

The `xint` source code

JEAN-FRANÇOIS BURNOL

jfbu (at) free (dot) fr

Package version: 1.2i (2016/12/13); documentation date: 2016/12/13.

From source file xint.dtx. Time-stamp: <13-12-2016 at 12:22:08 CET>.

Contents

1	Package <code>xintkernel</code> implementation	2
2	Package <code>xinttools</code> implementation	15
3	Package <code>xintcore</code> implementation	47
4	Package <code>xint</code> implementation	98
5	Package <code>xintbinhex</code> implementation	140
6	Package <code>xintgcd</code> implementation	154
7	Package <code>xintfrac</code> implementation	166
8	Package <code>xintseries</code> implementation	237
9	Package <code>xintcfrac</code> implementation	246
10	Package <code>xintexpr</code> implementation	268

This is 1.2i of 2016/12/13.

- Some portions of the code still date back to the initial release, and at that time I was learning my trade in expandable TeX macro programming. At some point in the future, I will have to re-examine the older parts of the code.
- Release 1.2i of 2016/12/13 has rewritten some legacy macros like `\xintDSR` or `\xintDecSplit` in the style of the techniques of 1.2. But this means also that they are now limited to about 22480 digits for the former and 19970 digits for the latter (this is with the input stack size at 5000 and the maximal expansion depth at 10000.) This is not really an issue from the point of view of calling macros (such as `\xintTrunc`, `\xintRound`), because they usually had since 1.2 their own limitation at about 19970 digits from other code parts (such as division.) The macro `\xintXTTrunc` (which is not f-expandable however) can produce tens of thousands of digits and it escapes these limitations. Old macros such as `\xintLength` are not limited either (incidentally it got a lifting in 1.2i.) The macros from `xinttools` (`\xintKeep`, `\xintTrim`, `\xintNthElt`) also are not limited (but slower.)
- Release 1.2 of 2015/10/10 has entirely rewritten the core arithmetic routines in `xintcore`. Many macros benefit indirectly from the faster core routines. The new model is yet to be extended to other portions of the code: for example the routines of `xintbinhex` could be made faster for very big inputs if they adopted some techniques from the implementation of the basic arithmetic routines. The parser of `xintexpr` is faster and does not have a limit at 5000 digits per number anymore.

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

Warning: pay attention when looking at the code to the catcode configuration as found in `\XINT\setcatcodes`. Additional temporary configuration is used at some locations. For example `!` is of catcode letter in `xintexpr` and there are locations with funny catcodes e.g. using some letters with the math shift catcode.

1 Package `xintkernel` implementation

.1	Catcodes, ε -TeX and reload detection . . .	2	.10	<code>\odef, \oodef, \fdef</code>	8
.2	Package identification	5	.11	<code>\xintReverseOrder</code>	8
.3	Constants	5	.12	<code>\xintLength</code>	9
.4	Token management utilities	6	.13	<code>\xintLastItem</code>	9
.5	“gob til” macros and UD style fork . . .	6	.14	<code>\xintLengthUpTo</code>	10
.6	<code>\xint_afterfi</code>	7	.15	<code>\xintreplicate</code>	11
.7	<code>\xint_bye, \xint_Bye</code>	7	.16	<code>\xintgobble</code>	12
.8	<code>\xintdothis, \xintorthat</code>	7	.17	<code>\xintMessage, \ifxintverbose</code>	14
.9	<code>\xint_zapspace</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.

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

1.1 Catcodes, ε -TeX and reload detection

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

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

Starting with version 1.06 of the package, also ``` must be catcode-protected, because we replace everywhere in the code the twice-expansion done with `\expandafter` by the systematic use of `\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 ε TeX3.

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 % :
```

1 Package *xintkernel* implementation

```

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%
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 % ,

```

1 Package *xintkernel* implementation

```

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

```

```
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 [2016/12/13 1.2i Paraphernalia for the xint packages (JFB)]%
```

1.3 Constants

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

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

```
153 \mathchardef\xint_c_x^iv 10000
```

1.4 Token management utilities

```
154 \def\xint_tmpa { }%
155 \ifx\xint_tmpa\space\else
156   \immediate\write-1{Package xintkernel Warning: ATTENTION!}%
157   \immediate\write-1{\string\space\xint_tmpa macro does not have its normal
158     meaning.}%
159   \immediate\write-1{\xint_tmpa\xint_tmpa\xint_tmpa\xint_tmpa
160     All kinds of catastrophes will ensue!!!!}%
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 \let\xint_gobble_\empty
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_gobble_thenstop #1{ }%
186 \long\def\xint_firstofone_thenstop #1{ #1}%
187 \long\def\xint_firstoftwo_thenstop #1#2{ #1}%
188 \long\def\xint_secondoftwo_thenstop #1#2{ #2}%
189 \long\def\xint_exchangetwo_keepbraces #1#2{{#2}{#1}}%
```

1.5 “gob til” macros and UD style fork

Some moved here from *xintcore* by release 1.2.

1.2i finally makes all gobbling macros long.

```
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 \long\def\xint_gob_til_zero #10{}%
194 \long\def\xint_gob_til_one #11{}%
195 \long\def\xint_gob_til_zeros_iii #1000{}%
196 \long\def\xint_gob_til_zeros_iv #10000{}%
```

```

197 \long\def\xint_gob_til_eightzeroes #1000000000{%
198 \long\def\xint_gob_til_exclam #1!{% catcode 12 exclam
199 \long\def\xint_gob_til_dot #1.{}%
200 \long\def\xint_gob_til_G #1G{%
201 \long\def\xint_gob_til_minus #1-{%
202 \long\def\xint_gob_til_relax #1\relax {%
203 \long\def\xint_UDzerominusfork #10-#2#3\krof {#2}%
204 \long\def\xint_UDzerofork #10#2#3\krof {#2}%
205 \long\def\xint_UDsignfork #1-#2#3\krof {#2}%
206 \long\def\xint_UDwfork #1\W#2#3\krof {#2}%
207 \long\def\xint_UDXINTWfork #1\XINT_W#2#3\krof {#2}%
208 \long\def\xint_UDzerosfork #100#2#3\krof {#2}%
209 \long\def\xint_UDonezerofork #110#2#3\krof {#2}%
210 \long\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`, `\xint_Bye`

`\xint_Bye` is new with 1.2i for `\xintDSRr` and `\xintRound`. Also `\xint_bye_thenstop`.

```

215 \long\def\xint_bye #1\xint_bye {%
216 \long\def\xint_Bye #1\xint_bye {%
217 \long\def\xint_bye_thenstop #1\xint_bye { }%

```

1.8 `\xintdothis`, `\xintorthat`

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

```

218 \long\def\xint_dothis #1#2\xint_orthat #3{\fi #1}% 1.1
219 \let\xint_orthat \xint_firstofone
220 \long\def\xintdothis #1#2\xintorthat #3{\fi #1}%
221 \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`.

1 Package *xintkernel* implementation

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

```
\the\numexpr \xint_zapspaces 1 2 \xint_gobble_i\relax
expands to 12, and not 12\relax. Imagine also:
```

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

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

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

Made long with 1.2i.

```
222 \long\def\xint_zapspaces #1 #2{#1#2\xint_zapspaces }% 1.1
223 \long\def\xint_zapspaces_o #1{\expandafter\xint_zapspaces#1 \xint_gobble_i}%
```

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

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

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

1.11 `\xintReverseOrder`

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

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

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

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



```
248 {%
249   \noexpand\expandafter\space\noexpand\xint_gob_til_xint_relax #1%
250 }%
```

1.12 \xintLength

\xintLength does not expand its argument. See \xintNthElt{0} from xinttools.sty which f-expands its argument.

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

1.2i has rewritten this venerable macro. New code is about 40% faster across all lengths.

```
251 \def\xintLength {\romannumeral0\xintlength }%
252 \long\edef\xintlength #1%
253 {%
254   \noexpand\expandafter\space
255   \noexpand\the\numexpr\xint_c_ix\noexpand\XINT_length_loop
256   #1\xint_relax\xint_relax\xint_relax\xint_relax
257   \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
258   \xint_c_i\xint_c_ii\xint_c_iii\xint_c_iv
259   \xint_c_v\xint_c_vi\xint_c_vii\xint_c_viii\xint_c_ix\noexpand\xint_bye
260   \relax
261 }%
262 \long\def\XINT_length_loop #1#2#3#4#5#6#7#8#9%
263 {%
264   \xint_gob_til_xint_relax #9\XINT_length_finish_a\xint_relax
265   +\xint_c_ix\XINT_length_loop
266 }%
267 \def\XINT_length_finish_a\xint_relax+\xint_c_ix\XINT_length_loop
268   #1#2#3#4#5#6#7#8#9%
269 {%
270   -#9\xint_bye
271 }%
```

1.13 \xintLastItem

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

```
272 \def\xintLastItem {\romannumeral0\xintlastitem }%
273 \long\def\xintlastitem #1%
274 {%
275   \XINT_last_loop {}. #1%
276   {\xint_relax\XINT_last_loop_enda}{\xint_relax\XINT_last_loop_endb}%
277   {\xint_relax\XINT_last_loop_endc}{\xint_relax\XINT_last_loop_endd}%
278   {\xint_relax\XINT_last_loop_ende}{\xint_relax\XINT_last_loop_endf}%
279   {\xint_relax\XINT_last_loop_endg}{\xint_relax\XINT_last_loop_endh}\xint_bye
280 }%
281 \long\def\XINT_last_loop #1.#2#3#4#5#6#7#8#9%
282 {%
283   \xint_gob_til_xint_relax #9%
284   {#8}{#7}{#6}{#5}{#4}{#3}{#2}{#1}\xint_relax
285   \XINT_last_loop {#9}.%
286 }%
287 \long\def\XINT_last_loop_enda #1#2\xint_bye{ #1}%
288 \long\def\XINT_last_loop_endb #1#2\xint_bye{ #1}%
289 \long\def\XINT_last_loop_endc #1#2\xint_bye{ #1}%
290 \long\def\XINT_last_loop_endd #1#2\xint_bye{ #1}%
291 \long\def\XINT_last_loop_ende #1#2\xint_bye{ #1}%
292 \long\def\XINT_last_loop_endf #1#2\xint_bye{ #1}%
293 \long\def\XINT_last_loop_endg #1#2\xint_bye{ #1}%
294 \long\def\XINT_last_loop_endh #1#2\xint_bye{ #1}%
295 }
```

```

288 \long\def\XINT_last_loop_endb #1#2#3\xint_bye{ #2}%
289 \long\def\XINT_last_loop_endc #1#2#3#4\xint_bye{ #3}%
290 \long\def\XINT_last_loop_endd #1#2#3#4#5\xint_bye{ #4}%
291 \long\def\XINT_last_loop_ende #1#2#3#4#5#6\xint_bye{ #5}%
292 \long\def\XINT_last_loop_endf #1#2#3#4#5#6#7\xint_bye{ #6}%
293 \long\def\XINT_last_loop_endg #1#2#3#4#5#6#7#8\xint_bye{ #7}%
294 \long\def\XINT_last_loop_endh #1#2#3#4#5#6#7#8#9\xint_bye{ #8}%

```

1.14 \xintLengthUpTo

1.2i for use by \xintKeep and \xintTrim.

\xintLengthUpTo{N}{L} produces -0 if length(L)>N, else it returns N-length(L). Hence subtracting it from N always computes min(N,length(L)).

```

295 \def\xintLengthUpTo {\romannumeral0\xintlengthupto}%
296 \long\def\xintlengthupto #1#2%
297 {%
298   \expandafter\XINT_lengthupto_loop
299   \the\numexpr#1.#2\xint_relax\xint_relax\xint_relax\xint_relax
300   \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
301 }%
302 \def\XINT_lengthupto_loop #1%
303 {%
304   \xint_UDsignfork #1\XINT_lengthupto_gt-{\XINT_lengthupto_loop_a#1}\krof
305 }%
306 \long\def\XINT_lengthupto_gt #1\xint_bye{-0}%
307 \long\def\XINT_lengthupto_loop_a #1.#2#3#4#5#6#7#8#9%
308 {%
309   \xint_gob_til_xint_relax #9\XINT_lengthupto_finish_a\xint_relax
310   \expandafter\XINT_lengthupto_loop\the\numexpr #1-\xint_c_viii.%
311 }%
312 \def\XINT_lengthupto_finish_a\xint_relax\expandafter\XINT_lengthupto_loop
313 \the\numexpr #1-\xint_c_viii.#2\xint_bye
314 {%
315   \XINT_lengthupto_finish_b #2\W\W\W\W\W\W\W\Z {#1}%
316 }%
317 \def\XINT_lengthupto_finish_b #1#2#3#4#5#6#7#8\Z
318 {%
319   \xint_gob_til_W
320   #1\XINT_lengthupto_finish_c \xint_c_
321   #2\XINT_lengthupto_finish_c \xint_c_i
322   #3\XINT_lengthupto_finish_c \xint_c_ii
323   #4\XINT_lengthupto_finish_c \xint_c_iii
324   #5\XINT_lengthupto_finish_c \xint_c_iv
325   #6\XINT_lengthupto_finish_c \xint_c_v
326   #7\XINT_lengthupto_finish_c \xint_c_vi
327   \W\XINT_lengthupto_finish_c \xint_c_vii\Z
328 }%
329 \def\XINT_lengthupto_finish_c #1#2\Z #3%
330 {\ifnum #1>#3 \xint_dothis{-0}\fi
331   \xint_orthat{\expandafter\space\the\numexpr#3-#1\relax}}%

```

1.15 `\xintreplicate`

Added with 1.2i. This is cloned from `\prg_replicate:nn` from `expl3`, see Joseph's post at <http://tex.stackexchange.com/questions/16189/repeat-command-n-times>

I posted there an alternative not using the chained `\csname`'s but it is a bit less efficient (except perhaps for thousands of repetitions). Notice that the code in Joseph's post is inefficient when input `#1` is negative. The variant here filters it out upfront, at the time when `#1=0` is also handled.

Expansion must be triggered by a `\romannumeral`.

```

332 \def\xintreplicate#1%
333   {\expandafter\XINT_replicate\the\numexpr#1\endcsname}%
334 \def\XINT_replicate #1{\xint_UDsignfork
335   #1\XINT_rep_neg
336   -\XINT_rep
337   \krof #1}%
338 \long\def\XINT_rep_neg #1\endcsname #2{\xint_c_}%
339 \def\XINT_rep #1{\csname XINT_rep_f#1\XINT_rep_a}%
340 \def\XINT_rep_a #1{\csname XINT_rep_#1\XINT_rep_a}%
341 \def\XINT_rep_\XINT_rep_a{\endcsname}%
342 \long\expandafter\def\csname XINT_rep_0\endcsname #1%
343   {\endcsname{#1#1#1#1#1#1#1#1#1#1}}%
344 \long\expandafter\def\csname XINT_rep_1\endcsname #1%
345   {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1}%
346 \long\expandafter\def\csname XINT_rep_2\endcsname #1%
347   {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1}%
348 \long\expandafter\def\csname XINT_rep_3\endcsname #1%
349   {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1}%
350 \long\expandafter\def\csname XINT_rep_4\endcsname #1%
351   {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1}%
352 \long\expandafter\def\csname XINT_rep_5\endcsname #1%
353   {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1#1}%
354 \long\expandafter\def\csname XINT_rep_6\endcsname #1%
355   {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1#1#1}%
356 \long\expandafter\def\csname XINT_rep_7\endcsname #1%
357   {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1#1#1#1}%
358 \long\expandafter\def\csname XINT_rep_8\endcsname #1%
359   {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1#1#1#1#1}%
360 \long\expandafter\def\csname XINT_rep_9\endcsname #1%
361   {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1#1#1#1#1#1}%
362 \long\expandafter\def\csname XINT_rep_f0\endcsname #1%
363   {\xint_c_}%
364 \long\expandafter\def\csname XINT_rep_f1\endcsname #1%
365   {\xint_c_ #1}%
366 \long\expandafter\def\csname XINT_rep_f2\endcsname #1%
367   {\xint_c_ #1#1}%
368 \long\expandafter\def\csname XINT_rep_f3\endcsname #1%
369   {\xint_c_ #1#1#1}%
370 \long\expandafter\def\csname XINT_rep_f4\endcsname #1%
371   {\xint_c_ #1#1#1#1}%
372 \long\expandafter\def\csname XINT_rep_f5\endcsname #1%
373   {\xint_c_ #1#1#1#1#1}%
374 \long\expandafter\def\csname XINT_rep_f6\endcsname #1%

```

```

375     {\xint_c_ #1#1#1#1#1}%
376 \long\expandafter\def\csname XINT_rep_f7\endcsname #1%
377     {\xint_c_ #1#1#1#1#1#1}%
378 \long\expandafter\def\csname XINT_rep_f8\endcsname #1%
379     {\xint_c_ #1#1#1#1#1#1#1}%
380 \long\expandafter\def\csname XINT_rep_f9\endcsname #1%
381     {\xint_c_ #1#1#1#1#1#1#1#1}%

```

1.16 \xintgobble

Added with 1.2i. I hesitated about allowing as many as $9^6-1=531440$ tokens to gobble, but $9^5-1=59058$ is too low for playing with long decimal expansions.

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

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

```

382 \def\xintgobble #1%
383     {\csname xint_c_\expandafter\XINT_gobble_a\the\numexpr#1.0}%
384 \def\XINT_gobble #1.{\csname xint_c_\XINT_gobble_a #1.0}%
385 \def\XINT_gobble_a #1{\xint_gob_til_zero#1\XINT_gobble_d0\XINT_gobble_b#1}%
386 \def\XINT_gobble_b #1.#2%
387     {\expandafter\XINT_gobble_c
388         \the\numexpr (#1+\xint_c_v)/\xint_c_ix-\xint_c_i\expandafter.%
389         \the\numexpr #2+\xint_c_i.#1.}%
390 \def\XINT_gobble_c #1.#2.#3.%
391     {\csname XINT_g#2\the\numexpr#3-\xint_c_ix*#1\relax\XINT_gobble_a #1.#2}%
392 \def\XINT_gobble_d0\XINT_gobble_b0.#1{\endcsname}%
393 \expandafter\let\csname XINT_g10\endcsname\endcsname
394 \long\expandafter\def\csname XINT_g11\endcsname#1{\endcsname}%
395 \long\expandafter\def\csname XINT_g12\endcsname#1#2{\endcsname}%
396 \long\expandafter\def\csname XINT_g13\endcsname#1#2#3{\endcsname}%
397 \long\expandafter\def\csname XINT_g14\endcsname#1#2#3#4{\endcsname}%
398 \long\expandafter\def\csname XINT_g15\endcsname#1#2#3#4#5{\endcsname}%
399 \long\expandafter\def\csname XINT_g16\endcsname#1#2#3#4#5#6{\endcsname}%
400 \long\expandafter\def\csname XINT_g17\endcsname#1#2#3#4#5#6#7{\endcsname}%
401 \long\expandafter\def\csname XINT_g18\endcsname#1#2#3#4#5#6#7#8{\endcsname}%
402 \expandafter\let\csname XINT_g20\endcsname\endcsname
403 \long\expandafter\def\csname XINT_g21\endcsname #1#2#3#4#5#6#7#8#9%
404     {\endcsname}%
405 \long\expandafter\edef\csname XINT_g22\endcsname #1#2#3#4#5#6#7#8#9%
406     {\expandafter\noexpand\csname XINT_g21\endcsname}%
407 \long\expandafter\edef\csname XINT_g23\endcsname #1#2#3#4#5#6#7#8#9%
408     {\expandafter\noexpand\csname XINT_g22\endcsname}%
409 \long\expandafter\edef\csname XINT_g24\endcsname #1#2#3#4#5#6#7#8#9%
410     {\expandafter\noexpand\csname XINT_g23\endcsname}%
411 \long\expandafter\edef\csname XINT_g25\endcsname #1#2#3#4#5#6#7#8#9%
412     {\expandafter\noexpand\csname XINT_g24\endcsname}%
413 \long\expandafter\edef\csname XINT_g26\endcsname #1#2#3#4#5#6#7#8#9%
414     {\expandafter\noexpand\csname XINT_g25\endcsname}%
415 \long\expandafter\edef\csname XINT_g27\endcsname #1#2#3#4#5#6#7#8#9%
416     {\expandafter\noexpand\csname XINT_g26\endcsname}%
417 \long\expandafter\edef\csname XINT_g28\endcsname #1#2#3#4#5#6#7#8#9%
418     {\expandafter\noexpand\csname XINT_g27\endcsname}%

```

1 Package *xintkernel* implementation

