\expandafter
2770     {\the\numexpr (#1+#2)/(2*#2)-\xint_c_i }{#1}{#2}%
2771 }%
2772 \def\XINT_flsqrt_small_g #1%
2773 {%
2774     \ifnum #1>\xint_c_
2775         \expandafter\XINT_flsqrt_small_h
2776     \else
2777         \expandafter\XINT_flsqrt_small_end
2778     \fi
2779     {#1}%
2780 }%
2781 \def\XINT_flsqrt_small_h #1#2#3%
2782 {%
2783     \expandafter\XINT_flsqrt_small_f\expandafter
2784     {\the\numexpr #2-\xint_c_ii*#1*#3+#1*#1\expandafter}\expandafter
2785     {\the\numexpr #3-#1}%
2786 }%
2787 \def\XINT_flsqrt_small_end #1#2#3%
2788 {%
2789     \expandafter\space\expandafter
2790     {\the\numexpr \xint_c_i+#3*\xint_c_x^iv-
2791         (#2*\xint_c_x^iv+#3)/(\xint_c_ii*#3)}%
2792 }%
2793 \def\XINT_flsqrt_big_f #1%
2794 {%
2795     \expandafter\XINT_flsqrt_big_fa\expandafter
2796     {\romannumeral0\xintiisqr {#1}}{#1}%
2797 }%
2798 \def\XINT_flsqrt_big_fa #1#2#3#4%
2799 {%
2800     \expandafter\XINT_flsqrt_big_fb\expandafter
2801     {\romannumeral0\XINT_dsx_addzerosnofuss
2802         {\numexpr #3-\xint_c_viii\relax}{#2}}%
2803     {\romannumeral0\xintiisub
2804         {\XINT_dsx_addzerosnofuss

```



```

2805      {\numexpr \xint_c_ii*(#3-\xint_c_viii)\relax}{#1}{#4}}%
2806      {#3}}%
2807 }%
2808 \def\xINT_flsqrt_big_fb #1#2%
2809 {%
2810   \expandafter\xINT_flsqrt_big_g\expandafter {#2}{#1}%
2811 }%
2812 \def\xINT_flsqrt_big_g #1#2%
2813 {%
2814   \expandafter\xINT_flsqrt_big_j
2815   \romannumeral0\xintiidivision
2816   {#1}{\romannumeral0\xINT_dbl_pos #2\Z}{#2}%
2817 }%
2818 \def\xINT_flsqrt_big_j #1%
2819 {%
2820   \if0\xINT_Sgn #1\Z
2821     \expandafter \xINT_flsqrt_big_end_a
2822   \else \expandafter \xINT_flsqrt_big_k
2823   \fi {#1}%
2824 }%
2825 \def\xINT_flsqrt_big_k #1#2#3%
2826 {%
2827   \expandafter\xINT_flsqrt_big_l\expandafter
2828   {\romannumeral0\xintiisub {#3}{#1}}%
2829   {\romannumeral0\xintiiaadd {#2}{\romannumeral0\xINT_sqr #1\Z}}%
2830 }%
2831 \def\xINT_flsqrt_big_l #1#2%
2832 {%
2833   \expandafter\xINT_flsqrt_big_g\expandafter
2834   {#2}{#1}%
2835 }%
2836 \def\xINT_flsqrt_big_end_a #1#2#3#4#5%
2837 {%
2838   \expandafter\xINT_flsqrt_big_end_b\expandafter
2839   {\the\numexpr -#4+#5/\xint_c_ii\expandafter}\expandafter
2840   {\romannumeral0\xintiisub
2841     {\XINT_dsx_addzerosnofuss {#4}{#3}}%
2842     {\xintHalf{\xintiiQuo{\XINT_dsx_addzerosnofuss {#4}{#2}}{#3}}}%
2843 }%
2844 \def\xINT_flsqrt_big_end_b #1#2{#2[#1]}%
2845 \XINT_restorecatcodes_endinput%

```

8 Package *xintseries* implementation

.1	Catcodes, ε -TeX and reload detection . . .	210	.7	<code>\xintRationalSeries</code>	213
.2	Package identification	210	.8	<code>\xintRationalSeriesX</code>	214
.3	<code>\xintSeries</code>	211	.9	<code>\xintFxFtPowerSeries</code>	215
.4	<code>\xintiSeries</code>	211	.10	<code>\xintFxFtPowerSeriesX</code>	216
.5	<code>\xintPowerSeries</code>	212	.11	<code>\xintFloatPowerSeries</code>	216
.6	<code>\xintPowerSeriesX</code>	213	.12	<code>\xintFloatPowerSeriesX</code>	218

The commenting is currently (2015/11/22) very sparse.

8.1 Catcodes, ε -TeX and reload detection

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

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

```

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 [2015/11/22 v1.2e Expandable partial sums with xint package (jfB)]%

```

8.3 `\xintSeries`

Modified in 1.06 to give the indices first to a `\numexpr` rather than expanding twice. I just use `\the\numexpr` and maintain the previous code after that. 1.08a adds the forgotten optimization following that previous change.

```

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

```

8.4 `\xintiSeries`

Modified in 1.06 to give the indices first to a `\numexpr` rather than expanding twice. I just use `\the\numexpr` and maintain the previous code after that. 1.08a adds the forgotten optimization following that previous change.

```

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

```

```

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

```

8.5 \xintPowerSeries

The 1.03 version was very lame and created a build-up of denominators. (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\xintiadd {#4}{\xintiTrunc {#7}{\xintMul {#6}{#2}}{#1}}}%

```

```

251      {#1}{#3}{#5}{#6}{#7}%
252 }%
253 \def\XINT_fppowseries_exit_i\fi\expandafter\XINT_fppowseries_loop_ii
254      {\fi \expandafter\XINT_fppowseries_exit_ii }%
255 \def\XINT_fppowseries_exit_ii #1#2#3#4#5#6#7%
256 {%
257     \xinttrunc {#7}
258     {\xintiiadd {#4}{\xintiTrunc {#7}{\xintMul {#6{#2}}{#1}}[-#7]}%
259 }%

```

8.10 \xintFxFtPowerSeriesX

a,b,coeff,x,D

Modified in 1.06 to give the indices first to a `\numexpr` rather than expanding twice. I just use `\the\numexpr` and maintain the previous code after that. 1.08a adds the forgotten optimization following that previous change.

```

260 \def\xintFxFtPowerSeriesX {\romannumeral0\xintfxptpowerseriesx }%
261 \def\xintfxptpowerseriesx #1#2%
262 {%
263     \expandafter\XINT_fppowseriesx\expandafter
264     {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
265 }%
266 \def\XINT_fppowseriesx #1#2#3#4#5%
267 {%
268     \ifnum #2<#1
269         \xint_afterfi { 0}%
270     \else
271         \xint_afterfi
272         {\expandafter \XINT_fppowseriesx_pre \expandafter
273          {\romannumeral`&&@#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, ε - \TeX and reload detection . . .	219	.16	<code>\xintiGctoF</code>	231
.2	Package identification	220	.17	<code>\xintCtoCv</code> , <code>\xintCstoCv</code>	232
.3	<code>\xintCFrac</code>	220	.18	<code>\xintiCstoCv</code>	233
.4	<code>\xintGCFrac</code>	221	.19	<code>\xintGctoCv</code>	233
.5	<code>\xintGGCFrac</code>	223	.20	<code>\xintiGctoCv</code>	235
.6	<code>\xintGctoGCx</code>	224	.21	<code>\xintFtoCv</code>	236
.7	<code>\xintFtoCs</code>	224	.22	<code>\xintFtoCCv</code>	236
.8	<code>\xintFtoCx</code>	225	.23	<code>\xintCntoF</code>	236
.9	<code>\xintFtoC</code>	225	.24	<code>\xintGCntoF</code>	237
.10	<code>\xintFtoGC</code>	226	.25	<code>\xintCntoCs</code>	238
.11	<code>\xintFGtoC</code>	226	.26	<code>\xintCntoGC</code>	238
.12	<code>\xintFtoCC</code>	227	.27	<code>\xintGCntoGC</code>	239
.13	<code>\xintCtoF</code> , <code>\xintCstoF</code>	228	.28	<code>\xintCstoGC</code>	240
.14	<code>\xintiCstoF</code>	229	.29	<code>\xintGctoGC</code>	240
.15	<code>\xintGctoF</code>	229			

The commenting is currently (2015/11/22) very sparse. Release 1.09m (2014/02/26) has modified a few things: `\xintFtoCs` and `\xintCntoCs` insert spaces after the commas, `\xintCstoF` and `\xintCstoCv` authorize spaces in the input also before the commas, `\xintCntoCs` does not brace the produced coefficients, new macros `\xintFtoC`, `\xintCtoF`, `\xintCtoCv`, `\xintFGtoC`, and `\xintGGCFrac`.

9.1 Catcodes, ε - \TeX and reload detection

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

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

```

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 [2015/11/22 v1.2e Expandable continued fractions with xint package (jfb)]%

```

9.3 *\xintCFrac*

```

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

```

```

74 }%
75 \def\XINT_cfrac_optr #1%
76 {%
77   \expandafter\XINT_cfrac_A\romannumeral0\xintraewithzeros {#1}\Z
78   \hfill\relax
79 }%
80 \def\XINT_cfrac_A #1/#2\Z
81 {%
82   \expandafter\XINT_cfrac_B\romannumeral0\xintiidivision {#1}{#2}{#2}%
83 }%
84 \def\XINT_cfrac_B #1#2%
85 {%
86   \XINT_cfrac_C #2\Z {#1}%
87 }%
88 \def\XINT_cfrac_C #1%
89 {%
90   \xint_gob_til_zero #1\XINT_cfrac_integer 0\XINT_cfrac_D #1%
91 }%
92 \def\XINT_cfrac_integer 0\XINT_cfrac_D 0#1\Z #2#3#4#5{ #2}%
93 \def\XINT_cfrac_D #1\Z #2#3{\XINT_cfrac_loop_a {#1}{#3}{#1}{#2}}}%
94 \def\XINT_cfrac_loop_a
95 {%
96   \expandafter\XINT_cfrac_loop_d\romannumeral0\XINT_div_prepare
97 }%
98 \def\XINT_cfrac_loop_d #1#2%
99 {%
100   \XINT_cfrac_loop_e #2.{#1}%
101 }%
102 \def\XINT_cfrac_loop_e #1%
103 {%
104   \xint_gob_til_zero #1\xint_cfrac_loop_exit0\XINT_cfrac_loop_f #1%
105 }%
106 \def\XINT_cfrac_loop_f #1.#2#3#4%
107 {%
108   \XINT_cfrac_loop_a {#1}{#3}{#1}{#2#4}%
109 }%
110 \def\xint_cfrac_loop_exit0\XINT_cfrac_loop_f #1.#2#3#4#5#6%
111   {\XINT_cfrac_T #5#6{#2#4}\Z }%
112 \def\XINT_cfrac_T #1#2#3#4%
113 {%
114   \xint_gob_til_Z #4\XINT_cfrac_end\Z\XINT_cfrac_T #1#2{#4+\cfrac{#1#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\xintraewithzeros {#2}\Z {#1}%
282 }%
283 \def\XINT_ftcx_A #1/#2\Z
284 {%
285   \expandafter\XINT_ftcx_B\romannumeral0\xintiidivision {#1}{#2}{#2}%
286 }%
287 \def\XINT_ftcx_B #1#2%
288 {%
289   \XINT_ftcx_C #2.{#1}%
290 }%
291 \def\XINT_ftcx_C #1%
292 {%
293   \xint_gob_til_zero #1\XINT_ftcx_integer 0\XINT_ftcx_D #1%
294 }%
295 \def\XINT_ftcx_integer 0\XINT_ftcx_D 0#1.#2#3#4{ #2}%
296 \def\XINT_ftcx_D #1.#2#3#4{\XINT_ftcx_loop_a {#1}{#3}{#1}{#2#4}{#4}}%
297 \def\XINT_ftcx_loop_a
298 {%
299   \expandafter\XINT_ftcx_loop_d\romannumeral0\XINT_div_prepare
300 }%
301 \def\XINT_ftcx_loop_d #1#2%
302 {%
303   \XINT_ftcx_loop_e #2.{#1}%
304 }%
305 \def\XINT_ftcx_loop_e #1%
306 {%
307   \xint_gob_til_zero #1\xint_ftcx_loop_exit0\XINT_ftcx_loop_f #1%
308 }%
309 \def\XINT_ftcx_loop_f #1.#2#3#4#5%
310 {%
311   \XINT_ftcx_loop_a {#1}{#3}{#1}{#4#2#5}{#5}%
312 }%
313 \def\xint_ftcx_loop_exit0\XINT_ftcx_loop_f #1.#2#3#4#5{ #4#2}%

```

9.9 \xintFtoC

New in 1.09m: this is the same as `\xintFtoCx` with empty separator. I had temporarily during preparation of 1.09m removed braces from `\xintFtoCx`, but I recalled later why that was useful (see doc), thus let's just here do `\xintFtoCx {}`

```

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

```

9.10 `\xintFtoGC`

```

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

```

9.11 `\xintFGtoC`

New with 1.09m of 2014/02/26. Computes the common initial coefficients for the two fractions *f* and *g*, and outputs them as a sequence of braced items.

```

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

```

```

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

```

9.12 \xintFtoCC

```

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

```

```

409 }%
410 \def\XINT_ftcc_loop_c #1#2%
411 {%
412   \expandafter\XINT_ftcc_loop_d
413   \romannumeral0\xintsub {#2}{#1[0]}\Z {#1}%
414 }%
415 \def\XINT_ftcc_loop_d #1%
416 {%
417   \xint_UDzerominusfork
418   #1-\XINT_ftcc_end
419   0#1\XINT_ftcc_loop_N
420   0-\XINT_ftcc_loop_P #1}%
421 \krof
422 }%
423 \def\XINT_ftcc_end #1\Z #2#3{ #3#2}%
424 \def\XINT_ftcc_loop_P #1\Z #2#3%
425 {%
426   \expandafter\XINT_ftcc_loop_a\expandafter
427   {\romannumeral0\xintdiv {1[0]}\Z {#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]}\Z {#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\xintraawithzeros {#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\xintraawithzeros {#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\xintrawithzeros {#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\xintrawithzeros {#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, ε -TeX and reload detection	245
10.2	Package identification	246
10.3	Locking and unlocking	246
10.4	<code>\XINT_expr_wrap</code> , <code>\XINT_iiexpr_wrap</code>	247
10.5	<code>\XINT_protectii</code> , <code>\XINT_expr_usethe</code>	247
10.6	<code>\XINT_expr_print</code> , <code>\XINT_iiexpr_print</code> , <code>\XINT_boolexpr_print</code>	247
10.7	<code>\xintexpr</code> , <code>\xintiexpr</code> , <code>\xintfloatexpr</code> , <code>\xintiieexpr</code> , <code>\xinttheexpr</code> , etc...	247
10.8	<code>\xintthe</code>	247
10.9	<code>\xintthecoords</code>	248
10.10	<code>\xintbareeval</code> , <code>\xintbarefloateval</code> , <code>\xintbareiieval</code>	248
10.11	<code>\xintthebareeval</code> , <code>\xintthebarefloateval</code> , <code>\xintthebareiieval</code>	248
10.12	<code>\xinteval</code> , <code>\xintiieval</code>	248
10.13	<code>\xintieval</code> , <code>\XINT_iexpr_wrap</code>	248
10.14	<code>\xintfloateval</code> , <code>\XINT_flexpr_wrap</code> , <code>\XINT_flexpr_print</code>	249
10.15	<code>\xintboolexpr</code> , <code>\xinttheboolexpr</code>	249
10.16	<code>\xintifboolexpr</code> , <code>\xintifboolfloatexpr</code> , <code>\xintifbooliiexpr</code>	249
10.17	Macros handling csv lists on output (for <code>\XINT_expr_print</code> et al. routines)	250
10.18	<code>\XINT_expr_getnext</code> : fetching some number then an operator	252
10.19	The integer or decimal number or hexa-decimal number or function name or variable name or special hacky things big parser	253
10.20	<code>\XINT_expr_getop</code> : finding the next operator or closing parenthesis or end of expression	259
10.21	Expansion spanning; opening and closing parentheses	261
10.22	<code> </code> , <code> </code> , <code>&</code> , <code>&&</code> , <code><</code> , <code>></code> , <code>=</code> , <code>==</code> , <code><=</code> , <code>>=</code> , <code>!=</code> , <code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>^</code> , <code>**</code> , <code>//</code> , <code>/:</code> , <code>...</code> , <code>..[</code> , <code>]</code> , <code>][</code> , <code>][:</code> , <code>:</code> , and <code>++</code> operators	262
10.23	Macros for a..b list generation	271
10.24	Macros for a..[d]..b list generation	273
10.25	The comma as binary operator	275
10.26	The minus as prefix operator of variable precedence level	276
10.27	<code>?</code> as two-way and <code>??</code> as three-way conditionals with braced branches	277
10.28	<code>!</code> as postfix factorial operator	277
10.29	The A/B[N] mechanism	277
10.30	<code>\XINT_expr_op_`</code> for recognizing functions	278
10.31	The <code>bool</code> , <code>to gl</code> , <code>protect</code> pseudo “functions”	278
10.32	The <code>break</code> function	279
10.33	The <code>qint</code> , <code>qfrac</code> , <code>qfloat</code> “functions”	279
10.34	<code>\XINT_expr_op__</code> for recognizing variables	279
10.35	User defined variables: <code>\xintdefvar</code> , <code>\xintdefiivar</code> , <code>\xintdeffloatvar</code>	279
10.36	<code>\xintunassignvar</code>	280
10.37	<code>seq</code> and the implementation of dummy variables	281
10.38	<code>add</code> , <code>mul</code>	287
10.39	<code>subs</code>	288
10.40	<code>rseq</code>	288
10.41	<code>rrseq</code>	290
10.42	<code>iter</code>	292
10.43	Macros handling csv lists for functions with multiple comma separated arguments in expressions	294

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, and reversed functions	297
10.45	f-expandable versions of the <code>\xintSeqB::csv</code> and alike routines, for <code>\xintNewExpr</code>	302
10.46	User defined functions: <code>\xintdeffunc</code> , <code>\xintdefiifunc</code> , <code>\xintdeffloatfunc</code>	304
10.47	<code>\xintNewExpr</code> , <code>\xintNewIExpr</code> , <code>\xintNewFloatExpr</code> , <code>\xintNewIExpr</code>	305

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.

- `\xinttexpr`, `\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 `"`, 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 **1.2e** of [2015/11/22].

10.1 Catcodes, ε -TeX and reload detection

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

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

```

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

```

```

28     \ifx\w\relax % but xintfrac.sty not yet loaded.
29         \expandafter\def\expandafter\z\expandafter
30             {\z\input xintfrac.sty\relax}%
31     \fi
32     \ifx\t\relax % but xinttools.sty not yet loaded.
33         \expandafter\def\expandafter\z\expandafter
34             {\z\input xinttools.sty\relax}%
35     \fi
36 \else
37     \def\empty {}%
38     \ifx\x\empty % LaTeX, first loading,
39     % variable is initialized, but \ProvidesPackage not yet seen
40         \ifx\w\relax % xintfrac.sty not yet loaded.
41             \expandafter\def\expandafter\z\expandafter
42                 {\z\RequirePackage{xintfrac}}%
43         \fi
44         \ifx\t\relax % xinttools.sty not yet loaded.
45             \expandafter\def\expandafter\z\expandafter
46                 {\z\RequirePackage{xinttools}}%
47         \fi
48     \else
49         \aftergroup\endinput % xintexpr already loaded.
50     \fi
51 \fi
52 \fi
53 \z%
54 \XINTsetupcatcodes%

```

10.2 Package identification

```

55 \XINT_providespackage
56 \ProvidesPackage{xintexpr}%
57 [2015/11/22 v1.2e 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 v1.2 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 %%%%%%%%%%%
75 \def\XINT_expr_unlock {\expandafter\XINT_expr_unlock_a\string }%
76 \def\XINT_expr_unlock_a #1.={}%
77 \def\XINT_expr_unexpectedtoken {\xintError:ignored }%
78 \let\XINT_expr_done\space

```

10.4 \XINT_expr_wrap, \XINT_iiexpr_wrap

```

79 \def\XINT_expr_wrap { !\XINT_expr_usethe\XINT_protectii\XINT_expr_print }%
80 \def\XINT_iiexpr_wrap { !\XINT_expr_usethe\XINT_protectii\XINT_iiexpr_print }%

```

10.5 \XINT_protectii, \XINT_expr_usethe

```

81 \def\XINT_protectii #1{\noexpand\XINT_protectii\noexpand #1\noexpand }%
82 \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.