```
419 \expandafter\let\csname XINT_g30\endcsname\endcsname
420 \long\expandafter\edef\csname XINT_g31\endcsname #1#2#3#4#5#6#7#8#9%
421 {\expandafter\noexpand\csname XINT_g28\endcsname}%
422 \long\expandafter\edef\csname XINT_g32\endcsname #1#2#3#4#5#6#7#8#9%
423 {\noexpand\csname XINT_g31\expandafter\noexpand\csname XINT_g28\endcsname}%
424 \long\expandafter\edef\csname XINT_g33\endcsname #1#2#3#4#5#6#7#8#9%
425 {\noexpand\csname XINT_g32\expandafter\noexpand\csname XINT_g28\endcsname}%
426 \long\expandafter\edef\csname XINT_g34\endcsname #1#2#3#4#5#6#7#8#9%
427 {\noexpand\csname XINT_g33\expandafter\noexpand\csname XINT_g28\endcsname}%
428 \long\expandafter\edef\csname XINT_g35\endcsname #1#2#3#4#5#6#7#8#9%
429 {\noexpand\csname XINT_g34\expandafter\noexpand\csname XINT_g28\endcsname}%
430 \long\expandafter\edef\csname XINT_g36\endcsname #1#2#3#4#5#6#7#8#9%
431 {\noexpand\csname XINT_g35\expandafter\noexpand\csname XINT_g28\endcsname}%
432 \long\expandafter\edef\csname XINT_g37\endcsname #1#2#3#4#5#6#7#8#9%
433 {\noexpand\csname XINT_g36\expandafter\noexpand\csname XINT_g28\endcsname}%
434 \long\expandafter\edef\csname XINT_g38\endcsname #1#2#3#4#5#6#7#8#9%
435 {\noexpand\csname XINT_g37\expandafter\noexpand\csname XINT_g28\endcsname}%
436 \expandafter\let\csname XINT_g40\endcsname\endcsname
437 \expandafter\edef\csname XINT_g41\endcsname
438 {\noexpand\csname XINT_g38\expandafter\noexpand\csname XINT_g31\endcsname}%
439 \expandafter\edef\csname XINT_g42\endcsname
440 {\noexpand\csname XINT_g41\expandafter\noexpand\csname XINT_g41\endcsname}%
441 \expandafter\edef\csname XINT_g43\endcsname
442 {\noexpand\csname XINT_g42\expandafter\noexpand\csname XINT_g41\endcsname}%
443 \expandafter\edef\csname XINT_g44\endcsname
444 {\noexpand\csname XINT_g43\expandafter\noexpand\csname XINT_g41\endcsname}%
445 \expandafter\edef\csname XINT_g45\endcsname
446 {\noexpand\csname XINT_g44\expandafter\noexpand\csname XINT_g41\endcsname}%
447 \expandafter\edef\csname XINT_g46\endcsname
448 {\noexpand\csname XINT_g45\expandafter\noexpand\csname XINT_g41\endcsname}%
449 \expandafter\edef\csname XINT_g47\endcsname
450 {\noexpand\csname XINT_g46\expandafter\noexpand\csname XINT_g41\endcsname}%
451 \expandafter\edef\csname XINT_g48\endcsname
452 {\noexpand\csname XINT_g47\expandafter\noexpand\csname XINT_g41\endcsname}%
453 \expandafter\let\csname XINT_g50\endcsname\endcsname
454 \expandafter\edef\csname XINT_g51\endcsname
455 {\noexpand\csname XINT_g48\expandafter\noexpand\csname XINT_g41\endcsname}%
456 \expandafter\edef\csname XINT_g52\endcsname
457 {\noexpand\csname XINT_g51\expandafter\noexpand\csname XINT_g51\endcsname}%
458 \expandafter\edef\csname XINT_g53\endcsname
459 {\noexpand\csname XINT_g52\expandafter\noexpand\csname XINT_g51\endcsname}%
460 \expandafter\edef\csname XINT_g54\endcsname
461 {\noexpand\csname XINT_g53\expandafter\noexpand\csname XINT_g51\endcsname}%
462 \expandafter\edef\csname XINT_g55\endcsname
463 {\noexpand\csname XINT_g54\expandafter\noexpand\csname XINT_g51\endcsname}%
464 \expandafter\edef\csname XINT_g56\endcsname
465 {\noexpand\csname XINT_g55\expandafter\noexpand\csname XINT_g51\endcsname}%
466 \expandafter\edef\csname XINT_g57\endcsname
467 {\noexpand\csname XINT_g56\expandafter\noexpand\csname XINT_g51\endcsname}%
468 \expandafter\edef\csname XINT_g58\endcsname
469 {\noexpand\csname XINT_g57\expandafter\noexpand\csname XINT_g51\endcsname}%
470 \expandafter\let\csname XINT_g60\endcsname\endcsname
```

1 Package *xintkernel* implementation

```
471 \expandafter\edef\csname XINT_g61\endcsname
472 {\noexpand\csname XINT_g58\expandafter\noexpand\csname XINT_g51\endcsname}%
473 \expandafter\edef\csname XINT_g62\endcsname
474 {\noexpand\csname XINT_g61\expandafter\noexpand\csname XINT_g61\endcsname}%
475 \expandafter\edef\csname XINT_g63\endcsname
476 {\noexpand\csname XINT_g62\expandafter\noexpand\csname XINT_g61\endcsname}%
477 \expandafter\edef\csname XINT_g64\endcsname
478 {\noexpand\csname XINT_g63\expandafter\noexpand\csname XINT_g61\endcsname}%
479 \expandafter\edef\csname XINT_g65\endcsname
480 {\noexpand\csname XINT_g64\expandafter\noexpand\csname XINT_g61\endcsname}%
481 \expandafter\edef\csname XINT_g66\endcsname
482 {\noexpand\csname XINT_g65\expandafter\noexpand\csname XINT_g61\endcsname}%
483 \expandafter\edef\csname XINT_g67\endcsname
484 {\noexpand\csname XINT_g66\expandafter\noexpand\csname XINT_g61\endcsname}%
485 \expandafter\edef\csname XINT_g68\endcsname
486 {\noexpand\csname XINT_g67\expandafter\noexpand\csname XINT_g61\endcsname}%
```

1.17 *\xintMessage*, *\ifxintverbose*

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

```
487 \def\xintMessage #1#2#3{%
488     \immediate\write128{Package #1 #2: (on line \the\inputlineno)}%
489     \immediate\write128{\space\space\space\space#3}%
490 }%
491 \newif\ifxintverbose
492 \XINT_restorecatcodes_endinput%
```

2 Package `xinttools` implementation

.1	Catcodes, ε -TeX and reload detection . . .	15	.20	<code>\xintilooop</code> , <code>\xintilooopindex</code> , <code>\xintouter-</code>	
.2	Package identification	16		<code>ilooopindex</code> , <code>\xintbreakilooop</code> , <code>\xint-</code>	
.3	<code>\xintgodef</code> , <code>\xintgoodef</code> , <code>\xintgfdef</code> . . .	16		<code>breakilooopanddo</code> , <code>\xintilooopskiptonext</code> ,	
.4	<code>\xintRevWithBraces</code>	16		<code>\xintilooopskipandredo</code>	31
.5	<code>\xintZapFirstSpaces</code>	17	.21	<code>\XINT_xflet</code>	31
.6	<code>\xintZapLastSpaces</code>	18	.22	<code>\xintApplyInline</code>	32
.7	<code>\xintZapSpaces</code>	19	.23	<code>\xintFor</code> , <code>\xintFor*</code> , <code>\xintBreakFor</code> ,	
.8	<code>\xintZapSpacesB</code>	19		<code>\xintBreakForAndDo</code>	33
.9	<code>\xintCSVtoList</code> , <code>\xintCSVtoListNon-</code>		.24	<code>\XINT_forever</code> , <code>\xintintegers</code> , <code>\xintdi-</code>	
	<code>Stripped</code>	19		<code>mensions</code> , <code>\xintrationals</code>	35
.10	<code>\xintListWithSep</code>	21	.25	<code>\xintForpair</code> , <code>\xintForthree</code> , <code>\xintFor-</code>	
.11	<code>\xintNthElt</code>	21		<code>four</code>	37
.12	<code>\xintKeep</code>	22	.26	<code>\xintAssign</code> , <code>\xintAssignArray</code> , <code>\xint-</code>	
.13	<code>\xintKeepUnbraced</code>	24		<code>DigitsOf</code>	38
.14	<code>\xintTrim</code>	25	.27	CSV (non user documented) variants of Trim,	
.15	<code>\xintTrimUnbraced</code>	26		Keep, NthElt, Reverse, Length	41
.16	<code>\xintApply</code>	27	.27.1	<code>\xintTrim:f:csv</code>	41
.17	<code>\xintApplyUnbraced</code>	28	.27.2	<code>\xintKeep:f:csv</code>	42
.18	<code>\xintSeq</code>	28	.27.3	<code>\xintNthEltPy:f:csv</code>	44
.19	<code>\xintloop</code> , <code>\xintbreakloop</code> , <code>\xintbreak-</code>		.27.4	<code>\xintReverse:f:csv</code>	45
	<code>loopanddo</code> , <code>\xintloopskiptonext</code>	30	.27.5	<code>\xintLength:f:csv</code>	46
			.27.6	Public names for the undocumented csv macros	46

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 [2016/12/13 1.2i 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}%

```


2 Package *xinttools* implementation

```
57 \romannumeral`&&@#1\xint_relax\xint_relax\xint_relax\xint_relax
58 \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
59 }%
60 \long\def\xintrevwithbracesnoexpand #1%
61 {%
62 \XINT_revwbr_loop }%
63 #1\xint_relax\xint_relax\xint_relax\xint_relax
64 \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
65 }%
66 \long\def\XINT_revwbr_loop #1#2#3#4#5#6#7#8#9%
67 {%
68 \xint_gob_til_xint_relax #9\XINT_revwbr_finish_a\xint_relax
69 \XINT_revwbr_loop {{#9}{#8}{#7}{#6}{#5}{#4}{#3}{#2}{#1}}%
70 }%
71 \long\def\XINT_revwbr_finish_a\xint_relax\XINT_revwbr_loop #1#2\xint_bye
72 {%
73 \XINT_revwbr_finish_b #2\R\R\R\R\R\R\R\R\Z #1%
74 }%
75 \def\XINT_revwbr_finish_b #1#2#3#4#5#6#7#8\Z
76 {%
77 \xint_gob_til_R
78 #1\XINT_revwbr_finish_c \xint_gobble_viii
79 #2\XINT_revwbr_finish_c \xint_gobble_vii
80 #3\XINT_revwbr_finish_c \xint_gobble_vi
81 #4\XINT_revwbr_finish_c \xint_gobble_v
82 #5\XINT_revwbr_finish_c \xint_gobble_iv
83 #6\XINT_revwbr_finish_c \xint_gobble_iii
84 #7\XINT_revwbr_finish_c \xint_gobble_ii
85 \R\XINT_revwbr_finish_c \xint_gobble_i\Z
86 }%
```

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

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

2.5 \xintZapFirstSpaces

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

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

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

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

If the original #1 started with a space, the grabbed #1 is empty. Thus `_again?` will see `#1=\xint_bye`, and hand over control to `_again` which will loop back into `\XINT_zapbsp_a`, with one initial space less. If the original #1 did not start with a space, or was empty, then the #1 below will be a

<sptoken>, then an extract of the original #1, not empty and not starting with a space, which contains what was up to the first <sp><sp> present in original #1, or, if none preexisted, <sptoken> and all of #1 (possibly empty) plus an ending \xint_relax. The added initial space will stop later the \romannumeral0. No brace stripping is possible. Control is handed over to \XINT_zapbsp_b which strips out the ending \xint_relax<sp><sp>\xint_relax

```

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

```

2.6 \xintZapLastSpaces

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

```

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

```

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

```

99 \long\edef\xintzaplastspaces #1{\noexpand\xINT_zapbsp_a {} \noexpand\empty#1%
100 \space\space\noexpand\xint_bye\xint_relax}%

```

The \empty from \xintzaplastspaces is to prevent brace removal in the #2 below. The \expandafter chain removes it.

```

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

```

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

```

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

```

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

```

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

```

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

```

106 \long\def\xINT_zapbsp_end\xINT_zapbsp_e #1#2\xint_relax { #1}%

```

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

```

107 \long\edef\xINT_zapbsp_e #1{\noexpand \XINT_zapbsp_a {#1\space\space}}%

```

2.7 `\xintZapSpaces`

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

```

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

```

2.8 `\xintZapSpacesB`

1.09f, written [2013/11/01]. Strips up to one pair of braces (but then does not strip spaces inside).

```

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

```

2.9 `\xintCSVtoList`, `\xintCSVtoListNonStripped`

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

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

```

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

```

2 Package *xinttools* implementation

```

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

```

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

```

164 \def\xINT_csvtol_finish_b #1,#2,#3,#4,#5,#6,#7,#8\Z
165 {%
166   \xint_gob_til_R
167   #1\expandafter\xINT_csvtol_finish_dviii\xint_gob_til_Z
168   #2\expandafter\xINT_csvtol_finish_dvii \xint_gob_til_Z
169   #3\expandafter\xINT_csvtol_finish_dvi \xint_gob_til_Z
170   #4\expandafter\xINT_csvtol_finish_dv \xint_gob_til_Z
171   #5\expandafter\xINT_csvtol_finish_div \xint_gob_til_Z
172   #6\expandafter\xINT_csvtol_finish_diii \xint_gob_til_Z
173   #7\expandafter\xINT_csvtol_finish_dii \xint_gob_til_Z
174   \R\xINT_csvtol_finish_di \Z
175 }%
176 \long\def\xINT_csvtol_finish_dviii #1#2#3#4#5#6#7#8#9{ #9}%
177 \long\def\xINT_csvtol_finish_dvii #1#2#3#4#5#6#7#8#9{ #9{#1}}%
178 \long\def\xINT_csvtol_finish_dvi #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}}%
179 \long\def\xINT_csvtol_finish_dv #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}}%
180 \long\def\xINT_csvtol_finish_div #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}{#4}}%
181 \long\def\xINT_csvtol_finish_diii #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}{#4}{#5}}%
182 \long\def\xINT_csvtol_finish_dii #1#2#3#4#5#6#7#8#9%
183   { #9{#1}{#2}{#3}{#4}{#5}{#6}}%
184 \long\def\xINT_csvtol_finish_di\Z #1#2#3#4#5#6#7#8#9%
185   { #9{#1}{#2}{#3}{#4}{#5}{#6}{#7}}%

```

2.10 `\xintListWithSep`

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

```

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

```

2.11 `\xintNthElt`

First included in release 1.06.

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

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

Negative values return the *|i|*th element from the end.

When *i* is out of range, an empty value is returned.

```

205 \def\xintNthElt {\romannumeral0\xintnthelt}%
206 \def\xintNthEltNoExpand {\romannumeral0\xintntheltnoexpand}%
207 \long\def\xintnthelt #1#2{\expandafter\XINT_nthelt_a\the\numexpr #1\expandafter.%
208     \expandafter{\romannumeral`&&@#2}}%
209 \def\xintntheltnoexpand #1{\expandafter\XINT_nthelt_a\the\numexpr #1.}%
210 \def\XINT_nthelt_a #1%
211 {%
212     \xint_UDzerominusfork
213     #1-\XINT_nthelt_zero
214     0#1\XINT_nthelt_neg
215     0-{\XINT_nthelt_pos #1}%
216     \krof
217}%
218 \def\XINT_nthelt_zero .{\xintlength}%
219 \long\def\XINT_nthelt_neg #1.#2%

```

```

220 {%
221   \expandafter\XINT_nthelt_neg_a\the\numexpr\xint_c_x\XINT_length_loop
222   #2\xint_relax\xint_relax\xint_relax\xint_relax
223   \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
224   \xint_c_i\xint_c_ii\xint_c_iii\xint_c_iv
225   \xint_c_v\xint_c_vi\xint_c_vii\xint_c_viii\xint_c_ix\xint_bye
226   -#1.#2\xint_bye
227 }%
228 \def\XINT_nthelt_neg_a #1%
229 {%
230   \xint_UDzerominusfork
231   #1-\xint_byeandstop
232   0#1\xint_byeandstop
233   0-{}%
234   \krof
235   \expandafter\XINT_nthelt_neg_b
236   \romannumeral\expandafter\XINT_gobble\the\numexpr-\xint_c_i+#1%
237 }%
238 \long\def\XINT_nthelt_neg_b #1#2\xint_bye{ #1}%
239 \long\def\XINT_nthelt_pos #1.#2%
240 {%
241   \expandafter\XINT_nthelt_loop\the\numexpr\xint_c_ix-#1.%
242   #2\xint_relax\xint_relax\xint_relax\xint_relax
243   \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
244 }%
245 \def\XINT_nthelt_loop #1%
246 {%
247   \xint_UDsignfork
248   #1\XINT_nthelt_loop_a
249   -{\XINT_nthelt_getit #1}%
250   \krof
251 }%
252 \long\def\XINT_nthelt_loop_a #1.#2#3#4#5#6#7#8#9%
253 {%
254   \xint_gob_til_xint_relax #9\xint_byeandstop\xint_relax
255   \expandafter\XINT_nthelt_loop\the\numexpr\xint_c_viii-#1.%
256 }%
257 \def\XINT_nthelt_getit #1.%
258 {%
259   \expandafter\expandafter\expandafter\XINT_nthelt_finish
260   \csname xint_gobble_\romannumeral\numexpr\xint_c_viii-#1\endcsname
261 }%
262 \long\edef\XINT_nthelt_finish #1#2\xint_bye
263 {\noexpand\xint_gob_til_xint_relax #1\noexpand\expandafter\space
264 \noexpand\xint_gobble_ii\xint_relax\space #1}%

```

2.12 \xintKeep

First included in release 1.09m.

`\xintKeep{i}{L}` f-expands its second argument `L`. It then grabs the first `i` items from `L` and discards the rest.

2 Package *xinttools* implementation

ATTENTION: **each such kept item is returned inside a brace pair** Use `\xintKeepUnbraced` to avoid that.

For i equal or larger to the number N of items in (expanded) L , the full L is returned (with braced items). For $i=0$, the macro returns an empty output. For $i<0$, the macro discards the first $N-|i|$ items. No brace pairs added to the remaining items. For i is less or equal to $-N$, the full L is returned (with no braces added.)

`\xintKeepNoExpand` does not expand the L argument.

Prior to 1.2i the code proceeded along a loop with no pre-computation of the length of L , for the $i>0$ case. The faster 1.2i version takes advantage of novel `\xintLengthUpTo` from `xintkernel.sty`.

```

265 \def\xintKeep          {\romannumeral0\xintkeep }%
266 \def\xintKeepNoExpand {\romannumeral0\xintkeepnoexpand }%
267 \long\def\xintkeep #1#2{\expandafter\XINT_keep_a\the\numexpr #1\expandafter.%
268                      \expandafter{\romannumeral`&&@#2}}%
269 \def\xintkeepnoexpand #1{\expandafter\XINT_keep_a\the\numexpr #1.%}
270 \def\XINT_keep_a #1%
271 {%
272   \xint_UDzerominusfork
273     #1-\XINT_keep_keepnone
274     0#1\XINT_keep_neg
275     0-{\XINT_keep_pos #1}%
276   \krof
277 }%
278 \long\def\XINT_keep_keepnone .#1{ }%
279 \long\def\XINT_keep_neg #1.#2%
280 {%
281   \expandafter\XINT_keep_neg_a\the\numexpr
282   #1-\numexpr\xint_c_ix\XINT_length_loop
283   #2\xint_relax\xint_relax\xint_relax\xint_relax
284   \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
285   \xint_c_i\xint_c_ii\xint_c_iii\xint_c_iv
286   \xint_c_v\xint_c_vi\xint_c_vii\xint_c_viii\xint_c_ix\xint_bye.#2%
287 }%
288 \def\XINT_keep_neg_a #1%
289 {%
290   \xint_UDsignfork
291     #1{\expandafter\space\romannumeral\XINT_gobble}%
292     -\XINT_keep_keepall
293   \krof
294 }%
295 \def\XINT_keep_keepall #1.{ }%
296 \long\def\XINT_keep_pos #1.#2%
297 {%
298   \expandafter\XINT_keep_loop
299   \the\numexpr#1-\XINT_lengthupto_loop
300   #1.#2\xint_relax\xint_relax\xint_relax\xint_relax
301   \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
302   -\xint_c_viii.{}#2\xint_bye%
303 }%
304 \def\XINT_keep_loop #1#2.%
305 {%
306   \xint_gob_til_minus#1\XINT_keep_loop_end-%
307   \expandafter\XINT_keep_loop

```

```

308 \the\numexpr#1#2-\xint_c_viii\expandafter.\XINT_keep_loop_pickeight
309 }%
310 \long\def\XINT_keep_loop_pickeight
311   #1#2#3#4#5#6#7#8#9{{#1{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}}}%
312 \def\XINT_keep_loop_end-\expandafter\XINT_keep_loop
313   \the\numexpr-#1-\xint_c_viii\expandafter.\XINT_keep_loop_pickeight
314   {\csname XINT_keep_end#1\endcsname}%
315 \long\expandafter\def\csname XINT_keep_end1\endcsname
316   #1#2#3#4#5#6#7#8#9\xint_bye { #1{#2}{#3}{#4}{#5}{#6}{#7}{#8}}%
317 \long\expandafter\def\csname XINT_keep_end2\endcsname
318   #1#2#3#4#5#6#7#8\xint_bye { #1{#2}{#3}{#4}{#5}{#6}{#7}}%
319 \long\expandafter\def\csname XINT_keep_end3\endcsname
320   #1#2#3#4#5#6#7\xint_bye { #1{#2}{#3}{#4}{#5}{#6}}%
321 \long\expandafter\def\csname XINT_keep_end4\endcsname
322   #1#2#3#4#5#6\xint_bye { #1{#2}{#3}{#4}{#5}}%
323 \long\expandafter\def\csname XINT_keep_end5\endcsname
324   #1#2#3#4#5\xint_bye { #1{#2}{#3}{#4}}%
325 \long\expandafter\def\csname XINT_keep_end6\endcsname
326   #1#2#3#4\xint_bye { #1{#2}{#3}}%
327 \long\expandafter\def\csname XINT_keep_end7\endcsname
328   #1#2#3\xint_bye { #1{#2}}%
329 \long\expandafter\def\csname XINT_keep_end8\endcsname
330   #1#2\xint_bye { #1}%

```

2.13 \xintKeepUnbraced

1.2a. Same as `\xintKeep` but will *not* add (or maintain) brace pairs around the kept items when `length(L)>i>0`.

The name may cause a mis-understanding: for $i < 0$, (i.e. keeping only trailing items), there is no brace removal at all happening.

Modified for 1.2i like `\xintKeep`.

```

331 \def\xintKeepUnbraced      {\romannumeral0\xintkeepunbraced }%
332 \def\xintKeepUnbracedNoExpand {\romannumeral0\xintkeepunbracednoexpand }%
333 \long\def\xintkeepunbraced #1#2%
334   {\expandafter\XINT_keepunbr_a\the\numexpr #1\expandafter.%
335     \expandafter{\romannumeral`&&@#2}}%
336 \def\xintkeepunbracednoexpand #1%
337   {\expandafter\XINT_keepunbr_a\the\numexpr #1.}%
338 \def\XINT_keepunbr_a #1%
339 {%
340   \xint_UDzerominusfork
341   #1-\XINT_keep_keepnone
342   0#1\XINT_keep_neg
343   0-{\XINT_keepunbr_pos #1}%
344   \krof
345 }%
346 \long\def\XINT_keepunbr_pos #1.#2%
347 {%
348   \expandafter\XINT_keepunbr_loop
349   \the\numexpr#1-\XINT_lengthupto_loop
350   #1.#2\xint_relax\xint_relax\xint_relax\xint_relax
351   \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye

```



```

352   -\xint_c_viii.{\#2\xint_bye%
353 }%
354 \def\xINT_keepunbr_loop #1#2.%
355 {%
356   \xint_gob_til_minus#1\xINT_keepunbr_loop_end-%
357   \expandafter\xINT_keepunbr_loop
358   \the\numexpr#1#2-\xint_c_viii\expandafter.\xINT_keepunbr_loop_pickeight
359 }%
360 \long\def\xINT_keepunbr_loop_pickeight
361   #1#2#3#4#5#6#7#8#9{{\#1#2#3#4#5#6#7#8#9}}%
362 \def\xINT_keepunbr_loop_end-\expandafter\xINT_keepunbr_loop
363   \the\numexpr-#1-\xint_c_viii\expandafter.\xINT_keepunbr_loop_pickeight
364   {\csname XINT_keepunbr_end#1\endcsname}%
365 \long\expandafter\def\csname XINT_keepunbr_end1\endcsname
366   #1#2#3#4#5#6#7#8#9\xint_bye { #1#2#3#4#5#6#7#8}%
367 \long\expandafter\def\csname XINT_keepunbr_end2\endcsname
368   #1#2#3#4#5#6#7#8\xint_bye { #1#2#3#4#5#6#7}%
369 \long\expandafter\def\csname XINT_keepunbr_end3\endcsname
370   #1#2#3#4#5#6#7\xint_bye { #1#2#3#4#5#6}%
371 \long\expandafter\def\csname XINT_keepunbr_end4\endcsname
372   #1#2#3#4#5#6\xint_bye { #1#2#3#4#5}%
373 \long\expandafter\def\csname XINT_keepunbr_end5\endcsname
374   #1#2#3#4#5\xint_bye { #1#2#3#4}%
375 \long\expandafter\def\csname XINT_keepunbr_end6\endcsname
376   #1#2#3#4\xint_bye { #1#2#3}%
377 \long\expandafter\def\csname XINT_keepunbr_end7\endcsname
378   #1#2#3\xint_bye { #1#2}%
379 \long\expandafter\def\csname XINT_keepunbr_end8\endcsname
380   #1#2\xint_bye { #1}%

```

2.14 *\xintTrim*

First included in release 1.09m.

\xintTrim{i}{L} f-expands its second argument L. It then removes the first i items from L and keeps the rest. For i equal or larger to the number N of items in (expanded) L, the macro returns an empty output. For i=0, the original (expanded) L is returned. For i<0, the macro proceeds from the tail. It thus removes the last |i| items, i.e. it keeps the first N-|i| items. For |i|>= N, the empty list is returned.

\xintTrimNoExpand does not expand the L argument.

Speed improvements for 1.2i mainly for i<0 branch (which hands over to *\xintKeep*).

```

381 \def\xintTrim          {\romannumeral0\xinttrim }%
382 \def\xintTrimNoExpand {\romannumeral0\xinttrimnoexpand }%
383 \long\def\xinttrim #1#2{\expandafter\xINT_trim_a\the\numexpr #1\expandafter.%
384   \expandafter{\romannumeral`&&@#2}}%
385 \def\xinttrimnoexpand #1{\expandafter\xINT_trim_a\the\numexpr #1.}%
386 \def\xINT_trim_a #1%
387 {%
388   \xint_UDzerominusfork
389   #1-\xINT_trim_trimnone
390   0#1\xINT_trim_neg
391   0-{\xINT_trim_pos #1}%

```

```

392 \krof
393 }%
394 \long\def\XINT_trim_trimnone .#1{ #1}%
395 \long\def\XINT_trim_neg #1.#2%
396 {%
397 \expandafter\XINT_trim_neg_a\the\numexpr
398 #1-\numexpr\xint_c_ix\XINT_length_loop
399 #2\xint_relax\xint_relax\xint_relax\xint_relax
400 \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
401 \xint_c_i\xint_c_ii\xint_c_iii\xint_c_iv
402 \xint_c_v\xint_c_vi\xint_c_vii\xint_c_viii\xint_c_ix\xint_bye
403 .{ }#2\xint_bye
404 }%
405 \def\XINT_trim_neg_a #1%
406 {%
407 \xint_UDsignfork
408 #1{\expandafter\XINT_keep_loop\the\numexpr-\xint_c_viii+}%
409 -\XINT_trim_trimall
410 \krof
411 }%
412 \edef\XINT_trim_trimall {\noexpand\expandafter\space\noexpand\xint_bye}%

```

This branch doesn't pre-evaluate the length of the list argument.

```

413 \long\def\XINT_trim_pos #1.#2%
414 {%
415 \expandafter\XINT_trim_loop\the\numexpr#1-\xint_c_viii.%
416 #2\xint_relax\xint_relax\xint_relax\xint_relax
417 \xint_relax\xint_relax\xint_relax\xint_relax\xint_bye
418 }%
419 \def\XINT_trim_loop #1%
420 {%
421 \xint_UDsignfork
422 #1\XINT_trim_finish
423 -\XINT_trim_loop_a
424 \krof #1%
425 }%
426 \long\def\XINT_trim_loop_a #1.#2#3#4#5#6#7#8#9%
427 {%
428 \xint_gob_til_xint_relax #9\xint_bye_thenstop\xint_relax
429 \expandafter\XINT_trim_loop\the\numexpr#1-\xint_c_viii.%
430 }%
431 \def\XINT_trim_finish -#1.%
432 {%
433 \expandafter\expandafter\expandafter\XINT_trim_finish_a
434 \expandafter\expandafter\expandafter\space
435 \csname xint_gobble_\romannumeral\numexpr\xint_c_viii-#1\endcsname
436 }%
437 \long\def\XINT_trim_finish_a #1\xint_relax #2\xint_bye {#1}%

```

2.15 `\xintTrimUnbraced`

1.2a. Modified in 1.2i like `\xintTrim`

2 Package *xinttools* implementation

```
438 \def\xintTrimUnbraced      {\romannumeral0\xinttrimunbraced}%
439 \def\xintTrimUnbracedNoExpand {\romannumeral0\xinttrimunbracednoexpand}%
440 \long\def\xinttrimunbraced #1#2%
441     {\expandafter\XINT_trimunbr_a\the\numexpr #1\expandafter.%
442         \expandafter{\romannumeral`&&#2}}%
443 \def\xinttrimunbracednoexpand #1%
444     {\expandafter\XINT_trimunbr_a\the\numexpr #1.%}%
445 \def\XINT_trimunbr_a #1%
446 {%
447     \xint_UDzerominusfork
448     #1-\XINT_trim_trimnone
449     0#1\XINT_trimunbr_neg
450     0-{\XINT_trim_pos #1}%
451     \krof
452}%
453 \long\def\XINT_trimunbr_neg #1.#2%
454 {%
455     \expandafter\XINT_trimunbr_neg_a\the\numexpr
456     #1-\numexpr\xint_c_ix\XINT_length_loop
457     #2\xint_relax\xint_relax\xint_relax\xint_relax
458     \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
459     \xint_c_i\xint_c_ii\xint_c_iii\xint_c_iv
460     \xint_c_v\xint_c_vi\xint_c_vii\xint_c_viii\xint_c_ix\xint_bye
461     .{\#2\xint_bye
462}%
463 \def\XINT_trimunbr_neg_a #1%
464 {%
465     \xint_UDsignfork
466     #1{\expandafter\XINT_keepunbr_loop\the\numexpr-\xint_c_viii+}%
467     -\XINT_trim_trimall
468     \krof
469}%
```

2.16 *\xintApply*

\xintApply $\{\text{\macro}\}\{a\}\{b\}...\{z\}$ returns $\{\text{\macro}\{a\}\}...\{\text{\macro}\{b\}\}$ 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.

```
470 \def\xintApply      {\romannumeral0\xintapply}%
471 \def\xintApplyNoExpand {\romannumeral0\xintapplynoexpand}%
472 \long\def\xintapply #1#2%
473 {%
474     \expandafter\XINT_apply\expandafter {\romannumeral`&&#2}%
475     {\#1}%
476}%
477 \long\def\XINT_apply #1#2{\XINT_apply_loop_a {}{\#2}\#1\xint_bye}%
478 \long\def\xintapplynoexpand #1#2{\XINT_apply_loop_a {}{\#1}\#2\xint_bye}%
479 \long\def\XINT_apply_loop_a #1#2#3%
480 {%
481     \xint_bye #3\XINT_apply_end\xint_bye
482     \expandafter
483     \XINT_apply_loop_b
```

```

484 \expandafter {\romannumeral`&&@#2{#3}}{#1}{#2}%
485 }%
486 \long\def\XINT_apply_loop_b #1#2{\XINT_apply_loop_a {#2{#1}}}%
487 \long\def\XINT_apply_end\xint_bye\expandafter\XINT_apply_loop_b
488 \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.

```

489 \def\xintApplyUnbraced {\romannumeral0\xintapplyunbraced }%
490 \def\xintApplyUnbracedNoExpand {\romannumeral0\xintapplyunbracednoexpand }%
491 \long\def\xintapplyunbraced #1#2%
492 {%
493 \expandafter\XINT_applyunbr\expandafter {\romannumeral`&&@#2}%
494 {#1}%
495 }%
496 \long\def\XINT_applyunbr #1#2{\XINT_applyunbr_loop_a {}{#2}#1\xint_bye }%
497 \long\def\xintapplyunbracednoexpand #1#2%
498 {\XINT_applyunbr_loop_a {}{#1}#2\xint_bye }%
499 \long\def\XINT_applyunbr_loop_a #1#2#3%
500 {%
501 \xint_bye #3\XINT_applyunbr_end\xint_bye
502 \expandafter\XINT_applyunbr_loop_b
503 \expandafter {\romannumeral`&&@#2{#3}}{#1}{#2}%
504 }%
505 \long\def\XINT_applyunbr_loop_b #1#2{\XINT_applyunbr_loop_a {#2#1}}%
506 \long\def\XINT_applyunbr_end\xint_bye\expandafter\XINT_applyunbr_loop_b
507 \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 generate thousands of terms then.

```

508 \def\xintSeq {\romannumeral0\xintseq }%
509 \def\xintseq #1{\XINT_seq_chkopt #1\xint_bye }%
510 \def\XINT_seq_chkopt #1%
511 {%
512 \ifx [#1\expandafter\XINT_seq_opt
513 \else\expandafter\XINT_seq_noopt
514 \fi #1%
515 }%
516 \def\XINT_seq_noopt #1\xint_bye #2%
517 {%
518 \expandafter\XINT_seq\expandafter
519 {\the\numexpr#1\expandafter}\expandafter{\the\numexpr #2}%
520 }%
521 \def\XINT_seq #1#2%
522 {%

```

2 Package *xinttools* implementation

```

523 \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
524 \expandafter\xint_firstoftwo_thenstop
525 \or
526 \expandafter\XINT_seq_p
527 \else
528 \expandafter\XINT_seq_n
529 \fi
530 {#2}{#1}%
531 }%
532 \def\XINT_seq_p #1#2%
533 {%
534 \ifnum #1>#2
535 \expandafter\expandafter\expandafter\XINT_seq_p
536 \else
537 \expandafter\XINT_seq_e
538 \fi
539 \expandafter{\the\numexpr #1-\xint_c_i}{#2}{#1}%
540 }%
541 \def\XINT_seq_n #1#2%
542 {%
543 \ifnum #1<#2
544 \expandafter\expandafter\expandafter\XINT_seq_n
545 \else
546 \expandafter\XINT_seq_e
547 \fi
548 \expandafter{\the\numexpr #1+\xint_c_i}{#2}{#1}%
549 }%
550 \def\XINT_seq_e #1#2#3{ }%
551 \def\XINT_seq_opt [\xint_bye #1]#2#3%
552 {%
553 \expandafter\XINT_seqo\expandafter
554 {\the\numexpr #2\expandafter}\expandafter
555 {\the\numexpr #3\expandafter}\expandafter
556 {\the\numexpr #1}%
557 }%
558 \def\XINT_seqo #1#2%
559 {%
560 \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
561 \expandafter\XINT_seqo_a
562 \or
563 \expandafter\XINT_seqo_pa
564 \else
565 \expandafter\XINT_seqo_na
566 \fi
567 {#1}{#2}%
568 }%
569 \def\XINT_seqo_a #1#2#3{ {#1}}%
570 \def\XINT_seqo_o #1#2#3#4{ #4}%
571 \def\XINT_seqo_pa #1#2#3%
572 {%
573 \ifcase\ifnum #3=\xint_c_ 0\else\ifnum #3>\xint_c_ 1\else -1\fi\fi\space
574 \expandafter\XINT_seqo_o

```

```

575 \or
576 \expandafter\XINT_seqo_pb
577 \else
578 \xint_afterfi{\expandafter\space\xint_gobble_iv}%
579 \fi
580 {#1}{#2}{#3}{{#1}}%
581 }%
582 \def\XINT_seqo_pb #1#2#3%
583 {%
584 \expandafter\XINT_seqo_pc\expandafter{\the\numexpr #1+#3}{#2}{#3}%
585 }%
586 \def\XINT_seqo_pc #1#2%
587 {%
588 \ifnum #1>#2
589 \expandafter\XINT_seqo_o
590 \else
591 \expandafter\XINT_seqo_pd
592 \fi
593 {#1}{#2}%
594 }%
595 \def\XINT_seqo_pd #1#2#3#4{\XINT_seqo_pb {#1}{#2}{#3}{#4{#1}}}%
596 \def\XINT_seqo_na #1#2#3%
597 {%
598 \ifcase\ifnum #3=\xint_c_ 0\else\ifnum #3>\xint_c_ 1\else -1\fi\fi\space
599 \expandafter\XINT_seqo_o
600 \or
601 \xint_afterfi{\expandafter\space\xint_gobble_iv}%
602 \else
603 \expandafter\XINT_seqo_nb
604 \fi
605 {#1}{#2}{#3}{{#1}}%
606 }%
607 \def\XINT_seqo_nb #1#2#3%
608 {%
609 \expandafter\XINT_seqo_nc\expandafter{\the\numexpr #1+#3}{#2}{#3}%
610 }%
611 \def\XINT_seqo_nc #1#2%
612 {%
613 \ifnum #1<#2
614 \expandafter\XINT_seqo_o
615 \else
616 \expandafter\XINT_seqo_nd
617 \fi
618 {#1}{#2}%
619 }%
620 \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.

```

621 \long\def\xintloop #1#2\repeat {#1#2\xintloop_again\fi\xint_gobble_i {#1#2}}%
622 \long\def\xintloop_again\fi\xint_gobble_i #1{\fi

```

```

623             #1\xintloop_again\fi\xint_gobble_i {#1}}%
624 \long\def\xintbreakloop #1\xintloop_again\fi\xint_gobble_i #2{%
625 \long\def\xintbreakloopanddo #1#2\xintloop_again\fi\xint_gobble_i #3{#1}%
626 \long\def\xintloopskiptonext #1\xintloop_again\fi\xint_gobble_i #2{%
627             #2\xintloop_again\fi\xint_gobble_i {#2}}%

```

2.20 \xintilooop, \xintilooopindex, \xintouterilooopindex, \xintbreakilooop, \xintbreakilooopanddo, \xintilooopskiptonext, \xintilooopskipandredo

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

```

628 \def\xintilooop [#1+#2]{%
629     \expandafter\xintilooop_a\the\numexpr #1\expandafter.\the\numexpr #2.%
630 \long\def\xintilooop_a #1.#2.#3#4\repeat{%
631     #3#4\xintilooop_again\fi\xint_gobble_iii {#1}{#2}{#3#4}}%
632 \def\xintilooop_again\fi\xint_gobble_iii #1#2{%
633     \fi\expandafter\xintilooop_again_b\the\numexpr#1+#2.#2.%
634 \long\def\xintilooop_again_b #1.#2.#3{%
635     #3\xintilooop_again\fi\xint_gobble_iii {#1}{#2}{#3}}%
636 \long\def\xintbreakilooop #1\xintilooop_again\fi\xint_gobble_iii #2#3#4{%
637 \long\def\xintbreakilooopanddo
638     #1.#2\xintilooop_again\fi\xint_gobble_iii #3#4#5{#1}%
639 \long\def\xintilooopindex #1\xintilooop_again\fi\xint_gobble_iii #2%
640     {#2#1\xintilooop_again\fi\xint_gobble_iii {#2}}%
641 \long\def\xintouterilooopindex #1\xintilooop_again
642     #2\xintilooop_again\fi\xint_gobble_iii #3%
643     {#3#1\xintilooop_again #2\xintilooop_again\fi\xint_gobble_iii {#3}}%
644 \long\def\xintilooopskiptonext #1\xintilooop_again\fi\xint_gobble_iii #2#3{%
645     \expandafter\xintilooop_again_b \the\numexpr#2+#3.#3.%
646 \long\def\xintilooopskipandredo #1\xintilooop_again\fi\xint_gobble_iii #2#3#4{%
647     #4\xintilooop_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.

```

648 \def\XINT_xflet #1%
649 {%
650     \def\XINT_xflet_macro {#1}\XINT_xflet_zapsp
651 }%
652 \def\XINT_xflet_zapsp
653 {%
654     \expandafter\futurelet\expandafter\XINT_token
655     \expandafter\XINT_xflet_sp?\romannumeral`&&@%
656 }%
657 \def\XINT_xflet_sp?
658 {%
659     \ifx\XINT_token\XINT_sptoken
660         \expandafter\XINT_xflet_zapsp
661     \else\expandafter\XINT_xflet_zapspB
662     \fi
663 }%

```

```

664 \def\XINT_xflet_zapspB
665 {%
666   \expandafter\futurelet\expandafter\XINT_tokenB
667   \expandafter\XINT_xflet_spB?\romannumeral`&&@%
668 }%
669 \def\XINT_xflet_spB?
670 {%
671   \ifx\XINT_tokenB\XINT_sptoken
672     \expandafter\XINT_xflet_zapspB
673   \else\expandafter\XINT_xflet_eq?
674   \fi
675 }%
676 \def\XINT_xflet_eq?
677 {%
678   \ifx\XINT_token\XINT_tokenB
679     \expandafter\XINT_xflet_macro
680   \else\expandafter\XINT_xflet_zapsp
681   \fi
682 }%

```

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.

```

683 \catcode`Z 3
684 \long\def\xintApplyInline #1#2%
685 {%
686   \long\expandafter\def\expandafter\XINT_inline_macro
687   \expandafter ##\expandafter 1\expandafter {#1{##1}}%
688   \XINT_xflet\XINT_inline_b #2Z% this Z has catcode 3
689 }%
690 \def\XINT_inline_b
691 {%
692   \ifx\XINT_token Z\expandafter\xint_gobble_i
693   \else\expandafter\XINT_inline_d\fi
694 }%
695 \long\def\XINT_inline_d #1%
696 {%
697   \long\def\XINT_item{#1}\XINT_xflet\XINT_inline_e
698 }%
699 \def\XINT_inline_e
700 {%
701   \ifx\XINT_token Z\expandafter\XINT_inline_w
702   \else\expandafter\XINT_inline_f\fi
703 }%
704 \def\XINT_inline_f
705 {%
706   \expandafter\XINT_inline_g\expandafter{\XINT_inline_macro {##1}}%
707 }%

```



```

708 \long\def\XINT_inline_g #1%
709 {%
710   \expandafter\XINT_inline_macro\XINT_item
711   \long\def\XINT_inline_macro ##1{##1}\XINT_inline_d
712 }%
713 \def\XINT_inline_w #1%
714 {%
715   \expandafter\XINT_inline_macro\XINT_item
716 }%

```

2.23 `\xintFor`, `\xintFor*`, `\xintBreakFor`, `\xintBreakForAndDo`

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

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

1.09e adds `\XINT_forever` with `\xintintegers`, `\xintdimensions`, `\xintrationals` and `\xintBreakFor`, `\xintBreakForAndDo`, `\xintifForFirst`, `\xintifForLast`. On this occasion `\xint_firstoftwo` and `\xint_secondoftwo` are made long.