```

83 \def\XINT_expr_print      #1{\xintSPRaw::csv {\XINT_expr_unlock #1}}%
84 \def\XINT_iiexpr_print   #1{\xintCSV::csv {\XINT_expr_unlock #1}}%
85 \def\XINT_boolexpr_print #1{\xintIsTrue::csv {\XINT_expr_unlock #1}}%

```

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

```

86 \def\xintexpr      {\romannumeral0\xinteval      }%
87 \def\xintiexpr     {\romannumeral0\xintieval     }%
88 \def\xintfloatexpr {\romannumeral0\xintfloateval }%
89 \def\xintiiexpr     {\romannumeral0\xintiieval    }%
90 \def\xinttheexpr
91   {\romannumeral`&&\expandafter\XINT_expr_print\romannumeral0\xintbareeval }%
92 \def\xinttheiexpr   {\romannumeral`&&\xintthe\xintiexpr }%
93 \def\xintthefloatexpr {\romannumeral`&&\xintthe\xintfloatexpr }%
94 \def\xinttheiiexpr
95   {\romannumeral`&&\expandafter\XINT_iiexpr_print\romannumeral0\xintbareiieval }%

```

10.8 \xintthe

```

96 \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!

```

97 \def\xintthecoords #1{\romannumeral`&&\expandafter\expandafter\expandafter
98     \XINT_thecoords_a
99     \expandafter\xint_gobble_iii\romannumeral0#1}%
100 \def\XINT_thecoords_a #1#2% #1=print macro, indispensable for scientific notation
101     {\expandafter\XINT_expr_unlock\csname.=\expandafter\XINT_thecoords_b
102         \romannumeral`&&@#1#2,!,!,^ \endcsname }%
103 \def\XINT_thecoords_b #1#2,#3#4,%
104     {\xint_gob_til_! #3\XINT_thecoords_c ! (#1#2, #3#4)\XINT_thecoords_b }%
105 \def\XINT_thecoords_c #1^{}%
```

10.10 `\xintbareeval`, `\xintbarefloateval`, `\xintbareiieval`

```

106 \def\xintbareeval
107     {\expandafter\XINT_expr_until_end_a\romannumeral`&&\XINT_expr_getnext }%
108 \def\xintbarefloateval
109     {\expandafter\XINT_flexpr_until_end_a\romannumeral`&&\XINT_expr_getnext }%
110 \def\xintbareiieval
111     {\expandafter\XINT_iexpr_until_end_a\romannumeral`&&\XINT_expr_getnext }%
```

10.11 `\xintthebareeval`, `\xintthebarefloateval`, `\xintthebareiieval`

```

112 \def\xintthebareeval      {\expandafter\XINT_expr_unlock\romannumeral0\xintbareeval}%
113 \def\xintthebarefloateval {\expandafter\XINT_expr_unlock\romannumeral0\xintbarefloateval}%
114 \def\xintthebareiieval    {\expandafter\XINT_expr_unlock\romannumeral0\xintbareiieval}%

```

10.12 `\xinteval`, `\xintiieval`

```

115 \def\xinteval      {\expandafter\XINT_expr_wrap\romannumeral0\xintbareeval }%
116 \def\xintiieval    {\expandafter\XINT_iexpr_wrap\romannumeral0\xintbareiieval }%

```

10.13 `\xintieval`, `\XINT_iexpr_wrap`

Optional argument since 1.1.

```

117 \def\xintieval #1%
118     {\ifx [#1\expandafter\XINT_iexpr_withopt\else\expandafter\XINT_iexpr_noopt \fi #1}%
119 \def\XINT_iexpr_noopt
120     {\expandafter\XINT_iexpr_wrap \expandafter 0\romannumeral0\xintbareeval }%
121 \def\XINT_iexpr_withopt [#1]%
122     {%
123     \expandafter\XINT_iexpr_wrap\expandafter
124     {\the\numexpr \xint_zapspace #1 \xint_gobble_i\expandafter}%
125     \romannumeral0\xintbareeval
126     }%
```



```

127 \def\XINT_iexpr_wrap #1#2%
128 {%
129     \expandafter\XINT_expr_wrap
130     \csname .=\xintRound::csv {#1}\XINT_expr_unlock #2}\endcsname
131 }%
```

10.14 \xintfloateval, \XINT_flexpr_wrap, \XINT_flexpr_print

Optional argument since 1.1

```

132 \def\xintfloateval #1%
133 {%
134     \ifx [#1\expandafter\XINT_flexpr_withopt_a\else\expandafter\XINT_flexpr_noopt
135     \fi #1%
136 }%
137 \def\XINT_flexpr_noopt
138 {%
139     \expandafter\XINT_flexpr_withopt_b\expandafter\xinttheDigits
140     \romannumeral0\xintbarefloateval
141 }%
142 \def\XINT_flexpr_withopt_a [#1]%
143 {%
144     \expandafter\XINT_flexpr_withopt_b\expandafter
145     {\the\numexpr\xint_zapspace #1 \xint_gobble_i\expandafter}%
146     \romannumeral0\xintbarefloateval
147 }%
148 \def\XINT_flexpr_withopt_b #1#2%
149 {%
150     \expandafter\XINT_flexpr_wrap\csname .;#1.=% ; and not : as before b'cause NewExpr
151     \XINTinFloat::csv {#1}\XINT_expr_unlock #2}\endcsname
152 }%
153 \def\XINT_flexpr_wrap { !\XINT_expr_usethe\XINT_protectii\XINT_flexpr_print }%
154 \def\XINT_flexpr_print #1%
155 {%
156     \expandafter\xintPFloat::csv
157     \romannumeral`&&\expandafter\XINT_expr_unlock_sp\string #1!%
158 }%
159 \catcode`: 12
160     \def\XINT_expr_unlock_sp #1.;#2.=#3!{{#2}{#3}}%
161 \catcode`: 11
```

10.15 \xintboolexpr, \xinttheboolexpr

```

162 \def\xintboolexpr      {\romannumeral0\expandafter\expandafter\expandafter
163     \XINT_boolexpr_done \expandafter\xint_gobble_iv\romannumeral0\xinteval }%
164 \def\xinttheboolexpr   {\romannumeral`&&\expandafter\expandafter\expandafter
165     \XINT_boolexpr_print\expandafter\xint_gobble_iv\romannumeral0\xinteval }%
166 \def\XINT_boolexpr_done { !\XINT_expr_usethe\XINT_protectii\XINT_boolexpr_print }%
```

10.16 \xintifboolexpr, \xintifboolfloatexpr, \xintifbooliiexpr

Do not work with comma separated expressions.

```

167 \def\xintifboolexpr      #1{\romannumeral0\xintifnotzero {\xinttheexpr #1\relax}}%
```

```
168 \def\xintifboolfloatexpr #1{\romannumeral0\xintifnotzero {\xintthefloatexpr #1\relax}}%
169 \def\xintifbooliiexpr #1{\romannumeral0\xintifnotzero {\xinttheiexpr #1\relax}}%
```

10.17 Macros handling csv lists on output (for `\XINT_expr_print` et al. routines)

10.17.1	<code>\XINT::_end</code>	250
10.17.2	<code>\xintCSV::csv</code>	250
10.17.3	<code>\xintSPRaw, \xintSPRaw::csv</code>	250
10.17.4	<code>\xintIsTrue::csv</code>	251
10.17.5	<code>\xintRound::csv</code>	251
10.17.6	<code>\XINTinFloat::csv</code>	251
10.17.7	<code>\xintPFloat::csv</code>	251

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`

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

10.17.2 `\xintCSV::csv`

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

```
171 \def\xintCSV::csv #1{\expandafter\XINT_csv::_a\romannumeral`&&@#1,^}%
172 \def\XINT_csv::_a {\XINT_csv::_b {}}%
173 \def\XINT_csv::_b #1#2,{\expandafter\XINT_csv::_c \romannumeral`&&@#2,{#1}}%
174 \def\XINT_csv::_c #1{\if ^#1\expandafter\XINT::_end\fi\XINT_csv::_d #1}%
175 \def\XINT_csv::_d #1,#2{\XINT_csv::_b {#2, #1}}% possibly, item #1 is empty.
```

10.17.3 `\xintSPRaw, \xintSPRaw::csv`

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

Same added `\romannumeral-`0` in 1.1a for `\xintNewExpr` purposes.

```
176 \def\xintSPRaw {\romannumeral0\xintsprawl}%
177 \def\xintsprawl #1{\expandafter\XINT_sprawl\romannumeral`&&@#1[\W]}%
178 \def\XINT_sprawl #1[#2#3]{\xint_gob_til_W #2\XINT_sprawl_a\W\XINT_sprawl_p #1[#2#3]}%
179 \def\XINT_sprawl_a\W\XINT_sprawl_p #1[\W]{ #1}%
180 \def\XINT_sprawl_p #1[\W]{\xintprawl {#1}}%
181 \def\xintSPRaw::csv #1{\romannumeral0\expandafter\XINT_sprawl::_a\romannumeral`&&@#1,^}%
182 \def\XINT_sprawl::_a {\XINT_sprawl::_b {}}%
183 \def\XINT_sprawl::_b #1#2,{\expandafter\XINT_sprawl::_c \romannumeral`&&@#2,{#1}}%
184 \def\XINT_sprawl::_c #1{\if ,#1\xint_dothis\XINT_sprawl::_e\fi
185     \if ^#1\xint_dothis\XINT::_end\fi
186     \xint_orthat\XINT_sprawl::_d #1}%
187 \def\XINT_sprawl::_d #1,{\expandafter\XINT_sprawl::_e\romannumeral0\XINT_sprawl #1[\W],}%
188 \def\XINT_sprawl::_e #1,#2{\XINT_sprawl::_b {#2, #1}}%
```

10.17.4 `\xintIsTrue::csv`

```

189 \def\xintIsTrue::csv #1{\romannumeral0\expandafter\xint_istrue::_a\romannumeral`&&@#1,^,}%
190 \def\xint_istrue::_a {\xint_istrue::_b {}}%
191 \def\xint_istrue::_b #1#2,{\expandafter\xint_istrue::_c \romannumeral`&&@#2,{#1}}%
192 \def\xint_istrue::_c #1{\if ,#1\xint_dothis\xint_istrue::_e\fi
193         \if ^#1\xint_dothis\xint_istrue::_d #1}%
194         \xint_orthat\xint_istrue::_d #1}%
195 \def\xint_istrue::_d #1,{\expandafter\xint_istrue::_e\romannumeral0\xintisnotzero {#1},}%
196 \def\xint_istrue::_e #1,#2{\xint_istrue::_b {#2, #1}}%

```

10.17.5 `\xintRound::csv`

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

```

197 \def\xint_:::end #1,#2#3{\xint_gobble_i #3}%
198 \def\xintRound::csv #1#2{\romannumeral0\expandafter\xint_round::_b\expandafter
199     {\the\numexpr#1\expandafter}\expandafter{\expandafter}\romannumeral`&&@#2,^,}%
200 \def\xint_round::_b #1#2#3,{\expandafter\xint_round::_c \romannumeral`&&@#3,{#1}{#2}}%
201 \def\xint_round::_c #1{\if ,#1\xint_dothis\xint_round::_e\fi
202         \if ^#1\xint_dothis\xint_:::end\fi
203         \xint_orthat\xint_round::_d #1}%
204 \def\xint_round::_d #1,#2{%
205     \expandafter\xint_round::_e\romannumeral0\ifnum#2>\xint_c_
206     \expandafter\xintround\else\expandafter\xintiround\fi {#2}{#1},{#2}}%
207 \def\xint_round::_e #1,#2#3{\XINT_round::_b {#2}{#3, #1}}%

```

10.17.6 `\XINTinFloat::csv`

Pour `\xintfloatexpr`. Attention, prépare sous la forme `digits[N]` pour traitement par les macros. Pas utilisé en sortie. Utilise `\XINT_:::end`.

1.1a: I believe this is not needed for `\xintNewExpr`, as it is removed by re-defined `\XINT_flexpr_wrap` code, hence no need to add the extra `\romannumeral`0`. Sub-expressions in `\xintNewExpr` are not supported.

I didn't start and don't want now to think about it at all.

```

208 \def\XINTinFloat::csv #1#2{\romannumeral0\expandafter\XINT_infloat::_b\expandafter
209     {\the\numexpr #1\expandafter}\expandafter{\expandafter}\romannumeral`&&@#2,^,}%
210 \def\XINT_infloat::_b #1#2#3,{\XINT_infloat::_c #3,{#1}{#2}}%
211 \def\XINT_infloat::_c #1{\if ,#1\xint_dothis\XINT_infloat::_e\fi
212         \if ^#1\xint_dothis\XINT_:::end\fi
213         \xint_orthat\XINT_infloat::_d #1}%
214 \def\XINT_infloat::_d #1,#2%
215     {\expandafter\XINT_infloat::_e\romannumeral0\XINTinfloat [#2]{#1},{#2}}%
216 \def\XINT_infloat::_e #1,#2#3{\XINT_infloat::_b {#2}{#3, #1}}%

```

10.17.7 `\xintPFloat::csv`

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

```

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

```

```

218 {\the\numexpr #1\expandafter}\expandafter{\expandafter}\romannumeral`&&@#2,^,%
219 \def\XINT_pfloat::_b #1#2#3,{\expandafter\XINT_pfloat::_c \romannumeral`&&@#3,{#1}{#2}}%
220 \def\XINT_pfloat::_c #1{\if ,#1\xint_dothis\XINT_pfloat::_e\fi
221 \if ^#1\xint_dothis\XINT::_end\fi
222 \xint_orthat\XINT_pfloat::_d #1}%
223 \def\XINT_pfloat::_d #1,#2%
224 {\expandafter\XINT_pfloat::_e\romannumeral0\XINT_pfloat_opt [\xint_relax #2]{#1},{#2}}%
225 \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.

```

226 \def\XINT_expr_getnext #1%
227 {%
228 \expandafter\XINT_expr_getnext_a\romannumeral`&&@#1%
229 }%
230 \def\XINT_expr_getnext_a #1%
231 {% screens out sub-expressions and \count or \dimen registers/variables
232 \xint_gob_til_! #1\XINT_expr_subexpr !% recall this ! has catcode 11
233 \ifcat\relax#1% \count or \numexpr etc... token or count, dimen, skip cs
234 \expandafter\XINT_expr_countetc
235 \else
236 \expandafter\expandafter\expandafter\XINT_expr_getnextfork\expandafter\string
237 \fi
238 #1%
239 }%
240 \def\XINT_expr_subexpr !#1\fi !{\expandafter\XINT_expr_getop\xint_gobble_iii }%

```

1.2 adds \ht, \dp, \wd and the eTeX font things.

```

241 \def\XINT_expr_countetc #1%
242 {%
243 \ifx\count#1\else\ifx\dimen#1\else\ifx\numexpr#1\else\ifx\dimexpr#1\else
244 \ifx\skip#1\else\ifx\glueexpr#1\else\ifx\fontdimen#1\else\ifx\ht#1\else
245 \ifx\dp#1\else\ifx\wd#1\else\ifx\fontcharht#1\else\ifx\fontcharwd#1\else
246 \ifx\fontcharhp#1\else\ifx\fontcharic#1\else
247 \XINT_expr_unpackvar
248 \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
249 \expandafter\XINT_expr_getnext\number #1%
250 }%
251 \def\XINT_expr_unpackvar\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
252 \expandafter\XINT_expr_getnext\number #1%
253 {\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
254 \expandafter\XINT_expr_getop\csname .=\number#1\endcsname }%
255 \begingroup
256 \lccode`*=`#
257 \lowercase{\endgroup
258 \def\XINT_expr_getnextfork #1{%
259 \if#1*\xint_dothis {\XINT_expr_scan_macropar *}\fi
260 \if#1[\xint_dothis {\xint_c_xviii ({} }\fi

```

```

261 \if#1+\xint_dothis \XINT_expr_getnext \fi
262 \if#1.\xint_dothis {\XINT_expr_startdec}\fi
263 \if#1-\xint_dothis -\fi
264 \if#1(\xint_dothis {\xint_c_xviii ({})}\fi
265 \xint_orthat {\XINT_expr_scan_nbr_or_func #1}%
266 }}%
267 \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)	254
10.19.2	Fractional part	255
10.19.3	Scientific notation	256
10.19.4	Hexadecimal numbers	257
10.19.5	Parsing names of functions and variables	258

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 v1.2, `\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.

```

268 \catcode96 11 % `
269 \def\xint_expr_scan_nbr_or_func #1% this #1 has necessarily here catcode 12
270 {%
271 \if "#1\xint_dothis \XINT_expr_scanhex_I\fi
272 \if `#1\xint_dothis {\XINT_expr_onlitteral_}\fi
273 \ifnum \xint_c_ix<1#1 \xint_dothis \XINT_expr_startint\fi
274 \xint_orthat \XINT_expr_scanfunc #1%
275 }%
276 \def\xint_expr_onlitteral_` #1#2#3({\xint_c_xviii `{#2}}}%

```

```

277 \catcode96 12 % `
278 \def\XINT_expr_startint #1%
279 {%
280   \if #10\expandafter\XINT_expr_gobz_a\else\XINT_expr_scanint_a\fi #1%
281 }%
282 \def\XINT_expr_scanint_a #1#2%
283   {\expandafter\XINT_expr_getop\csname.=#1%
284     \expandafter\XINT_expr_scanint_b\romannumeral`&&@#2}%
285 \def\XINT_expr_gobz_a #1%
286   {\expandafter\XINT_expr_getop\csname.=#1%
287     \expandafter\XINT_expr_gobz_scanint_b\romannumeral`&&@#1}%
288 \def\XINT_expr_startdec #1%
289   {\expandafter\XINT_expr_getop\csname.=#1%
290     \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-negligible. I don't think the doubled \string is a serious penalty.

```

291 \def\XINT_expr_scanint_b #1%
292 {%
293   \ifcat \relax #1\expandafter\XINT_expr_scanint_endbycs\expandafter #1\fi
294   \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanint_c\fi
295   \string#1\XINT_expr_scanint_d
296 }%
297 \def\XINT_expr_scanint_d #1%
298 {%
299   \expandafter\XINT_expr_scanint_b\romannumeral`&&@#1%
300 }%
301 \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.

```

302 \def\XINT_expr_scanint_c\string #1\XINT_expr_scanint_d
303 {%
304   \if e#1\xint_dothis{[\the\numexpr0\XINT_expr_scanexp_a +}\fi
305   \if E#1\xint_dothis{[\the\numexpr0\XINT_expr_scanexp_a +}\fi
306   \if .#1\xint_dothis{\XINT_expr_startdec_a .}\fi
307   \xint_orthat {\endcsname #1}%
308 }%
309 \def\XINT_expr_startdec_a .#1%
310 {%
311   \expandafter\XINT_expr_scandec_a\romannumeral`&&@#1%
312 }%

```

```

313 \def\XINT_expr_scandec_a #1%
314 {%
315   \if .#1\xint_dothis{\endcsname..}\fi
316   \xint_orthat {\XINT_expr_scandec_b 0.#1}%
317 }%
318 \def\XINT_expr_gobz_scanint_b #1%
319 {%
320   \ifcat \relax #1\expandafter\XINT_expr_gobz_scanint_endbycs\expandafter #1\fi
321   \ifnum\xint_c_x<1\string#1 \else\expandafter\XINT_expr_gobz_scanint_c\fi
322   \string#1\XINT_expr_scanint_d
323 }%
324 \def\XINT_expr_gobz_scanint_endbycs#1#2\XINT_expr_scanint_d{0\endcsname #1}%
325 \def\XINT_expr_gobz_scanint_c\string #1\XINT_expr_scanint_d
326 {%
327   \if e#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
328   \if E#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
329   \if .#1\xint_dothis{\XINT_expr_gobz_startdec_a .}\fi
330   \if 0#1\xint_dothis\XINT_expr_gobz_scanint_d\fi
331   \xint_orthat {0\endcsname #1}%
332 }%
333 \def\XINT_expr_gobz_scanint_d #1%
334 {%
335   \expandafter\XINT_expr_gobz_scanint_b\romannumeral`&&@#1%
336 }%
337 \def\XINT_expr_gobz_startdec_a .#1%
338 {%
339   \expandafter\XINT_expr_gobz_scandec_a\romannumeral`&&@#1%
340 }%
341 \def\XINT_expr_gobz_scandec_a #1%
342 {%
343   \if .#1\xint_dothis{0\endcsname..}\fi
344   \xint_orthat {\XINT_expr_gobz_scandec_b 0.#1}%
345 }%

```

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 ?

```

346 \def\XINT_expr_scandec_b #1.#2%
347 {%
348   \ifcat \relax #2\expandafter\XINT_expr_scandec_endbycs\expandafter#2\fi
349   \ifnum\xint_c_ix<1\string#2 \else\expandafter\XINT_expr_scandec_c\fi
350   \string#2\expandafter\XINT_expr_scandec_d\the\numexpr #1-\xint_c_i.%
351 }%
352 \def\XINT_expr_scandec_endbycs #1#2\XINT_expr_scandec_d
353   \the\numexpr#3-\xint_c_i.{[#3]\endcsname #1}%
354 \def\XINT_expr_scandec_d #1.#2%
355 {%
356   \expandafter\XINT_expr_scandec_b

```

```

357 \the\numexpr #1\expandafter.\romannumeral`&&@#2%
358 }%
359 \def\XINT_expr_scandec_c\string #1#2\the\numexpr#3-\xint_c_i.%
360 {%
361 \if e#1\xint_dothis{[\the\numexpr#3\XINT_expr_scanexp_a +}\fi
362 \if E#1\xint_dothis{[\the\numexpr#3\XINT_expr_scanexp_a +}\fi
363 \xint_orthat {[#3]\endcsname #1}%
364 }%

```

For bugfix release 1.2a, I only need code that works, I will think another day about making it perhaps more elegant/efficient.

```

365 \def\XINT_expr_gobz_scandec_b #1.#2%
366 {%
367 \ifcat \relax #2\expandafter\XINT_expr_gobz_scandec_endbycs\expandafter#2\fi
368 \ifnum\xint_c_ix<1\string#2 \else\expandafter\XINT_expr_gobz_scandec_c\fi
369 \if0#2\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
370 {\expandafter\XINT_expr_gobz_scandec_b}%
371 {\string#2\expandafter\XINT_expr_scandec_d}\the\numexpr#1-\xint_c_i.%
372 }%

```

Even if number is zero leave a trace in [...] of its formation ? for code tracing purposes ? Finally no. But in case of exponential part, yes as I don't want to write extra code just to handle that case.

```

373 \def\XINT_expr_gobz_scandec_endbycs #1#2\xint_c_i.{0[0]\endcsname #1}%
374 \def\XINT_expr_gobz_scandec_c\if0#1#2\fi #3\xint_c_i.%
375 {%
376 \if e#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
377 \if E#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
378 \xint_orthat {0[0]\endcsname #1}%
379 }%

```

10.19.3 Scientific notation

Some pluses and minuses are allowed at the start of the scientific part, however not later, and no parenthesis.

```

380 \def\XINT_expr_scanexp_a #1#2%
381 {%
382 #1\expandafter\XINT_expr_scanexp_b\romannumeral`&&@#2%
383 }%
384 \def\XINT_expr_scanexp_b #1%
385 {%
386 \ifcat \relax #1\expandafter\XINT_expr_scanexp_endbycs\expandafter #1\fi
387 \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanexp_c\fi
388 \string#1\XINT_expr_scanexp_d
389 }%
390 \def\XINT_expr_scanexp_endbycs#1#2\XINT_expr_scanexp_d []\endcsname #1}%
391 \def\XINT_expr_scanexp_d #1%
392 {%
393 \expandafter\XINT_expr_scanexp_bb\romannumeral`&&@#1%
394 }%
395 \def\XINT_expr_scanexp_c\string #1\XINT_expr_scanexp_d

```



```

396 {%
397   \if + #1\xint_dothis {\XINT_expr_scanexp_a +}\fi
398   \if - #1\xint_dothis {\XINT_expr_scanexp_a -}\fi
399   \xint_orthat {} \endcsname #1}%
400}%
401 \def \XINT_expr_scanexp_bb #1%
402 {%
403   \ifcat \relax #1\expandafter \XINT_expr_scanexp_endbycs_b \expandafter #1\fi
404   \ifnum \xint_c_ix < 1 \string #1 \else \expandafter \XINT_expr_scanexp_cb \fi
405   \string #1 \XINT_expr_scanexp_db
406}%
407 \def \XINT_expr_scanexp_endbycs_b #1 #2 \XINT_expr_scanexp_db {} \endcsname #1}%
408 \def \XINT_expr_scanexp_db #1%
409 {%
410   \expandafter \XINT_expr_scanexp_bb \romannumeral `&& @ #1%
411}%
412 \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`).

```

413 \def \XINT_expr_scanhex_I #1% #1="
414 {%
415   \expandafter \XINT_expr_getop \csname . = \expandafter
416   \XINT_expr_unlock_hex_in \csname . = \XINT_expr_scanhexI_a
417}%
418 \def \XINT_expr_scanhexI_a #1%
419 {%
420   \ifcat #1 \relax \xint_dothis {.> \endcsname \endcsname #1} \fi
421   \ifx ! #1 \xint_dothis {.> \endcsname \endcsname !} \fi
422   \xint_orthat { \expandafter \XINT_expr_scanhexI_aa \string #1}%
423}%
424 \def \XINT_expr_scanhexI_aa #1%
425 {%
426   \if \ifnum `#1 > ` /
427     \ifnum `#1 > ` 9
428       \ifnum `#1 > ` @
429         \ifnum `#1 > ` F
430           0 \else 1 \fi \else 0 \fi \else 1 \fi \else 0 \fi 1%
431     \expandafter \XINT_expr_scanhexI_b
432   \else
433     \if . #1%
434       \expandafter \xint_firstoftwo
435     \else % gather what we got so far, leave catcode 12 #1 in stream
436       \expandafter \xint_secondoftwo
437     \fi
438     { \expandafter \XINT_expr_scanhex_transition}%
439     { \xint_afterfi {.> \endcsname \endcsname}}}%

```

```

440 \fi
441 #1%
442 }%
443 \def\XINT_expr_scanhexI_b #1#2%
444 {%
445   #1\expandafter\XINT_expr_scanhexI_a\romannumeral`&&@#2%
446 }%
447 \def\XINT_expr_scanhex_transition .#1%
448 {%
449   \expandafter.\expandafter.\expandafter
450   \XINT_expr_scanhexII_a\romannumeral`&&@#1%
451 }%
452 \def\XINT_expr_scanhexII_a #1%
453 {%
454   \ifcat #1\relax\xint_dothis{\endcsname\endcsname#1}\fi
455   \ifx !#1\xint_dothis{\endcsname\endcsname !}\fi
456   \xint_orthat {\expandafter\XINT_expr_scanhexII_aa\string #1}%
457 }%
458 \def\XINT_expr_scanhexII_aa #1%
459 {%
460   \if\ifnum`#1>`/
461     \ifnum`#1>`9
462     \ifnum`#1>`@
463     \ifnum`#1>`F
464     0\else1\fi\else0\fi\else1\fi\else0\fi 1%
465     \expandafter\XINT_expr_scanhexII_b
466   \else
467     \xint_afterfi {\endcsname\endcsname}%
468   \fi
469   #1%
470 }%
471 \def\XINT_expr_scanhexII_b #1#2%
472 {%
473   #1\expandafter\XINT_expr_scanhexII_a\romannumeral`&&@#2%
474 }%

```

10.19.5 Parsing names of functions and variables

```

475 \def\XINT_expr_scanfunc
476 {%
477   \expandafter\XINT_expr_func\romannumeral`&&@\XINT_expr_scanfunc_a
478 }%
479 \def\XINT_expr_scanfunc_a #1#2%
480 {%
481   \expandafter #1\romannumeral`&&@\expandafter\XINT_expr_scanfunc_b\romannumeral`&&@#2%
482 }%

```

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 v1.2d 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.