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

1.09f rewrites (2013/11/03) the code which now allows all macro parameters from #1 to #9 in `\xintFor`, `\xintFor*`, and `\XINT_forever`. 1.2i: slightly more robust `\xintifForFirst/Last` in case of nesting.

```

717 \def\XINT_tmpa #1#2{\ifnum #2<#1 \xint_afterfi {{#####2}}\fi}%
718 \def\XINT_tmpb #1#2{\ifnum #1<#2 \xint_afterfi {{#####2}}\fi}%
719 \def\XINT_tmpc #1%
720 {%
721   \expandafter\edef \csname XINT_for_left#1\endcsname
722     {\xintApplyUnbraced {\XINT_tmpa #1}{123456789}}%
723   \expandafter\edef \csname XINT_for_right#1\endcsname
724     {\xintApplyUnbraced {\XINT_tmpb #1}{123456789}}%
725 }%
726 \xintApplyInline \XINT_tmpc {123456789}%
727 \long\def\xintBreakFor      #1Z{%}%
728 \long\def\xintBreakForAndDo #1#2Z{#1}%
729 \def\xintFor {\let\xintifForFirst\xint_firstoftwo
730               \let\xintifForLast\xint_secondoftwo
731               \futurelet\XINT_token\XINT_for_ifstar }%
732 \def\XINT_for_ifstar {\ifx\XINT_token*\expandafter\XINT_forx
733                      \else\expandafter\XINT_for \fi }%
734 \catcode`U 3 % with numexpr
735 \catcode`V 3 % with xintfrac.sty (xint.sty not enough)
736 \catcode`D 3 % with dimexpr
737 \def\XINT_flet_zapsp
738 {%
739   \futurelet\XINT_token\XINT_flet_sp?
740 }%

```

2 Package *xinttools* implementation

```

741 \def\XINT_flet_sp?
742 {%
743   \ifx\XINT_token\XINT_sptoken
744     \xint_afterfi{\expandafter\XINT_flet_zapsp\romannumeral0}%
745   \else\expandafter\XINT_flet_macro
746   \fi
747 }%
748 \long\def\XINT_for #1#2in#3#4#5%
749 {%
750   \expandafter\XINT_toks\expandafter
751   {\expandafter\XINT_for_d\the\numexpr #2\relax {#5}}%
752   \def\XINT_flet_macro {\expandafter\XINT_for_forever?\space}%
753   \expandafter\XINT_flet_zapsp #3Z%
754 }%
755 \def\XINT_for_forever? #1Z%
756 {%
757   \ifx\XINT_token U\XINT_to_forever\fi
758   \ifx\XINT_token V\XINT_to_forever\fi
759   \ifx\XINT_token D\XINT_to_forever\fi
760   \expandafter\the\expandafter\XINT_toks\romannumeral0\xintcsvtolist {#1}Z%
761 }%
762 \def\XINT_to_forever\fi #1\xintcsvtolist #2{\fi \XINT_forever #2}%
763 \long\def\XINT_forx *#1#2in#3#4#5%
764 {%
765   \expandafter\XINT_toks\expandafter
766   {\expandafter\XINT_forx_d\the\numexpr #2\relax {#5}}%
767   \XINT_xflet\XINT_forx_forever? #3Z%
768 }%
769 \def\XINT_forx_forever?
770 {%
771   \ifx\XINT_token U\XINT_to_forxever\fi
772   \ifx\XINT_token V\XINT_to_forxever\fi
773   \ifx\XINT_token D\XINT_to_forxever\fi
774   \XINT_forx_empty?
775 }%
776 \def\XINT_to_forxever\fi #1\XINT_forx_empty? {\fi \XINT_forever }%
777 \catcode`U 11
778 \catcode`D 11
779 \catcode`V 11
780 \def\XINT_forx_empty?
781 {%
782   \ifx\XINT_token Z\expandafter\xintBreakFor\fi
783   \the\XINT_toks
784 }%
785 \long\def\XINT_for_d #1#2#3%
786 {%
787   \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
788   \XINT_toks {{#3}}%
789   \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
790     \the\XINT_toks \csname XINT_for_right#1\endcsname }%
791   \XINT_toks {\XINT_x\let\xintifForFirst\xint_seconddoftwo
792     \let\xintifForLast\xint_seconddoftwo\XINT_for_d #1{#2}}%

```

```

793 \futurelet\XINT_token\XINT_for_last?
794 }%
795 \long\def\XINT_forx_d #1#2#3%
796 {%
797 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
798 \XINT_toks {{#3}}%
799 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
800 \the\XINT_toks \csname XINT_for_right#1\endcsname }%
801 \XINT_toks {\XINT_x\let\xintifForFirst\xint_secondoftwo
802 \let\xintifForLast\xint_secondoftwo\XINT_forx_d #1{#2}}%
803 \XINT_xflet\XINT_for_last?
804 }%
805 \def\XINT_for_last?
806 {%
807 \ifx\XINT_token Z\expandafter\XINT_for_last?yes\fi
808 \the\XINT_toks
809 }%
810 \def\XINT_for_last?yes
811 {%
812 \let\xintifForLast\xint_firstoftwo
813 \xintBreakForAndDo{\XINT_x\xint_gobble_i Z}%
814 }%

```

2.24 \XINT_forever, \xintintegers, \xintdimensions, \xintrationals

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

```

815 \catcode`U 3
816 \catcode`D 3
817 \catcode`V 3
818 \let\xintegers U%
819 \let\xintintegers U%
820 \let\xintdimensions D%
821 \let\xintrationals V%
822 \def\XINT_forever #1%
823 {%
824 \expandafter\XINT_forever_a
825 \csname XINT?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi a\expandafter\endcsname
826 \csname XINT?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi i\expandafter\endcsname
827 \csname XINT?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi \endcsname
828 }%
829 \catcode`U 11
830 \catcode`D 11
831 \catcode`V 11
832 \def\XINT?expr_Ua #1#2%
833 {\expandafter{\expandafter\numexpr\the\numexpr #1\expandafter\relax
834 \expandafter\relax\expandafter}%
835 \expandafter{\the\numexpr #2}}%

```

2 Package *xinttools* implementation

```

836 \def\XINT_?expr_Da #1#2%
837   {\expandafter{\expandafter\dimexpr\number\dimexpr #1\expandafter\relax
838     \expandafter s\expandafter p\expandafter\relax\expandafter}%
839   \expandafter{\number\dimexpr #2}}%
840 \catcode`Z 11
841 \def\XINT_?expr_Va #1#2%
842 {%
843   \expandafter\XINT_?expr_Vb\expandafter
844     {\romannumeral`&&\xintrowwithzeros{\xintZapSpacesB{#2}}}%
845     {\romannumeral`&&\xintrowwithzeros{\xintZapSpacesB{#1}}}%
846 }%
847 \catcode`Z 3
848 \def\XINT_?expr_Vb #1#2{\expandafter\XINT_?expr_Vc #2.#1}%
849 \def\XINT_?expr_Vc #1/#2.#3/#4.%
850 {%
851   \xintifEq {#2}{#4}%
852     {\XINT_?expr_Vf {#3}{#1}{#2}}%
853     {\expandafter\XINT_?expr_Vd\expandafter
854       {\romannumeral0\xintiimul {#2}{#4}}%
855       {\romannumeral0\xintiimul {#1}{#4}}%
856       {\romannumeral0\xintiimul {#2}{#3}}%
857     }%
858 }%
859 \def\XINT_?expr_Vd #1#2#3{\expandafter\XINT_?expr_Ve\expandafter {#2}{#3}{#1}}%
860 \def\XINT_?expr_Ve #1#2{\expandafter\XINT_?expr_Vf\expandafter {#2}{#1}}%
861 \def\XINT_?expr_Vf #1#2#3{{#2/#3}{0}{#1}{#2}{#3}}%
862 \def\XINT_?expr_Ui {{\numexpr 1\relax}{1}}%
863 \def\XINT_?expr_Di {{\dimexpr 0pt\relax}{65536}}%
864 \def\XINT_?expr_Vi {{1/1}{0111}}%
865 \def\XINT_?expr_U #1#2%
866   {\expandafter{\expandafter\numexpr\the\numexpr #1+#2\relax\relax}{#2}}%
867 \def\XINT_?expr_D #1#2%
868   {\expandafter{\expandafter\dimexpr\the\numexpr #1+#2\relax sp\relax}{#2}}%
869 \def\XINT_?expr_V #1#2{\XINT_?expr_Vx #2}%
870 \def\XINT_?expr_Vx #1#2%
871 {%
872   \expandafter\XINT_?expr_Vy\expandafter
873     {\romannumeral0\xintiiadd {#1}{#2}}{#2}%
874 }%
875 \def\XINT_?expr_Vy #1#2#3#4%
876 {%
877   \expandafter{\romannumeral0\xintiiadd {#3}{#1}/#4}{#1}{#2}{#3}{#4}}%
878 }%
879 \def\XINT_forever_a #1#2#3#4%
880 {%
881   \ifx #4[\expandafter\XINT_forever_opt_a
882     \else\expandafter\XINT_forever_b
883   \fi #1#2#3#4%
884 }%
885 \def\XINT_forever_b #1#2#3Z{\expandafter\XINT_forever_c\the\XINT_toks #2#3}%
886 \long\def\XINT_forever_c #1#2#3#4#5%
887   {\expandafter\XINT_forever_d\expandafter #2#4#5{#3}Z}%

```

```

888 \def\XINT_forever_opt_a #1#2#3[#4+#5]#6Z%
889 {%
890   \expandafter\expandafter\expandafter
891   \XINT_forever_opt_c\expandafter\the\expandafter\XINT_toks
892   \romannumeral`&&@#1{#4}{#5}#3%
893 }%
894 \long\def\XINT_forever_opt_c #1#2#3#4#5#6{\XINT_forever_d #2{#4}{#5}#6{#3}Z}%
895 \long\def\XINT_forever_d #1#2#3#4#5%
896 {%
897   \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#5}%
898   \XINT_toks {{#2}}%
899   \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
900     \the\XINT_toks \csname XINT_for_right#1\endcsname }%
901   \XINT_x
902   \let\xintifForFirst\xint_seconddoftwo
903   \let\xintifForLast\xint_seconddoftwo
904   \expandafter\XINT_forever_d\expandafter #1\romannumeral`&&@#4{#2}{#3}#4{#5}%
905 }%

```

2.25 \xintForpair, \xintForthree, \xintForfour

1.09c.

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

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

1.2i: slightly more robust \xintifForFirst/Last in case of nesting.

```

906 \catcode`j 3
907 \long\def\xintForpair #1#2#3in#4#5#6%
908 {%
909   \let\xintifForFirst\xint_firstoftwo
910   \let\xintifForLast\xint_seconddoftwo
911   \XINT_toks {\XINT_forpair_d #2{#6}}%
912   \expandafter\the\expandafter\XINT_toks #4jZ%
913 }%
914 \long\def\XINT_forpair_d #1#2#3(#4)#5%
915 {%
916   \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
917   \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
918   \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
919     \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_i\endcsname}%
920   \ifx #5j\expandafter\XINT_for_last?yes\fi
921   \XINT_x
922   \let\xintifForFirst\xint_seconddoftwo
923   \let\xintifForLast\xint_seconddoftwo
924   \XINT_forpair_d #1{#2}%
925 }%
926 \long\def\xintForthree #1#2#3in#4#5#6%
927 {%
928   \let\xintifForFirst\xint_firstoftwo
929   \let\xintifForLast\xint_seconddoftwo
930   \XINT_toks {\XINT_forthree_d #2{#6}}%

```

```

931 \expandafter\the\expandafter\XINT_toks #4jZ%
932 }%
933 \long\def\XINT_forthree_d #1#2#3(#4)#5%
934 {%
935 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
936 \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
937 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
938 \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_ii\endcsname}%
939 \ifx #5j\expandafter\XINT_for_last?yes\fi
940 \XINT_x
941 \let\xintifForFirst\xint_seconddoftwo
942 \let\xintifForLast\xint_seconddoftwo
943 \XINT_forthree_d #1{#2}%
944 }%
945 \long\def\xintForfour #1#2#3in#4#5#6%
946 {%
947 \let\xintifForFirst\xint_firstoftwo
948 \let\xintifForLast\xint_seconddoftwo
949 \XINT_toks {\XINT_forfour_d #2{#6}}%
950 \expandafter\the\expandafter\XINT_toks #4jZ%
951 }%
952 \long\def\XINT_forfour_d #1#2#3(#4)#5%
953 {%
954 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
955 \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
956 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
957 \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_iii\endcsname}%
958 \ifx #5j\expandafter\XINT_for_last?yes\fi
959 \XINT_x
960 \let\xintifForFirst\xint_seconddoftwo
961 \let\xintifForLast\xint_seconddoftwo
962 \XINT_forfour_d #1{#2}%
963 }%
964 \catcode`Z 11
965 \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.

```

966 \def\xintAssign{\def\XINT_flet_macro {\XINT_assign_fork}\XINT_flet_zapsp }%
967 \def\XINT_assign_fork
968 {%
969 \let\XINT_assign_def\def
970 \ifx\XINT_token[\expandafter\XINT_assign_opt
971 \else\expandafter\XINT_assign_a
972 \fi
973 }%
974 \def\XINT_assign_opt [#1]%

```

2 Package *xinttools* implementation

```

975 {%
976   \ifcsname #1def\endcsname
977     \expandafter\let\expandafter\XINT_assign_def \csname #1def\endcsname
978   \else
979     \expandafter\let\expandafter\XINT_assign_def \csname xint#1def\endcsname
980   \fi
981   \XINT_assign_a
982 }%
983 \long\def\XINT_assign_a #1\to
984 {%
985   \def\XINT_flet_macro{\XINT_assign_b}%
986   \expandafter\XINT_flet_zapsp\romannumeral`&&@#1\xint_relax\to
987 }%
988 \long\def\XINT_assign_b
989 {%
990   \ifx\XINT_token\bgroup
991     \expandafter\XINT_assign_c
992   \else\expandafter\XINT_assign_f
993   \fi
994 }%
995 \long\def\XINT_assign_f #1\xint_relax\to #2%
996 {%
997   \XINT_assign_def #2{#1}%
998 }%
999 \long\def\XINT_assign_c #1%
1000 {%
1001   \def\xint_temp {#1}%
1002   \ifx\xint_temp\xint_brelax
1003     \expandafter\XINT_assign_e
1004   \else
1005     \expandafter\XINT_assign_d
1006   \fi
1007 }%
1008 \long\def\XINT_assign_d #1\to #2%
1009 {%
1010   \expandafter\XINT_assign_def\expandafter #2\expandafter{\xint_temp}%
1011   \XINT_assign_c #1\to
1012 }%
1013 \def\XINT_assign_e #1\to {}%
1014 \def\xintRelaxArray #1%
1015 {%
1016   \edef\XINT_restoreescapechar {\escapechar\the\escapechar\relax}%
1017   \escapechar -1
1018   \expandafter\def\expandafter\xint_arrayname\expandafter {\string #1}%
1019   \XINT_restoreescapechar
1020   \xintilop [\csname\xint_arrayname 0\endcsname+-1]
1021   \global
1022   \expandafter\let\csname\xint_arrayname\xintilopindex\endcsname\relax
1023   \ifnum \xintilopindex > \xint_c_
1024   \repeat
1025   \global\expandafter\let\csname\xint_arrayname 00\endcsname\relax
1026   \global\let #1\relax

```

2 Package *xinttools* implementation

```
1027 }%
1028 \def\xintAssignArray{\def\XINT_flet_macro {\XINT_assignarray_fork}%
1029 \XINT_flet_zapsp }%
1030 \def\XINT_assignarray_fork
1031 {%
1032 \let\XINT_assignarray_def\def
1033 \ifx\XINT_token[\expandafter\XINT_assignarray_opt
1034 \else\expandafter\XINT_assignarray
1035 \fi
1036 }%
1037 \def\XINT_assignarray_opt [#1]%
1038 {%
1039 \ifcsname #1def\endcsname
1040 \expandafter\let\expandafter\XINT_assignarray_def \csname #1def\endcsname
1041 \else
1042 \expandafter\let\expandafter\XINT_assignarray_def
1043 \csname xint#1def\endcsname
1044 \fi
1045 \XINT_assignarray
1046 }%
1047 \long\def\XINT_assignarray #1\to #2%
1048 {%
1049 \edef\XINT_restoreescapechar {\escapechar\the\escapechar\relax }%
1050 \escapechar -1
1051 \expandafter\def\expandafter\xint_arrayname\expandafter {\string #2}%
1052 \XINT_restoreescapechar
1053 \def\xint_itemcount {0}%
1054 \expandafter\XINT_assignarray_loop \romannumeral`&&@#1\xint_relax
1055 \csname\xint_arrayname 00\expandafter\endcsname
1056 \csname\xint_arrayname 0\expandafter\endcsname
1057 \expandafter {\xint_arrayname}#2%
1058 }%
1059 \long\def\XINT_assignarray_loop #1%
1060 {%
1061 \def\xint_temp {#1}%
1062 \ifx\xint_brelax\xint_temp
1063 \expandafter\def\csname\xint_arrayname 0\expandafter\endcsname
1064 \expandafter{\the\numexpr\xint_itemcount}%
1065 \expandafter\expandafter\expandafter\XINT_assignarray_end
1066 \else
1067 \expandafter\def\expandafter\xint_itemcount\expandafter
1068 {\the\numexpr\xint_itemcount+\xint_c_i}%
1069 \expandafter\XINT_assignarray_def
1070 \csname\xint_arrayname\xint_itemcount\expandafter\endcsname
1071 \expandafter{\xint_temp }%
1072 \expandafter\XINT_assignarray_loop
1073 \fi
1074 }%
1075 \def\XINT_assignarray_end #1#2#3#4%
1076 {%
1077 \def #4##1%
1078 }
```



```

1079     \romannumeral0\expandafter #1\expandafter{\the\numexpr ##1}%
1080 }%
1081 \def #1##1%
1082 {%
1083     \ifnum ##1<\xint_c_
1084         \xint_afterfi {\xintError:ArrayIndexIsNegative\space }%
1085     \else
1086         \xint_afterfi {%
1087             \ifnum ##1>#2
1088                 \xint_afterfi {\xintError:ArrayIndexBeyondLimit\space }%
1089             \else\xint_afterfi
1090         {\expandafter\expandafter\expandafter\space\csname #3##1\endcsname}%
1091         \fi}%
1092     \fi
1093 }%
1094 }%
1095 \let\xintDigitsOf\xintAssignArray

```

2.27 CSV (non user documented) variants of Trim, Keep, NthElt, Reverse, Length

These routines are for use by `\xintListSel:f:csv` from *xintexpr*. We don't take any precaution here regarding brace stripping or spaces. And the macros are short. They will remain undocumented in the user manual for the time being for the reasons above and to preserve the possibility to modify the interface in case some evolution on the *xintexpr* side requires it.

Nevertheless, directly usable names are also provided.

All these macros are in the $O(N^2)$ style when they need to accumulate things. They handle comma separated values. There is no final comma, neither on input nor on output. Special care has been taken to handle safely the case of an empty list.

2.27.1 `\xintTrim:f:csv`

1.2g 2016/03/17.

```

1096 \def\xintTrim:f:csv {\romannumeral0\xinttrim:f:csv }%
1097 \def\xinttrim:f:csv #1#2%
1098 {%
1099     \expandafter\XINT_trim:f:csv_a\the\numexpr #1\expandafter.%
1100     \romannumeral`&&@#2\xint_relax,\xint_relax,\xint_relax,\xint_relax,%
1101         \xint_relax,\xint_relax,\xint_relax,\xint_relax,\xint_bye
1102 }%
1103 \def\XINT_trim:f:csv_a #1%
1104 {%
1105     \xint_UDzerominusfork
1106     #1-{\XINT_trim:f:csv_zero }%
1107     0#1{\XINT_trim:f:csv_neg }%
1108     0-{\XINT_trim:f:csv_loop_a #1}%
1109     \krof
1110 }%
1111 \def\XINT_trim:f:csv_zero .#1\xint_relax #2\xint_bye{ #1}%
1112 \def\XINT_trim:f:csv_neg #1.#2\xint_relax%
1113 {%
1114     \expandafter\XINT_trim:f:csv_neg_b

```

```

1115 \the\numexpr\XINT_length:f:csv_a #2%
1116 \xint_relax,\xint_relax,\xint_relax,\xint_relax,%
1117 \xint_relax,\xint_relax,\xint_relax,\xint_relax,\xint_bye
1118 -#1.#2\xint_relax
1119 }%
1120 \def\XINT_trim:f:csv_neg_b #1%
1121 {%
1122 \xint_UDzerominusfork
1123 #1-{\expandafter\space\xint_bye }%
1124 0#1{\expandafter\space\xint_bye }%
1125 0-{\XINT_keep:f:csv_pos #1}%
1126 \krof
1127 }%
1128 \def\XINT_trim:f:csv_loop_a #1.%
1129 {%
1130 \ifnum #1>\xint_c_vii
1131 \expandafter\XINT_trim:f:csv_loop_b
1132 \else
1133 \expandafter\XINT_trim:f:csv_finish
1134 \fi
1135 #1.%
1136 }%
1137 \def\XINT_trim:f:csv_loop_b #1.#2,#3,#4,#5,#6,#7,#8,#9,%
1138 {%
1139 \xint_gob_til_xint_relax #9\xint_bye_thenstop\xint_relax
1140 \expandafter\XINT_trim:f:csv_loop_a\the\numexpr #1-\xint_c_viii.%
1141 }%
1142 \def\XINT_trim:f:csv_finish #1.%
1143 {%
1144 \expandafter\XINT_trim:f:csv_finish_a
1145 \romannumeral`&&@\csname xint_gobble:csv_\romannumeral#1\endcsname
1146 }%
1147 \def\xint_gobble:csv_ {}%
1148 \def\xint_gobble:csv_i #1,{}%
1149 \def\xint_gobble:csv_ii #1,#2,{}%
1150 \def\xint_gobble:csv_iii #1,#2,#3,{}%
1151 \def\xint_gobble:csv_iv #1,#2,#3,#4,{}%
1152 \def\xint_gobble:csv_v #1,#2,#3,#4,#5,{}%
1153 \def\xint_gobble:csv_vi #1,#2,#3,#4,#5,#6,{}%
1154 \def\xint_gobble:csv_vii #1,#2,#3,#4,#5,#6,#7,{}%
1155 \def\XINT_trim:f:csv_finish_a #1\xint_relax #2\xint_bye { #1}%

```

2.27.2 *\xintKeep:f:csv*

1.2g 2016/03/17.