```

483 \def\XINT_expr_scanfunc_b #1%
484 {%
485   \ifx !#1\xint_dothis{(_)\fi
486   \ifcat \relax#1\xint_dothis{(_)\fi
487   \if (#1\xint_dothis{\xint_firstoftwo{(`)}}\fi
488   \if @#1\xint_dothis \XINT_expr_scanfunc_a \fi
489   \if _#1\xint_dothis \XINT_expr_scanfunc_a \fi
490   \ifnum \xint_c_ix<1\string#1 \xint_dothis \XINT_expr_scanfunc_a \fi
491   \ifcat a#1\xint_dothis \XINT_expr_scanfunc_a \fi
492   \xint_orthat {(_}%
493     #1%
494 }%
```

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.

```

495 \def\XINT_expr_func #1(#2{\if _#2\xint_dothis\XINT_expr_op__\fi
496   \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.

```

497 \def\XINT_expr_getop #1#2% this #1 is the current locked computed value
498 {%
499   \expandafter\XINT_expr_getop_a\expandafter #1\romannumeral`&&@#2%
500 }%
501 \catcode`* 11
502 \def\XINT_expr_getop_a #1#2%
503 {%
504   \ifx   \relax #2\xint_dothis\xint_firstofthree\fi
505   \ifcat \relax #2\xint_dothis\xint_secondofthree\fi
506   \if   _#2\xint_dothis      \xint_secondofthree\fi
507   \if   @#2\xint_dothis      \xint_secondofthree\fi
508   \if   (#2\xint_dothis      \xint_secondofthree\fi
509   \ifcat a#2\xint_dothis      \xint_secondofthree\fi
510   \xint_orthat \xint_thirdofthree
511   {\XINT_expr_founded #1}%
512   {\XINT_expr_precedence_*** *#1#2}% tacit multiplication with higher precedence
513   {\expandafter\XINT_expr_getop_b \string#2#1}%
514 }%
515 \catcode`* 12
516 \def\XINT_expr_founded {\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.

```

517 \def\XINT_expr_getop_b #1%
518 {%
519   \if '#1\xint_dothis{\XINT_expr_binopwrd }\fi
520   \if ?#1\xint_dothis{\XINT_expr_precedence_? ?}\fi
521   \xint_orthat      {\XINT_expr_scanop_a #1}%
522 }%
523 \def\XINT_expr_binopwrd #1#2'\{\expandafter\XINT_expr_foundedop_a
524   \csname XINT_expr_itself_\xint_zapspaces #2 \xint_gobble_i\endcsname #1}%
525 \def\XINT_expr_scanop_a #1#2#3%
526   {\expandafter\XINT_expr_scanop_b\expandafter #1\expandafter #2\romannumeral`&&@#3}%
527 \def\XINT_expr_scanop_b #1#2#3%
528 {%
529   \ifcat#3\relax\xint_dothis{\XINT_expr_foundedop_a #1#2#3}\fi
530   \ifcsname XINT_expr_itself_#1#3\endcsname
531     \xint_dothis

```

```

532      {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#3\endcsname #2}\fi
533 \xint_orthat {\XINT_expr_foundop_a #1#2#3}%
534 }%
535 \def\XINT_expr_scanop_c #1#2#3%
536 {%
537   \expandafter\XINT_expr_scanop_d\expandafter #1\expandafter #2\romannumeral`&&@#3%
538 }%
539 \def\XINT_expr_scanop_d #1#2#3%
540 {%
541   \ifcat#3\relax \xint_dothis{\XINT_expr_foundop #1#2#3}\fi
542   \ifcsname XINT_expr_itself_#1#3\endcsname
543     \xint_dothis
544       {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#3\endcsname #2}\fi
545   \xint_orthat {\csname XINT_expr_precedence_#1\endcsname #1#2#3}%
546 }%
547 \def\XINT_expr_foundop_a #1%
548 {%
549   \ifcsname XINT_expr_precedence_#1\endcsname
550     \csname XINT_expr_precedence_#1\endcsname\expandafter\endcsname
551     \expandafter #1%
552   \else
553     \xint_afterfi{\XINT_expr_unknown_operator {#1}\XINT_expr_getop}%
554   \fi
555 }%
556 \def\XINT_expr_unknown_operator #1{\xintError:removed \xint_gobble_i {#1}}%
557 \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.

```

558 \catcode`) 11
559 \def\XINT_tmpa #1#2#3#4% (avant #4#5)
560 {%
561   \def#1##1%
562   {%
563     \xint_UDsignfork
564       ##1{\expandafter#1\romannumeral`&&@#3}%
565       -{#2##1}%
566     \krof
567   }%
568   \def#2##1##2%
569   {%
570     \ifcase ##1\expandafter\XINT_expr_done
571     \or\xint_afterfi{\XINT_expr_extra_}
572       \expandafter #1\romannumeral`&&\XINT_expr_getop }%
573     \else
574       \xint_afterfi{\expandafter#1\romannumeral`&&\csname XINT_#4_op_##2\endcsname }%
575     \fi
576   }%
577 }%

```

```

578 \def\XINT_expr_extra_){\xintError:removed }%
579 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
580   \expandafter\XINT_tmpa
581   \csname XINT_#1_until_end_a\expandafter\endcsname
582   \csname XINT_#1_until_end_b\expandafter\endcsname
583   \csname XINT_#1_op_-vi\endcsname
584   {#1}%
585 }%
586 \def\XINT_tmpa #1#2#3#4#5#6%
587 {%
588   \def #1##1{\expandafter #3\romannumeral`&&\XINT_expr_getnext }%
589   \def #2{\expandafter #3\romannumeral`&&\XINT_expr_getnext }%
590   \def #3##1{\xint_UDsignfork
591     ##1{\expandafter #3\romannumeral`&&#5}%
592     -{#4##1}%
593     \krof }%
594   \def #4##1##2{\ifcase ##1\expandafter\XINT_expr_missing_
595     \or \csname XINT_#6_op_##2\expandafter\endcsname
596     \else
597     \xint_afterfi{\expandafter #3\romannumeral`&&\csname XINT_#6_op_##2\endcsname }%
598     \fi
599   }%
600 }%
601 \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_`) said more.

```

602 \catcode`)\ 12
603 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
604   \expandafter\XINT_tmpa
605   \csname XINT_#1_op_(\expandafter\endcsname
606   \csname XINT_#1_oparen\expandafter\endcsname
607   \csname XINT_#1_until_)_a\expandafter\endcsname
608   \csname XINT_#1_until_)_b\expandafter\endcsname
609   \csname XINT_#1_op_-vi\endcsname
610   {#1}%
611 }%
612 \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	263
10.22.2	The , &, xor, <, >, =, <=, >=, !=, //, /: and .. operators for expr and floatexpr . . .	264
10.22.3	The +, -, *, /, ^, ..[, and].. operators for expr and floatexpr	265
10.22.4	The previous operators for iiexpr	265
10.22.5	The]+,]-,]*,]/,]^, +[, -[, *[, /[, and ^[list operators	266
10.22.6	The 'and', 'or', 'xor', and 'mod' as infix operator words	268
10.22.7	The , &&, **, **[,]** operators as synonyms	268
10.22.8	List selectors: [list][N], [list][:b], [list][a:], [list][a:b]	269
10.22.9	\xintListSel::csv	271

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.

```
613 \expandafter\let\csname XINT_expr_precedence_]\endcsname\xint_c_i
614 \expandafter\let\csname XINT_expr_precedence_;\endcsname\xint_c_i
615 \let\XINT_expr_precedence_a \xint_c_xviii
616 \let\XINT_expr_precedence_!? \xint_c_ii
617 \expandafter\let\csname XINT_expr_precedence_++)\endcsname \xint_c_i
```

Comments added 2015/11/13 Here we have in particular the mechanism for post action on lists via `op_]` The precedence_]`]` is the one of a closing parenthesis. We need the closing parenthesis to do its job, hence we can not define a `op_]+` operator for example, as we want to assign it the precedence of addition not the one of closing parenthesis. The trick I used in 1.1 was to let the `op_]` insert the letter `a`, this letter exceptionnally also being a legitimate operator, launch the `_getop` and let it find a `a*`, `a+`, `a/`, `a-`, `a^`, `a**` operator standing for `]*`, `]+`, `]/`, `]^`, `]**` postfix item by item list operator. I thought I had in mind an example to show that having defined `op_a` and `precedence_a` for the letter `a` caused a reduction in syntax for this letter, but it seems I am lacking now an example.

2015/11/18: for 1.2d I accelerate `\XINT_expr_op_]` to jump over the `\XINT_expr_getop_a` which now does tacit multiplications also in front of letters, for reasons of things like, `(x+y)z`, hence it must not see the "`a`". I could have used a `catcode12` a possibly, but anyhow jumping straight to `\XINT_expr_scanop_a` skips a few expansion steps (up to the potential price of less conceptual programming if I change things in the future.)

```
618 \catcode`. 11 \catcode`= 11 \catcode`+ 11
619 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
620   \expandafter\let\csname XINT_#1_op_)\endcsname \XINT_expr_getop
621   \expandafter\let\csname XINT_#1_op_;\endcsname \space
622   \expandafter\def\csname XINT_#1_op_]\endcsname ##1{\XINT_expr_scanop_a a##1}%
623   \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.

```
624   \expandafter\def\csname XINT_#1_op_++)\endcsname ##1##2\relax
625   {\expandafter\XINT_expr_foundend \expandafter
626     {\expandafter\.=+\csname .=\xintiCeil{\XINT_expr_unlock ##1}\endcsname }}%
627 }%
628 \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.

```

629 \catcode`& 12
630 \xintFor* #1 in {{==}{<=}{>=}{!=}{&&}{||}{**}{//}{/:}{..}{...}{.}{.}{.}}%
631      {+[]{-[]{*[]}[/[]{**[]}^{[]} {a+}{a-}{a*}{a/}{a**}{a^}%
632      {}[]{}[:]{:}{}{!?}{++}{++)}}%{***}}
633   \do {\expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}}%
634 \catcode`& 7
635 \expandafter\let\csname XINT_expr_precedence_***\endcsname \xint_c_viii

```

10.22.2 The `|`, `&`, `xor`, `<`, `>`, `=`, `<=`, `>=`, `!=`, `//`, `/:` and `..` operators for `expr` and `floatexpr`

```

636 \def\XINT_tmpc #1#2#3#4#5#6#7#8%
637 {%
638   \def #1#1% \XINT_expr_op_<op> ou flexpr ou iiexpr
639   {% keep value, get next number and operator, then do until
640     \expandafter #2\expandafter ##1%
641     \romannumeral`&&\expandafter\XINT_expr_getnext}%
642   \def #2#1#1#2% \XINT_expr_until_<op>_a ou flexpr ou iiexpr
643   {\xint_UDsignfork ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
644     -{#3##1##2}%
645     \krof}%
646   \def #3##1##2##3##4% \XINT_expr_until_<op>_b ou flexpr ou iiexpr
647   {% either execute next operation now, or first do next (possibly unary)
648     \ifnum ##2>#5%
649       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
650         \csname XINT_#8_op_##3\endcsname {##4}}%
651       \else \xint_afterfi {\expandafter ##2\expandafter ##3%
652         \csname .=#6{\XINT_expr_unlock ##1}{\XINT_expr_unlock ##4}\endcsname}%
653       \fi}%
654   \let #7#5%
655}%
656 \def\XINT_tmppb #1#2#3#4#5#6%
657 {%
658   \expandafter\XINT_tmpc
659   \csname XINT_#1_op_#3\endcsname\expandafter\endcsname
660   \csname XINT_#1_until_#3_a\endcsname\expandafter\endcsname
661   \csname XINT_#1_until_#3_b\endcsname\expandafter\endcsname
662   \csname XINT_#1_op_-#5\endcsname\expandafter\endcsname
663   \csname xint_c_#4\endcsname\expandafter\endcsname
664   \csname #2#6\endcsname\expandafter\endcsname
665   \csname XINT_expr_precedence_#3\endcsname {#1}%
666}%
667 \catcode`& 12
668 \xintFor #1 in {expr, flexpr} \do {%
669   \def\XINT_tmppa ##1{\XINT_tmppb {#1}{xint}##1}%
670   \xintApplyInline {\XINT_tmppa }{%
671     {||{iii}{vi}{OR}}%
672     {&{iv}{vi}{AND}}%
673     {{xor}{iii}{vi}{XOR}}%
674     {<{v}{vi}{Lt}}%
675     {>{v}{vi}{Gt}}%

```



```

676     {={v}{vi}{Eq}}%
677     {{<=}{v}{vi}{LtorEq}}%
678     {{>=}{v}{vi}{GtorEq}}%
679     {{!=}{v}{vi}{Neq}}%
680     {{.}{iii}{vi}{Seq::csv}}%
681     {{/{vii}{vii}{DivTrunc}}%
682     {{/{vii}{vii}{Mod}}%
683   }%
684 }%
685 \catcode`& 7

```

10.22.3 The +, −, *, /, ^, ..[, and].. operators for expr and floatexpr

1.2d needed some room between /, * and ^. Hence precedence for ^ is now at 9.

```

686 \def\XINT_tmpa #1{\XINT_tmpb {expr}{xint}#1}%
687 \xintApplyInline {\XINT_tmpa }{%
688   {+{vi}{vi}{Add}}%
689   {-{vi}{vi}{Sub}}%
690   {*{vii}{vii}{Mul}}%
691   {/{vii}{vii}{Div}}%
692   {^{ix}{ix}{Pow}}%
693   {{.}{iii}{vi}{SeqA::csv}}%
694   {{.}{iii}{vi}{SeqB::csv}}%
695 }%
696 \def\XINT_tmpa #1{\XINT_tmpb {flexpr}{XINTinFloat}#1}%
697 \xintApplyInline {\XINT_tmpa }{%
698   {+{vi}{vi}{Add}}%
699   {-{vi}{vi}{Sub}}%
700   {*{vii}{vii}{Mul}}%
701   {/{vii}{vii}{Div}}%
702   {^{ix}{ix}{Power}}%
703   {{.}{iii}{vi}{SeqA::csv}}%
704   {{.}{iii}{vi}{SeqB::csv}}%
705 }%

```

10.22.4 The previous operators for iiexpr

```

706 \def\XINT_tmpa #1{\XINT_tmpb {iiexpr}{xint}#1}%
707 \catcode`& 12
708 \xintApplyInline {\XINT_tmpa }{%
709   {{iii}{vi}{OR}}%
710   {{iv}{vi}{AND}}%
711   {{xor}{iii}{vi}{XOR}}%
712   {{<}{v}{vi}{iiLt}}%
713   {{>}{v}{vi}{iiGt}}%
714   {{=}{v}{vi}{iiEq}}%
715   {{<=}{v}{vi}{iiLtorEq}}%
716   {{>=}{v}{vi}{iiGtorEq}}%
717   {{!=}{v}{vi}{iiNeq}}%
718   {+{vi}{vi}{iiAdd}}%
719   {-{vi}{vi}{iiSub}}%
720   {*{vii}{vii}{iiMul}}%
721   {/{vii}{vii}{iiDivRound}}% CHANGED IN 1.1! PREVIOUSLY DID EUCLIDEAN QUOTIENT

```

```

722  {^{\ix}{\ix}{iiPow}}}%
723  {{\..\}{iii}{vi}{iiSeqA::csv}}}%
724  {{\..\}{iii}{vi}{iiSeqB::csv}}}%
725  {{\..\}{iii}{vi}{iiSeq::csv}}}%
726  {{/{/}{vii}{vii}{iiDivTrunc}}}%
727  {{/{/}{vii}{vii}{iiMod}}}%
728 }%
729 \catcode\& 7

```

10.22.5 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`).

```

730 \def\XINT_expr_binop_inline_a
731   {\expandafter\xint_gobble_i\romannumeral`&&\XINT_expr_binop_inline_b }%
732 \def\XINT_expr_binop_inline_b #1#2,{\XINT_expr_binop_inline_c #2,{#1}}%
733 \def\XINT_expr_binop_inline_c #1{%
734   \if ,#1\xint_dothis\XINT_expr_binop_inline_e\fi
735   \if ^#1\xint_dothis\XINT_expr_binop_inline_end\fi
736   \xint_orthat\XINT_expr_binop_inline_d #1}%
737 \def\XINT_expr_binop_inline_d #1,#2{,#2{#1}\XINT_expr_binop_inline_b {#2}}%
738 \def\XINT_expr_binop_inline_e #1,#2{,\XINT_expr_binop_inline_b {#2}}%
739 \def\XINT_expr_binop_inline_end #1,#2{}%
740 \def\XINT_tmpc #1#2#3#4#5#6#7#8%
741 {%
742   \def #1#1% \XINT_expr_op_<op> ou flexpr ou iiexpr
743   {% keep value, get next number and operator, then do until
744     \expandafter #2\expandafter ##1%
745     \romannumeral`&&\expandafter\XINT_expr_getnext }%
746   \def #2#1#1#2% \XINT_expr_until_<op>_a ou flexpr ou iiexpr
747   {\xint_UDsignfork ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
748    -{#3##1#2}%
749    \krof }%
750   \def #3#1#1#2#3#4% \XINT_expr_until_<op>_b ou flexpr ou iiexpr
751   {% either execute next operation now, or first do next (possibly unary)
752     \ifnum ##2>#5%
753       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
754         \csname XINT_#8_op_##3\endcsname {##4}}%
755       \else \xint_afterfi {\expandafter ##2\expandafter ##3%
756         \csname .=\expandafter\XINT_expr_binop_inline_a\expandafter
757         {\expandafter\expandafter\expandafter#6\expandafter
758         \xint_exchangetwo_keepbraces\expandafter
759         {\expandafter\XINT_expr_unlock\expandafter ##4\expandafter}\expandafter}%
760         \romannumeral`&&\XINT_expr_unlock ##1,^,\endcsname }%
761       \fi }%
762   \let #7#5%
763 }%
764 \def\XINT_tmpb #1#2#3#4%
765 {%
766   \expandafter\XINT_tmpc
767   \csname XINT_#1_op_#2\expandafter\endcsname
768   \csname XINT_#1_until_#2_a\expandafter\endcsname
769   \csname XINT_#1_until_#2_b\expandafter\endcsname

```

```

770 \csname XINT_#1_op_-#3\expandafter\endcsname
771 \csname xint_c_#3\expandafter\endcsname
772 \csname #4\expandafter\endcsname
773 \csname XINT_expr_precedence_#2\endcsname {#1}%
774 }%

```

This is for $[x..y]*z$ syntax etc.... Attention that with 1.2d, precedence level of $^$ raised to ix to make room for $***$.

```

775 \xintApplyInline {\expandafter\XINT_tmpb \xint_firstofone}%
776 {{\expr}{a+}{vi}{xintAdd}}}%
777 {{\expr}{a-}{vi}{xintSub}}}%
778 {{\expr}{a*}{vii}{xintMul}}}%
779 {{\expr}{a/}{vii}{xintDiv}}}%
780 {{\expr}{a^}{ix}{xintPow}}}%
781 {{iiexpr}{a+}{vi}{xintiiAdd}}}%
782 {{iiexpr}{a-}{vi}{xintiiSub}}}%
783 {{iiexpr}{a*}{vii}{xintiiMul}}}%
784 {{iiexpr}{a/}{vii}{xintiiDivRound}}}%
785 {{iiexpr}{a^}{ix}{xintiiPow}}}%
786 {{flexpr}{a+}{vi}{XINTinFloatAdd}}}%
787 {{flexpr}{a-}{vi}{XINTinFloatSub}}}%
788 {{flexpr}{a*}{vii}{XINTinFloatMul}}}%
789 {{flexpr}{a/}{vii}{XINTinFloatDiv}}}%
790 {{flexpr}{a^}{ix}{XINTinFloatPower}}}%
791 }%
792 \def\XINT_tmpc #1#2#3#4#5#6#7%
793 {%
794 \def #1##1{\expandafter#2\expandafter##1\romannumeral`&&%
795 \expandafter #3\romannumeral`&&\XINT_expr_getnext }%
796 \def #2##1##2##3##4%
797 {% either execute next operation now, or first do next (possibly unary)
798 \ifnum ##2>#4%
799 \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&%
800 \csname XINT_#7_op_##3\endcsname {##4}}}%
801 \else \xint_afterfi {\expandafter ##2\expandafter ##3%
802 \csname .=\expandafter\XINT_expr_binop_inline_a\expandafter
803 {\expandafter#5\expandafter
804 {\expandafter\XINT_expr_unlock\expandafter ##1\expandafter}\expandafter}%
805 \romannumeral`&&\XINT_expr_unlock ##4,^,\endcsname }%
806 \fi }%
807 \let #6#4%
808 }%
809 \def\XINT_tmpb #1#2#3#4%
810 {%
811 \expandafter\XINT_tmpc
812 \csname XINT_#1_op_#2\expandafter\endcsname
813 \csname XINT_#1_until_#2\expandafter\endcsname
814 \csname XINT_#1_until_)_a\expandafter\endcsname
815 \csname xint_c_#3\expandafter\endcsname
816 \csname #4\expandafter\endcsname
817 \csname XINT_expr_precedence_#2\endcsname {#1}%
818 }%

```

This is for $z*[x..y]$ syntax etc...

```

819 \xintApplyInline {\expandafter\XINT_tmpb\xint_firstofone }{%
820   {{expr}}{+}{vi}{xintAdd}}%
821   {{expr}}{-}{vi}{xintSub}}%
822   {{expr}}{*}{vii}{xintMul}}%
823   {{expr}}{/}{vii}{xintDiv}}%
824   {{expr}}{^}{ix}{xintPow}}%
825   {{iiexpr}}{+}{vi}{xintiiAdd}}%
826   {{iiexpr}}{-}{vi}{xintiiSub}}%
827   {{iiexpr}}{*}{vii}{xintiiMul}}%
828   {{iiexpr}}{/}{vii}{xintiiDivRound}}%
829   {{iiexpr}}{^}{ix}{xintiiPow}}%
830   {{flexpr}}{+}{vi}{XINTinFloatAdd}}%
831   {{flexpr}}{-}{vi}{XINTinFloatSub}}%
832   {{flexpr}}{*}{vii}{XINTinFloatMul}}%
833   {{flexpr}}{/}{vii}{XINTinFloatDiv}}%
834   {{flexpr}}{^}{ix}{XINTinFloatPower}}%
835 }%

```

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

```

836 \xintFor #1 in {and,or,xor,mod} \do {%
837   \expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}}%
838 \expandafter\let\csname XINT_expr_precedence_and\expandafter\endcsname
839   \csname XINT_expr_precedence_&\endcsname
840 \expandafter\let\csname XINT_expr_precedence_or\expandafter\endcsname
841   \csname XINT_expr_precedence_|\endcsname
842 \expandafter\let\csname XINT_expr_precedence_mod\expandafter\endcsname
843   \csname XINT_expr_precedence_/\endcsname
844 \xintFor #1 in {expr, flexpr, iiexpr} \do {%
845   \expandafter\let\csname XINT_#1_op_and\expandafter\endcsname
846     \csname XINT_#1_op_&\endcsname
847   \expandafter\let\csname XINT_#1_op_or\expandafter\endcsname
848     \csname XINT_#1_op_|\endcsname
849   \expandafter\let\csname XINT_#1_op_mod\expandafter\endcsname
850     \csname XINT_#1_op_/\endcsname
851 }%

```

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

```

852 \expandafter\let\csname XINT_expr_precedence_==\expandafter\endcsname
853   \csname XINT_expr_precedence_=\endcsname
854 \expandafter\let\csname XINT_expr_precedence_&string&\expandafter\endcsname
855   \csname XINT_expr_precedence_&\endcsname
856 \expandafter\let\csname XINT_expr_precedence_||\expandafter\endcsname
857   \csname XINT_expr_precedence_|\endcsname
858 \expandafter\let\csname XINT_expr_precedence_**\expandafter\endcsname
859   \csname XINT_expr_precedence_^\endcsname
860 \expandafter\let\csname XINT_expr_precedence_a**\expandafter\endcsname
861   \csname XINT_expr_precedence_a^\endcsname
862 \expandafter\let\csname XINT_expr_precedence_**[\expandafter\endcsname
863   \csname XINT_expr_precedence_^\endcsname
864 \xintFor #1 in {expr, flexpr, iiexpr} \do {%

```

```

865 \expandafter\let\csname XINT_#1_op_==\expandafter\endcsname
866 \csname XINT_#1_op_==\endcsname
867 \expandafter\let\csname XINT_#1_op_&\string&\expandafter\endcsname
868 \csname XINT_#1_op_&\endcsname
869 \expandafter\let\csname XINT_#1_op_||\expandafter\endcsname
870 \csname XINT_#1_op_||\endcsname
871 \expandafter\let\csname XINT_#1_op_**\expandafter\endcsname
872 \csname XINT_#1_op_**\endcsname
873 \expandafter\let\csname XINT_#1_op_a**\expandafter\endcsname
874 \csname XINT_#1_op_a**\endcsname
875 \expandafter\let\csname XINT_#1_op_**[\expandafter\endcsname
876 \csname XINT_#1_op_**[\endcsname
877 }%

```

10.22.8 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.

```

878 \def\xINT_tmpa #1#2#3#4#5#6%
879 {%
880 \def #1##1% \XINT_expr_op_[
881 {%
882 \expandafter #2\expandafter ##1\romannumeral`&&\XINT_expr_getnext
883 }%
884 \def #2##1##2% \XINT_expr_until_[_a
885 {\xint_UDsignfork
886 ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
887 -{#3##1##2}%
888 \krof }%
889 \def #3##1##2##3##4% \XINT_expr_until_[_b
890 {%
891 \ifnum ##2>\xint_c_ii
892 \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
893 \csname XINT_#6_op_##3\endcsname {##4}}%
894 \else
895 \xint_afterfi
896 {\expandafter ##2\expandafter ##3\csname
897 .=\expandafter\xintListSel::csv \romannumeral`&&\XINT_expr_unlock ##4;%
898 \XINT_expr_unlock ##1;\endcsname % unlock added for \xintNewExpr
899 }%
900 \fi
901 }%
902 \let #5\xint_c_ii
903 }%
904 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
905 \expandafter\xINT_tmpa

```

```

906 \csname XINT_#1_op_][\expandafter\endcsname
907 \csname XINT_#1_until_][_a\expandafter\endcsname
908 \csname XINT_#1_until_][_b\expandafter\endcsname
909 \csname XINT_#1_op_-vi\expandafter\endcsname
910 \csname XINT_expr_precedence_][\endcsname {#1}%
911 }%
912 \def\XINT_tmpa #1#2#3#4#5#6%
913 {%
914 \def #1##1% \XINT_expr_op_:
915 {%
916 \expandafter #2\expandafter ##1\romannumeral`&&@\XINT_expr_getnext
917 }%
918 \def #2##1##2% \XINT_expr_until_:_a
919 {\xint_UDsignfork
920 ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
921 -{#3##1##2}%
922 \krof }%
923 \def #3##1##2##3##4% \XINT_expr_until_:_b
924 {%
925 \ifnum ##2>\xint_c_iii
926 \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
927 \csname XINT_#6_op_##3\endcsname {##4}}%
928 \else
929 \xint_afterfi
930 {\expandafter ##2\expandafter ##3\csname
931 .:=\xintiiifSgn{\XINT_expr_unlock ##1}NPP.%
932 \xintiiifSgn{\XINT_expr_unlock ##4}NPP.%
933 \xintNum{\XINT_expr_unlock ##1};\xintNum{\XINT_expr_unlock ##4}\endcsname
934 }%
935 \fi
936 }%
937 \let #5\xint_c_iii
938 }%
939 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
940 \expandafter\XINT_tmpa
941 \csname XINT_#1_op_:\expandafter\endcsname
942 \csname XINT_#1_until_:_a\expandafter\endcsname
943 \csname XINT_#1_until_:_b\expandafter\endcsname
944 \csname XINT_#1_op_-vi\expandafter\endcsname
945 \csname XINT_expr_precedence_:\endcsname {#1}%
946 }%
947 \catcode`[ 11 \catcode`] 11
948 \let\XINT_expr_precedence_:\xint_c_iii
949 \def\XINT_expr_op_:] #1{\expandafter\xint_c_i\expandafter }%
950 \csname .:=\xintiiifSgn{\XINT_expr_unlock #1}npp\XINT_expr_unlock #1\endcsname }%
951 \let\XINT_flexpr_op_:] \XINT_expr_op_:]
952 \let\XINT_iiexpr_op_:] \XINT_expr_op_:]
953 \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).

```
954 \edef\XINT_expr_op_[: #1{\xint_c_ii \expandafter\noexpand
955 \csname XINT_expr_itself_][\endcsname #10\string :}%
```

10.22.9 \xintListSel::csv

Some complications here are due to \xintNewExpr matters.

```
956 \let\XINT_flexpr_op_[: \XINT_expr_op_[:
957 \let\XINT_iexpr_op_[: \XINT_expr_op_[:
958 \catcode`[ 12 \catcode`] 12
959 \def\xintListSel::csv #1{%
960   \if ]\noexpand#1\xint_dothis{\expandafter\XINT_listsel:_s\romannumeral`&&@\fi
961   \if :\noexpand#1\xint_dothis{\XINT_listsel:_:}\fi
962   \xint_orthat {\XINT_listsel:_nth #1}%
963 }%
964 \def\XINT_listsel:_s #1{\if p#1\expandafter\XINT_listsel:_trim\else
965   \expandafter\XINT_listsel:_keep\fi }%
966 \def\XINT_listsel:_: #1.#2.{\csname XINT_listsel:_#1#2\endcsname }%
967 \def\XINT_listsel:_trim #1;#2;%
968   {\xintListWithSep,{\xintTrim {\xintNum{#1}}{\xintCSVtoListNonStripped{#2}}}}%
969 \def\XINT_listsel:_keep #1;#2;%
970   {\xintListWithSep,{\xintKeep {\xintNum{#1}}{\xintCSVtoListNonStripped{#2}}}}%
971 \def\XINT_listsel:_nth#1;#2;%
972   {\xintNthElt {\xintNum{#1}}{\xintCSVtoListNonStripped{#2}}}%
973 \def\XINT_listsel:_PP #1;#2;#3;%
974   {\xintListWithSep,%
975     {\xintTrim {\xintNum{#1}}{\xintKeep {\xintNum{#2}}{\xintCSVtoListNonStripped{#3}}}}}%
976   }%
977 \def\XINT_listsel:_NN #1;#2;#3;%
978   {\xintListWithSep,%
979     {\xintTrim {\xintNum{#2}}{\xintKeep {\xintNum{#1}}{\xintCSVtoListNonStripped{#3}}}}}%
980   }%
981 \def\XINT_listsel:_NP #1;#2;#3;%
982   {\expandafter\XINT_listsel:_NP_a \the\numexpr #1+%
983     \xintNthElt{0}{\xintCSVtoListNonStripped{#3}};#2;#3;%
984 \def\XINT_listsel:_NP_a #1#2;{\if -#1\expandafter\XINT_listsel:_OP\fi
985   \XINT_listsel:_PP #1#2;%
986 \def\XINT_listsel:_OP\XINT_listsel:_PP #1;{\XINT_listsel:_PP 0;%
987 \def\XINT_listsel:_PN #1;#2;#3;%
988   {\expandafter\XINT_listsel:_PN_a \the\numexpr #2+%
989     \xintNthElt{0}{\xintCSVtoListNonStripped{#3}};#1;#3;%
990 \def\XINT_listsel:_PN_a #1#2;#3;{\if -#1\expandafter\XINT_listsel:_PO\fi
991   \XINT_listsel:_PP #3;#1#2;%
992 \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	\xintSeq::csv	272
10.23.2	\xintiiSeq::csv	272

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 $\text{ceil}(a)$ et b par $\text{floor}(b)$ et renvoie ensuite les entiers entre les deux, possiblement en décroissant, et extrémités comprises. Si $a=b$ est non entier en obtient donc $\text{ceil}(a)$ et $\text{floor}(a)$. Ne renvoie jamais une liste vide.

Note: le $a..b$ dans `\xintfloatexpr` utilise cette routine.

```

993 \def\xintSeq::csv {\romannumeral0\xintseq::csv}%
994 \def\xintseq::csv #1#2%
995 {%
996   \expandafter\XINT_seq::csv\expandafter
997     {\the\numexpr \xintiCeil{#1}\expandafter}\expandafter
998     {\the\numexpr \xintiFloor{#2}}%
999}%
1000 \def\XINT_seq::csv #1#2%
1001 {%
1002   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
1003   \expandafter\XINT_seq::csv_z
1004   \or
1005   \expandafter\XINT_seq::csv_p
1006   \else
1007   \expandafter\XINT_seq::csv_n
1008   \fi
1009   {#2}{#1}%
1010}%
1011 \def\XINT_seq::csv_z #1#2{ #1/1[0]}%
1012 \def\XINT_seq::csv_p #1#2%
1013 {%
1014   \ifnum #1>#2
1015     \expandafter\expandafter\expandafter\XINT_seq::csv_p
1016   \else
1017     \expandafter\XINT_seq::csv_e
1018   \fi
1019   \expandafter{\the\numexpr #1-\xint_c_i}{#2},#1/1[0]%
1020}%
1021 \def\XINT_seq::csv_n #1#2%
1022 {%
1023   \ifnum #1<#2
1024     \expandafter\expandafter\expandafter\XINT_seq::csv_n
1025   \else
1026     \expandafter\XINT_seq::csv_e
1027   \fi
1028   \expandafter{\the\numexpr #1+\xint_c_i}{#2},#1/1[0]%
1029}%
1030 \def\XINT_seq::csv_e #1,{ }%

```

10.23.2 `\xintiiSeq::csv`

```

1031 \def\xintiiSeq::csv {\romannumeral0\xintiiseq::csv}%
1032 \def\xintiiseq::csv #1#2%
1033 {%
1034   \expandafter\XINT_iiSeq::csv\expandafter
1035     {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%

```



```

1036 }%
1037 \def\XINT_iiseq::csv #1#2%
1038 {%
1039   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
1040   \expandafter\XINT_iiseq::csv_z
1041   \or
1042   \expandafter\XINT_iiseq::csv_p
1043   \else
1044   \expandafter\XINT_iiseq::csv_n
1045   \fi
1046   {#2}{#1}%
1047 }%
1048 \def\XINT_iiseq::csv_z #1#2{ #1}%
1049 \def\XINT_iiseq::csv_p #1#2%
1050 {%
1051   \ifnum #1>#2
1052     \expandafter\expandafter\expandafter\XINT_iiseq::csv_p
1053   \else
1054     \expandafter\XINT_seq::csv_e
1055   \fi
1056   \expandafter{\the\numexpr #1-\xint_c_i}{#2},#1%
1057 }%
1058 \def\XINT_iiseq::csv_n #1#2%
1059 {%
1060   \ifnum #1<#2
1061     \expandafter\expandafter\expandafter\XINT_iiseq::csv_n
1062   \else
1063     \expandafter\XINT_seq::csv_e
1064   \fi
1065   \expandafter{\the\numexpr #1+\xint_c_i}{#2},#1%
1066 }%
1067 \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>	273
10.24.2	<code>\xintSeqB::csv</code>	274
10.24.3	<code>\xintiiSeqB::csv</code>	274
10.24.4	<code>\XINTinFloatSeqB::csv</code>	275

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`

2015/11/11 Naturally, I did not document anything in 2014, and today I was perplexed about what these macros do; and why something was wrong with `\xintNewIIExpr` and a..*[b]*..*c* things therein. In fact `\xintiiSeqB:f:csv` had a typo in its name, but this had escaped my 2014 tests; and if I had corrected it I would have seen another problem with a..*[b]*..*C* in `\xintNewIIExpr`, the `\xinti-iSeqB:f:csv` macro calls `\xintiiSeqA::csv` with arguments which have no more a `\XINT_expr_unlock`. But `\xintiiSeqA::csv` tried to be clever and assumed the `\XINT_expr_unlock` were there. The other two expanded either in `\xintraw` or `\XINTinfloat`, hence no problem arose in `\xintNew-Expr/\xintNewFloatExpr`. The fix has been to let `\xintiiSeqA::csv` act a bit more like the other two.

```

1068 \def\xintSeqA::csv #1%
1069   {\expandafter\XINT_seqa::csv\expandafter{\romannumeral0\xintra {#1}}}%
1070 \def\XINT_seqa::csv #1#2{\expandafter\XINT_seqa::csv_a \romannumeral0\xintra {#2};#1}%
1071 \def\xintiiSeqA::csv #1{\expandafter\XINT_iiseqa::csv\expandafter{\romannumeral`&&@#1}}%
1072 \def\XINT_iiseqa::csv #1#2{\expandafter\XINT_seqa::csv_a\romannumeral`&&@#2;#1}%
1073 \def\XINTinFloatSeqA::csv #1{\expandafter\XINT_flseqa::csv\expandafter
1074   {\romannumeral0\XINTinfloat [\XINTdigits]{#1}}}%
1075 \def\XINT_flseqa::csv #1#2%
1076   {\expandafter\XINT_seqa::csv_a\romannumeral0\XINTinfloat [\XINTdigits]{#2};#1}%
1077 \def\XINT_seqa::csv_a #1{\xint_UDzerominusfork
1078   #1-{z}%
1079   0#1{n}%
1080   0-{p}%
1081   \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.

```

1082 \def\xintSeqB::csv #1#2%
1083   {\expandafter\XINT_seqb::csv \expandafter{\romannumeral0\xintra{#2}}{#1}}%
1084 \def\XINT_seqb::csv #1#2{\expandafter\XINT_seqb::csv_a\romannumeral`&&@#2#1!}%
1085 \def\XINT_seqb::csv_a #1#2;#3;#4!{\expandafter\XINT_expr_seq_empty?
1086   \romannumeral0\csname XINT_seqb::csv_#1\endcsname {#3}{#4}{#2}}%
1087 \def\XINT_seqb::csv_p #1#2#3%
1088 {%
1089   \xintifCmp {#1}{#2}{, #1\expandafter\XINT_seqb::csv_p\expandafter}%
1090   {, #1\xint_gobble_iii}{\xint_gobble_iii}%

```

\romannumeral0 stopped by \endcsname, XINT_expr_seq_empty? constructs "nil".

```

1091   {\romannumeral0\xintadd {#3}{#1}}{#2}{#3}%
1092 }%
1093 \def\XINT_seqb::csv_n #1#2#3%
1094 {%
1095   \xintifCmp {#1}{#2}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1096   {, #1\expandafter\XINT_seqb::csv_n\expandafter}%
1097   {\romannumeral0\xintadd {#3}{#1}}{#2}{#3}%
1098 }%
1099 \def\XINT_seqb::csv_z #1#2#3{, #1}%

```

10.24.3 \xintiiSeqB::csv

```

1100 \def\xintiiSeqB::csv #1#2{\XINT_iiseqb::csv #1#2}%
1101 \def\XINT_iiseqb::csv #1#2#3#4%
1102   {\expandafter\XINT_iiseqb::csv_a
1103     \romannumeral`&&\expandafter \XINT_expr_unlock\expandafter#2%
1104     \romannumeral`&&\XINT_expr_unlock #4!}%
1105 \def\XINT_iiseqb::csv_a #1#2;#3;#4!{\expandafter\XINT_expr_seq_empty?
1106   \romannumeral`&&\csname XINT_iiseqb::csv_#1\endcsname {#3}{#4}{#2}}%
1107 \def\XINT_iiseqb::csv_p #1#2#3%
1108 {%
1109   \xintSgnFork{\XINT_Cmp {#1}{#2}}{, #1\expandafter\XINT_iiseqb::csv_p\expandafter}%

```

```

1110 {,#1\xint_gobble_iii}{\xint_gobble_iii}%
1111 {\romannumeral0\xintiiadd {#3}{#1}}{#2}{#3}%
1112 }%
1113 \def\xINT_iiseqb::csv_n #1#2#3%
1114 {%
1115 \xintSgnFork{\XINT_Cmp {#1}{#2}}{\xint_gobble_iii}{,#1\xint_gobble_iii}%
1116 {,#1\expandafter\xINT_iiseqb::csv_n\expandafter}%
1117 {\romannumeral0\xintiiadd {#3}{#1}}{#2}{#3}%
1118 }%
1119 \def\xINT_iiseqb::csv_z #1#2#3{,#1}%

```

10.24.4 \XINTinFloatSeqB::csv

```

1120 \def\xINTinFloatSeqB::csv #1#2{\expandafter\xINT_flseqb::csv \expandafter
1121 {\romannumeral0\xINTinfloat [\XINTdigits]{#2}}{#1}}%
1122 \def\xINT_flseqb::csv #1#2{\expandafter\xINT_flseqb::csv_a\romannumeral`&&#2#1!}%
1123 \def\xINT_flseqb::csv_a #1#2;#3;#4!\expandafter\xINT_expr_seq_empty?
1124 \romannumeral`&&\csname XINT_flseqb::csv_#1\endcsname {#3}{#4}{#2}}%
1125 \def\xINT_flseqb::csv_p #1#2#3%
1126 {%
1127 \xintifCmp {#1}{#2}{,#1\expandafter\xINT_flseqb::csv_p\expandafter}%
1128 {,#1\xint_gobble_iii}{\xint_gobble_iii}%
1129 {\romannumeral0\xINTinfloatadd {#3}{#1}}{#2}{#3}%
1130 }%
1131 \def\xINT_flseqb::csv_n #1#2#3%
1132 {%
1133 \xintifCmp {#1}{#2}{\xint_gobble_iii}{,#1\xint_gobble_iii}%
1134 {,#1\expandafter\xINT_flseqb::csv_n\expandafter}%
1135 {\romannumeral0\xINTinfloatadd {#3}{#1}}{#2}{#3}%
1136 }%
1137 \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.

```

1138 \def\xINT_tmpa #1#2#3#4#5#6%
1139 {%
1140 \def #1##1% \XINT_expr_op_,
1141 {%
1142 \expandafter #2\expandafter ##1\romannumeral`&&\XINT_expr_getnext
1143 }%
1144 \def #2##1##2% \XINT_expr_until_,_a
1145 {\xint_UDsignfork
1146 ##2{\expandafter #2\expandafter ##1\romannumeral`&&#4}%
1147 -{#3##1##2}%
1148 \krof }%
1149 \def #3##1##2##3##4% \XINT_expr_until_,_b
1150 {%
1151 \ifnum ##2>\xint_c_ii
1152 \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
1153 \csname XINT_#6_op_##3\endcsname {##4}}%
1154 \else
1155 \xint_afterfi
1156 {\expandafter ##2\expandafter ##3%

```

```

1157      \csname .=\XINT_expr_unlock ##1,\XINT_expr_unlock ##4\endcsname }%
1158      \fi
1159    }%
1160    \let #5\xint_c_ii
1161  }%
1162  \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1163  \expandafter\XINT_tmpa
1164    \csname XINT_#1_op_,\expandafter\endcsname
1165    \csname XINT_#1_until_,_a\expandafter\endcsname
1166    \csname XINT_#1_until_,_b\expandafter\endcsname
1167    \csname XINT_#1_op_-vi\expandafter\endcsname
1168    \csname XINT_expr_precedence_,\endcsname {#1}%
1169  }%

```

10.26 The minus as prefix operator of variable precedence level

Inherits the precedence level of the previous infix operator.

```

1170 \def\XINT_tmpa #1#2#3%
1171 {%
1172   \expandafter\XINT_tmppb
1173   \csname XINT_#1_op_-#3\expandafter\endcsname
1174   \csname XINT_#1_until_-#3_a\expandafter\endcsname
1175   \csname XINT_#1_until_-#3_b\expandafter\endcsname
1176   \csname xint_c_#3\endcsname {#1}#2%
1177 }%
1178 \def\XINT_tmppb #1#2#3#4#5#6%
1179 {%
1180   \def #1% \XINT_expr_op_-<level>
1181   {% get next number+operator then switch to _until macro
1182     \expandafter #2\romannumeral`&&\XINT_expr_getnext
1183   }%
1184   \def #2##1% \XINT_expr_until_-<l>_a
1185   {\xint_UDsignfork
1186     ##1{\expandafter #2\romannumeral`&&@#1}%
1187     -{#3##1}%
1188   \krof }%
1189   \def #3##1##2##3% \XINT_expr_until_-<l>_b
1190   {% _until tests precedence level with next op, executes now or postpones
1191     \ifnum ##1>#4%
1192       \xint_afterfi {\expandafter #2\romannumeral`&&@%
1193         \csname XINT_#5_op_##2\endcsname {##3}}%
1194     \else
1195       \xint_afterfi {\expandafter ##1\expandafter ##2%
1196         \csname .=#6{\XINT_expr_unlock ##3}\endcsname }%
1197     \fi
1198   }%
1199 }%

```

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.

```

1200 \xintApplyInline{\XINT_tmpa {expr}\xintOpp}{\vi}{vii}{viii}{ix}}%

```

```
1201 \xintApplyInline{\XINT_tmpa {fexpr}\xintOpp}{vi}{vii}{viii}{ix}}%
1202 \xintApplyInline{\XINT_tmpa {iexpr}\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.

```
1203 \let\XINT_expr_precedence_? \xint_c_x
1204 \def\XINT_expr_op_? #1#2{\if ?#2\expandafter \XINT_expr_op_??\fi
1205     \XINT_expr_op_?a #1{#2}}%
1206 \def\XINT_expr_op_?a #1#2#3%
1207 {%
1208     \xintiiifNotZero{\XINT_expr_unlock #1}{\XINT_expr_getnext #2}{\XINT_expr_getnext #3}%
1209 }%
1210 \let\XINT_flexpr_op_?\XINT_expr_op_?
1211 \let\XINT_iexpr_op_?\XINT_expr_op_?
1212 \def\XINT_expr_op_?? #1#2#3#4#5#6%
1213 {%
1214     \xintiiifSgn {\XINT_expr_unlock #2}{\XINT_expr_getnext #4}{\XINT_expr_getnext #5}%
1215     {\XINT_expr_getnext #6}%
1216 }%
```

10.28 ! as postfix factorial operator

A float version \xintFloatFac was at last done 2015/10/06.

```
1217 \let\XINT_expr_precedence_! \xint_c_x
1218 \def\XINT_expr_op_! #1{\expandafter\XINT_expr_getop
1219     \csname .=\xintFac{\XINT_expr_unlock #1}\endcsname }%
1220 \def\XINT_flexpr_op_! #1{\expandafter\XINT_expr_getop
1221     \csname .=\XINTinFloatFac{\XINT_expr_unlock #1}\endcsname }%
1222 \def\XINT_iexpr_op_! #1{\expandafter\XINT_expr_getop
1223     \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.

```
1224 \catcode`[ 11
1225 \catcode`* 11
1226 \let\XINT_expr_precedence_[ \xint_c_vii
1227 \def\XINT_expr_op_[ #1#2]{\expandafter\XINT_expr_getop
1228     \csname .=\xintE{\XINT_expr_unlock #1}%
```

```

1229          {\xint_zapspaces #2 \xint_gobble_i}\endcsname}%
1230 \def\XINT_iiexpr_op_[ #1#2]{\expandafter\XINT_expr_getop
1231          \csname .=\xintiiE{\XINT_expr_unlock #1}%
1232          {\xint_zapspaces #2 \xint_gobble_i}\endcsname}%
1233 \def\XINT_flexpr_op_[ #1#2]{\expandafter\XINT_expr_getop
1234          \csname .=\XINTinFloatE{\XINT_expr_unlock #1}%
1235          {\xint_zapspaces #2 \xint_gobble_i}\endcsname}%
1236 \catcode`[ 12
1237 \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.

```

1238 \def\XINT_tmpa #1#2#3{%
1239   \def #1##1%
1240   {%
1241     \ifcsname XINT_#3_func_##1\endcsname
1242     \xint_dothis{\expandafter\expandafter
1243       \csname XINT_#3_func_##1\endcsname\romannumeral`&&@#2}\fi
1244     \ifcsname XINT_expr_onlitteral_##1\endcsname
1245     \xint_dothis{\csname XINT_expr_onlitteral_##1\endcsname}\fi
1246     \xint_orthat{\XINT_expr_unknown_function {##1}%
1247       \expandafter\XINT_expr_func_unknown\romannumeral`&&@#2}%
1248   }%
1249 }%
1250 \def\XINT_expr_unknown_function #1{\xintError:removed \xint_gobble_i {#1}}%
1251 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1252   \expandafter\XINT_tmpa
1253   \csname XINT_#1_op_`\expandafter\endcsname
1254   \csname XINT_#1_oparen\endcsname
1255   {#1}%
1256 }%
1257 \def\XINT_expr_func_unknown #1#2#3%
1258   {\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".