```

1156 \def\xintKeep:f:csv {\romannumeral0\xintkeep:f:csv }%
1157 \def\xintkeep:f:csv #1#2%
1158 {%
1159 \expandafter\XINT_keep:f:csv_a\the\numexpr #1\expandafter.%
1160 \romannumeral`&&@#2\xint_relax,\xint_relax,\xint_relax,\xint_relax,%
1161 \xint_relax,\xint_relax,\xint_relax,\xint_relax,\xint_bye
1162 }%

```

2 Package *xinttools* implementation

```
1163 \def\XINT_keep:f:csv_a #1%
1164 {%
1165   \xint_UDzerominusfork
1166     #1-\xint_bye_thenstop
1167     0#1\XINT_keep:f:csv_neg
1168     0-\XINT_keep:f:csv_pos #1}%
1169   \krof
1170 }%
1171 \def\XINT_keep:f:csv_neg #1.#2\xint_relax%
1172 {%
1173   \expandafter\XINT_keep:f:csv_neg_b
1174   \the\numexpr\XINT_length:f:csv_a #2%
1175     \xint_relax,\xint_relax,\xint_relax,\xint_relax,%
1176     \xint_relax,\xint_relax,\xint_relax,\xint_relax,\xint_bye
1177   -#1.#2\xint_relax
1178 }%
1179 \def\XINT_keep:f:csv_neg_b #1%
1180 {%
1181   \xint_UDzerominusfork
1182     #1-\XINT_keep:f:csv_all }%
1183     0#1\XINT_keep:f:csv_all }%
1184     0-\XINT_trim:f:csv_loop_a #1}%
1185   \krof
1186 }%
1187 \def\XINT_keep:f:csv_all #1.#2\xint_relax #3\xint_bye { #2}%
1188 \def\XINT_keep:f:csv_pos #1.{\expandafter\xint_gobble_thenstop
1189   \romannumeral`&&\XINT_keep:f:csv_loop_a #1.{}%
1190 \def\XINT_keep:f:csv_loop_a #1.%
1191 {%
1192   \ifnum #1>\xint_c_vi
1193     \expandafter\XINT_keep:f:csv_loop_b
1194   \else
1195     \expandafter\XINT_keep:f:csv_finish
1196   \fi
1197   #1.%
1198 }%
1199 \def\XINT_keep:f:csv_loop_b #1.#2#3,#4,#5,#6,#7,#8,#9,%
1200 {%
1201   \xint_gob_til_xint_relax #9\XINT_keep:f:csv_enda\xint_relax
1202   \expandafter\XINT_keep:f:csv_loop_c\the\numexpr #1-\xint_c_vii.%
1203   {,#3,#4,#5,#6,#7,#8,#9}{#2}%
1204 }%
1205 \def\XINT_keep:f:csv_loop_c #1.#2#3{\XINT_keep:f:csv_loop_a #1.{#3#2}}%
1206 \def\XINT_keep:f:csv_enda #1.#2#3#4\xint_bye
1207 {%
1208   \XINT_keep:f:csv_endb #2\xint_bye {#3}%
1209 }%
1210 \def\XINT_keep:f:csv_endb #1\xint_relax #2\xint_bye #3{ #3#1}%
1211 \def\XINT_keep:f:csv_finish #1.#2#3,#4,#5,#6,#7,#8,#9\xint_bye
1212 {%
1213   \csname XINT_keep:f:csv_finish_\romannumeral#1\endcsname
1214   ,#3,#4,#5,#6,#7,#8,\Z {#2}%

```

```

1215 }%
1216 \def\XINT_keep:f:csv_finish_ #1\Z #2{#2}%
1217 \def\XINT_keep:f:csv_finish_i ,#1,#2\Z
1218   {\XINT_keep:f:csv_finish_f ,#1\xint_relax\Z}%
1219 \def\XINT_keep:f:csv_finish_ii ,#1,#2,#3\Z
1220   {\XINT_keep:f:csv_finish_f ,#1,#2\xint_relax\Z}%
1221 \def\XINT_keep:f:csv_finish_iii ,#1,#2,#3,#4\Z
1222   {\XINT_keep:f:csv_finish_f ,#1,#2,#3\xint_relax\Z}%
1223 \def\XINT_keep:f:csv_finish_iv ,#1,#2,#3,#4,#5\Z
1224   {\XINT_keep:f:csv_finish_f ,#1,#2,#3,#4\xint_relax\Z}%
1225 \def\XINT_keep:f:csv_finish_v ,#1,#2,#3,#4,#5,#6\Z
1226   {\XINT_keep:f:csv_finish_f ,#1,#2,#3,#4,#5\xint_relax\Z}%
1227 \def\XINT_keep:f:csv_finish_vi ,#1,#2,#3,#4,#5,#6,#7\Z
1228   {\XINT_keep:f:csv_finish_f ,#1,#2,#3,#4,#5,#6\xint_relax\Z}%
1229 \def\XINT_keep:f:csv_finish_f #1\xint_relax #2\Z #3{#3#1}%

```

2.27.3 \xintNthEltPy:f:csv

1.2g. 2016/03/17. Counts like Python starting at zero.

```

1230 \def\xintNthEltPy:f:csv {\romannumeral0\xintntheltypy:f:csv}%
1231 \def\xintntheltypy:f:csv #1{\expandafter\XINT_ntheltypy:f:csv\the\numexpr #1.}%
1232 \def\XINT_ntheltypy:f:csv #1%
1233 {%
1234   \xint_UDsignfork
1235     #1{\XINT_nthelt:f:csv_neg}%
1236     -{\XINT_nthelt:f:csv_pos #1}%
1237   \krof
1238 }%
1239 \long\def\XINT_nthelt:f:csv_neg #1.#2%
1240 {%
1241   \expandafter\XINT_nthelt:f:csv_neg_a
1242   \the\numexpr\expandafter\XINT_length:f:csv_a\romannumeral`&&#2%
1243     \xint_relax,\xint_relax,\xint_relax,\xint_relax,%
1244     \xint_relax,\xint_relax,\xint_relax,\xint_relax,\xint_bye
1245   -#1+\xint_c_i\expandafter.\romannumeral`&&#2%
1246     \xint_relax,\xint_relax,\xint_relax,\xint_relax,%
1247     \xint_relax,\xint_relax,\xint_relax,\xint_relax,\xint_bye
1248 }%
1249 \def\XINT_nthelt:f:csv_neg_a #1%
1250 {%
1251   \xint_UDzerominusfork
1252     #1-\xint_bye_thenstop
1253     0#1\xint_bye_thenstop
1254     0-{\XINT_nthelt:f:csv_loop_a #1}%
1255   \krof
1256 }%
1257 \long\def\XINT_nthelt:f:csv_pos #1.#2%
1258 {%
1259   \expandafter\XINT_nthelt:f:csv_loop_a\the\numexpr #1+\xint_c_i\expandafter.%
1260   \romannumeral`&&#2%
1261   \xint_relax,\xint_relax,\xint_relax,\xint_relax,%
1262   \xint_relax,\xint_relax,\xint_relax,\xint_relax,\xint_bye

```

```

1263 }%
1264 \def\XINT_nthelt:f:csv_loop_a #1.%
1265 {%
1266   \ifnum #1>\xint_c_viii
1267     \expandafter\XINT_nthelt:f:csv_loop_b
1268   \else
1269     \expandafter\XINT_nthelt:f:csv_getit
1270   \fi #1.%
1271 }%
1272 \long\def\XINT_nthelt:f:csv_loop_b #1.#2,#3,#4,#5,#6,#7,#8,#9,%
1273 {%
1274   \xint_gob_til_xint_relax #9\XINT_nthelt:f:csv_none\xint_relax
1275   \expandafter\XINT_nthelt:f:csv_loop_a\the\numexpr #1-\xint_c_viii.%
1276 }%
1277 \def\XINT_nthelt:f:csv_getit #1.%
1278 {%
1279   \expandafter\XINT_nthelt:f:csv_finish
1280   \romannumeral`&&\csname xint_gobble:csv_\romannumeral
1281     \numexpr#1-\xint_c_i\endcsname
1282 }%
1283 \def\XINT_nthelt:f:csv_finish #1,#2\xint_bye
1284 {%
1285   \XINT_nthelt:f:csv_finishb #1\xint_relax\xint_bye
1286 }%
1287 \def\XINT_nthelt:f:csv_finishb #1\xint_relax #2\xint_bye { #1}%

```

2.27.4 \xintReverse:f:csv

1.2g. Contrarily to `\xintReverseOrder` from `xintkernel.sty`, this one expands its argument. Handles empty list too. 2016/03/17.

```

1288 \def\xintReverse:f:csv {\romannumeral0\xintreverse:f:csv }%
1289 \def\xintreverse:f:csv #1%
1290 {%
1291   \expandafter\XINT_reverse:f:csv_loop
1292   \expandafter{\expandafter}\romannumeral`&&@#1,%
1293   \xint_relax,%
1294   \xint_bye,\xint_bye,\xint_bye,\xint_bye,%
1295   \xint_bye,\xint_bye,\xint_bye,\xint_bye,%
1296   \xint_relax
1297 }%
1298 \def\XINT_reverse:f:csv_loop #1#2,#3,#4,#5,#6,#7,#8,#9,%
1299 {%
1300   \xint_bye #9\XINT_reverse:f:csv_cleanup\xint_bye
1301   \XINT_reverse:f:csv_loop {,#9,#8,#7,#6,#5,#4,#3,#2#1}%
1302 }%
1303 \def\XINT_reverse:f:csv_cleanup\xint_bye\XINT_reverse:f:csv_loop #1#2\xint_relax
1304 {%
1305   \XINT_reverse:f:csv_finish #1%
1306 }%
1307 \def\XINT_reverse:f:csv_finish #1\xint_relax,{ }%

```

2.27.5 \xintLength:f:csv

1.2g. Contrarily to ancestor \xintLength from xintkernel.sty, this one expands its argument.
2016/03/17.

```

1308 \def\xintLength:f:csv {\romannumeral0\xintlength:f:csv }%
1309 \def\xintlength:f:csv #1%
1310 {%
1311   \expandafter\XINT_length:f:csv_a
1312   \romannumeral`&&@#1\xint_relax,\xint_relax,\xint_relax,\xint_relax,%
1313   \xint_relax,\xint_relax,\xint_relax,\xint_relax,\xint_bye
1314 }%
1315 \def\XINT_length:f:csv_a #1%
1316 {%
1317   \xint_gob_til_xint_relax #1\XINT_length:f:csv_empty\xint_relax
1318   \XINT_length:f:csv_loop 0.#1%
1319 }%
1320 \def\XINT_length:f:csv_empty #1\xint_bye { 0}%
1321 \def\XINT_length:f:csv_loop #1.#2,#3,#4,#5,#6,#7,#8,#9,%
1322 {%
1323   \xint_gob_til_xint_relax #9\XINT_length:f:csv_finish_a\xint_relax
1324   \expandafter\XINT_length:f:csv_loop\the\numexpr #1+\xint_c_viii.%
1325 }%
1326 \def\XINT_length:f:csv_finish_a\xint_relax
1327   \expandafter\XINT_length:f:csv_loop\the\numexpr #1+\xint_c_viii.#2\xint_bye
1328 {%
1329   \XINT_length:f:csv_finish_b #2\W,\W,\W,\W,\W,\W,\W,\Z {#1}%
1330 }%
1331 \def\XINT_length:f:csv_finish_b #1,#2,#3,#4,#5,#6,#7,#8\Z
1332 {%
1333   \xint_gob_til_W
1334   #1\XINT_length:f:csv_finish_c \xint_c_i
1335   #2\XINT_length:f:csv_finish_c \xint_c_ii
1336   #3\XINT_length:f:csv_finish_c \xint_c_iii
1337   #4\XINT_length:f:csv_finish_c \xint_c_iv
1338   #5\XINT_length:f:csv_finish_c \xint_c_v
1339   #6\XINT_length:f:csv_finish_c \xint_c_vi
1340   #7\XINT_length:f:csv_finish_c \xint_c_vii
1341   \W\XINT_length:f:csv_finish_c \xint_c_viii\Z
1342 }%
1343 \edef\XINT_length:f:csv_finish_c #1#2\Z #3%
1344   {\noexpand\expandafter\space\noexpand\the\numexpr #3+#1\relax}%

```

2.27.6 Public names for the undocumented csv macros

1.2g. 2016/03/17.

```

1345 \let\xintCSVTrim      \xintTrim:f:csv
1346 \let\xintCSVKeep      \xintKeep:f:csv
1347 \let\xintCSVNthEltPy  \xintNthEltPy:f:csv
1348 \let\xintCSVReverse   \xintReverse:f:csv
1349 \let\xintCSVLength     \xintLength:f:csv
1350 \let\XINT_tmpa\relax \let\XINT_tmpp\relax \let\XINT_tmppc\relax
1351 \XINT_restorecatcodes_endinput%

```

3 Package `xintcore` implementation

.1	Catcodes, ε -TeX and reload detection . . .	47	.17	<code>\xintDSL</code>	59
.2	Package identification	48	.18	<code>\xintDSR</code>	59
.3	Counts for holding needed constants . . .	48	.19	<code>\xintDSRr</code>	59
.4	<code>\xintNum</code>	48	.20	Core arithmetic	60
.5	Zeroes	49	.21	<code>\xintiAdd</code> , <code>\xintiiAdd</code>	60
.6	Blocks of eight digits	50	.22	<code>\xintiSub</code> , <code>\xintiiSub</code>	63
.7	<code>\xintReverseDigits</code>	54	.23	<code>\xintiMul</code> , <code>\xintiiMul</code>	68
.8	<code>\xintSgn</code> , <code>\xintiiSgn</code> , <code>\XINT_Sgn</code> , <code>\XINT_cntSgn</code>	55	.24	<code>\xintiSqr</code> , <code>\xintiiSqr</code>	73
.9	<code>\xintiOpp</code> , <code>\xintiiOpp</code>	55	.25	<code>\xintiPow</code> , <code>\xintiiPow</code>	74
.10	<code>\xintiAbs</code> , <code>\xintiiAbs</code>	56	.26	<code>\xintiFac</code> , <code>\xintiiFac</code>	77
.11	<code>\xintFDg</code> , <code>\xintiiFDg</code>	56	.27	<code>\xintiDivision</code> , <code>\xintiQuo</code> , <code>\xintiRem</code> , <code>\xintiiDivision</code> , <code>\xintiiQuo</code> , <code>\xintiiRem</code>	79
.12	<code>\xintLDg</code> , <code>\xintiiLDg</code>	56	.28	<code>\xintiDivRound</code> , <code>\xintiiDivRound</code> . . .	95
.13	<code>\xintDouble</code>	57	.29	<code>\xintiDivTrunc</code> , <code>\xintiiDivTrunc</code> . . .	96
.14	<code>\xintHalf</code>	57	.30	<code>\xintiMod</code> , <code>\xintiiMod</code>	96
.15	<code>\xintInc</code>	58	.31	“Load <code>xintfrac</code> ” macros	97
.16	<code>\xintDec</code>	58			

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

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

The commenting continues (2016/12/13) to be very sparse: actually it got worse than ever with release 1.2. I will possibly add comments at a later date, but for the time being the new routines are not commented at all.

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

3.1 Catcodes, ε -TeX and reload detection

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

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

```

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

```

3 Package *xintcore* implementation

```

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

```

3.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintcore}%
46 [2016/12/13 1.2i Expandable arithmetic on big integers (JFB)]%

```

3.3 Counts for holding needed constants

```

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

```

3.4 *\xintNum*

For example *\xintNum {-----+-----00000000000003}*

```

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

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

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

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

```

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

```

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

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

3 Package *xintcore* implementation

```

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

```

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

```

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

```

3.6 Blocks of eight digits

Lingua of release 1.2.

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

```

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

```

This is used as

```

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

```

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

```

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

```

3 Package *xintcore* implementation

```

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

```

This is used typically as

```

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

```

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

```

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

```

This is used typically as

```

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

```

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

```

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

```

3 Package *xintcore* implementation

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

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

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

```

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

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

```

This is used as

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

```

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

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

```

This is used as

```
\romannumeral0\XINT_rev_nounsep {}<blocks 1<8d>!>\R!\R!\R!\R!\R!\R!\R!\W

```

3 Package *xintcore* implementation

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

```

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

```

This is used as

```

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

```

```

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

```

This is used as

```

\romannumeral0\XINT_unsep_cuzsmall {}<blocks of 1<8d>!>1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W

```

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

```

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

```

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

```

217 \def\XINT_div_unsepQ #11#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
218 {%
219   \xint_gob_til_R #9\XINT_div_unsepQ_end\R
220   \XINT_div_unsepQ {#1#2#3#4#5#6#7#8#9}%
221 }%
222 \def\XINT_div_unsepQ_end\R\XINT_div_unsepQ #1{\XINT_div_unsepQ_x #1}%
223 \def\XINT_div_unsepQ_x #1#2#3#4#5#6#7#8#9%
224 {%
225   \xint_gob_til_R #9\XINT_div_unsepQ_e \R
226   \xint_gob_til_eightzeroes #1#2#3#4#5#6#7#8\XINT_div_unsepQ_y 00000000%
227   \expandafter\XINT_div_unsepQ_done \the\numexpr #1#2#3#4#5#6#7#8.#9%

```

```

228 }%
229 \def\XINT_div_unsepQ_e\R\xint_gob_til_eightzeroes #1\XINT_div_unsepQ_y #2\W
230   {\the\numexpr #1\relax \Z}%
231 \def\XINT_div_unsepQ_y #1.#2\R #3\W{\XINT_cuz_small #2\Z}%
232 \def\XINT_div_unsepQ_done #1.#2\R #3\W { #1#2\Z}%

```

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

```

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

```

3.7 *\xintReverseDigits*

1.2.

```

240 \def\XINT_microrevsep #1#2#3#4#5#6#7#8%
241 {%
242   1#8#7#6#5#4#3#2#1\expandafter!\the\numexpr\XINT_microrevsep
243 }%
244 \def\XINT_microrevsep_end #1\W #2\expandafter #3\Z{#2!}%
245 \def\xintReverseDigits {\romannumeral0\xintreversedigits }%
246 \def\xintreversedigits #1{\expandafter\XINT_reversedigits\romannumeral`&&@#1\Z}%
247 \def\XINT_reversedigits #1%
248 {%
249   \xint_UDsignfork
250   #1{\expandafter-\romannumeral0\XINT_reversedigits_a}%
251   -{\XINT_reversedigits_a #1}%
252   \krof
253 }%
254 \def\XINT_reversedigits_a #1\Z
255 {%
256   \expandafter\XINT_revdigits_a\the\numexpr\expandafter\XINT_microrevsep
257   \romannumeral`&&@#1{\XINT_microrevsep_end\W}\XINT_microrevsep_end
258   \XINT_microrevsep_end\XINT_microrevsep_end
259   \XINT_microrevsep_end\XINT_microrevsep_end
260   \XINT_microrevsep_end\XINT_microrevsep_end\Z
261   1\Z!1\backslash R!1\backslash R!1\backslash R!1\backslash R!1\backslash R!1\backslash R!1\backslash W
262 }%
263 \def\XINT_revdigits_a {\XINT_revdigits_b {}}%
264 \def\XINT_revdigits_b #11#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
265 {%
266   \xint_gob_til_R #9\XINT_revdigits_end\R
267   \XINT_revdigits_b {#9#8#7#6#5#4#3#2#1}%
268 }%
269 \edef\XINT_revdigits_end\R\XINT_revdigits_b #1#2\W
270 {\noexpand\expandafter\space\noexpand\xint_gob_til_Z #1}%

```

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

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

```

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

```

3.9 `\xintiOpp`, `\xintiiOpp`

```

305 \def\xintiiOpp {\romannumeral0\xintiioff }%
306 \def\xintiioff #1%
307 {%
308   \expandafter\XINT_opp \romannumeral`&&@#1%
309 }%
310 \def\xintiOpp {\romannumeral0\xintiopp }%
311 \def\xintiopp #1%
312 {%
313   \expandafter\XINT_opp \romannumeral0\xintnum{#1}%
314 }%
315 \def\XINT_Opp #1{\romannumeral0\XINT_opp #1}%
316 \def\XINT_opp #1%

```

```

317 {%
318   \xint_UDzerominusfork
319   #1-{ 0}%      zero
320   0#1{ }%      negative
321   0-{ -#1}%    positive
322   \krof
323 }%

```

3.10 \xintiAbs, \xintiiAbs

```

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

```

3.11 \xintFDg, \xintiiFDg

```

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

```

3.12 \xintLDg, \xintiiLDg

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


```

360 \def\xintLDg   {\romannumeral0\xintldg }%
361 \def\xintlDg #1{\expandafter\XINT_ldg_fork\romannumeral0\xintnum{#1}%
362     \XINT_ldg_c{}{}{}{}{}{}{}{}\xint_bye\relax}%
363 \def\xintiILDg {\romannumeral0\xintiildg }%
364 \def\xintiilDg #1{\expandafter\XINT_ldg_fork\romannumeral`&&@#1%
365     \XINT_ldg_c{}{}{}{}{}{}{}{}\xint_bye\relax}%
366 \def\XINT_ldg_fork #1%
367 {%
368     \xint_UDsignfork
369         #1\XINT_ldg
370         -{\XINT_ldg#1}%
371     \krof
372 }%
373 \edef\XINT_ldg #1#2#3#4#5#6#7#8#9%
374     {\noexpand\expandafter\space
375     \noexpand\the\numexpr#9#8#7#6#5#4#3#2#1*\xint_c_+\noexpand\XINT_ldg_a#9}%
376 \def\XINT_ldg_a#1#2{\XINT_ldg_cbye#2\XINT_ldg_d#1\XINT_ldg_c\XINT_ldg_b#2}%
377 \def\XINT_ldg_b#1#2#3#4#5#6#7#8#9{#9#8#7#6#5#4#3#2#1*\xint_c_+\XINT_ldg_a#9}%
378 \def\XINT_ldg_c    #1#2\xint_bye{#1}%
379 \def\XINT_ldg_cbye #1\XINT_ldg_c{}%
380 \def\XINT_ldg_d#1#2\xint_bye{#1}%

```

3.13 \xintDouble

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

```

381 \def\xintDouble {\romannumeral0\xintdouble}%
382 \def\xintdouble #1{\expandafter\XINT_dbl\romannumeral`&&#1%
383         \xint_bye2345678\xint_bye*\xint_c_ii\relax}%
384 \def\XINT_dbl #1#2#3#4#5#6#7#8%
385     {\expandafter\space\the\numexpr#1#2#3#4#5#6#7#8\XINT_dbl_a}%
386 \def\XINT_dbl_a #1#2#3#4#5#6#7#8%
387     {\expandafter\XINT_dbl_e\the\numexpr #1#2#3#4#5#6#7#8\XINT_dbl_a}%
388 \def\XINT_dbl_e#1{* \xint_c_ii\if#13+ \xint_c_i\fi\relax}%

```

3.14 \xintHalf

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

```

389 \def\xintHalf {\romannumeral0\xinthal}%
390 \def\xinthal #1{\expandafter\XINT_half_fork\romannumeral`&&@#1%
391   \xint_bye\xint_Bye345678\xint_bye
392   *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax}%
393 \def\XINT_half_fork #1%
394 {%
395   \xint_UDsignfork
396   #1\XINT_half_neg
397   -{\XINT_half #1}%
398   \krof
399 }%
400 \def\XINT_half_neg{\xintiopp\XINT_half}%
401 \def\XINT_half #1#2#3#4#5#6#7#8%
402   {\expandafter\space\the\numexpr(#1#2#3#4#5#6#7#8\XINT_half_a}%

```

```

403 \def\XINT_half_a#1{\xint_Bye#1\xint_bye\XINT_half_b#1}%
404 \def\XINT_half_b #1#2#3#4#5#6#7#8%
405   {\expandafter\XINT_half_e\the\numexpr(1#1#2#3#4#5#6#7#8\XINT_half_a}%
406 \def\XINT_half_e#1{* \xint_c_v+#1- \xint_c_v)\relax}%

```

3.15 \xintInc

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

As we take 9 by 9 with the input save stack at 5000 this allows a bit less than 9 times 2500 = 22500 digits on input.