```

1259 \def\XINT_expr_onlitteral_bool #1)%
1260   {\expandafter\XINT_expr_getop\csname .=\xintBool{#1}\endcsname }%
1261 \def\XINT_expr_onlitteral_togl #1)%

```

```

1262      {\expandafter\XINT_expr_getop\csname .=\xintToggle{#1}\endcsname }%
1263 \def\XINT_expr_onlitteral_protect #1)%
1264      {\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.

```

1265 \def\XINT_expr_func_break #1#2#3%
1266     {\expandafter #1\expandafter #2\csname .=?\romannumeral`&&\XINT_expr_unlock #3\endcsname }%
1267 \let\XINT_flexpr_func_break \XINT_expr_func_break
1268 \let\XINT_iexpr_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.

```

1269 \def\XINT_expr_onlitteral_qint #1)%
1270     {\expandafter\XINT_expr_getop\csname .=\xintiNum{#1}\endcsname }%
1271 \def\XINT_expr_onlitteral_qfrac #1)%
1272     {\expandafter\XINT_expr_getop\csname .=\xintRaw{#1}\endcsname }%
1273 \def\XINT_expr_onlitteral_qfloat #1)%
1274     {\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`.

```

1275 \def\XINT_expr_op__ #1% op__ with two _'s
1276     {%
1277         \ifcsname XINT_expr_var_#1\endcsname
1278             \expandafter\xint_firstoftwo
1279         \else
1280             \expandafter\xint_secondoftwo
1281         \fi
1282         {\expandafter\expandafter\expandafter
1283          \XINT_expr_getop\csname XINT_expr_var_#1\endcsname}%
1284         {\XINT_expr_unknown_variable {#1}%
1285          \expandafter\XINT_expr_getop\csname .:=0\endcsname}%
1286     }%
1287 \def\XINT_expr_unknown_variable #1{\xintError:removed \xint_gobble_i {#1}}%
1288 \let\XINT_flexpr_op__ \XINT_expr_op__
1289 \let\XINT_iexpr_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 an underscore are reserved.

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

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

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

2015/11/13: I now first do an a priori complete expansion of #1, and then apply \detokenize to the result, and remove spaces. 2015/11/21: finally I do not detokenize the variable name. Because this complicated the \xintunassignvar if it did the same and we wanted to use it to redeclare a letter as dummy variable.

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

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

```

1290 \catcode`* 11
1291 \def\XINT_expr_defvar #1#2#3;%
1292   \edef\XINT_expr_tmpa{#2}%
1293   \edef\XINT_expr_tmpa {\xint_zapspaces_o\XINT_expr_tmpa}%
1294   \ifnum\expandafter\xintLength\expandafter{\XINT_expr_tmpa}=\z@
1295     \xintMessage {xintexpr}{Warning}
1296     {Error: impossible to declare variable with empty name.}%
1297   \else
1298     \edef\XINT_expr_tmpb {\romannumeral0#1#3\relax }%
1299     \expandafter\edef\csname XINT_expr_var_\XINT_expr_tmpa\endcsname
1300       {\expandafter\noexpand\XINT_expr_tmpb}%
1301     \expandafter\edef\csname XINT_expr_onlitteral_\XINT_expr_tmpa\endcsname
1302       {\XINT_expr_precedence_*** *\expandafter\noexpand\XINT_expr_tmpb }%
1303     \ifxintverbose\xintMessage {xintexpr}{Info}
1304       {Variable "\XINT_expr_tmpa" defined with value
1305         \expandafter\XINT_expr_unlock\XINT_expr_tmpb.}%
1306     \fi
1307   \fi
1308 }%
1309 \catcode`* 12
1310 \catcode`: 12
1311 \def\xintdefvar      #1:={\XINT_expr_defvar\xintbareeval      {#1}}%
1312 \def\xintdefiivar   #1:={\XINT_expr_defvar\xintbareiieval   {#1}}%
1313 \def\xintdeffloatvar #1:={\XINT_expr_defvar\xintbarefloateval {#1}}%
1314 \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_`.

1315 \def\xintunassignvar #1{%
1316   \edef\XINT_expr_tmpa{#1}%
1317   \edef\XINT_expr_tmpa {\xint_zapspaces_o\XINT_expr_tmpa}%
1318   \ifcsname XINT_expr_var_\XINT_expr_tmpa\endcsname
1319     \ifnum\expandafter\xintLength\expandafter{\XINT_expr_tmpa}=\@ne
1320       \expandafter\XINT_expr_makedummy \XINT_expr_tmpa
1321       \ifxintverbose\xintMessage {xintexpr}{Info}
1322         {Character \XINT_expr_tmpa\space usable as dummy variable (if letter).}%
1323       \fi
1324     \else
1325       \expandafter\edef\csname XINT_expr_var_\XINT_expr_tmpa\endcsname
1326         {\csname .=\0\endcsname\noexpand\XINT_expr_undefined {\XINT_expr_tmpa}}%
1327       \expandafter\edef\csname XINT_expr_onlitteral_\XINT_expr_tmpa\endcsname
1328         {\csname .=\0\endcsname\noexpand\XINT_expr_undefined {\XINT_expr_tmpa}*}%
1329       \ifxintverbose\xintMessage {xintexpr}{Info}
1330         {Variable \XINT_expr_tmpa\space has been ``unassigned''.}%
1331       \fi
1332     \fi
1333   \else
1334     \xintMessage {xintexpr}{Warning}
1335     {Error: there was no such variable \XINT_expr_tmpa\space to unassign.}%
1336   \fi
1337 }%
1338 \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	282
10.37.2	omit and abort	282
10.37.3	The special variables @, @1, @2, @3, @4, @@, @@(1), ..., @@@, @@@(1), ... for recursion	283
10.37.4	\XINT_expr_onlitteral_seq	284
10.37.5	\XINT_expr_onlitteral_seq_a	284
10.37.6	\XINT_isbalanced_a for \XINT_expr_onlitteral_seq_a	285
10.37.7	\XINT_allexpr_func_seqx	285
10.37.8	Evaluation over list, \XINT_expr_seq:_a with break, abort, omit	286
10.37.9	Evaluation over ++ generated lists with \XINT_expr_seq:_A	286

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.

```
1339 \catcode`* 11
1340 \def\xint_expr_makedummy #1%
1341 {%
1342   \expandafter\def\csname XINT_expr_var_#1\endcsname ##1\relax !#1##2%
1343   {##2##1\relax !#1##2}%
1344   \expandafter\def\csname XINT_expr_onlitteral_#1\endcsname ##1\relax !#1##2%
1345   {\XINT_expr_precedence_*** *##2(##1\relax !#1##2}%
1346 }%
1347 \xintApplyUnbraced \XINT_expr_makedummy {abcdefghijklmnopqrstuvwxyz}%
1348 \xintApplyUnbraced \XINT_expr_makedummy {ABCDEFGHIJKLMNOPQRSTUVWXYZ}%
1349 \edef\xint_expr_var_nil {\expandafter\noexpand\csname . = \endcsname}%
1350 \edef\xint_expr_onlitteral_nil
1351   {\XINT_expr_precedence_*** *\expandafter\noexpand\csname . = \endcsname {}%
1352 \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 \wedge . Celui-ci étant de précédence zéro provoque la finalisation de tous les calculs antérieurs dans le sous-bareeval. Mais j'ai dû hacker le `until_end_b` (et le `until_*)_b`) qui confronté à \wedge , 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 \wedge a été fait alors que je n'avais pas encore les variables muettes. Je dois trouver autre chose, car `seq(2 \wedge C, C=1..5)` est alors impossible. De toute façon ce \wedge était à usage interne uniquement.

Il me faut un symbole d'opérateur qui ne rentre pas en conflit. Bon je vais prendre `!?`. Ensuite au lieu de hacker `until_end`, il vaut mieux lui donner précédence 2 (mais ça ne pourra pas marcher à l'intérieur de parenthèses il faut d'abord les fermer manuellement) et lui associer un simplement un op spécial. Je n'avais pas fait cela peut-être pour éviter d'avoir à définir plusieurs macros. Le `#1` dans la définition de `\XINT_expr_op_!?` est le résultat de l'évaluation forcée précédente.

Attention que les premier `!` doivent être de catcode 12 sinon ils signalent une sous-expression qui déclenche une multiplication tacite.

2015/11/13

```
1353 \edef\XINT_expr_var_omit #1\relax !{1\string !?! \relax !}%
1354 \edef\XINT_expr_var_abort #1\relax !{1\string !?^ \relax !}%
1355 \def\XINT_expr_op_!#2\relax {\expandafter\XINT_expr_foundend\csname .=#2\endcsname}%
1356 \let\XINT_iiexpr_op_!?\XINT_expr_op_!
1357 \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.

```
1358 \catcode`? 3 \catcode`* 11
1359 \def\XINT_expr_var_@ #1~#2{#2#1~#2}%
1360 \expandafter\let\csname XINT_expr_var_@1\endcsname \XINT_expr_var_@
1361 \expandafter\def\csname XINT_expr_var_@2\endcsname #1~#2#3{#3#1~#2#3}%
1362 \expandafter\def\csname XINT_expr_var_@3\endcsname #1~#2#3#4{#4#1~#2#3#4}%
1363 \expandafter\def\csname XINT_expr_var_@4\endcsname #1~#2#3#4#5{#5#1~#2#3#4#5}%
1364 \def\XINT_expr_onlitteral_@ #1~#2{\XINT_expr_precedence_*** *#2(#1~#2)%
1365 \expandafter\let\csname XINT_expr_onlitteral_@1\endcsname \XINT_expr_onlitteral_@
1366 \expandafter\def\csname XINT_expr_onlitteral_@2\endcsname #1~#2#3%
1367      {\XINT_expr_precedence_*** *#3(#1~#2#3)%
1368 \expandafter\def\csname XINT_expr_onlitteral_@3\endcsname #1~#2#3#4%
1369      {\XINT_expr_precedence_*** *#4(#1~#2#3#4)%
1370 \expandafter\def\csname XINT_expr_onlitteral_@4\endcsname #1~#2#3#4#5%
1371      {\XINT_expr_precedence_*** *#5(#1~#2#3#4#5)%
1372 \catcode`* 12
1373 \def\XINT_expr_func_@@ #1#2#3#4~#5?%
1374 {%
1375   \expandafter#1\expandafter#2\romannumeral0\XINTntheltnoexpand
1376      {\XINTNum{\XINT_expr_unlock#3}}{#5}#4~#5?%
```

```

1377 }%
1378 \def\XINT_expr_func_@@ #1#2#3#4~#5~#6?%
1379 {%
1380   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1381   {\xintNum{\XINT_expr_unlock#3}}{#6}#4~#5~#6?%
1382 }%
1383 \def\XINT_expr_func_@@@ #1#2#3#4~#5~#6~#7?%
1384 {%
1385   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1386   {\xintNum{\XINT_expr_unlock#3}}{#7}#4~#5~#6~#7?%
1387 }%
1388 \let\XINT_flexpr_func_@@\XINT_expr_func_@@
1389 \let\XINT_flexpr_func_@@@\XINT_expr_func_@@@
1390 \let\XINT_flexpr_func_@@@@\XINT_expr_func_@@@
1391 \def\XINT_iexpr_func_@@ #1#2#3#4~#5?%
1392 {%
1393   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1394   {\XINT_expr_unlock#3}{#5}#4~#5?%
1395 }%
1396 \def\XINT_iexpr_func_@@@ #1#2#3#4~#5~#6?%
1397 {%
1398   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1399   {\XINT_expr_unlock#3}{#6}#4~#5~#6?%
1400 }%
1401 \def\XINT_iexpr_func_@@@@ #1#2#3#4~#5~#6~#7?%
1402 {%
1403   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1404   {\XINT_expr_unlock#3}{#7}#4~#5~#6~#7?%
1405 }%
1406 \catcode`? 11

```

10.37.4 \XINT_expr_onlitteral_seq

```

1407 \def\XINT_expr_onlitteral_seq
1408 {\expandafter\XINT_expr_onlitteral_seq_f\romannumeral`&&\XINT_expr_onlitteral_seq_a {}}%
1409 \def\XINT_expr_onlitteral_seq_f #1#2{\xint_c_xviii `{seqx}#2)\relax #1}%

```

10.37.5 \XINT_expr_onlitteral_seq_a

```

1410 \def\XINT_expr_onlitteral_seq_a #1#2,%
1411 {% checks balancing of parentheses
1412   \ifcase\XINT_isbalanced_a \relax #1#2(\xint_bye)\xint_bye
1413   \expandafter\XINT_expr_onlitteral_seq_c
1414   \or\expandafter\XINT_expr_onlitteral_seq_b
1415   \else\expandafter\xintError:we_are_doomed
1416   \fi {#1#2},%
1417 }%
1418 \def\XINT_expr_onlitteral_seq_b #1,{\XINT_expr_onlitteral_seq_a {#1},}%
1419 \def\XINT_expr_onlitteral_seq_c #1,#2#3% #3 pour absorber le =
1420 {%
1421   \XINT_expr_onlitteral_seq_d {#2{#1}}}%
1422 }%
1423 \def\XINT_expr_onlitteral_seq_d #1#2#3)%
1424 {%

```

```

1425 \ifcase\XINT_isbalanced_a \relax #2#3(\xint_bye)\xint_bye
1426 \or\expandafter\XINT_expr_onlitteral_seq_e
1427 \else\expandafter\xintError:we_are_doomed
1428 \fi
1429 {#1}{#2#3}%
1430 }%
1431 \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.

```

1432 % use as \XINT_isbalanced_a \relax #1(\xint_bye)\xint_bye
1433 \def\XINT_isbalanced_a #1({\XINT_isbalanced_b #1)\xint_bye }%
1434 \def\XINT_isbalanced_b #1)#2%
1435 {\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

1436 \def\XINT_isbalanced_error #1)\xint_bye {\xint_c_mone}%

  #2 was \xint_bye, was there a ) in original #1?

1437 \def\XINT_isbalanced_c\xint_bye\XINT_isbalanced_error #1%
1438 {\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.

1439 \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.

1440 \def\XINT_isbalanced_d #1)#2%
1441 {\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.

1442 \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>}.

```

1443 \def\XINT_expr_func_seqx #1#2{\XINT_allexpr_seqx \xintthebareval }%
1444 \def\XINT_flexpr_func_seqx #1#2{\XINT_allexpr_seqx \xintthebarefloateval}%
1445 \def\XINT_iexpr_func_seqx #1#2{\XINT_allexpr_seqx \xintthebareiieval }%
1446 \def\XINT_allexpr_seqx #1#2#3#4%
1447 {%
1448 \expandafter \XINT_expr_getop

```

```

1449 \csname .=\expandafter\XINT_expr_seq:_aa
1450 \romannumeral`&&\XINT_expr_unlock #2!{#1#4\relax !#3}\endcsname
1451 }%
1452 \def\XINT_expr_seq:_aa #1{\if +#1\expandafter\XINT_expr_seq:_A\else
1453 \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.)

```

1454 \def\XINT_expr_seq:_a #1!#2{\expandafter\XINT_expr_seq_empty?
1455 \romannumeral0\XINT_expr_seq:_b {#2}#1,^,}%
1456 \def\XINT_expr_seq:_b #1#2#3,{%
1457 \if ,#2\xint_dothis\XINT_expr_seq:_noop\fi
1458 \if ^#2\xint_dothis\XINT_expr_seq:_end\fi
1459 \xint_orthat{\expandafter\XINT_expr_seq:_c}\csname.=#2#3\endcsname {#1}%
1460 }%
1461 \def\XINT_expr_seq:_noop\csname.=,#1\endcsname #2{\XINT_expr_seq:_b {#2}#1,}%
1462 \def\XINT_expr_seq:_end \csname.=^ \endcsname #1}%
1463 \def\XINT_expr_seq:_c #1#2{\expandafter\XINT_expr_seq:_d\romannumeral`&&@#2#1{#2}}%
1464 \def\XINT_expr_seq:_d #1{\if #1^ \xint_dothis\XINT_expr_seq:_abort\fi
1465 \if #1? \xint_dothis\XINT_expr_seq:_break\fi
1466 \if #1! \xint_dothis\XINT_expr_seq:_omit\fi
1467 \xint_orthat{\XINT_expr_seq:_goon #1}}%
1468 \def\XINT_expr_seq:_abort #1!#2#3#4#5^,{}%
1469 \def\XINT_expr_seq:_break #1!#2#3#4#5^,{,#1}%
1470 \def\XINT_expr_seq:_omit #1!#2#3#4{\XINT_expr_seq:_b {#4}}%
1471 \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`.

```

1472 \def\XINT_expr_seq_empty? #1{%
1473 \def\XINT_expr_seq_empty? ##1{\if ,##1\expandafter\xint_gobble_i\fi #1\endcsname }}%
1474 \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.

```

1475 \def\XINT_expr_seq:_A +#1!%

```

```

1476 {\expandafter\XINT_expr_seq_empty?\romannumeral0\XINT_expr_seq:_D #1}%
1477 \def\XINT_expr_seq:_D #1#2{\expandafter\XINT_expr_seq:_E\romannumeral`&&@#2#1{#2}}%
1478 \def\XINT_expr_seq:_E #1{\if #1^\xint_dothis\XINT_expr_seq:_Abort\fi
1479         \if #1?\xint_dothis\XINT_expr_seq:_Break\fi
1480         \if #1!\xint_dothis\XINT_expr_seq:_Omit\fi
1481         \xint_orthat{\XINT_expr_seq:_Goon #1}}%
1482 \def\XINT_expr_seq:_Abort #1!#2#3#4{%}
1483 \def\XINT_expr_seq:_Break #1!#2#3#4{,#1}%
1484 \def\XINT_expr_seq:_Omit #1!#2#3%
1485     {\expandafter\XINT_expr_seq:_D
1486     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
1487 \def\XINT_expr_seq:_Goon #1!#2#3%
1488     {,#1\expandafter\XINT_expr_seq:_D
1489     \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/opxmul` 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.

```

1490 \def\XINT_expr_onlitteral_add
1491 {\expandafter\XINT_expr_onlitteral_add_f\romannumeral`&&\XINT_expr_onlitteral_seq_a {}}%
1492 \def\XINT_expr_onlitteral_add_f #1#2{\xint_c_xviii `{opxadd}#2)\relax #1}%
1493 \def\XINT_expr_onlitteral_mul
1494 {\expandafter\XINT_expr_onlitteral_mul_f\romannumeral`&&\XINT_expr_onlitteral_seq_a {}}%
1495 \def\XINT_expr_onlitteral_mul_f #1#2{\xint_c_xviii `{opxmul}#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.

```

1496 \def\XINT_expr_func_opxadd #1#2{\XINT_allexpr_opx \xintbareeval {\xintAdd 0}}%
1497 \def\XINT_flexpr_func_opxadd #1#2{\XINT_allexpr_opx \xintbarefloateval {\XINTinFloatAdd 0}}%
1498 \def\XINT_iiexpr_func_opxadd #1#2{\XINT_allexpr_opx \xintbareiieval {\xintiiAdd 0}}%
1499 \def\XINT_expr_func_opxmul #1#2{\XINT_allexpr_opx \xintbareeval {\xintMul 1}}%
1500 \def\XINT_flexpr_func_opxmul #1#2{\XINT_allexpr_opx \xintbarefloateval {\XINTinFloatMul 1}}%
1501 \def\XINT_iiexpr_func_opxmul #1#2{\XINT_allexpr_opx \xintbareiieval {\xintiiMul 1}}%

```

#1=bareval etc, #2={Add0} ou {Mul1}, #3=liste encapsulée, #4=la variable, #5=expression

```

1502 \def\XINT_allexpr_opx #1#2#3#4#5%
1503 {%
1504     \expandafter\XINT_expr_getop
1505     \csname.=\romannumeral`&&\expandafter\XINT_expr_op:_a
1506         \romannumeral`&&\XINT_expr_unlock #3!{#1#5\relax !#4}{#2}\endcsname
1507 }%
1508 \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.

```

1509 \def\XINT_expr_op:_b #1#2#3#4#5,{%

```



```

1510 \if ,#4\xint_dothis\XINT_expr_op:_noop\fi
1511 \if ^#4\xint_dothis\XINT_expr_op:_end\fi
1512 \xint_orthat{\expandafter\XINT_expr_op:_c}\csname.=#4#5\endcsname {#3}#1{#2}%
1513 }%
1514 \def\XINT_expr_op:_c #1#2#3#4{\expandafter\XINT_expr_op:_d\romannumeral0#2#1#3{#4}{#2}}%
1515 \def\XINT_expr_op:_d #1!#2#3#4#5%
1516 {\expandafter\XINT_expr_op:_b\expandafter #4\expandafter
1517 {\romannumeral`&&@#4{\XINT_expr_unlock#1}{#5}}}%
1518 \def\XINT_expr_op:_noop\csname.=,#1\endcsname #2#3#4{\XINT_expr_seq:_b #3{#4}{#2}#1,}%
1519 \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.

```

1520 \def\XINT_expr_onlitteral_subx
1521 {\expandafter\XINT_expr_onlitteral_subx_f\romannumeral`&&@\XINT_expr_onlitteral_seq_a {}}%
1522 \def\XINT_expr_onlitteral_subx_f #1#2{\xint_c_xviii `{subx}#2)\relax #1}%
1523 \def\XINT_expr_func_subx #1#2{\XINT_allexpr_subx \xintbareeval }%
1524 \def\XINT_flexpr_func_subx #1#2{\XINT_allexpr_subx \xintbarefloateval}%
1525 \def\XINT_iiexpr_func_subx #1#2{\XINT_allexpr_subx \xintbareiieval }%
1526 \def\XINT_allexpr_subx #1#2#3#4% #2 is the value to assign to the dummy variable
1527 {% #3 is the dummy variable, #4 is the expression to evaluate
1528 \expandafter\expandafter\expandafter\XINT_expr_getop
1529 \expandafter\XINT_expr_subx:_end\romannumeral0#1#4\relax !#3#2%
1530 }%
1531 \def\XINT_expr_subx:_end #1!#2#3{#1}%

```

10.40 rseq

10.40.1	\XINT_expr_rseqx	289
10.40.2	\XINT_expr_rseqy	289
10.40.3	\XINT_expr_rseq:_a etc...	289
10.40.4	\XINT_expr_rseq:_A etc...	289

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...

```

1532 \def\XINT_expr_func_rseq {\XINT_allexpr_rseq \xintbareeval \xintthebareeval }%
1533 \def\XINT_flexpr_func_rseq {\XINT_allexpr_rseq \xintbarefloateval \xintthebarefloateval }%
1534 \def\XINT_iiexpr_func_rseq {\XINT_allexpr_rseq \xintbareiieval \xintthebareiieval }%
1535 \def\XINT_allexpr_rseq #1#2#3%
1536 {%
1537 \expandafter\XINT_expr_rseqx\expandafter #1\expandafter#2\expandafter
1538 #3\romannumeral`&&@\XINT_expr_onlitteral_seq_a {}}%
1539 }%

```


10.40.1 \XINT_expr_rseqx

The (#5) is for ++ mechanism which must have its closing parenthesis.

```
1540 \def\XINT_expr_rseqx #1#2#3#4#5%
1541 {%
1542   \expandafter\XINT_expr_rseqy\romannumeral0#1(#5)\relax #3#4#2%
1543 }%
```

10.40.2 \XINT_expr_rseqy

#1=valeurs pour variable (locked), #2=toutes les valeurs initiales (csv,locked), #3=variable, #4=expr, #5=\xintthebareeval ou \xintthebarefloateval ou \xintthebareiieval

```
1544 \def\XINT_expr_rseqy #1#2#3#4#5%
1545 {%
1546   \expandafter \XINT_expr_getop
1547   \csname .=\XINT_expr_unlock #2%
1548   \expandafter\XINT_expr_rseq:_aa
1549   \romannumeral`&&@\XINT_expr_unlock #1!{#5#4\relax !#3}#2\endcsname
1550 }%
1551 \def\XINT_expr_rseq:_aa #1{\if +#1\expandafter\XINT_expr_rseq:_A\else
1552   \expandafter\XINT_expr_rseq:_a\fi #1}%

```

10.40.3 \XINT_expr_rseq:_a etc...

```
1553 \def\XINT_expr_rseq:_a #1!#2#3{\XINT_expr_rseq:_b {#3}{#2}#1,^,%}
1554 \def\XINT_expr_rseq:_b #1#2#3#4,{%
1555   \if ,#3\xint_dothis\XINT_expr_rseq:_noop\fi
1556   \if ^#3\xint_dothis\XINT_expr_rseq:_end\fi
1557   \xint_orthat{\expandafter\XINT_expr_rseq:_c}\csname.=#3#4\endcsname
1558   {#1}{#2}%
1559 }%
1560 \def\XINT_expr_rseq:_noop\csname.=#1\endcsname #2#3{\XINT_expr_rseq:_b {#2}{#3}#1,%}
1561 \def\XINT_expr_rseq:_end \csname.=^'\endcsname #1#2{%}
1562 \def\XINT_expr_rseq:_c #1#2#3%
1563   {\expandafter\XINT_expr_rseq:_d\romannumeral`&&@#3#1~#2{#3}}%
1564 \def\XINT_expr_rseq:_d #1{%
1565   \if ^#1\xint_dothis\XINT_expr_rseq:_abort\fi
1566   \if ?#1\xint_dothis\XINT_expr_rseq:_break\fi
1567   \if !#1\xint_dothis\XINT_expr_rseq:_omit\fi
1568   \xint_orthat{\XINT_expr_rseq:_goon #1}%}
1569 \def\XINT_expr_rseq:_goon #1!#2#3~#4#5{,#1\expandafter\XINT_expr_rseq:_b
1570   \romannumeral0\XINT_expr_lockit {#1}{#5}}%
1571 \def\XINT_expr_rseq:_omit #1!#2#3~{\XINT_expr_rseq:_b }%
1572 \def\XINT_expr_rseq:_abort #1!#2#3~#4#5#6^,{,%}
1573 \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.

```
1574 \def\XINT_expr_rseq:_A +#1!#2#3{\XINT_expr_rseq:_D #1#3{#2}}%
1575 \def\XINT_expr_rseq:_D #1#2#3%
```

```

1576 {\expandafter\XINT_expr_rseq:_E\romannumeral`&&@#3#1~#2{#3}}%
1577 \def\XINT_expr_rseq:_E #1{\if #1^\xint_dothis\XINT_expr_rseq:_Abort\fi
1578     \if #1?\xint_dothis\XINT_expr_rseq:_Break\fi
1579     \if #1!\xint_dothis\XINT_expr_rseq:_Omit\fi
1580     \xint_orthat{\XINT_expr_rseq:_Goon #1}}%
1581 \def\XINT_expr_rseq:_Goon #1!#2#3~#4#5%
1582     {,#1\expandafter\XINT_expr_rseq:_D
1583     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1584     \romannumeral0\XINT_expr_lockit{#1}{#5}}%
1585 \def\XINT_expr_rseq:_Omit #1!#2#3~#4#5%
1586     {\expandafter\XINT_expr_rseq:_D
1587     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname }%
1588 \def\XINT_expr_rseq:_Abort #1!#2#3~#4#5}%
1589 \def\XINT_expr_rseq:_Break #1!#2#3~#4#5{,#1}%

```

10.41 rrseq

10.41.1	\XINT_expr_rrseqx	290
10.41.2	\XINT_expr_rrseqy	290
10.41.3	\XINT_expr_rrseq:_a etc.	291
10.41.4	\XINT_expr_rrseq:_A etc.	291

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.

```

1590 \def\XINT_expr_func_rrseq {\XINT_allexpr_rrseq \xintbareeval \xintthebareeval }%
1591 \def\XINT_flexpr_func_rrseq {\XINT_allexpr_rrseq \xintbarefloateval \xintthebarefloateval }%
1592 \def\XINT_iiepr_func_rrseq {\XINT_allexpr_rrseq \xintbareiieval \xintthebareiieval }%
1593 \def\XINT_allexpr_rrseq #1#2#3%
1594 {%
1595     \expandafter\XINT_expr_rrseqx\expandafter #1\expandafter#2\expandafter
1596     #3\romannumeral`&&\XINT_expr_onlitteral_seq_a {}%
1597 }%

```

10.41.1 \XINT_expr_rrseqx

The (#5) is for ++ mechanism which must have its closing parenthesis.

```

1598 \def\XINT_expr_rrseqx #1#2#3#4#5%
1599 {%
1600     \expandafter\XINT_expr_rrseqy\romannumeral0#1(#5)\expandafter\relax
1601     \expandafter{\romannumeral0\xintapply \XINT_expr_lockit
1602         {\xintRevWithBraces{\xintCSVtoListNonStripped{\XINT_expr_unlock #3}}}%
1603         #3#4#2%
1604 }%

```

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

```

1605 \def\XINT_expr_rrseqy #1#2#3#4#5#6%

```

```

1606 {%
1607   \expandafter \XINT_expr_getop
1608   \csname .=\XINT_expr_unlock #3%
1609   \expandafter\XINT_expr_rrseq:_aa
1610       \romannumeral`&&@\XINT_expr_unlock #1!{#6#5\relax !#4}{#2}\endcsname
1611 }%
1612 \def\XINT_expr_rrseq:_aa #1{\if +#1\expandafter\XINT_expr_rrseq:_A\else
1613     \expandafter\XINT_expr_rrseq:_a\fi #1}%

```

10.41.3 \XINT_expr_rrseq:_a etc...

Attention que ? a catcode 3 ici et dans iter.

```

1614 \catcode`? 3
1615 \def\XINT_expr_rrseq:_a #1!#2#3{\XINT_expr_rrseq:_b {#3}{#2}#1,^,%}
1616 \def\XINT_expr_rrseq:_b #1#2#3#4,{%
1617     \if ,#3\xint_dothis\XINT_expr_rrseq:_noop\fi
1618     \if ^#3\xint_dothis\XINT_expr_rrseq:_end\fi
1619     \xint_orthat{\expandafter\XINT_expr_rrseq:_c}\csname .=#3#4\endcsname
1620     {#1}{#2}%
1621 }%
1622 \def\XINT_expr_rrseq:_noop\csname .=#1\endcsname #2#3{\XINT_expr_rrseq:_b {#2}{#3}#1,%}
1623 \def\XINT_expr_rrseq:_end \csname .=#1\endcsname #1#2{%}
1624 \def\XINT_expr_rrseq:_c #1#2#3%
1625     {\expandafter\XINT_expr_rrseq:_d\romannumeral`&&@#3#1~#2?{#3}}%
1626 \def\XINT_expr_rrseq:_d #1{%
1627     \if ^#1\xint_dothis\XINT_expr_rrseq:_abort\fi
1628     \if ?#1\xint_dothis\XINT_expr_rrseq:_break\fi
1629     \if !#1\xint_dothis\XINT_expr_rrseq:_omit\fi
1630     \xint_orthat{\XINT_expr_rrseq:_goon #1}%
1631 }%
1632 \def\XINT_expr_rrseq:_goon #1!#2#3~#4?#5{,#1\expandafter\XINT_expr_rrseq:_b\expandafter
1633     {\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}{#4}}{#5}}%
1634 \def\XINT_expr_rrseq:_omit #1!#2#3~{\XINT_expr_rrseq:_b }%
1635 \def\XINT_expr_rrseq:_abort #1!#2#3~#4?#5#6^,{,%}
1636 \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.

```

1637 \def\XINT_expr_rrseq:_A +#1!#2#3{\XINT_expr_rrseq:_D #1{#3}{#2}}%
1638 \def\XINT_expr_rrseq:_D #1#2#3%
1639     {\expandafter\XINT_expr_rrseq:_E\romannumeral`&&@#3#1~#2?{#3}}%
1640 \def\XINT_expr_rrseq:_Goon #1!#2#3~#4?#5%
1641     {,#1\expandafter\XINT_expr_rrseq:_D
1642         \csname .=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1643         \expandafter{\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}{#4}}{#5}}%
1644 \def\XINT_expr_rrseq:_Omit #1!#2#3~%#4?#5%
1645     {\expandafter\XINT_expr_rrseq:_D
1646         \csname .=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
1647 \def\XINT_expr_rrseq:_Abort #1!#2#3~#4?#5{%}
1648 \def\XINT_expr_rrseq:_Break #1!#2#3~#4?#5{,#1}%

```

```

1649 \def\XINT_expr_rrseq:_E #1{\if #1^\xint_dothis\XINT_expr_rrseq:_Abort\fi
1650           \if #1?\xint_dothis\XINT_expr_rrseq:_Break\fi
1651           \if #1!\xint_dothis\XINT_expr_rrseq:_Omit\fi
1652           \xint_orthat{\XINT_expr_rrseq:_Goon #1}}%

```

10.42 iter

10.42.1	<code>\XINT_expr_iterx</code>	292
10.42.2	<code>\XINT_expr_itory</code>	292
10.42.3	<code>\XINT_expr_iter:_a</code> etc...	292
10.42.4	<code>\XINT_expr_iter:_A</code> etc...	293

```

1653 \def\XINT_expr_func_iter {\XINT_allexpr_iter \xintbareeval \xintthebareeval }%
1654 \def\XINT_flexpr_func_iter {\XINT_allexpr_iter \xintbarefloateval \xintthebarefloateval }%
1655 \def\XINT_iexpr_func_iter {\XINT_allexpr_iter \xintbareiieval \xintthebareiieval }%
1656 \def\XINT_allexpr_iter #1#2#3%
1657 {%
1658   \expandafter\XINT_expr_iterx\expandafter #1\expandafter #2\expandafter
1659   #3\romannumeral`&&\XINT_expr_onlitteral_seq_a {}%
1660 }%

```

10.42.1 `\XINT_expr_iterx`

The (#5) is for ++ mechanism which must have its closing parenthesis.

```

1661 \def\XINT_expr_iterx #1#2#3#4#5%
1662 {%
1663   \expandafter\XINT_expr_itory\romannumeral0#1(#5)\expandafter\relax
1664   \expandafter{\romannumeral0\xintapply \XINT_expr_lockit
1665     {\xintRevWithBraces{\xintCSVtoListNonStripped{\XINT_expr_unlock #3}}}}%
1666   #3#4#2%
1667 }%

```

10.42.2 `\XINT_expr_itory`

#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

```

1668 \def\XINT_expr_itory #1#2#3#4#5#6%
1669 {%
1670   \expandafter \XINT_expr_getop
1671   \csname .=%
1672   \expandafter\XINT_expr_iter:_aa
1673     \romannumeral`&&\XINT_expr_unlock #1!{#6#5\relax !#4}{#2}\endcsname
1674 }%
1675 \def\XINT_expr_iter:_aa #1{\if +#1\expandafter\XINT_expr_iter:_A\else
1676   \expandafter\XINT_expr_iter:_a\fi #1}%

```

10.42.3 `\XINT_expr_iter:_a` etc...

```

1677 \def\XINT_expr_iter:_a #1!#2#3{\XINT_expr_iter:_b {#3}{#2}#1,^,%
1678 \def\XINT_expr_iter:_b #1#2#3#4,%

```

```

1679 \if ,#3\xint_dothis\xINT_expr_iter:_noop\fi
1680 \if ^#3\xint_dothis\xINT_expr_iter:_end\fi
1681 \xint_orthat{\expandafter\xINT_expr_iter:_c}\csname.=#3#4\endcsname
1682 {#1}{#2}%
1683 }%
1684 \def\xINT_expr_iter:_noop\csname.=#1\endcsname #2#3{\xINT_expr_iter:_b {#2}{#3}#1,%
1685 \def\xINT_expr_iter:_end \csname.=^ \endcsname #1#2%
1686 {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1687 {\,\xINT_expr:_unlock}{\xintReverseOrder{#1\space}}}%
1688 \def\xINT_expr_iter:_c #1#2#3%
1689 {\expandafter\xINT_expr_iter:_d\romannumeral`&&#3#1~#2?{#3}}%
1690 \def\xINT_expr_iter:_d #1{%
1691 \if ^#1\xint_dothis\xINT_expr_iter:_abort\fi
1692 \if ?#1\xint_dothis\xINT_expr_iter:_break\fi
1693 \if !#1\xint_dothis\xINT_expr_iter:_omit\fi
1694 \xint_orthat{\xINT_expr_iter:_goon #1}%
1695 }%
1696 \def\xINT_expr_iter:_goon #1!#2#3~#4?#5{\expandafter\xINT_expr_iter:_b\expandafter
1697 {\romannumeral0\xinttrim{-1}{\xINT_expr_lockit{#1}#4}}{#5}}%
1698 \def\xINT_expr_iter:_omit #1!#2#3~{\xINT_expr_iter:_b }%
1699 \def\xINT_expr_iter:_abort #1!#2#3~#4?#5#6^,%
1700 {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1701 {\,\xINT_expr:_unlock}{\xintReverseOrder{#4\space}}}%
1702 \def\xINT_expr_iter:_break #1!#2#3~#4?#5#6^,%
1703 {\expandafter\xint_gobble_iv\romannumeral0\xintapplyunbraced
1704 {\,\xINT_expr:_unlock}{\xintReverseOrder{#4\space}}},#1}%
1705 \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.

```

1706 \def\xINT_expr_iter:_A +#1!#2#3{\xINT_expr_iter:_D #1{#3}{#2}}%
1707 \def\xINT_expr_iter:_D #1#2#3%
1708 {\expandafter\xINT_expr_iter:_E\romannumeral`&&#3#1~#2?{#3}}%
1709 \def\xINT_expr_iter:_Goon #1!#2#3~#4?#5%
1710 {\expandafter\xINT_expr_iter:_D
1711 \csname.=\the\numexpr \xINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1712 \expandafter{\romannumeral0\xinttrim{-1}{\xINT_expr_lockit{#1}#4}}{#5}}%
1713 \def\xINT_expr_iter:_Omit #1!#2#3~%#4?#5%
1714 {\expandafter\xINT_expr_iter:_D
1715 \csname.=\the\numexpr \xINT_expr_unlock#3+\xint_c_i\endcsname}%
1716 \def\xINT_expr_iter:_Abort #1!#2#3~#4?#5%
1717 {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1718 {\,\xINT_expr:_unlock}{\xintReverseOrder{#4\space}}}%
1719 \def\xINT_expr_iter:_Break #1!#2#3~#4?#5%
1720 {\expandafter\xint_gobble_iv\romannumeral0\xintapplyunbraced
1721 {\,\xINT_expr:_unlock}{\xintReverseOrder{#4\space}}},#1}%
1722 \def\xINT_expr_iter:_E #1{\if #1^\xint_dothis\xINT_expr_iter:_Abort\fi
1723 \if #1?\xint_dothis\xINT_expr_iter:_Break\fi
1724 \if #1!\xint_dothis\xINT_expr_iter:_Omit\fi
1725 \xint_orthat{\xINT_expr_iter:_Goon #1}}%
1726 \catcode`? 11

```

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

10.43.1	<code>\xintANDof:csv</code>	294
10.43.2	<code>\xintORof:csv</code>	294
10.43.3	<code>\xintXORof:csv</code>	294
10.43.4	Generic csv routine	295
10.43.5	<code>\xintMaxof:csv, \xintiiMaxof:csv</code>	295
10.43.6	<code>\xintMinof:csv, \xintiiMinof:csv</code>	295
10.43.7	<code>\xintSum:csv, \xintiiSum:csv</code>	295
10.43.8	<code>\xintPrd:csv, \xintiiPrd:csv</code>	296
10.43.9	<code>\xintGCDof:csv, \xintLCMof:csv</code>	296
10.43.10	<code>\xintiiGCDof:csv, \xintiiLCMof:csv</code>	296
10.43.11	<code>\XINTinFloatdigits, \XINTinFloatSqrtdigits</code>	296
10.43.12	<code>\XINTinFloatMaxof:csv, \XINTinFloatMinof:csv</code>	296
10.43.13	<code>\XINTinFloatSum:csv, \XINTinFloatPrd:csv</code>	296

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.`

```

1727 \def\xintANDof:csv #1{\expandafter\XINT_andof:_a\romannumeral`&&@#1,,^}%
1728 \def\XINT_andof:_a #1{\if ,#1\expandafter\XINT_andof:_e
1729     \else\expandafter\XINT_andof:_c\fi #1}%
1730 \def\XINT_andof:_c #1,{\xintiiifNotZero {#1}{\XINT_andof:_a}{\XINT_andof:_no}}%
1731 \def\XINT_andof:_no #1^{0}%
1732 \def\XINT_andof:_e #1^{1}% works with empty list

```

10.43.2 `\xintORof:csv`

1.09a. For use by `\xintexpr`.

```

1733 \def\xintORof:csv #1{\expandafter\XINT_orof:_a\romannumeral`&&@#1,,^}%
1734 \def\XINT_orof:_a #1{\if ,#1\expandafter\XINT_orof:_e
1735     \else\expandafter\XINT_orof:_c\fi #1}%
1736 \def\XINT_orof:_c #1,{\xintiiifNotZero{#1}{\XINT_orof:_yes}{\XINT_orof:_a}}%
1737 \def\XINT_orof:_yes #1^{1}%
1738 \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`).

```

1739 \def\xintXORof:csv #1{\expandafter\XINT_xorof:_a\expandafter 0\romannumeral`&&@#1,,^}%
1740 \def\XINT_xorof:_a #1#2,{\XINT_xorof:_b #2,#1}%
1741 \def\XINT_xorof:_b #1{\if ,#1\expandafter\XINT_xorof:_e

```

```

1742             \else\expandafter\XINT_xorof:_c\fi #1}%
1743 \def\XINT_xorof:_c #1,#2%
1744     {\xintiiifNotZero {#1}{\if #20\xint_afterfi{\XINT_xorof:_a 1}%
1745         \else\xint_afterfi{\XINT_xorof:_a 0}\fi}%
1746         {\XINT_xorof:_a #2}%
1747     }%
1748 \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.

```

1749 \def\XINT_oncsv:_empty #1,^,#2{#2}%
1750 \def\XINT_oncsv:_end ^,#1#2#3#4{#1}%
1751 \def\XINT_oncsv:_a #1#2#3%
1752     {\if ,#3\expandafter\XINT_oncsv:_empty\else\expandafter\XINT_oncsv:_b\fi #1#2#3}%
1753 \def\XINT_oncsv:_b #1#2#3,%
1754     {\expandafter\XINT_oncsv:_c \expandafter{\romannumeral`&&@#2{#3}}#1#2}%
1755 \def\XINT_oncsv:_c #1#2#3#4,{\expandafter\XINT_oncsv:_d \romannumeral`&&@#4,{#1}#2#3}%
1756 \def\XINT_oncsv:_d #1%
1757     {\if ^#1\expandafter\XINT_oncsv:_end\else\expandafter\XINT_oncsv:_e\fi #1}%
1758 \def\XINT_oncsv:_e #1,#2#3#4%
1759     {\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.

```

1760 \def\xintMaxof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmax
1761     \expandafter\xint_firstofone\romannumeral`&&@#1,^,{0/1[0]}}%
1762 \def\xintiiMaxof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiimax
1763     \expandafter\xint_firstofone\romannumeral`&&@#1,^,0}%

```

10.43.6 \xintMinof:csv, \xintiiMinof:csv

1.09i. Rewritten for 1.1. For use by \xintiiexpr.

```

1764 \def\xintMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmin
1765     \expandafter\xint_firstofone\romannumeral`&&@#1,^,{0/1[0]}}%
1766 \def\xintiiMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiimin
1767     \expandafter\xint_firstofone\romannumeral`&&@#1,^,0}%

```

10.43.7 \xintSum:csv, \xintiiSum:csv

1.09a. Rewritten for 1.1. For use by \xintexpr.

```

1768 \def\xintSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintadd
1769     \expandafter\xint_firstofone\romannumeral`&&@#1,^,{0/1[0]}}%
1770 \def\xintiiSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiadd
1771     \expandafter\xint_firstofone\romannumeral`&&@#1,^,0}%

```

10.43.8 `\xintPrd:csv`, `\xintiiPrd:csv`

1.09a. Rewritten for 1.1. For use by `\xintexpr`.

```
1772 \def\xintPrd:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintmul
1773         \expandafter\xint_firstofone\romannumeral`&&@#1,^,{1/1[0]}}}%
1774 \def\xintiiPrd:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintiimul
1775         \expandafter\xint_firstofone\romannumeral`&&@#1,^,1}%

```

10.43.9 `\xintGCDof:csv`, `\xintLCMof:csv`

1.09a. Rewritten for 1.1. For use by `\xintexpr`. Expansion réinstallée pour besoins de `xintNewExpr` de version 1.1

```
1776 \def\xintGCDof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintgcd
1777         \expandafter\xint_firstofone\romannumeral`&&@#1,^,1}%
1778 \def\xintLCMof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintlcm
1779         \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.

```
1780 \def\xintiiGCDof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintiigcd
1781         \expandafter\xint_firstofone\romannumeral`&&@#1,^,1}%
1782 \def\xintiiLCMof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintiilcm
1783         \expandafter\xint_firstofone\romannumeral`&&@#1,^,0}%

```

10.43.11 `\XINTinFloatdigits`, `\XINTinFloatSqrtdigits`

for `\xintNewExpr` matters, mainly.

```
1784 \def\XINTinFloatdigits {\XINTinFloat [\XINTdigits]}%
1785 \def\XINTinFloatSqrtdigits {\XINTinFloatSqrt [\XINTdigits]}%

```

10.43.12 `\XINTinFloatMaxof:csv`, `\XINTinFloatMinof:csv`

1.09a. Rewritten for 1.1. For use by `\xintfloatexpr`. Name changed in 1.09h

```
1786 \def\XINTinFloatMaxof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintmax
1787         \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^,{0[0]}}}%
1788 \def\XINTinFloatMinof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintmin
1789         \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^,{0[0]}}}%

```

10.43.13 `\XINTinFloatSum:csv`, `\XINTinFloatPrd:csv`

1.09a. Rewritten for 1.1. For use by `\xintfloatexpr`.

```
1790 \def\XINTinFloatSum:csv #1{\expandafter\XINT_onscv:_a\expandafter\XINTinfloatadd
1791         \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^,{0[0]}}}%
1792 \def\XINTinFloatPrd:csv #1{\expandafter\XINT_onscv:_a\expandafter\XINTinfloatmul
1793         \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^,{1[0]}}}%

```


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

```

1794 \def\XINT_expr_twoargs #1,#2,{\{#1\}{#2}}%
1795 \def\XINT_expr_argandopt #1,#2,#3.#4#5%
1796 {%
1797     \if\relax#3\relax\expandafter\xint_firstoftwo\else
1798         \expandafter\xint_secondoftwo\fi
1799     {\#4}{\#5[\xintNum {\#2}]]{\#1}%
1800 }%
1801 \def\XINT_expr_oneortwo #1#2#3,#4,#5.%
1802 {%
1803     \if\relax#5\relax\expandafter\xint_firstoftwo\else
1804         \expandafter\xint_secondoftwo\fi
1805     {\#1{0}}{\#2{\xintNum {\#4}}}{\#3}%
1806 }%
1807 \def\XINT_iiexpr_oneortwo #1#2,#3,#4.%
1808 {%
1809     \if\relax#4\relax\expandafter\xint_firstoftwo\else
1810         \expandafter\xint_secondoftwo\fi
1811     {\#1{0}}{\#1{\#3}}{\#2}%
1812 }%
1813 \def\XINT_expr_func_num #1#2#3%
1814     {\expandafter #1\expandafter #2\csname=\xintNum {\XINT_expr_unlock #3}\endcsname }%
1815 \let\XINT_flexpr_func_num\XINT_expr_func_num
1816 \let\XINT_iiexpr_func_num\XINT_expr_func_num
1817 % [0] added Oct 25. For interaction with SPRaw::csv
1818 \def\XINT_expr_func_reduce #1#2#3%
1819     {\expandafter #1\expandafter #2\csname=\xintIrr {\XINT_expr_unlock #3}[0]\endcsname }%
1820 \let\XINT_flexpr_func_reduce\XINT_expr_func_reduce
1821 % no \XINT_iiexpr_func_reduce
1822 \def\XINT_expr_func_abs #1#2#3%
1823     {\expandafter #1\expandafter #2\csname=\xintAbs {\XINT_expr_unlock #3}\endcsname }%
1824 \let\XINT_flexpr_func_abs\XINT_expr_func_abs
1825 \def\XINT_iiexpr_func_abs #1#2#3%
1826     {\expandafter #1\expandafter #2\csname=\xintiAbs {\XINT_expr_unlock #3}\endcsname }%
1827 \def\XINT_expr_func_sgn #1#2#3%
1828     {\expandafter #1\expandafter #2\csname=\xintSgn {\XINT_expr_unlock #3}\endcsname }%
1829 \let\XINT_flexpr_func_sgn\XINT_expr_func_sgn
1830 \def\XINT_iiexpr_func_sgn #1#2#3%
1831     {\expandafter #1\expandafter #2\csname=\xintiiSgn {\XINT_expr_unlock #3}\endcsname }%
1832 \def\XINT_expr_func_frac #1#2#3%
1833     {\expandafter #1\expandafter #2\csname=\xintTFrac {\XINT_expr_unlock #3}\endcsname }%
1834 \def\XINT_flexpr_func_frac #1#2#3{\expandafter #1\expandafter #2\csname
1835     .=\XINTinFloatFracdigits {\XINT_expr_unlock #3}\endcsname }%

no \XINT_iiexpr_func_frac

1836 \def\XINT_expr_func_floor #1#2#3%
1837     {\expandafter #1\expandafter #2\csname .=\xintFloor {\XINT_expr_unlock #3}\endcsname }%
1838 \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.