```

407 \def\xintInc {\romannumeral0\xintinc}%
408 \def\xintinc #1{\expandafter\XINT_inc_fork\romannumeral`&&@#1%
409   \xint_bye23456789\xint_bye+\xint_c_i\relax}%
410 \def\XINT_inc_fork #1%
411 {%
412   \xint_UDsignfork
413   #1\XINT_inc_neg
414   -{\XINT_inc #1}%
415   \krof
416 }%
417 \def\XINT_inc_neg #1\xint_bye#2\relax
418   {\xintiopp\XINT_dec #1\XINT_dec_bye234567890\xint_bye}%
419 \def\XINT_inc #1#2#3#4#5#6#7#8#9%
420   {\expandafter\space\the\numexpr#1#2#3#4#5#6#7#8#9\XINT_inc_a}%
421 \def\XINT_inc_a #1#2#3#4#5#6#7#8#9%
422   {\expandafter\XINT_inc_e\the\numexpr 1#1#2#3#4#5#6#7#8#9\XINT_inc_a}%
423 \def\XINT_inc_e#1{\if#12+\xint_c_i\fi\relax}%

```

3.16 \xintDec

1.2i much delayed complete rewrite in the 1.2 style. Things are a bit more complicated than \xintInc because 2999999999 is too big for TeX.

```

424 \def\xintDec {\romannumeral0\xintdec}%
425 \def\xintdec #1{\expandafter\XINT_dec_fork\romannumeral`&&@#1%
426   \XINT_dec_bye234567890\xint_bye}%
427 \def\XINT_dec_fork #1%
428 {%
429   \xint_UDsignfork
430   #1\XINT_dec_neg
431   -{\XINT_dec #1}%
432   \krof
433 }%
434 \def\XINT_dec_neg #1\XINT_dec_bye#2\xint_bye
435   {\expandafter-%
436   \romannumeral0\XINT_inc #1\xint_bye23456789\xint_bye+\xint_c_i\relax}%
437 \def\XINT_dec #1#2#3#4#5#6#7#8#9%
438   {\expandafter\space\the\numexpr#1#2#3#4#5#6#7#8#9\XINT_dec_a}%
439 \def\XINT_dec_a #1#2#3#4#5#6#7#8#9%
440   {\expandafter\XINT_dec_e\the\numexpr 1#1#2#3#4#5#6#7#8#9\XINT_dec_a}%
441 \def\XINT_dec_bye #1\XINT_dec_a#2#3\xint_bye

```

```

442      {\if#20-\xint_c_ii\relax+\else-\fi\xint_c_i\relax}%
443 \def\xINT_dec_e#1{\unless\if#11\xint_dothis{-\xint_c_i#1}\fi\xint_orthat\relax}%

```

3.17 \xintDSL

DECIMAL SHIFT LEFT (=MULTIPLICATION PAR 10). Rewritten for 1.2i. This was very old code... I never came back to it, but I should have rewritten it long time ago.

```

444 \def\xintDSL {\romannumeral0\xintdsl}%
445 \def\xintdsl #1{\expandafter\xINT_dsl\romannumeral`&&@#1}%
446 \edef\xINT_dsl #1%
447   {\noexpand\xint_gob_til_zero #1\noexpand\xint_dsl_zero 0\space #1}%
448 \def\xint_dsl_zero 0 0{ }%

```

3.18 \xintDSR

Decimal shift right, truncates towards zero. Rewritten for 1.2i. Limited to 22483 digits on input.

```

449 \def\xintDSR{\romannumeral0\xintdsr}%
450 \def\xintdsr #1{\expandafter\xINT_dsr_fork\romannumeral`&&@#1%
451   \xint_bye\xint_Bye3456789\xint_bye+\xint_c_v)/\xint_c_x-\xint_c_i\relax}%
452 \def\xINT_dsr_fork #1%
453 {%
454   \xint_UDsignfork
455   #1\xINT_dsr_neg
456   -{\xINT_dsr #1}%
457   \krof
458 }%
459 \def\xINT_dsr_neg{\xintiopp\xINT_dsr}%
460 \def\xINT_dsr #1#2#3#4#5#6#7#8#9%
461   {\expandafter\space\the\numexpr(#1#2#3#4#5#6#7#8#9\xINT_dsr_a}%
462 \def\xINT_dsr_a#1{\xint_Bye#1\xint_bye\xINT_dsr_b#1}%
463 \def\xINT_dsr_b #1#2#3#4#5#6#7#8#9%
464   {\expandafter\xINT_dsr_e\the\numexpr(1#1#2#3#4#5#6#7#8#9\xINT_dsr_a}%
465 \def\xINT_dsr_e #1{\)\relax}%

```

3.19 \xintDSRr

New with 1.2i. Decimal shift right, rounds away from zero; done in the 1.2 spirit (with much delay, sorry). Used by \xintRound, \xintDivRound.

This is about the first time I am happy that the division in \numexpr rounds!

```

466 \def\xintDSRr{\romannumeral0\xintdsrr}%
467 \def\xintdsrr #1{\expandafter\xINT_dsrr_fork\romannumeral`&&@#1%
468   \xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax}%
469 \def\xINT_dsrr_fork #1%
470 {%
471   \xint_UDsignfork
472   #1\xINT_dsrr_neg
473   -{\xINT_dsrr #1}%
474   \krof
475 }%
476 \def\xINT_dsrr_neg{\xintiopp\xINT_dsrr}%

```

```

477 \def\XINT_dsrr #1#2#3#4#5#6#7#8#9%
478   {\expandafter\space\the\numexpr#1#2#3#4#5#6#7#8#9\XINT_dsrr_a}%
479 \def\XINT_dsrr_a#1{\xint_Bye#1\xint_bye\XINT_dsrr_b#1}%
480 \def\XINT_dsrr_b #1#2#3#4#5#6#7#8#9%
481   {\expandafter\XINT_dsrr_e\the\numexpr1#1#2#3#4#5#6#7#8#9\XINT_dsrr_a}%
482 \let\XINT_dsrr_e\XINT_inc_e

```

3.20 Core arithmetic

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

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

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

3.21 `\xintiAdd`, `\xintiiAdd`

```

483 \def\xintiAdd   {\romannumeral0\xintiadd }%
484 \def\xintiadd  #1{\expandafter\XINT_iadd\romannumeral0\xintnum{#1}\Z }%
485 \def\xintiiAdd  {\romannumeral0\xintiiadd }%
486 \def\xintiiadd #1{\expandafter\XINT_iiadd\romannumeral`&&@#1\Z }%
487 \def\XINT_iiadd #1#2\Z #3%
488 {%
489   \expandafter\XINT_add_nfork\expandafter #1\romannumeral`&&@#3\Z #2\Z
490 }%
491 \def\XINT_iadd #1#2\Z #3%
492 {%
493   \expandafter\XINT_add_nfork\expandafter #1\romannumeral0\xintnum{#3}\Z #2\Z
494 }%
495 \def\XINT_add_fork #1#2\Z #3\Z {\XINT_add_nfork #1#3\Z #2\Z}%
496 \def\XINT_add_nfork #1#2%
497 {%
498   \xint_UDzerofork
499     #1\XINT_add_firstiszero
500     #2\XINT_add_secondiszero
501     0}%
502   \krof
503   \xint_UDsignsfork
504     #1#2\XINT_add_minusminus
505     #1-\XINT_add_minusplus
506     #2-\XINT_add_plusminus
507     --\XINT_add_plusplus
508   \krof #1#2%
509 }%
510 \def\XINT_add_firstiszero #1\krof 0#2#3\Z #4\Z { #2#3}%
511 \def\XINT_add_secondiszero #1\krof #20#3\Z #4\Z { #2#4}%
512 \def\XINT_add_minusminus #1#2%
513   {\expandafter-\romannumeral0\XINT_add_pp_a {}{}}%

```

3 Package *xintcore* implementation

```

514 \def\XINT_add_minusplus #1#2{\XINT_sub_mm_a {}#2}%
515 \def\XINT_add_plusminus #1#2%
516   {\expandafter\XINT_opp\romannumeral0\XINT_sub_mm_a #1{}}%
517 \def\XINT_add_pp_a #1#2#3\Z
518 {%
519   \expandafter\XINT_add_pp_b
520     \romannumeral0\expandafter\XINT_sepandrev_andcount
521     \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
522     #2#3\XINT_rsepbyviii_end_A 2345678%
523     \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_iii
524     \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
525     \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
526   \X #1%
527 }%
528 \let\XINT_add_plusplus \XINT_add_pp_a
529 \def\XINT_add_pp_b #1.#2\X #3\Z
530 {%
531   \expandafter\XINT_add_checklengths
532   \the\numexpr #1\expandafter.%
533   \romannumeral0\expandafter\XINT_sepandrev_andcount
534   \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
535   #3\XINT_rsepbyviii_end_A 2345678%
536   \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_iii
537   \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
538   \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
539   1\Z!1\Z!1\Z!1\Z!\W #21\Z!1\Z!1\Z!1\Z!\W
540   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
541 }%

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

542 \def\XINT_add_checklengths #1.#2.%
543 {%
544   \ifnum #2>#1
545     \expandafter\XINT_add_exchange
546   \else
547     \expandafter\XINT_add_A
548   \fi
549   #1.#2.%
550 }%
551 \def\XINT_add_exchange #1.#2.#3\W #4\W
552 {%
553   \XINT_add_A #2.#1.#4\W #3\W
554 }%
555 \def\XINT_add_A #1.#2.%
556 {%
557   \ifnum #1>\xint_c_vi
558     \expandafter\XINT_add_aa
559   \else \expandafter\XINT_add_aa_small
560   \fi
561 }%
562 \def\XINT_add_aa {\expandafter\XINT_add_out\the\numexpr\XINT_add_a \xint_c_ii}%
563 \def\XINT_add_out{\expandafter\XINT_cuz_small\romannumeral0\XINT_unrevbyviii {}}%

```

3 Package *xintcore* implementation

```

564 \def\XINT_add_aa_small
565   {\expandafter\XINT_smallunrevbyviii\the\numexpr\XINT_add_a \xint_c_ii}%

2 as first token of #1 stands for "no carry", 3 will mean a carry (we are adding 1<8digits> to
1<8digits>.) Version 1.2c has terminators of the shape 1\Z!, replacing the \Z! used in 1.2.
Call: \the\numexpr\XINT_add_a 2#11\Z!1\Z!1\Z!1\Z!\W #21\Z!1\Z!1\Z!1\Z!\W where #1 and #2 are
blocks of 1<8d>!, and #1 is at most as long as #2. This last requirement is a bit annoying (if one
wants to do recursive algorithms but not have to check lengths), and I will probably remove it at
some point.
Output: blocks of 1<8d>! representing the addition, (least significant first), and a final 1\Z!.
In recursive algorithm this 1\Z! terminator can thus conveniently be reused as part of input ter-
minator (up to the length problem).

566 \def\XINT_add_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
567 {%
568   \XINT_add_b #1!#6!#2!#7!#3!#8!#4!#9!#5\W
569 }%
570 \def\XINT_add_b #1#2#3!#4!%
571 {%
572   \xint_gob_til_Z #2\XINT_add_bi \Z
573   \expandafter\XINT_add_c\the\numexpr#1+1#2#3+#4-\xint_c_ii.%
574 }%
575 \def\XINT_add_bi\Z\expandafter\XINT_add_c
576   \the\numexpr#1+#2+#3-\xint_c_ii.#4!#5!#6!#7!#8!#9!\W
577 {%
578   \XINT_add_k #1#3!#5!#7!#9!%
579 }%
580 \def\XINT_add_c #1#2.%
581 {%
582   1#2\expandafter!\the\numexpr\XINT_add_d #1%
583 }%
584 \def\XINT_add_d #1#2#3!#4!%
585 {%
586   \xint_gob_til_Z #2\XINT_add_di \Z
587   \expandafter\XINT_add_e\the\numexpr#1+1#2#3+#4-\xint_c_ii.%
588 }%
589 \def\XINT_add_di\Z\expandafter\XINT_add_e
590   \the\numexpr#1+#2+#3-\xint_c_ii.#4!#5!#6!#7!#8\W
591 {%
592   \XINT_add_k #1#3!#5!#7!%
593 }%
594 \def\XINT_add_e #1#2.%
595 {%
596   1#2\expandafter!\the\numexpr\XINT_add_f #1%
597 }%
598 \def\XINT_add_f #1#2#3!#4!%
599 {%
600   \xint_gob_til_Z #2\XINT_add_fi \Z
601   \expandafter\XINT_add_g\the\numexpr#1+1#2#3+#4-\xint_c_ii.%
602 }%
603 \def\XINT_add_fi\Z\expandafter\XINT_add_g
604   \the\numexpr#1+#2+#3-\xint_c_ii.#4!#5!#6\W
605 {%
606   \XINT_add_k #1#3!#5!%

```

```

607 }%
608 \def\XINT_add_g #1#2.%
609 {%
610     1#2\expandafter!\the\numexpr\XINT_add_h #1%
611 }%
612 \def\XINT_add_h #1#2#3!#4!%
613 {%
614     \xint_gob_til_Z #2\XINT_add_hi \Z
615     \expandafter\XINT_add_i\the\numexpr#1+1#2#3+#4-\xint_c_ii.%
616 }%
617 \def\XINT_add_hi\Z
618     \expandafter\XINT_add_i\the\numexpr#1+#2+#3-\xint_c_ii.#4\W
619 {%
620     \XINT_add_k #1#3!%
621 }%
622 \def\XINT_add_i #1#2.%
623 {%
624     1#2\expandafter!\the\numexpr\XINT_add_a #1%
625 }%
626 \def\XINT_add_k #1{\if #12\expandafter\XINT_add_ke\else\expandafter\XINT_add_l \fi}%
627 \def\XINT_add_ke #11\Z #2\W {\XINT_add_kf #11\Z!}%
628 \def\XINT_add_kf 1{1\relax }%
629 \def\XINT_add_l 1#1#2{\xint_gob_til_Z #1\XINT_add_lf \Z \XINT_add_m 1#1#2}%
630 \def\XINT_add_lf #1\W {1\relax 00000001!1\Z!}%
631 \def\XINT_add_m #1!{\expandafter\XINT_add_n\the\numexpr\xint_c_i+#1.%}
632 \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>.)

```

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

```

3.22 \xintiSub, \xintiiSub

Entirely rewritten for 1.2.

```

634 \def\xintiSub {\romannumeral0\xintiisub }%
635 \def\xintiisub #1{\expandafter\XINT_iisub\romannumeral`&&#1\Z }%
636 \def\XINT_iisub #1#2\Z #3%
637 {%
638     \expandafter\XINT_sub_nfork\expandafter #1\romannumeral`&&#3\Z #2\Z
639 }%
640 \def\xintiSub {\romannumeral0\xintisub }%
641 \def\xintisub #1{\expandafter\XINT_isub\romannumeral0\xintnum{#1}\Z }%
642 \def\XINT_isub #1#2\Z #3%
643 {%
644     \expandafter\XINT_sub_nfork\expandafter #1\romannumeral0\xintnum{#3}\Z #2\Z
645 }%
646 \def\XINT_sub_nfork #1#2%
647 {%
648     \xint_UDzerofork
649     #1\XINT_sub_firstiszero
650     #2\XINT_sub_secondiszero
651     0{}%
652     \krof

```

3 Package *xintcore* implementation

```

653 \xint_UDsignsfork
654 #1#2\XINT_sub_minusminus
655 #1-\XINT_sub_minusplus
656 #2-\XINT_sub_plusminus
657 --\XINT_sub_plusplus
658 \krof #1#2%
659 }%
660 \def\XINT_sub_firstiszero #1\krof 0#2#3\Z #4\Z {\XINT_opp #2#3}%
661 \def\XINT_sub_secondiszero #1\krof #20#3\Z #4\Z { #2#4}%
662 \def\XINT_sub_plusminus #1#2{\XINT_add_pp_a #1{}}%
663 \def\XINT_sub_plusplus #1#2%
664 {\expandafter\XINT_opp\romannumeral0\XINT_sub_mm_a #1#2}%
665 \def\XINT_sub_minusplus #1#2%
666 {\expandafter-\romannumeral0\XINT_add_pp_a {}}#2}%
667 \def\XINT_sub_minusminus #1#2{\XINT_sub_mm_a {}}}%
668 \def\XINT_sub_mm_a #1#2#3\Z
669 {%
670 \expandafter\XINT_sub_mm_b
671 \romannumeral0\expandafter\XINT_sepandrev_andcount
672 \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
673 #2#3\XINT_rsepyviii_end_A 2345678%
674 \XINT_rsepyviii_end_B 2345678\relax\xint_c_ii\xint_c_iii
675 \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
676 \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
677 \X #1%
678 }%
679 \def\XINT_sub_mm_b #1.#2\X #3\Z
680 {%
681 \expandafter\XINT_sub_checklengths
682 \the\numexpr #1\expandafter.%
683 \romannumeral0\expandafter\XINT_sepandrev_andcount
684 \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
685 #3\XINT_rsepyviii_end_A 2345678%
686 \XINT_rsepyviii_end_B 2345678\relax \xint_c_ii\xint_c_iii
687 \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
688 \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
689 \Z!\Z!\Z!\Z!\W #2\Z!\Z!\Z!\Z!\W
690 }%
691 \def\XINT_sub_checklengths #1.#2.%
692 {%
693 \ifnum #2>#1
694 \expandafter\XINT_sub_exchange
695 \else
696 \expandafter\XINT_sub_aa
697 \fi
698 }%
699 \def\XINT_sub_exchange #1\W #2\W
700 {%
701 \expandafter\XINT_opp\romannumeral0\XINT_sub_aa #2\W #1\W
702 }%
703 \def\XINT_sub_aa {\expandafter\XINT_sub_out\the\numexpr\XINT_sub_a \xint_c_i }%

```

The {} after \XINT_unrevbyviii could be removed, but attention then at \XINT_sub_startrescue

3 Package *xintcore* implementation

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

```
704 \def\XINT_sub_out #1\Z #2#3\W
705 {%
706   \if-#2\expandafter\XINT_sub_startrescue\fi
707   \expandafter\XINT_cuz_small
708   \romannumeral0\XINT_unrevbyviii {}#11\Z!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
709 }%
```

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

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

The routine wants to compute #2-#1.

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

`\numexpr` governed expansion stops with various possibilities:

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

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

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

If there is a carry things are more complicated.

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

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

1<8d>!...1<8d>!1\XINT_cuz_byviii!\Z 0\W\R

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

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

1<8d>!...1<8d>!1\XINT_cuz_byviii!\Z 0\W\R

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

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

Thus `\XINT_sub_out` examines the token after the first `\Z`, which may be ! or 0 or -. If ! or 0, `\XINT_unrevbyviii` will be executed (gobbling a possible final 1!), and followed in case 2a or 1c by `\XINT_cuz_byviii` (note the extra `\R` which terminates it), and then in both 1* and 2a by `\XINT_cuz_small`.

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

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

For the record: subtraction was correct (I think) in xint releases up to 1.2, but 1.2 had a broken treatment of the 1d case. For example `\xintiiSub {10000000112345678}{12345679}` produced 99999999. This got fixed in 1.2c, but that fix broke the 1c case :((, for example `\xintiiSub {10000000000000000}{9999999999999997}` was now returning 000000003. Alas.