```

1839 \def\xINT_iiexpr_func_floor #1#2#3%
1840 {%
1841   \expandafter #1\expandafter #2\csname.=\xintiFloor {\XINT_expr_unlock #3}\endcsname }%
1842 \def\xINT_expr_func_ceil #1#2#3%
1843   {\expandafter #1\expandafter #2\csname .=\xintCeil {\XINT_expr_unlock #3}\endcsname }%
1844 \let\xINT_flexpr_func_ceil\xINT_expr_func_ceil
1845 \def\xINT_iiexpr_func_ceil #1#2#3%
1846 {%
1847   \expandafter #1\expandafter #2\csname.=\xintiCeil {\XINT_expr_unlock #3}\endcsname }%
1848 \def\xINT_expr_func_sqr #1#2#3%
1849   {\expandafter #1\expandafter #2\csname.=\xintSqr {\XINT_expr_unlock #3}\endcsname }%
1850 \def\xINT_flexpr_func_sqr #1#2#3%
1851 {%
1852   \expandafter #1\expandafter #2\csname
1853     .=\XINTinFloatMul{\XINT_expr_unlock #3}{\XINT_expr_unlock #3}\endcsname
1854 }%
1855 \def\xINT_iiexpr_func_sqr #1#2#3%
1856   {\expandafter #1\expandafter #2\csname.=\xintiiSqr {\XINT_expr_unlock #3}\endcsname }%
1857 \def\xINT_expr_func_sqrt #1#2#3%
1858 {%
1859   \expandafter #1\expandafter #2\csname .=%
1860   \expandafter\xINT_expr_argandopt
1861   \romannumeral`&&\XINT_expr_unlock#3,,.\XINTinFloatSqrtdigits\xINTinFloatSqrt
1862   \endcsname
1863 }%
1864 \let\xINT_flexpr_func_sqrt\xINT_expr_func_sqrt
1865 \def\xINT_iiexpr_func_sqrt #1#2#3%
1866   {\expandafter #1\expandafter #2\csname.=\xintiiSqrt {\XINT_expr_unlock #3}\endcsname }%
1867 \def\xINT_iiexpr_func_sqrtr #1#2#3%
1868   {\expandafter #1\expandafter #2\csname.=\xintiiSqrR {\XINT_expr_unlock #3}\endcsname }%
1869 \def\xINT_expr_func_round #1#2#3%
1870 {%
1871   \expandafter #1\expandafter #2\csname .=%
1872   \expandafter\xINT_expr_oneortwo
1873   \expandafter\xintiRound\expandafter\xintRound
1874   \romannumeral`&&\XINT_expr_unlock #3,,.\endcsname
1875 }%
1876 \let\xINT_flexpr_func_round\xINT_expr_func_round
1877 \def\xINT_iiexpr_func_round #1#2#3%
1878 {%
1879   \expandafter #1\expandafter #2\csname .=%
1880   \expandafter\xINT_iiexpr_oneortwo\expandafter\xintiRound
1881   \romannumeral`&&\XINT_expr_unlock #3,,.\endcsname
1882 }%
1883 \def\xINT_expr_func_trunc #1#2#3%
1884 {%
1885   \expandafter #1\expandafter #2\csname .=%
1886   \expandafter\xINT_expr_oneortwo
1887   \expandafter\xintiTrunc\expandafter\xintTrunc
1888   \romannumeral`&&\XINT_expr_unlock #3,,.\endcsname

```

```

1889 }%
1890 \let\XINT_flexpr_func_trunc\XINT_expr_func_trunc
1891 \def\XINT_iiexpr_func_trunc #1#2#3%
1892 {%
1893   \expandafter #1\expandafter #2\csname .=%
1894   \expandafter\XINT_iiexpr_oneortwo\expandafter\xintiTrunc
1895   \romannumeral`&&\XINT_expr_unlock #3,,.\endcsname
1896 }%
1897 \def\XINT_expr_func_float #1#2#3%
1898 {%
1899   \expandafter #1\expandafter #2\csname .=%
1900   \expandafter\XINT_expr_argandopt
1901   \romannumeral`&&\XINT_expr_unlock #3,,.\XINTinFloatdigits\XINTinFloat
1902   \endcsname
1903 }%
1904 \let\XINT_flexpr_func_float\XINT_expr_func_float
1905 % \XINT_iiexpr_func_float not defined
1906 \def\XINT_expr_func_mod #1#2#3%
1907 {%
1908   \expandafter #1\expandafter #2\csname .=%
1909   \expandafter\expandafter\expandafter\xintMod
1910   \expandafter\XINT_expr_twoargs
1911   \romannumeral`&&\XINT_expr_unlock #3,\endcsname
1912 }%
1913 \def\XINT_flexpr_func_mod #1#2#3%
1914 {%
1915   \expandafter #1\expandafter #2\csname .=%
1916   \expandafter\XINTinFloatMod
1917   \romannumeral`&&\expandafter\XINT_expr_twoargs
1918   \romannumeral`&&\XINT_expr_unlock #3,\endcsname
1919 }%
1920 \def\XINT_iiexpr_func_mod #1#2#3%
1921 {%
1922   \expandafter #1\expandafter #2\csname .=%
1923   \expandafter\expandafter\expandafter\xintiiMod
1924   \expandafter\XINT_expr_twoargs
1925   \romannumeral`&&\XINT_expr_unlock #3,\endcsname
1926 }%
1927 \def\XINT_expr_func_quo #1#2#3%
1928 {%
1929   \expandafter #1\expandafter #2\csname .=%
1930   \expandafter\expandafter\expandafter\xintiQuo
1931   \expandafter\XINT_expr_twoargs
1932   \romannumeral`&&\XINT_expr_unlock #3,\endcsname
1933 }%
1934 \let\XINT_flexpr_func_quo\XINT_expr_func_quo
1935 \def\XINT_iiexpr_func_quo #1#2#3%
1936 {%
1937   \expandafter #1\expandafter #2\csname .=%
1938   \expandafter\expandafter\expandafter\xintiiQuo
1939   \expandafter\XINT_expr_twoargs
1940   \romannumeral`&&\XINT_expr_unlock #3,\endcsname

```

```

1941 }%
1942 \def\XINT_expr_func_rem #1#2#3%
1943 {%
1944     \expandafter #1\expandafter #2\csname .=%
1945     \expandafter\expandafter\expandafter\xintiRem
1946     \expandafter\XINT_expr_twoargs
1947     \romannumeral`&&\XINT_expr_unlock #3,\endcsname
1948 }%
1949 \let\XINT_flexpr_func_rem\XINT_expr_func_rem
1950 \def\XINT_iiexpr_func_rem #1#2#3%
1951 {%
1952     \expandafter #1\expandafter #2\csname .=%
1953     \expandafter\expandafter\expandafter\xintiiRem
1954     \expandafter\XINT_expr_twoargs
1955     \romannumeral`&&\XINT_expr_unlock #3,\endcsname
1956 }%
1957 \def\XINT_expr_func_gcd #1#2#3%
1958     {\expandafter #1\expandafter #2\csname
1959         .=\xintGCDof:csv{\XINT_expr_unlock #3}\endcsname }%
1960 \let\XINT_flexpr_func_gcd\XINT_expr_func_gcd
1961 \def\XINT_iiexpr_func_gcd #1#2#3%
1962     {\expandafter #1\expandafter #2\csname
1963         .=\xintiiGCDof:csv{\XINT_expr_unlock #3}\endcsname }%
1964 \def\XINT_expr_func_lcm #1#2#3%
1965     {\expandafter #1\expandafter #2\csname
1966         .=\xintLCMof:csv{\XINT_expr_unlock #3}\endcsname }%
1967 \let\XINT_flexpr_func_lcm\XINT_expr_func_lcm
1968 \def\XINT_iiexpr_func_lcm #1#2#3%
1969     {\expandafter #1\expandafter #2\csname
1970         .=\xintiiLCMof:csv{\XINT_expr_unlock #3}\endcsname }%
1971 \def\XINT_expr_func_max #1#2#3%
1972     {\expandafter #1\expandafter #2\csname
1973         .=\xintMaxof:csv{\XINT_expr_unlock #3}\endcsname }%
1974 \def\XINT_iiexpr_func_max #1#2#3%
1975     {\expandafter #1\expandafter #2\csname
1976         .=\xintiiMaxof:csv{\XINT_expr_unlock #3}\endcsname }%
1977 \def\XINT_flexpr_func_max #1#2#3%
1978     {\expandafter #1\expandafter #2\csname
1979         .=\XINTinFloatMaxof:csv{\XINT_expr_unlock #3}\endcsname }%
1980 \def\XINT_expr_func_min #1#2#3%
1981     {\expandafter #1\expandafter #2\csname
1982         .=\xintMinof:csv{\XINT_expr_unlock #3}\endcsname }%
1983 \def\XINT_iiexpr_func_min #1#2#3%
1984     {\expandafter #1\expandafter #2\csname
1985         .=\xintiiMinof:csv{\XINT_expr_unlock #3}\endcsname }%
1986 \def\XINT_flexpr_func_min #1#2#3%
1987     {\expandafter #1\expandafter #2\csname
1988         .=\XINTinFloatMinof:csv{\XINT_expr_unlock #3}\endcsname }%
1989 \expandafter\def\csname XINT_expr_func_+\endcsname #1#2#3%
1990     {\expandafter #1\expandafter #2\csname
1991         .=\xintSum:csv{\XINT_expr_unlock #3}\endcsname }%
1992 \expandafter\def\csname XINT_flexpr_func_+\endcsname #1#2#3%

```

```

1993   {\expandafter #1\expandafter #2\csname
1994                                     .=\XINTinFloatSum:csv{\XINT_expr_unlock #3}\endcsname }%
1995 \expandafter\def\csname XINT_iiexpr_func_+\endcsname #1#2#3%
1996   {\expandafter #1\expandafter #2\csname
1997                                     .=\xintiiSum:csv{\XINT_expr_unlock #3}\endcsname }%
1998 \expandafter\def\csname XINT_expr_func_+\endcsname #1#2#3%
1999   {\expandafter #1\expandafter #2\csname
2000                                     .=\xintPrd:csv{\XINT_expr_unlock #3}\endcsname }%
2001 \expandafter\def\csname XINT_flexpr_func_+\endcsname #1#2#3%
2002   {\expandafter #1\expandafter #2\csname
2003                                     .=\XINTinFloatPrd:csv{\XINT_expr_unlock #3}\endcsname }%
2004 \expandafter\def\csname XINT_iiexpr_func_+\endcsname #1#2#3%
2005   {\expandafter #1\expandafter #2\csname
2006                                     .=\xintiiPrd:csv{\XINT_expr_unlock #3}\endcsname }%
2007 \def\XINT_expr_func_? #1#2#3%
2008   {\expandafter #1\expandafter #2\csname
2009                                     .=\xintiiIsNotZero {\XINT_expr_unlock #3}\endcsname }%
2010 \let\XINT_flexpr_func_? \XINT_expr_func_?
2011 \let\XINT_iiexpr_func_? \XINT_expr_func_?
2012 \def\XINT_expr_func_! #1#2#3%
2013 {\expandafter #1\expandafter #2\csname .=\xintiiIsZero {\XINT_expr_unlock #3}\endcsname }%
2014 \let\XINT_flexpr_func_! \XINT_expr_func_!
2015 \let\XINT_iiexpr_func_! \XINT_expr_func_!
2016 \def\XINT_expr_func_not #1#2#3%
2017 {\expandafter #1\expandafter #2\csname .=\xintiiIsZero {\XINT_expr_unlock #3}\endcsname }%
2018 \let\XINT_flexpr_func_not \XINT_expr_func_not
2019 \let\XINT_iiexpr_func_not \XINT_expr_func_not
2020 \def\XINT_expr_func_all #1#2#3%
2021   {\expandafter #1\expandafter #2\csname
2022                                     .=\xintANDof:csv{\XINT_expr_unlock #3}\endcsname }%
2023 \let\XINT_flexpr_func_all\XINT_expr_func_all
2024 \let\XINT_iiexpr_func_all\XINT_expr_func_all
2025 \def\XINT_expr_func_any #1#2#3%
2026   {\expandafter #1\expandafter #2\csname
2027                                     .=\xintORof:csv{\XINT_expr_unlock #3}\endcsname }%
2028 \let\XINT_flexpr_func_any\XINT_expr_func_any
2029 \let\XINT_iiexpr_func_any\XINT_expr_func_any
2030 \def\XINT_expr_func_xor #1#2#3%
2031   {\expandafter #1\expandafter #2\csname
2032                                     .=\xintXORof:csv{\XINT_expr_unlock #3}\endcsname }%
2033 \let\XINT_flexpr_func_xor\XINT_expr_func_xor
2034 \let\XINT_iiexpr_func_xor\XINT_expr_func_xor
2035 \def\xintifNotZero: #1,#2,#3,{\xintiiifNotZero{#1}{#2}{#3}}%
2036 \def\XINT_expr_func_if #1#2#3%
2037   {\expandafter #1\expandafter #2\csname
2038                                     .=\expandafter\xintifNotZero:\romannumeral`&&\XINT_expr_unlock #3,\endcsname }%
2039 \let\XINT_flexpr_func_if\XINT_expr_func_if
2040 \let\XINT_iiexpr_func_if\XINT_expr_func_if
2041 \def\xintifSgn: #1,#2,#3,#4,{\xintiiifSgn{#1}{#2}{#3}{#4}}%
2042 \def\XINT_expr_func_ifsgn #1#2#3%
2043 {%
2044   \expandafter #1\expandafter #2\csname

```

```

2045      .=\expandafter\xintifSgn:\romannumeral`&&\XINT_expr_unlock #3,\endcsname
2046 }%
2047 \let\XINT_flexpr_func_ifsgn\XINT_expr_func_ifsgn
2048 \let\XINT_iiexpr_func_ifsgn\XINT_expr_func_ifsgn
2049 \def\XINT_expr_func_first #1#2#3%
2050     {\expandafter #1\expandafter #2\csname.=\expandafter\XINT_expr_func_firsta
2051       \romannumeral`&&\XINT_expr_unlock #3,^\endcsname }%
2052 \def\XINT_expr_func_firsta #1,#2^{#1}%
2053 \let\XINT_flexpr_func_first\XINT_expr_func_first
2054 \let\XINT_iiexpr_func_first\XINT_expr_func_first
2055 \def\XINT_expr_func_last #1#2#3% will not work in \xintNewExpr if macro param involved
2056     {\expandafter #1\expandafter #2\csname.=\expandafter\XINT_expr_func_lasta
2057       \romannumeral`&&\XINT_expr_unlock #3,^\endcsname }%
2058 \def\XINT_expr_func_lasta #1,#2%
2059     {\if ^#2 #1\expandafter\xint_gobble_ii\fi \XINT_expr_func_lasta #2}%
2060 \let\XINT_flexpr_func_last\XINT_expr_func_last
2061 \let\XINT_iiexpr_func_last\XINT_expr_func_last
2062 \def\XINT_expr_func_odd #1#2#3%
2063     {\expandafter #1\expandafter #2\csname.=\xintOdd{\XINT_expr_unlock #3}\endcsname}%
2064 \let\XINT_flexpr_func_odd\XINT_expr_func_odd
2065 \def\XINT_iiexpr_func_odd #1#2#3%
2066     {\expandafter #1\expandafter #2\csname.=\xintiiOdd{\XINT_expr_unlock #3}\endcsname}%
2067 \def\XINT_expr_func_even #1#2#3%
2068     {\expandafter #1\expandafter #2\csname.=\xintEven{\XINT_expr_unlock #3}\endcsname}%
2069 \let\XINT_flexpr_func_even\XINT_expr_func_even
2070 \def\XINT_iiexpr_func_even #1#2#3%
2071     {\expandafter #1\expandafter #2\csname.=\xintiiEven{\XINT_expr_unlock #3}\endcsname}%
2072 \def\XINT_expr_func_nuple #1#2#3%
2073     {\expandafter #1\expandafter #2\csname .=\XINT_expr_unlock #3\endcsname }%
2074 \let\XINT_flexpr_func_nuple\XINT_expr_func_nuple
2075 \let\XINT_iiexpr_func_nuple\XINT_expr_func_nuple

```

1.2c 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`.

```

2076 \def\XINT_expr_func_reversed #1#2#3%
2077     {\expandafter #1\expandafter #2\csname .=%
2078       \xintReverse::csv {\XINT_expr_unlock #3}\endcsname }%
2079 \let\XINT_flexpr_func_reversed\XINT_expr_func_reversed
2080 \let\XINT_iiexpr_func_reversed\XINT_expr_func_reversed
2081 \def\xintReverse::csv #1% should be done directly, of course
2082     {\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>	302
10.45.2	<code>\xintiiSeqB:f:csv</code>	303
10.45.3	<code>\XINTinFloatSeqB:f:csv</code>	304

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.

```

2083 \def\xintSeqB:f:csv #1#2%
2084   {\expandafter\XINT_seqb:f:csv \expandafter{\romannumeral0\xintra{#2}}{#1}}%
2085 \def\XINT_seqb:f:csv #1#2{\expandafter\XINT_seqb:f:csv_a\romannumeral`&&@#2#1!}%
2086 \def\XINT_seqb:f:csv_a #1#2;#3;#4!{%
2087   \expandafter\xint_gobble_i\romannumeral`&&@%
2088   \xintifCmp {#3}{#4}\XINT_seqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_seqb:f:csv_bg
2089   #1{#3}{#4}{#2}}%
2090 \def\XINT_seqb:f:csv_be #1#2#3#4#5{,#2}%
2091 \def\XINT_seqb:f:csv_bl #1{\if #1p\expandafter\XINT_seqb:f:csv_pa\else
2092   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2093 \def\XINT_seqb:f:csv_pa #1#2#3#4{\expandafter\XINT_seqb:f:csv_p\expandafter
2094   {\romannumeral0\xintadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2095 \def\XINT_seqb:f:csv_p #1#2%
2096 {%
2097   \xintifCmp {#1}{#2}\XINT_seqb:f:csv_pa\XINT_seqb:f:csv_pb\XINT_seqb:f:csv_pc
2098   {#1}{#2}%
2099 }%
2100 \def\XINT_seqb:f:csv_pb #1#2#3#4{#3,#1}%
2101 \def\XINT_seqb:f:csv_pc #1#2#3#4{#3}%
2102 \def\XINT_seqb:f:csv_bg #1{\if #1n\expandafter\XINT_seqb:f:csv_na\else
2103   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2104 \def\XINT_seqb:f:csv_na #1#2#3#4{\expandafter\XINT_seqb:f:csv_n\expandafter
2105   {\romannumeral0\xintadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2106 \def\XINT_seqb:f:csv_n #1#2%
2107 {%
2108   \xintifCmp {#1}{#2}\XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_seqb:f:csv_na
2109   {#1}{#2}%
2110 }%
2111 \def\XINT_seqb:f:csv_nb #1#2#3#4{#3,#1}%
2112 \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).

```

2113 \def\xintiiSeqB:f:csv #1#2%
2114   {\expandafter\XINT_iiiseqb:f:csv \expandafter{\romannumeral`&&@#2}{#1}}%
2115 \def\XINT_iiiseqb:f:csv #1#2{\expandafter\XINT_iiiseqb:f:csv_a\romannumeral`&&@#2#1!}%
2116 \def\XINT_iiiseqb:f:csv_a #1#2;#3;#4!{%
2117   \expandafter\xint_gobble_i\romannumeral`&&@%
2118   \xintSgnFork{\XINT_Cmp {#3}{#4}}%
2119   \XINT_iiiseqb:f:csv_bl\XINT_iiiseqb:f:csv_be\XINT_iiiseqb:f:csv_bg
2120   #1{#3}{#4}{#2}}%
2121 \def\XINT_iiiseqb:f:csv_bl #1{\if #1p\expandafter\XINT_iiiseqb:f:csv_pa\else
2122   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2123 \def\XINT_iiiseqb:f:csv_pa #1#2#3#4{\expandafter\XINT_iiiseqb:f:csv_p\expandafter
2124   {\romannumeral0\xintiiadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2125 \def\XINT_iiiseqb:f:csv_p #1#2%
2126 {%

```



```

2127 \xintSgnFork{\XINT_Cmp {#1}{#2}}%
2128 \XINT_iiseqb:f:csv_pa\XINT_iiseqb:f:csv_pb\XINT_iiseqb:f:csv_pc {#1}{#2}%
2129 }%
2130 \def\XINT_iiseqb:f:csv_pb #1#2#3#4{#3,#1}%
2131 \def\XINT_iiseqb:f:csv_pc #1#2#3#4{#3}%
2132 \def\XINT_iiseqb:f:csv_bg #1{\if #1n\expandafter\XINT_iiseqb:f:csv_na\else
2133 \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2134 \def\XINT_iiseqb:f:csv_na #1#2#3#4{\expandafter\XINT_iiseqb:f:csv_n\expandafter
2135 {\romannumeral0\xintiiadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2136 \def\XINT_iiseqb:f:csv_n #1#2%
2137 {%
2138 \xintSgnFork{\XINT_Cmp {#1}{#2}}%
2139 \XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_iiseqb:f:csv_na {#1}{#2}%
2140 }%

```

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.