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

```
710 \def\XINT_sub_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
711 {%
```

3 Package *xintcore* implementation

```
712 \XINT_sub_b #1!#6!#2!#7!#3!#8!#4!#9!#5\W
713 }%
714 \def\XINT_sub_b #1#2#3!#4!%
715 {%
716 \xint_gob_til_Z #2\XINT_sub_bi \Z
717 \expandafter\XINT_sub_c\the\numexpr#1+1#4-#3-\xint_c.i.%
718 }%
719 \def\XINT_sub_c 1#1#2.%
720 {%
721 1#2\expandafter!\the\numexpr\XINT_sub_d #1%
722 }%
723 \def\XINT_sub_d #1#2#3!#4!%
724 {%
725 \xint_gob_til_Z #2\XINT_sub_di \Z
726 \expandafter\XINT_sub_e\the\numexpr#1+1#4-#3-\xint_c.i.%
727 }%
728 \def\XINT_sub_e 1#1#2.%
729 {%
730 1#2\expandafter!\the\numexpr\XINT_sub_f #1%
731 }%
732 \def\XINT_sub_f #1#2#3!#4!%
733 {%
734 \xint_gob_til_Z #2\XINT_sub_fi \Z
735 \expandafter\XINT_sub_g\the\numexpr#1+1#4-#3-\xint_c.i.%
736 }%
737 \def\XINT_sub_g 1#1#2.%
738 {%
739 1#2\expandafter!\the\numexpr\XINT_sub_h #1%
740 }%
741 \def\XINT_sub_h #1#2#3!#4!%
742 {%
743 \xint_gob_til_Z #2\XINT_sub_hi \Z
744 \expandafter\XINT_sub_i\the\numexpr#1+1#4-#3-\xint_c.i.%
745 }%
746 \def\XINT_sub_i 1#1#2.%
747 {%
748 1#2\expandafter!\the\numexpr\XINT_sub_a #1%
749 }%
750 \def\XINT_sub_bi\Z
751 \expandafter\XINT_sub_c\the\numexpr#1+1#2-#3.#4!#5!#6!#7!#8!#9!\W
752 {%
753 \XINT_sub_k #1#2!#5!#7!#9!%
754 }%
755 \def\XINT_sub_di\Z
756 \expandafter\XINT_sub_e\the\numexpr#1+1#2-#3.#4!#5!#6!#7!#8\W
757 {%
758 \XINT_sub_k #1#2!#5!#7!%
759 }%
760 \def\XINT_sub_fi\Z
761 \expandafter\XINT_sub_g\the\numexpr#1+1#2-#3.#4!#5!#6\W
762 {%
763 \XINT_sub_k #1#2!#5!%
```

3 Package *xintcore* implementation

```

764 }%
765 \def\XINT_sub_hi\Z
766   \expandafter\XINT_sub_i\the\numexpr#1+1#2-#3.#4\W
767 {%
768   \XINT_sub_k #1#2!%
769 }%

```

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

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

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

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

```

770 \def\XINT_sub_k #1#2%
771 {%
772   \xint_gob_til_Z #2\XINT_sub_p\Z \XINT_sub_l #1#2%
773 }%
774 \def\XINT_sub_l #1{\xint_UDzerofork #1\XINT_sub_l_carry 0\XINT_sub_l_nocarry\krof}%
775 \def\XINT_sub_l_nocarry 1{1\relax }%
776 \def\XINT_sub_l_carry 1#1!{\ifcase #1
777   \expandafter \XINT_sub_l_zeroa\or\expandafter\XINT_sub_l_one\else
778   \expandafter \XINT_sub_l_done\fi 1#1!}%
779 \def\XINT_sub_l_done {-\xint_c_i+}%
780 \def\XINT_sub_l_one 1#1!#2%
781 {%
782   \xint_gob_til_Z #2\XINT_sub_l_oneone\Z 1\relax 00000000!#2%
783 }%
784 \def\XINT_sub_l_oneone #1\W {1\relax \XINT_cuz_byviii!\Z 0\W\R }%
785 \def\XINT_sub_l_zeroa 1#1!{199999999\expandafter!\the\numexpr \XINT_sub_l_zerob }%
786 \def\XINT_sub_l_zerob 1#1!{\ifcase #1
787   \expandafter \XINT_sub_l_zeroa\or\expandafter\XINT_sub_l_zone\else
788   \expandafter \XINT_sub_l_done\fi 1#1!}%
789 \def\XINT_sub_l_zone 1#1!#2%
790 {%
791   \xint_gob_til_Z #2\XINT_sub_l_zoneone\Z 1\relax 00000000!#2%
792 }%
793 \def\XINT_sub_l_zoneone\Z 1\relax 00000000{1}%

```

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

```

794 \def\XINT_sub_p\Z\XINT_sub_l #1#2\W
795 {%
796   \xint_UDzerofork
797   #1{-1\relax\Z -\W}%

```

3 Package *xintcore* implementation

```
798      0{1\relax \XINT_cuz_byviii!\Z 0\W\R }%
799      \krof
800 }%
```

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

```
801 \def\XINT_sub_startrescue\expandafter\XINT_cuz_small
802      \romannumeral0\XINT_unrevbyviii #1#2\Z!#3\W
803 {%
804      \expandafter\XINT_sub_rescue_finish
805      \the\numexpr\XINT_sub_rescue_a #2!%
806      1\Z!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W \R
807 }%
808 \def\XINT_sub_rescue_finish
809      {\expandafter-\romannumeral0\expandafter\XINT_cuz\romannumeral0\XINT_unrevbyviii {}}%
810 \def\XINT_sub_rescue_a #1!%
811 {%
812      \expandafter\XINT_sub_rescue_c\the\numexpr \xint_c_xii_e_viii-#1.%
813 }%
814 \def\XINT_sub_rescue_c 1#1#2.%
815 {%
816      1#2\expandafter!\the\numexpr\XINT_sub_rescue_d #1%
817 }%
818 \def\XINT_sub_rescue_d #1#2#3!%
819 {%
820      \xint_gob_til_minus #2\XINT_sub_rescue_z -%
821      \expandafter\XINT_sub_rescue_c\the\numexpr \xint_c_xii_e_viii_mone-#2#3+#1.%
822 }%
823 \def\XINT_sub_rescue_z #1.{1!}%

```

3.23 `\xintiMul`, `\xintiiMul`

Completely rewritten for 1.2.

```
824 \def\xintiMul {\romannumeral0\xintimul }%
825 \def\xintimul #1%
826 {%
827      \expandafter\XINT_imul\romannumeral0\xintnum{#1}\Z
828 }%
829 \def\XINT_imul #1#2\Z #3%
830 {%
831      \expandafter\XINT_mul_nfork\expandafter #1\romannumeral0\xintnum{#3}\Z #2\Z
832 }%
833 \def\xintiiMul {\romannumeral0\xintiimul }%
834 \def\xintiimul #1%
835 {%
836      \expandafter\XINT_iimul\romannumeral`&&@#1\Z
837 }%
838 \def\XINT_iimul #1#2\Z #3%
839 {%
840      \expandafter\XINT_mul_nfork\expandafter #1\romannumeral`&&@#3\Z #2\Z
841 }%

```

3 Package *xintcore* implementation

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

```
842 \def\XINT_mul_fork #1#2\Z #3\Z{\XINT_mul_nfork #1#3\Z #2\Z}%
843 \def\XINT_mul_nfork #1#2%
844 {%
845   \xint_UDzerofork
846   #1\XINT_mul_zero
847   #2\XINT_mul_zero
848   0{}}%
849   \krof
850   \xint_UDsignsfork
851   #1#2\XINT_mul_minusminus
852   #1-\XINT_mul_minusplus
853   #2-\XINT_mul_plusminus
854   --\XINT_mul_plusplus
855   \krof #1#2%
856 }%
857 \def\XINT_mul_zero #1\krof #2#3\Z #4\Z { 0}%
858 \def\XINT_mul_minusminus #1#2{\XINT_mul_plusplus {}}}%
859 \def\XINT_mul_minusplus #1#2%
860   {\expandafter\romannumeral0\XINT_mul_plusplus {}#2}%
861 \def\XINT_mul_plusminus #1#2%
862   {\expandafter\romannumeral0\XINT_mul_plusplus #1{}}}%
863 \def\XINT_mul_plusplus #1#2#3\Z
864 {%
865   \expandafter\XINT_mul_pre_b
866   \romannumeral0\expandafter\XINT_sepandrev_andcount
867   \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
868   #2#3\XINT_rsepybviii_end_A 2345678%
869   \XINT_rsepybviii_end_B 2345678\relax\xint_c_ii\xint_c_iii
870   \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
871   \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
872   \W #1%
873 }%
874 \def\XINT_mul_pre_b #1.#2\W #3\Z
875 {%
876   \expandafter\XINT_mul_checklengths
877   \the\numexpr #1\expandafter.%
878   \romannumeral0\expandafter\XINT_sepandrev_andcount
879   \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
880   #3\XINT_rsepybviii_end_A 2345678%
881   \XINT_rsepybviii_end_B 2345678\relax\xint_c_ii\xint_c_iii
882   \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
883   \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
884   1\Z!\W #21\Z!%
885   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\W
886 }%
```

Cooking recipe, 2015/10/05.

```
887 \def\XINT_mul_checklengths #1.#2.%
888 {%
889   \ifnum #2=\xint_c_i\expandafter\XINT_mul_smallbyfirst\fi
```

3 Package *xintcore* implementation

```

890 \ifnum #1=\xint_c_i\expandafter\XINT_mul_smallbysecond\fi
891 \ifnum #2<#1
892 \ifnum \numexpr (#2-\xint_c_i)*(#1-#2)<383
893 \XINT_mul_exchange
894 \fi
895 \else
896 \ifnum \numexpr (#1-\xint_c_i)*(#2-#1)>383
897 \XINT_mul_exchange
898 \fi
899 \fi
900 \XINT_mul_start
901 }%
902 \def\XINT_mul_smallbyfirst #1\XINT_mul_start 1#2!1\Z!\W
903 {%
904 \ifnum#2=\xint_c_i\expandafter\XINT_mul_oneisone\fi
905 \ifnum#2<\xint_c_xxii\expandafter\XINT_mul_verysmall\fi
906 \expandafter\XINT_mul_out\the\numexpr\XINT_smallmul 1#2!%
907 }%
908 \def\XINT_mul_smallbysecond #1\XINT_mul_start #2\W 1#3!1\Z!%
909 {%
910 \ifnum#3=\xint_c_i\expandafter\XINT_mul_oneisone\fi
911 \ifnum#3<\xint_c_xxii\expandafter\XINT_mul_verysmall\fi
912 \expandafter\XINT_mul_out\the\numexpr\XINT_smallmul 1#3!#2%
913 }%
914 \def\XINT_mul_oneisone #1!\XINT_mul_out }%
915 \def\XINT_mul_verysmall\expandafter\XINT_mul_out
916 \the\numexpr\XINT_smallmul 1#1!%
917 {\expandafter\XINT_mul_out\the\numexpr\XINT_verysmallmul 0.#1!}%
918 \def\XINT_mul_exchange #1\XINT_mul_start #2\W #3!1\Z!%
919 {\fi\fi\XINT_mul_start #3!1\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`.

```

920 \def\XINT_mul_start
921 {\expandafter\XINT_mul_out\the\numexpr\XINT_mul_loop 100000000!1\Z!\W}%
922 \def\XINT_mul_out
923 {\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`.

3 Package *xintcore* implementation

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

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

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

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

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

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

```
930 \def\XINT_mul_a #1\W #2\W
931 {%
932   \expandafter\XINT_mul_b\the\numexpr
933   \XINT_add_a \xint_c_ii #21\Z!1\Z!1\Z!\W #11\Z!1\Z!1\Z!\W\W
934 }%
935 \def\XINT_mul_b 1#1!{1#1\expandafter!\the\numexpr\XINT_mul_loop }%
936 \def\XINT_mul_e\Z #1\W 1#2\W #3\W {1\relax #2}%
```

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

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

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

Multiplication by zero will produce blocks of zeros.

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

3 Package *xintcore* implementation

```
952 }%
953 \def\XINT_smallmul 1#1#2#3#4#5!{\XINT_smallmul_a 100000000.#1#2#3#4.#5!}%
954 \def\XINT_smallmul_a #1.#2.#3!#4!%
955 {%
956   \xint_gob_til_Z #4\XINT_smallmul_e\Z
957   \XINT_minimulwc_a #1.#2.#3!#4.#2.#3!%
958 }%
959 \def\XINT_smallmul_e\Z\XINT_minimulwc_a 1#1.#2\Z #3!%
960   {\xint_gob_til_eightzeroes #1\XINT_smallmul_f 000000001\relax #1!1\Z!}%
961 \def\XINT_smallmul_f 000000001\relax 00000000!1{1\relax}%
```

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

```
962 \def\XINT_verysmallmul #1.#2!1#3!%
963 {%
964   \xint_gob_til_Z #3\XINT_verysmallmul_e\Z
965   \expandafter\XINT_verysmallmul_a
966   \the\numexpr #2*#3+#1.#2!%
967 }%
968 \def\XINT_verysmallmul_e\Z\expandafter\XINT_verysmallmul_a\the\numexpr
969   #1+#2#3.#4!%
970 {\xint_gob_til_zero #2\XINT_verysmallmul_f 0\xint_c_x^viii+#2#3!1\Z!}%
971 \def\XINT_verysmallmul_f #1!1{1\relax}%
972 \def\XINT_verysmallmul_a #1#2.%
973 {%
974   \unless\ifnum #1#2<\xint_c_x^ix
975     \expandafter\XINT_verysmallmul_bi\else
976     \expandafter\XINT_verysmallmul_bj\fi
977     \the\numexpr \xint_c_x^ix+#1#2.%
978 }%
979 \def\XINT_verysmallmul_bj{\expandafter\XINT_verysmallmul_cj }%
980 \def\XINT_verysmallmul_cj 1#1#2.%
981   {1#2\expandafter!\the\numexpr\XINT_verysmallmul #1.}%
982 \def\XINT_verysmallmul_bi\the\numexpr\xint_c_x^ix+#1#2#3.%
983   {1#3\expandafter!\the\numexpr\XINT_verysmallmul #1#2.}%
```

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

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

```
984 \def\XINT_minimul_a #1.#2!#3#4#5#6#7!%
985 {%
986   \expandafter\XINT_minimul_b
987   \the\numexpr \xint_c_x^viii+#2*#7.#2*#3#4#5#6+#1*#7.#1*#3#4#5#6.%
988 }%
989 \def\XINT_minimul_b 1#1#2#3#4#5.#6.%
990 {%
991   \expandafter\XINT_minimul_c
992   \the\numexpr \xint_c_x^ix+#1#2#3#4+#6.#5.%
993 }%
994 \def\XINT_minimul_c 1#1#2#3#4#5#6.#7.#8.%
995 {%
```



```

996      1#6#7\expandafter!\the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8!%
997 }%

```

3.24 \xintiSqr, \xintiiSqr

Rewritten for 1.2.

```

998 \def\xintiiSqr {\romannumeral0\xintiisqr }%
999 \def\xintiisqr #1%
1000 {%
1001   \expandafter\XINT_sqr\romannumeral0\xintiisqr{#1}\Z
1002 }%
1003 \def\xintiSqr {\romannumeral0\xintisqr }%
1004 \def\xintisqr #1%
1005 {%
1006   \expandafter\XINT_sqr\romannumeral0\xintiisqr{#1}\Z
1007 }%
1008 \def\XINT_sqr #1\Z
1009 {%
1010   \expandafter\XINT_sqr_a
1011   \romannumeral0\expandafter\XINT_sepandrev_andcount
1012   \romannumeral0\XINT_zeroes_forviii #1\R\R\R\R\R\R\R\R{10}0000001\W
1013   #1\XINT_rsepbyviii_end_A 2345678%
1014   \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_iii
1015   \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
1016   \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
1017   \Z
1018 }%

```

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

```

1019 \def\XINT_sqr_a #1.%
1020 {%
1021   \ifnum #1=\xint_c_i \expandafter\XINT_sqr_small
1022   \else\expandafter\XINT_sqr_start\fi
1023 }%
1024 \def\XINT_sqr_small 1#1#2#3#4#5!\Z
1025 {%
1026   \ifnum #1#2#3#4#5<46341 \expandafter\XINT_sqr_verysmall\fi
1027   \expandafter\XINT_sqr_small_out
1028   \the\numexpr\XINT_minimul_a #1#2#3#4.#5!#1#2#3#4#5!%
1029 }%
1030 \edef\XINT_sqr_verysmall
1031   \expandafter\XINT_sqr_small_out\the\numexpr\XINT_minimul_a #1!#2!%
1032   {\noexpand\expandafter\space\noexpand\the\numexpr #2*#2\relax}%
1033 \def\XINT_sqr_small_out 1#1!1#2!%
1034 {%
1035   \XINT_cuz #2#1\R
1036 }%

```

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

```

1037 \def\XINT_sqr_start #1\Z
1038 {%
1039   \expandafter\XINT_mul_out
1040   \the\numexpr\XINT_mul_loop 100000000!1\Z!\W #11\Z!\W #11\Z!%
1041   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1042 }%

```

3.25 \xintiPow, \xintiiPow

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

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

```

1043 \def\xintiiPow {\romannumeral0\xintiipow }%
1044 \def\xintiipow #1#2%
1045 {%
1046   \expandafter\xint_pow\the\numexpr #2\expandafter.\romannumeral`&&@#1\Z%
1047 }%
1048 \def\xintiPow {\romannumeral0\xintipow }%
1049 \def\xintipow #1#2%
1050 {%
1051   \expandafter\xint_pow\the\numexpr #2\expandafter.\romannumeral0\xintnum{#1}\Z%
1052 }%
1053 \def\xint_pow #1.#2%#3\Z
1054 {%
1055   \xint_UDzerominusfork
1056   #2-\XINT_pow_AisZero
1057   0#2\XINT_pow_Aneg
1058   0-\XINT_pow_Apos #2}%
1059   \krof {#1}%
1060 }%
1061 \def\XINT_pow_AisZero #1#2\Z
1062 {%
1063   \ifcase\XINT_cntSgn #1\Z
1064     \xint_afterfi { 1}%
1065   \or
1066     \xint_afterfi { 0}%
1067   \else
1068     \xint_afterfi {\xintError:DivisionByZero\space 0}%
1069   \fi
1070 }%
1071 \def\XINT_pow_Aneg #1%
1072 {%
1073   \ifodd #1
1074     \expandafter\XINT_opp\romannumeral0%
1075   \fi
1076   \XINT_pow_Apos {}{#1}%
1077 }%
1078 \def\XINT_pow_Apos #1#2{\XINT_pow_Apos_a {#2}#1}%
1079 \def\XINT_pow_Apos_a #1#2#3%

```

3 Package *xintcore* implementation

```

1080 {%
1081   \xint_gob_til_Z #3\XINT_pow_Apos_short\Z
1082   \XINT_pow_AatleastTwo {#1}#2#3%
1083 }%
1084 \def\XINT_pow_Apos_short\Z\XINT_pow_AatleastTwo #1#2\Z
1085 {%
1086   \ifcase #2
1087     \xintError:thiscannothappen!
1088   \or \expandafter\XINT_pow_AisOne
1089   \else\expandafter\XINT_pow_AatleastTwo
1090   \fi {#1}#2\Z
1091 }%
1092 \def\XINT_pow_AisOne #1\Z{ 1}%
1093 \def\XINT_pow_AatleastTwo #1%
1094 {%
1095   \ifcase\XINT_cntSgn #1\Z
1096     \expandafter\XINT_pow_BisZero
1097   \or
1098     \expandafter\XINT_pow_I_in
1099   \else
1100     \expandafter\XINT_pow_BisNegative
1101   \fi
1102   {#1}%
1103 }%
1104 \edef\XINT_pow_BisNegative #1\Z
1105   {\noexpand\xintError:FractionRoundedToZero\space 0}%
1106 \def\XINT_pow_BisZero #1\Z{ 1}%

```

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

```

1107 \def\XINT_pow_I_in #1#2\Z
1108 {%
1109   \expandafter\XINT_pow_I_loop
1110   \the\numexpr #1\expandafter.%
1111   \romannumeral0\expandafter\XINT_sepandrev
1112   \romannumeral0\XINT_zeroes_forviii #2\R\R\R\R\R\R\R\{10}0000001\W
1113   #2\XINT_rsepbyviii_end_A 2345678%
1114   \XINT_rsepbyviii_end_B 2345678\relax XX%
1115   \R.\R.\R.\R.\R.\R.\R.\R.\W 1\Z!\W
1116   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1117 }%
1118 \def\XINT_pow_I_loop #1.%
1119 {%
1120   \ifnum #1 = \xint_c_i\expandafter\XINT_pow_I_exit\fi
1121   \ifodd #1
1122     \expandafter\XINT_pow_II_in
1123   \else
1124     \expandafter\XINT_pow_I_squareit
1125   \fi #1.%
1126 }%
1127 \def\XINT_pow_I_exit \ifodd #1\fi #2.#3\W {\XINT_mul_out #3}%

```

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

3 Package *xintcore* implementation

mall" is not a necessity as it was earlier with 1.2. On 2^{30} , it does bring roughly a 40% time gain though, and 30% gain for 2^{60} . The overhead on big computations should be negligible.

```
1128 \def\XINT_pow_I_squareit #1.#2\W%
1129 {%
1130   \expandafter\XINT_pow_I_loop
1131   \the\numexpr #1/\xint_c_ii\expandafter.%
1132   \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W
1133 }%
1134 \def\XINT_pow_mulbutcheckifsmall #1!1#2%
1135 {%
1136   \xint_gob_til_Z #2\XINT_pow_mul_small\Z
1137   \XINT_mul_loop 100000000!1\Z!\W #1!1#2%
1138 }%
1139 \def\XINT_pow_mul_small\Z \XINT_mul_loop 100000000!1\Z!\W 1#1!1\Z!\W
1140 {%
1141   \XINT_smallmul 1#1!%
1142 }%
1143 \def\XINT_pow_II_in #1.#2\W
1144 {%
1145   \expandafter\XINT_pow_II_loop
1146   \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter.%
1147   \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W #2\W
1148 }%
1149 \def\XINT_pow_II_loop #1.%
1150 {%
1151   \ifnum #1 = \xint_c_i\expandafter\XINT_pow_II_exit\fi
1152   \ifodd #1
1153     \expandafter\XINT_pow_II_odda
1154   \else
1155     \expandafter\XINT_pow_II_even
1156   \fi #1.%
1157 }%
1158 \def\XINT_pow_II_exit\ifodd #1\fi #2.#3\W #4\W
1159 {%
1160   \expandafter\XINT_mul_out
1161   \the\numexpr\XINT_pow_mulbutcheckifsmall #4\W #3%
1162 }%
1163 \def\XINT_pow_II_even #1.#2\W
1164 {%
1165   \expandafter\XINT_pow_II_loop
1166   \the\numexpr #1/\xint_c_ii\expandafter.%
1167   \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W
1168 }%
1169 \def\XINT_pow_II_odda #1.#2\W #3\W
1170 {%
1171   \expandafter\XINT_pow_II_oddb
1172   \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter.%
1173   \the\numexpr\XINT_pow_mulbutcheckifsmall #3\W #2\W #2\W
1174 }%
1175 \def\XINT_pow_II_oddb #1.#2\W #3\W
1176 {%
1177   \expandafter\XINT_pow_II_loop
```

```

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

```

3.26 `\xintiFac`, `\xintiiFac`

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

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

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

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

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

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

```

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

```

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

```

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