```

2141 \def\XINTinFloatSeqB:f:csv #1#2{\expandafter\XINT_flseqb:f:csv \expandafter
2142 {\romannumeral0\XINTinfloat [\XINTdigits]{#2}}{#1}}%
2143 \def\XINT_flseqb:f:csv #1#2{\expandafter\XINT_flseqb:f:csv_a\romannumeral`&&@#2#1!}%
2144 \def\XINT_flseqb:f:csv_a #1#2;#3;#4!{%
2145 \expandafter\xint_gobble_i\romannumeral`&&@%
2146 \xintifCmp {#3}{#4}\XINT_flseqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_flseqb:f:csv_bg
2147 #1{#3}{#4}}{#2}}%
2148 \def\XINT_flseqb:f:csv_bl #1{\if #1p\expandafter\XINT_flseqb:f:csv_pa\else
2149 \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2150 \def\XINT_flseqb:f:csv_pa #1#2#3#4{\expandafter\XINT_flseqb:f:csv_p\expandafter
2151 {\romannumeral0\XINTinfloatadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2152 \def\XINT_flseqb:f:csv_p #1#2%
2153 {%
2154 \xintifCmp {#1}{#2}%
2155 \XINT_flseqb:f:csv_pa\XINT_flseqb:f:csv_pb\XINT_flseqb:f:csv_pc {#1}{#2}%
2156 }%
2157 \def\XINT_flseqb:f:csv_pb #1#2#3#4{#3,#1}%
2158 \def\XINT_flseqb:f:csv_pc #1#2#3#4{#3}%
2159 \def\XINT_flseqb:f:csv_bg #1{\if #1n\expandafter\XINT_flseqb:f:csv_na\else
2160 \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2161 \def\XINT_flseqb:f:csv_na #1#2#3#4{\expandafter\XINT_flseqb:f:csv_n\expandafter
2162 {\romannumeral0\XINTinfloatadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2163 \def\XINT_flseqb:f:csv_n #1#2%
2164 {%
2165 \xintifCmp {#1}{#2}%
2166 \XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_flseqb:f:csv_na {#1}{#2}%
2167 }%

```

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 `:`.

```

2168 \catcode`: 12
2169 \def\XINT_tmpa #1#2#3#4%
2170 {%
2171   \def #1##1(##2)##3=##4;{%
2172     \edef\XINT_expr_tmpa {##1}%
2173     \edef\XINT_expr_tmpa {\xint_zapspace_o \XINT_expr_tmpa}%
2174     \def\XINT_expr_tmpb {0}%
2175     \def\XINT_expr_tmpc {##4}%
2176     \xintFor ###1 in {##2} \do
2177       {\edef\XINT_expr_tmpb {\the\numexpr\XINT_expr_tmpb+\xint_c_i}%
2178        \edef\XINT_expr_tmpc {subs(\unexpanded\expandafter{\XINT_expr_tmpc},%
2179                                     #####1=#####\XINT_expr_tmpb)}}%
2180     }%
2181     \expandafter#3\csname XINT_#2_userfunc_\XINT_expr_tmpa\endcsname
2182       [\XINT_expr_tmpb]{\XINT_expr_tmpc}%
2183     \expandafter\XINT_expr_defuserfunc
2184       \csname XINT_#2_func_\XINT_expr_tmpa\expandafter\endcsname
2185       \csname XINT_#2_userfunc_\XINT_expr_tmpa\endcsname
2186     \ifxintverbose\xintMessage {xintexpr}{Info}
2187       {Function \XINT_expr_tmpa\space for \string\xint #4 parser
2188        associated to \string\XINT_#2_userfunc_\XINT_expr_tmpa\space
2189        with meaning \expandafter\meaning
2190        \csname XINT_#2_userfunc_\XINT_expr_tmpa\endcsname}%
2191     \fi
2192   }%
2193 }%
2194 \catcode`: 11
2195 \XINT_tmpa\xintdeffunc      {expr} \XINT_NewFunc      {expr}%
2196 \XINT_tmpa\xintdefiifunc    {iiexpr}\XINT_NewIIFunc    {iiexpr}%
2197 \XINT_tmpa\xintdeffloatfunc {flexpr}\XINT_NewFloatFunc{floatexpr}%
2198 \def\XINT_expr_defuserfunc #1#2{%
2199   \def #1##1##2##3{\expandafter ##1\expandafter ##2%
2200     \csname .=\expandafter #2\romannumeral-`0\XINT_expr_unlock ##3,\endcsname
2201   }%
2202 }%

```

10.47 `\xintNewExpr`, `\xintNewIExpr`, `\xintNewFloatExpr`, `\xintNewIIExpr`

10.47.1	<code>\xintApply::csv</code>	306
10.47.2	<code>\xintApply:::csv</code>	306
10.47.3	<code>\XINT_expr_RApply::csv</code> , <code>\XINT_expr_LApply::csv</code> , <code>\XINT_expr_RLApply:::csv</code> . .	306
10.47.4	Mysterious stuff	306
10.47.5	<code>\xintNewExpr</code> , ..., at last.	310

10.47.1 \xintApply::csv

Don't ask me what this is for. I wrote it in June 2014, and we are now late October 2014.

```

2203 \def\xintApply::csv #1#2%
2204   {\expandafter\xint_applyon::_a\expandafter {\romannumeral`&&@#2}{#1}}%
2205 \def\xint_applyon::_a #1#2{\xint_applyon::_b {#2}{#1},,%
2206 \def\xint_applyon::_b #1#2#3,{\expandafter\xint_applyon::_c \romannumeral`&&@#3,{#1}{#2}}%
2207 \def\xint_applyon::_c #1{\if #1,\expandafter\xint_applyon::_end
2208   \else\expandafter\xint_applyon::_d\fi #1}%
2209 \def\xint_applyon::_d #1,#2{\expandafter\xint_applyon::_e\romannumeral`&&@#2{#1},{#2}}%
2210 \def\xint_applyon::_e #1,#2#3{\xint_applyon::_b {#2}{#3}, #1}%
2211 \def\xint_applyon::_end #1,#2#3{\xint_secondoftwo #3}%

```

10.47.2 \xintApply:::csv

```

2212 \def\xintApply:::csv #1#2#3%
2213   {\expandafter\xint_applyon::_a\expandafter{\romannumeral`&&@#2}{#1}{#3}}%
2214 \def\xint_applyon::_a #1#2#3{\xint_applyon::_b {#2}{#3}{#1},,%
2215 \def\xint_applyon::_b #1#2#3#4,%
2216   {\expandafter\xint_applyon::_c \romannumeral`&&@#4,{#1}{#2}{#3}}%
2217 \def\xint_applyon::_c #1{\if #1,\expandafter\xint_applyon::_end
2218   \else\expandafter\xint_applyon::_d\fi #1}%
2219 \def\xint_applyon::_d #1,#2#3%
2220   {\expandafter\xint_applyon::_e\expandafter
2221     {\romannumeral`&&\xintApply:::csv {#2}{#1}{#3}}{#2}{#3}}%
2222 \def\xint_applyon::_e #1,#2#3#4{\xint_applyon::_b {#2}{#3}{#4}, #1}%
2223 \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).

```

2224 \catcode`~ 12
2225 \def\XINT_expr_RApply::csv #1#2#3#4%
2226   {\xintApply:::csv{~\expandafter#1~\xint_exchangetwo_keepbraces{#4}}{#3}}%
2227 \def\XINT_expr_LApply:::csv #1#2#3#4{\xintApply:::csv{#1}{#3}{#4}}%
2228 \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.

```

2229 \catcode`$ 12 % $
2230 \def\xINT_xptwo_getab_b #1#2!#3%
2231   {\expandafter\xINT_xptwo_getab_c\romannumeral`&&@#3!#1{#1#2}}%
2232 \def\xINT_xptwo_getab_c #1#2!#3#4#5#6{#1#3{#5}{#6}{#1#2}{#4}}%
2233 \def\xint_ddfork #1$#2#3\krof {#2}% $$
2234 \def\xINT_NEfork #1#2{\xint_ddfork
2235   #1#2\xINT_expr_RLApply::csv
2236   #1$\xINT_expr_RApply::csv% $
2237   $#2\xINT_expr_LApply::csv% $
2238   $${\xINT_NEfork_nn #1#2}% $$
2239   \krof }%
2240 \def\xINT_NEfork_nn #1#2#3#4{%
2241   \if #1#\xint_dothis{#3}\fi
2242   \if #1~\xint_dothis{#3}\fi
2243   \if #2#\xint_dothis{#3}\fi
2244   \if #2~\xint_dothis{#3}\fi
2245   \xint_orthat {\csname #4NE\endcsname}%
2246   }%
2247 \def\xINT_NEfork_one #1#2!#3#4#5#6{%
2248   \if ###1\xint_dothis {#3}\fi
2249   \if ~#1\xint_dothis {#3}\fi
2250   \if $#1\xint_dothis {\xintApply::csv{#3#5}}\fi %$
2251   \xint_orthat {\csname #4NE\endcsname #6}{#1#2}%
2252 }%
2253 \toks0 {}%
2254 \xintFor #1 in {DivTrunc,iiDivTrunc,iiDivRound,Mod,iiMod,iRound,Round,iTrunc,Trunc,%
2255   Lt,Gt,Eq,LtorEq,GtorEq,Neq,AND,OR,XOR,iQuo,iRem,Add,Sub,Mul,Div,Pow,E,%
2256   iiAdd,iiSub,iiMul,iiPow,iiQuo,iiRem,iiE,SeqA::csv,iiSeqA::csv}\do
2257 {\toks0
2258   \expandafter{\the\toks0% no space!
2259   \expandafter\let\csname xint#1NE\expandafter\endcsname\csname xint#1\expandafter
2260   \endcsname\expandafter\def\csname xint#1\endcsname ####1####2{%
2261     \expandafter\xINT_NEfork
2262     \romannumeral`&&\expandafter\xINT_xptwo_getab_b
2263     \romannumeral`&&@####2!{####1}{~xint#1}{xint#1}}%
2264   }%
2265 }% cela aurait-il un sens d'ajouter Raw et iNum (à cause de qint, qfrac,
2266 % qfloat?). Pas le temps d'y réfléchir. Je ne fais rien.
2267 \xintFor #1 in {Num,Irr,Abs,iiAbs,Sgn,iiSgn,TFrac,Floor,iFloor,Ceil,iCeil,%
2268   Sqr,iiSqr,iiSqrt,iiSqrtR,iiIsZero,iiIsNotZero,iiifNotZero,iiifSgn,%

```

```

2269   Odd,Even,iiOdd,iiEven,Opp,iiOpp,iiifZero,Fac,iiFac,Bool,Toggle}\do
2270 {\toks0
2271   \expandafter{\the\toks0%
2272   \expandafter\let\csname xint#1NE\expandafter\endcsname\csname xint#1\expandafter
2273   \endcsname\expandafter\def\csname xint#1\endcsname ####1{%
2274     \expandafter\XINT_NEfork_one\romannumeral`&&@####1!{\sim xint#1}{xint#1}{}}}%
2275   }%
2276 }%
2277 \toks0
2278   \expandafter{\the\toks0
2279   \let\XINTinFloatFacNE\XINTinFloatFac
2280   \def\XINTinFloatFac ##1{%
2281     \expandafter\XINT_NEfork_one
2282     \romannumeral`&&@##1!{\sim XINTinFloatFac}{XINTinFloatFac}{}}}%
2283   }%
2284 \xintFor #1 in {Add,Sub,Mul,Div,Power,E,Mod,SeqA::csv}\do
2285 {\toks0
2286   \expandafter{\the\toks0%
2287   \expandafter\let\csname XINTinFloat#1NE\expandafter\endcsname
2288     \csname XINTinFloat#1\expandafter\endcsname
2289   \expandafter\def\csname XINTinFloat#1\endcsname ####1####2{%
2290     \expandafter\XINT_NEfork
2291     \romannumeral`&&@\expandafter\XINT_xptwo_getab_b
2292     \romannumeral`&&@####2!{####1}{\sim XINTinFloat#1}{XINTinFloat#1}}}%
2293   }%
2294 }%
2295 \xintFor #1 in {XINTinFloatdigits,XINTinFloatFracdigits,XINTinFloatSqrtdigits}\do
2296 {\toks0
2297   \expandafter{\the\toks0%
2298   \expandafter\let\csname #1NE\expandafter\endcsname\csname #1\expandafter
2299   \endcsname\expandafter\def\csname #1\endcsname ####1{\expandafter
2300     \XINT_NEfork_one\romannumeral`&&@####1!{\sim #1}{#1}{}}}%
2301   }%
2302 }%
2303 \xintFor #1 in {xintSeq::csv,xintiiSeq::csv,XINTinFloatSeq::csv}\do
2304 {\toks0
2305   \expandafter{\the\toks0% no space
2306   \expandafter\let\csname #1NE\expandafter\endcsname\csname #1\expandafter
2307   \endcsname\expandafter\def\csname #1\endcsname ####1####2{%
2308     \expandafter\XINT_NEfork
2309     \romannumeral`&&@\expandafter\XINT_xptwo_getab_b
2310     \romannumeral`&&@####2!{####1}{\noexpand$#1}{#1}}}%
2311   }%
2312 }%
2313 \xintFor #1 in {xintSeqB,xintiiSeqB,XINTinFloatSeqB}\do
2314 {\toks0
2315   \expandafter{\the\toks0% no space
2316   \expandafter\let\csname #1::csvNE\expandafter\endcsname\csname #1::csv\expandafter
2317   \endcsname\expandafter\def\csname #1::csv\endcsname ####1####2{%
2318     \expandafter\XINT_NEfork
2319     \romannumeral`&&@\expandafter\XINT_xptwo_getab_b
2320     \romannumeral`&&@####2!{####1}{\noexpand$#1:f:csv}{#1::csv}}}%

```

```

2321 }%
2322 }%
2323 \toks0
2324 \expandafter{\the\toks0
2325 \let\XINTinFloatNE\XINTinFloat
2326 \def\XINTinFloat [##1]##2{% not ultimately general, but got tired
2327     \expandafter\XINT_NEfork_one
2328     \romannumeral`&&@##2!{\XINTinFloat[##1]}{\XINTinFloat}{\{[##1]}}}%
2329 \let\XINTinFloatSqrtNE\XINTinFloatSqrt
2330 \def\XINTinFloatSqrt [##1]##2{%
2331     \expandafter\XINT_NEfork_one
2332     \romannumeral`&&@##2!{\XINTinFloatSqrt[##1]}{\XINTinFloatSqrt}{\{[##1]}}}%
2333 }%
2334 \xintFor #1 in {ANDof,ORof,XORof,iiMaxof,iiMinof,iiSum,iiPrd,
2335               GCDof,LCMof,Sum,Prd,Maxof,Minof}\do
2336 {\toks0
2337   \expandafter{\the\toks0\expandafter\def\csname xint#1:csv\endcsname {\~xint#1:csv}}%
2338 }%
2339 \xintFor #1 in
2340   {XINTinFloatMaxof,XINTinFloatMinof,XINTinFloatSum,XINTinFloatPrd}\do
2341 {\toks0
2342   \expandafter{\the\toks0\expandafter\def\csname #1:csv\endcsname {\~#1:csv}}%
2343 }%

```

~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).

```

2344 \toks0 \expandafter{\the\toks0
2345         \def\xintReverse::csv {\~xintReverse::csv }%
2346         \def\xintListSel::csv {\~xintListSel::csv }%
2347 }%
2348 \odef\XINT_expr_redefinmacros {\the\toks0}% Not \edef ! (subtle)
2349 \def\XINT_expr_redefineprints
2350 {%
2351   \def\XINT_flexpr_noopt
2352     {\expandafter\XINT_flexpr_withopt_b\expandafter-\romannumeral0\xintbarefloateval }%
2353   \def\XINT_flexpr_withopt_b ##1##2%
2354     {\expandafter\XINT_flexpr_wrap\csname .;##1.=\XINT_expr_unlock ##2\endcsname }%
2355   \def\XINT_expr_unlock_sp ##1.;##2##3.=##4!%
2356     {\if -##2\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
2357      \XINTdigits{\##2##3}{\##4}}}%
2358   \def\XINT_expr_print ##1{\expandafter\xintSPRaw::csv\expandafter
2359     {\romannumeral`&&\XINT_expr_unlock ##1}}%
2360   \def\XINT_iiexpr_print ##1{\expandafter\xintCSV::csv\expandafter
2361     {\romannumeral`&&\XINT_expr_unlock ##1}}%
2362   \def\XINT_boolexpr_print ##1{\expandafter\xintIsTrue::csv\expandafter
2363     {\romannumeral`&&\XINT_expr_unlock ##1}}%

```

1) spaces after *::csv* needed to separate from possible later stuff. Well I currently don't recall what I meant by that.

2) due to redefinitions done above of `\XINT_flexpr_noopt`, etc..., no need to redefine `\xint-Float::csv` as it is not used (sub-expressions not supported), it is `\xintPFloat::csv` which is neutralized.

```

2364 \def\xintCSV::csv      {\xintCSV::csv }%
2365 \def\xintSPRaw::csv    {\xintSPRaw::csv }%
2366 \def\xintPFloat::csv   {\xintPFloat::csv }%
2367 \def\xintIsTrue::csv   {\xintIsTrue::csv }%
2368 \def\xintRound::csv    {\xintRound::csv }%
2369 }%
2370 \toks0 {}%
```

10.47.5 `\xintNewExpr`, ..., at last.

1.2c modifications to accomodate `\XINT_expr_deffunc_newexpr` etc..

```

2371 \def\xintNewExpr      {\XINT_NewExpr{\XINT_expr_redefineprints\xint_firstofone\xinttheexpr }%
2372 \def\xintNewFloatExpr {\XINT_NewExpr{\XINT_expr_redefineprints\xint_firstofone\xintthefloatexpr }%
2373 \def\xintNewIExpr      {\XINT_NewExpr{\XINT_expr_redefineprints\xint_firstofone\xinttheiexpr }%
2374 \def\xintNewIIExpr     {\XINT_NewExpr{\XINT_expr_redefineprints\xint_firstofone\xinttheiexpr }%
2375 \def\xintNewBoolExpr   {\XINT_NewExpr{\XINT_expr_redefineprints\xint_firstofone\xinttheboolexpr }%
```

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

```

2376 \def\XINT_NewFunc      {\XINT_NewExpr,{\xint_gobble_i\xintthebareeval }%
2377 \def\XINT_NewFloatFunc {\XINT_NewExpr,{\xint_gobble_i\xintthebarefloateval }%
2378 \def\XINT_NewIIFunc     {\XINT_NewExpr,{\xint_gobble_i\xintthebareiieval }%
2379 \def\XINT_newexpr_clean #1>{\noexpand\romannumeral`&&@}%
```

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`.

The #4 stands for the macro to be defined. Versions earlier than 1.2c had #1#2[#3], which was very bad for people applying LaTeX syntax `\xintNewExpr {\foo} [5]` for example as #2 would be `{\foo}<space>`. That didn't bother me much, but there is also the issue of `\2<space>`. Changed in 1.2d.

Up to and including 1.2c the definition was global. Starting with 1.2d it is done locally.

```

2380 \def\XINT_NewExpr #1#2#3#4#5#6[#7]%
2381 {%
2382 \begingroup
2383 \ifcase #7\relax
2384 \toks0 {\endgroup\def#5}%
2385 \or \toks0 {\endgroup\def#5##1#1}%
2386 \or \toks0 {\endgroup\def#5##1#1##2#1}%
2387 \or \toks0 {\endgroup\def#5##1#1##2#1##3#1}%
2388 \or \toks0 {\endgroup\def#5##1#1##2#1##3#1##4#1}%
2389 \or \toks0 {\endgroup\def#5##1#1##2#1##3#1##4#1##5#1}%
2390 \or \toks0 {\endgroup\def#5##1#1##2#1##3#1##4#1##5#1##6#1}%
2391 \or \toks0 {\endgroup\def#5##1#1##2#1##3#1##4#1##5#1##6#1##7#1}%
2392 \or \toks0 {\endgroup\def#5##1#1##2#1##3#1##4#1##5#1##6#1##7#1##8#1}%
2393 \or \toks0 {\endgroup\def#5##1#1##2#1##3#1##4#1##5#1##6#1##7#1##8#1##9#1}%
2394 \fi
2395 \xintexprSafeCatcodes
2396 \XINT_expr_redefinmacros
```

```
2397      #2%
2398      \XINT_NewExpr_a #3#4#5%
2399 }%
```

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.

```
2400 \catcode`~ 13 \catcode`@ 14 \catcode`\% 6 \catcode`# 12 \catcode`$ 11 @ $
2401 \def\XINT_NewExpr_a %1%2%3%4@
2402 {@
2403   \def\XINT_tmpa %1%2%3%4%5%6%7%8%9{%4}@
2404   \def~{$noexpand$}@
2405   \catcode` : 11 \catcode`_ 11
2406   \catcode`# 12 \catcode`~ 13 \escapechar 126
2407   \endlinechar -1 \everyeof {\noexpand }@
2408   \edef\XINT_tmpb
2409   {\scantokens\expandafter
2410    {\romannumeral`&&@\expandafter%2\XINT_tmpa {#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}\relax}@
2411   }@
2412   \escapechar 92 \catcode`# 6 \catcode`$ 0 @ $
2413   \edef\XINT_tmpa %1%2%3%4%5%6%7%8%9@
2414   {\scantokens\expandafter{\expandafter\XINT_newexpr_clean\meaning\XINT_tmpb}}@
2415   \the\toks0\expandafter{\XINT_tmpa{%1}{%2}{%3}{%4}{%5}{%6}{%7}{%8}{%9}}@
2416   %1{\ifxintverbose
2417     \xintMessage{xintexpr}{Info}@
2418     {\string%3\space now with meaning \meaning%3}@
2419   \fi}@
2420 }@
2421 \catcode`% 14
2422 \let\xintexprRestoreCatcodes\empty
2423 \def\xintexprSafeCatcodes
2424 {%
2425   \edef\xintexprRestoreCatcodes {%
2426     \catcode59=\the\catcode59 % ;
2427     \catcode34=\the\catcode34 % "
2428     \catcode63=\the\catcode63 % ?
2429     \catcode124=\the\catcode124 % |
2430     \catcode38=\the\catcode38 % &
2431     \catcode33=\the\catcode33 % !
2432     \catcode93=\the\catcode93 % ]
2433     \catcode91=\the\catcode91 % [
2434     \catcode94=\the\catcode94 % ^
2435     \catcode95=\the\catcode95 % _
2436     \catcode47=\the\catcode47 % /
2437     \catcode41=\the\catcode41 % )
2438     \catcode40=\the\catcode40 % (
2439     \catcode42=\the\catcode42 % *
2440     \catcode43=\the\catcode43 % +
2441     \catcode62=\the\catcode62 % >
```

```

2442      \catcode60=\the\catcode60 % <
2443      \catcode58=\the\catcode58 % :
2444      \catcode46=\the\catcode46 % .
2445      \catcode45=\the\catcode45 % -
2446      \catcode44=\the\catcode44 % ,
2447      \catcode61=\the\catcode61 % =
2448      \catcode32=\the\catcode32\relax % space
2449  }%
2450      \catcode59=12 % ;
2451      \catcode34=12 % "
2452      \catcode63=12 % ?
2453      \catcode124=12 % |
2454      \catcode38=4 % &
2455      \catcode33=12 % !
2456      \catcode93=12 % ]
2457      \catcode91=12 % [
2458      \catcode94=7 % ^
2459      \catcode95=8 % _
2460      \catcode47=12 % /
2461      \catcode41=12 % )
2462      \catcode40=12 % (
2463      \catcode42=12 % *
2464      \catcode43=12 % +
2465      \catcode62=12 % >
2466      \catcode60=12 % <
2467      \catcode58=12 % :
2468      \catcode46=12 % .
2469      \catcode45=12 % -
2470      \catcode44=12 % ,
2471      \catcode61=12 % =
2472      \catcode32=10 % space
2473 }%
2474 \let\XINT_tmpa\relax \let\XINT_tmpb\relax \let\XINT_tmppc\relax
2475 \XINT_restorecatcodes_endinput%

xintkernel: 285. Total number of code lines: 12758. Among those, release 1.2
xinttools:1160. has about 3000 lines starting with either {% or }%.
xintcore:2073. Each package starts with circa 50 lines dealing with cat-
xint:1427. codes, package identification and reloading management,
xintbinhex: 631. also for Plain TeX. Version 1.2e of 2015/11/22.
xintgcd: 455.
xintfrac:2845.
xintseries: 386.
xintcfrc:1021.
xintexpr:2475.

```