```

3 Package *xintcore* implementation

```
1191 \krof #1#2.%
1192 }%
1193 \def\XINT_fac_zero #1.{ 1}%
1194 \edef\XINT_fac_neg #1.{\noexpand\xintError:FactorialOfNegative\space 1}%
1195 \def\XINT_fac_checksize #1.%
1196 {%
1197 \ifnum #1>\xint_c_x^iv
1198 \xint_dothis{\expandafter\xintError:TooBigFactorial
1199 \expandafter\space\expandafter 1\xint_gob_til_W }\fi
1200 \ifnum #1>465 \xint_dothis{\XINT_fac_bigloop_a #1.}\fi
1201 \ifnum #1>101 \xint_dothis{\XINT_fac_medloop_a #1.\XINT_mul_out}\fi
1202 \xint_orthat{\XINT_fac_smallloop_a #1.\XINT_mul_out}%
1203 1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1204 }%
1205 \def\XINT_fac_bigloop_a #1.%
1206 {%
1207 \expandafter\XINT_fac_bigloop_b \the\numexpr
1208 #1+\xint_c_i-\xint_c_ii*((#1-464)/\xint_c_ii).#1.%
1209 }%
1210 \def\XINT_fac_bigloop_b #1.#2.%
1211 {%
1212 \expandafter\XINT_fac_medloop_a
1213 \the\numexpr #1-\xint_c_i.{\XINT_fac_bigloop_loop #1.#2.}%
1214 }%
1215 \def\XINT_fac_bigloop_loop #1.#2.%
1216 {%
1217 \ifnum #1>#2 \expandafter\XINT_fac_bigloop_exit\fi
1218 \expandafter\XINT_fac_bigloop_loop
1219 \the\numexpr #1+\xint_c_ii\expandafter.%
1220 \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_bigloop_mul #1!%
1221 }%
1222 \def\XINT_fac_bigloop_exit #1!{\XINT_mul_out}%
1223 \def\XINT_fac_bigloop_mul #1!%
1224 {%
1225 \expandafter\XINT_smallmul
1226 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
1227 }%
1228 \def\XINT_fac_medloop_a #1.%
1229 {%
1230 \expandafter\XINT_fac_medloop_b
1231 \the\numexpr #1+\xint_c_i-\xint_c_iii*((#1-100)/\xint_c_iii).#1.%
1232 }%
1233 \def\XINT_fac_medloop_b #1.#2.%
1234 {%
1235 \expandafter\XINT_fac_smallloop_a
1236 \the\numexpr #1-\xint_c_i.{\XINT_fac_medloop_loop #1.#2.}%
1237 }%
1238 \def\XINT_fac_medloop_loop #1.#2.%
1239 {%
1240 \ifnum #1>#2 \expandafter\XINT_fac_loop_exit\fi
1241 \expandafter\XINT_fac_medloop_loop
1242 \the\numexpr #1+\xint_c_iii\expandafter.%
```

3 Package *xintcore* implementation

```
1243 \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_medloop_mul #1!%
1244 }%
1245 \def\XINT_fac_medloop_mul #1!%
1246 {%
1247 \expandafter\XINT_smallmul
1248 \the\numexpr
1249 \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1250 }%
1251 \def\XINT_fac_smallloop_a #1.%
1252 {%
1253 \csname
1254 XINT_fac_smallloop_\the\numexpr #1-\xint_c_iv*(#1/\xint_c_iv)\relax
1255 \endcsname #1.%
1256 }%
1257 \expandafter\def\csname XINT_fac_smallloop_1\endcsname #1.%
1258 {%
1259 \XINT_fac_smallloop_loop 2.#1.100000001!1\Z!%
1260 }%
1261 \expandafter\def\csname XINT_fac_smallloop_-2\endcsname #1.%
1262 {%
1263 \XINT_fac_smallloop_loop 3.#1.100000002!1\Z!%
1264 }%
1265 \expandafter\def\csname XINT_fac_smallloop_-1\endcsname #1.%
1266 {%
1267 \XINT_fac_smallloop_loop 4.#1.100000006!1\Z!%
1268 }%
1269 \expandafter\def\csname XINT_fac_smallloop_0\endcsname #1.%
1270 {%
1271 \XINT_fac_smallloop_loop 5.#1.1000000024!1\Z!%
1272 }%
1273 \def\XINT_fac_smallloop_loop #1.#2.%
1274 {%
1275 \ifnum #1>#2 \expandafter\XINT_fac_loop_exit\fi
1276 \expandafter\XINT_fac_smallloop_loop
1277 \the\numexpr #1+\xint_c_iv\expandafter.%
1278 \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_smallloop_mul #1!%
1279 }%
1280 \def\XINT_fac_smallloop_mul #1!%
1281 {%
1282 \expandafter\XINT_smallmul
1283 \the\numexpr
1284 \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1285 }%
1286 \def\XINT_fac_loop_exit #1!#2\Z!#3{#3#2\Z!}%
```

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

Completely rewritten for 1.2.

WARNING: some comments below try to describe the flow of tokens but they date back to xint 1.09j and I updated them on the fly while doing the 1.2 version. As the routine now works in base 10^8 , not 10^4 and "drops" the quotient digits, rather than store them upfront as the earlier code, I may

3 Package *xintcore* implementation

well have not correctly converted all such comments. At the last minute some previously #1 became stuff like #1#2#3#4, then of course the old comments describing what the macro parameters stand for are necessarily wrong.

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

1.2 2015/10/15 had a bad bug which got corrected in 1.2b of 2015/10/29: a divisor starting with 9999999xyz... would cause a failure, simply because it was attempted to use the \XINT_div_mini routine with a divisor of 1+99999999=100000000 having 9 digits. Fortunately the origin of the bug was easy to find out. Too bad that my obviously very deficient test files did not detect it.

```

1287 \def\xintiQuo {\romannumeral0\xinti quo }%
1288 \def\xintiRem {\romannumeral0\xinti rem }%
1289 \def\xinti quo {\expandafter\xint_firstoftwo_thenstop\romannumeral0\xinti division }%
1290 \def\xinti rem {\expandafter\xint_secondoftwo_thenstop\romannumeral0\xinti division }%
1291 \def\xintiQuo {\romannumeral0\xinti quo }%
1292 \def\xintiRem {\romannumeral0\xinti rem }%
1293 \def\xinti quo {\expandafter\xint_firstoftwo_thenstop\romannumeral0\xinti division }%
1294 \def\xinti rem {\expandafter\xint_secondoftwo_thenstop\romannumeral0\xinti division }%
1295 %\let\xintQuo\xintiQuo\let\xintquo\xinti quo % now removed
1296 %\let\xintRem\xintiRem\let\xintrem\xinti rem % now removed

```

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

```

1297 \def\xintiDivision {\romannumeral0\xinti division }%
1298 \def\xintidivision #1{\expandafter\XINT_idivision\romannumeral0\xintnum{#1}\Z }%
1299 \def\XINT_idivision #1#2\Z #3{\expandafter\XINT_idivision_a\expandafter #1%
1300 \romannumeral0\xintnum{#3}\Z #2\Z }%
1301 \def\xintiDivision {\romannumeral0\xinti division }%
1302 \def\xinti idivision #1{\expandafter\XINT_idivision \romannumeral`&&@#1\Z }%
1303 \def\XINT_idivision #1#2\Z #3{\expandafter\XINT_idivision_a\expandafter #1%
1304 \romannumeral`&&@#3\Z #2\Z }%

```

On regarde les signes de A et de B.

```

1305 \def\XINT_idivision_a #1#2% #1 de A, #2 de B.
1306 {%
1307 \if0#2\xint_dothis\XINT_idivision_divbyzero\fi
1308 \if0#1\xint_dothis\XINT_idivision_aiszero\fi
1309 \if-#2\xint_dothis{\expandafter\XINT_idivision_bneg
1310 \romannumeral0\XINT_idivision_bpos #1}\fi
1311 \xint_orthat{\XINT_idivision_bpos #1#2}%
1312 }%
1313 \def\XINT_idivision_divbyzero #1\Z #2\Z {\xintError:DivisionByZero{0}{0}}%
1314 \def\XINT_idivision_aiszero #1\Z #2\Z {{0}{0}}%
1315 \def\XINT_idivision_bneg #1% q->-q, r unchanged
1316 {\expandafter{\romannumeral0\XINT_opp #1}}%
1317 \def\XINT_idivision_bpos #1%
1318 {%
1319 \xint_UDsignfork
1320 #1\XINT_idivision_aneg

```


3 Package *xintcore* implementation

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

3 Package *xintcore* implementation

```

1360 }%
1361 \def\XINT_div_BisOne #1#2{{#2}{0}}%
1362 \def\XINT_div_BisTwo #1#2%
1363 {%
1364     \expandafter\expandafter\expandafter\XINT_div_BisTwo_a
1365     \ifodd\xintiilDg{#2} \expandafter1\else \expandafter0\fi {#2}%
1366 }%
1367 \def\XINT_div_BisTwo_a #1#2%
1368 {%
1369     \expandafter{\romannumeral0\XINT_half
1370     #2\xint_bye\xint_Bye345678\xint_bye
1371     *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax}{#1}%
1372 }%

```

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

```

1373 \def\XINT_div_small_a #1#2%
1374 {%
1375     \expandafter\XINT_div_small_b
1376     \the\numexpr #1/\xint_c_ii\expandafter
1377     .\the\numexpr \xint_c_x^viii+#1\expandafter!%
1378     \romannumeral0%
1379     \XINT_div_small_ba #2\R\R\R\R\R\R\R\R{10}0000001\W
1380     #2\XINT_sepbyviii_Z_end 2345678\relax
1381 }%

```

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

```

1382 \def\XINT_div_small_b #1!#2{#2#1!}%

```

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

```

1383 \def\XINT_div_small_ba #1#2#3#4#5#6#7#8#9%
1384 {%
1385     \xint_gob_til_R #9\XINT_div_smallsmall\R
1386     \expandafter\XINT_div_dosmallldiv
1387     \the\numexpr\expandafter\XINT_sepbyviii_Z
1388     \romannumeral0\XINT_zeroes_forviii
1389     #1#2#3#4#5#6#7#8#9%
1390 }%

```

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

```

1391 \def\XINT_div_smallsmall\R
1392     \expandafter\XINT_div_dosmallldiv
1393     \the\numexpr\expandafter\XINT_sepbyviii_Z
1394     \romannumeral0\XINT_zeroes_forviii #1\R #2\relax
1395     {{\XINT_div_dosmallsmall}{#1}}%
1396 \def\XINT_div_dosmallsmall #1.1#2!#3%
1397 {%
1398     \expandafter\XINT_div_smallsmallend
1399     \the\numexpr (#3+#1)/#2-\xint_c_i.#2.#3.%

```

3 Package *xintcore* implementation

```

1400 }%
1401 \def\XINT_div_smallsmallend #1.#2.#3.{\expandafter
1402   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #3-#1*#2}}%

  Si  $A \geq 10^8$ , il est maintenant sous la forme  $1\langle 8d \rangle! \dots 1\langle 8d \rangle!1\backslash Z!$  avec plus significatifs en pre-
  mier. Donc on poursuit par
  \expandafter\XINT_sdiv_out\the\numexpr\XINT_smalldivx_a x.1B!1\langle 8d \rangle! \dots 1\langle 8d \rangle!1\backslash Z! avec  $x = \text{round}(B/2)$ ,
   $1B = 10^8 + B$ .

1403 \def\XINT_div_dosmalldiv
1404   {{\expandafter\XINT_sdiv_out\the\numexpr\XINT_smalldivx_a}}%

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

1405 \def\XINT_div_prepare_b
1406   {\expandafter\XINT_div_prepare_c\romannumeral0\XINT_zeroes_forviii }%
1407 \def\XINT_div_prepare_c #1!%
1408 {%
1409   \XINT_div_prepare_d #1.00000000!{#1}%
1410 }%
1411 \def\XINT_div_prepare_d #1#2#3#4#5#6#7#8#9%
1412 {%
1413   \expandafter\XINT_div_prepare_e\xint_gob_til_dot #1#2#3#4#5#6#7#8#9!%
1414 }%
1415 \def\XINT_div_prepare_e #1!#2!#3#4%
1416 {%
1417   \XINT_div_prepare_f #4#3\X {#1}{#3}%
1418 }%

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

1419 \def\XINT_div_prepare_f #1#2#3#4#5#6#7#8#9\X
1420 {%
1421   \expandafter\XINT_div_prepare_g
1422     \the\numexpr #1#2#3#4#5#6#7#8+\xint_c_i\expandafter
1423     .\the\numexpr (#1#2#3#4#5#6#7#8+\xint_c_i)/\xint_c_ii\expandafter
1424     .\the\numexpr #1#2#3#4#5#6#7#8\expandafter
1425     .\romannumeral0\XINT_sepandrev_andcount
1426     #1#2#3#4#5#6#7#8#9\XINT_rsepbyviii_end_A 2345678%
1427     \XINT_rsepbyviii_end_B 2345678%
1428   \relax\xint_c_ii\xint_c_iii
1429     \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
1430     \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
1431   \X
1432 }%
1433 \def\XINT_div_prepare_g #1.#2.#3.#4.#5\X #6#7#8%
1434 {%
1435   \expandafter\XINT_div_prepare_h
1436   \the\numexpr\expandafter\XINT_sepbyviii_andcount

```

3 Package *xintcore* implementation

```

1437 \romannumeral0\XINT_zeroes_forviii #8#7\R\R\R\R\R\R\R\R{10}0000001\W
1438 #8#7\XINT_sepybviii_end 2345678\relax
1439 \xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!%
1440 \xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_\W
1441 {#1}{#2}{#3}{#4}{#5}{#6}%
1442 }%
1443 \def\XINT_div_prepare_h #1.#2.#3#4#5#6#7#8%
1444 {%
1445 \XINT_div_start_a {#2}{#6}{#1}{#3}{#4}{#5}{#7}{#8}%
1446 }%

```

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.

```

1447 \def\XINT_div_start_a #1#2%
1448 {%
1449 \ifnum #1 < #2
1450 \expandafter\XINT_div_zeroQ
1451 \else
1452 \expandafter\XINT_div_start_b
1453 \fi
1454 {#1}{#2}%
1455 }%
1456 \def\XINT_div_zeroQ #1#2#3#4#5#6#7%
1457 {%
1458 \expandafter\XINT_div_zeroQ_end
1459 \romannumeral0\XINT_unsep_cuzsmall
1460 #31\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W .%
1461 }%
1462 \def\XINT_div_zeroQ_end #1.#2%
1463 {\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»

```

1464 \def\XINT_div_start_b #1#2#3#4#5#6%
1465 {%
1466 \expandafter\XINT_div_finish\the\numexpr
1467 \XINT_div_start_c {#2}.#3.{#6}{#1}{#2}{#4}{#5}{#6}%
1468 }%
1469 \def\XINT_div_finish
1470 {%
1471 \expandafter\XINT_div_finish_a \romannumeral`&&\XINT_div_unsepQ
1472 }%
1473 \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».

```

1474 \def\XINT_div_finish_b #1%
1475 {%
1476 \if0#1%
1477 \expandafter\XINT_div_finish_bRzero
1478 \else
1479 \expandafter\XINT_div_finish_bRpos

```

3 Package *xintcore* implementation

```

1480 \fi
1481 #1%
1482 }%
1483 \def\XINT_div_finish_bRzero 0.#1#2{{#1}{0}}%
1484 \def\XINT_div_finish_bRpos #1.#2#3%
1485 {%
1486 \expandafter\xint_exchangetwo_keepbraces\XINT_div_cleanR #1#3.{#2}%
1487 }%
1488 \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.

```

1489 \def\XINT_div_start_c #1%
1490 {%
1491 \ifnum #1>\xint_c_vi
1492 \expandafter\XINT_div_start_ca
1493 \else
1494 \expandafter\XINT_div_start_cb
1495 \fi {#1}%
1496 }%
1497 \def\XINT_div_start_ca #1#2.#3!#4!#5!#6!#7!#8!#9!%
1498 {%
1499 \expandafter\XINT_div_start_c\expandafter
1500 {\the\numexpr #1-\xint_c_vii}#2#3!#4!#5!#6!#7!#8!#9!.%
1501 }%
1502 \def\XINT_div_start_cb #1%
1503 {\csname XINT_div_start_c_\romannumeral\numexpr#1\endcsname}%
1504 \def\XINT_div_start_c_i #1.#2!%
1505 {\XINT_div_start_c_ #1#2!.}%
1506 \def\XINT_div_start_c_ii #1.#2!#3!%
1507 {\XINT_div_start_c_ #1#2!#3!.}%
1508 \def\XINT_div_start_c_iii #1.#2!#3!#4!%
1509 {\XINT_div_start_c_ #1#2!#3!#4!.}%
1510 \def\XINT_div_start_c_iv #1.#2!#3!#4!#5!%
1511 {\XINT_div_start_c_ #1#2!#3!#4!#5!.}%
1512 \def\XINT_div_start_c_v #1.#2!#3!#4!#5!#6!%
1513 {\XINT_div_start_c_ #1#2!#3!#4!#5!#6!.}%
1514 \def\XINT_div_start_c_vi #1.#2!#3!#4!#5!#6!#7!%
1515 {\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».

```

1516 \def\XINT_div_start_c_ 1#1!#2.#3.#4#5#6%
1517 {%
1518 \XINT_div_I_a {#1}{#4}{1#1!#2}{#6}{000000000}#5{#3}{#6}%
1519 }%

```

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

```

1520 \def\XINT_div_I_a #1#2%
1521 {%

```

3 Package *xintcore* implementation

```

1522 \expandafter\XINT_div_I_b\the\numexpr #1/#2.{#1}{#2}%
1523 }%
1524 \def\XINT_div_I_b #1%
1525 {%
1526 \xint_gob_til_zero #1\XINT_div_I_czero 0\XINT_div_I_c #1%
1527 }%

```

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».

```

1528 \def\XINT_div_I_czero 0\XINT_div_I_c 0.#1#2#3#4#5{1#5\XINT_div_I_g {#3}}%
1529 \def\XINT_div_I_c #1.#2#3%
1530 {%
1531 \expandafter\XINT_div_I_da\the\numexpr #2-#1*#3.#1.{#2}{#3}%
1532 }%

```

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

```

1533 \def\XINT_div_I_da #1.%
1534 {%
1535 \ifnum #1>\xint_c_ix
1536 \expandafter\XINT_div_I_dP
1537 \else
1538 \ifnum #1<\xint_c_
1539 \expandafter\expandafter\expandafter\XINT_div_I_dN
1540 \else
1541 \expandafter\expandafter\expandafter\XINT_div_I_db
1542 \fi
1543 \fi
1544 }%

```

attention très mauvaises notations avec _b et _db.

```

1545 \def\XINT_div_I_dN #1.%
1546 {%
1547 \expandafter\XINT_div_I_b\the\numexpr #1-\xint_c_i.%
1548 }%
1549 \def\XINT_div_I_db #1.#2#3#4#5%
1550 {%
1551 \expandafter\XINT_div_I_dc\expandafter #1%
1552 \romannumeral0\expandafter\XINT_div_sub\expandafter
1553 {\romannumeral0\XINT_rev_nounsep {#4\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1554 {\the\numexpr\XINT_div_verysmallmul #1!#51\Z!}%
1555 \Z {#4}{#5}%
1556 }%

```

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

```

1557 \def\XINT_div_I_dc #1#2%
1558 {%
1559 \if-#2\expandafter\XINT_div_I_dd\else\expandafter\XINT_div_I_de\fi
1560 #1#2%
1561 }%
1562 \def\XINT_div_I_dd #1-\Z

```

3 Package *xintcore* implementation

```

1563 {%
1564   \if #11\expandafter\XINT_div_I_dz\fi
1565   \expandafter\XINT_div_I_dP\the\numexpr #1-\xint_c_i.XX%
1566 }%
1567 \def\XINT_div_I_dz #1XX#2#3#4%
1568 {%
1569   1#4\XINT_div_I_g {#2}%
1570 }%
1571 \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»

1572 \def\XINT_div_I_dP #1.#2#3#4#5#6%
1573 {%
1574   1#6+#1\expandafter\XINT_div_I_g\expandafter
1575   {\romannumeral0\expandafter\XINT_div_sub\expandafter
1576     {\romannumeral0\XINT_rev_nounsep {#4\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1577     {\the\numexpr\XINT_div_verysmallmul #1!#51\Z!}%
1578   }%
1579 }%

  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»

1580 \def\XINT_div_I_g #1#2#3#4#5#6#7%
1581 {%
1582   \expandafter !\the\numexpr
1583   \ifnum#2=#3
1584     \expandafter\XINT_div_exittofinish
1585   \else
1586     \expandafter\XINT_div_I_h
1587   \fi
1588   {#4}#1.#6.{{#4}{#5}{#3}{#2}}{#7}%
1589 }%

  {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»

1590 \def\XINT_div_exittofinish #1#2.#3.#4#5%
1591 {%
1592   1\expandafter\expandafter\expandafter!\expandafter\XINT_unsep_delim
1593   \romannumeral0\XINT_div_unsepR #2#31\R!1\R!1\R!1\R!1\R!1\R!1\R!1\W.%
1594 }%

  ATTENTION DESCRIPTION OBSOLETE. #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»

1595 \def\XINT_div_I_h #1.#2!#3.#4#5%
1596 {%
1597   \XINT_div_II_b #1#2!.{#5}{#4}{#3}{#5}%
1598 }%

```

3 Package *xintcore* implementation

$\{x'y\}\alpha.B, \{\{x'y\}, x, K, L\}, \text{nouveau alpha}', B, \langle c \rangle$

```
1599 \def\XINT_div_II_b #1#2!#3!%
1600 {%
1601     \xint_gob_til_eightzeroes #2\XINT_div_II_skipc 00000000%
1602     \XINT_div_II_c #1{1#2}{#3}%
1603 }%
```

$x'y\{100000000\}\{1<8\}$ reste de $\alpha.\#6=B, \#7=\{\{x'y\}, x, K, L\}, \alpha', B, \langle c \rangle \rightarrow \{x'y\}x, K, L$ (à diminuer de 4), $\{\alpha \text{ sur } K\}B\{q1=00000000\}\{\alpha'\}B, \langle c \rangle$

```
1604 \def\XINT_div_II_skipc 00000000\XINT_div_II_c #1#2#3#4#5.#6#7%
1605 {%
1606     \XINT_div_II_k #7{#4!#5}{#6}\{00000000\}%
1607 }%
```

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

ATTENTION!

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

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

```
1608 \def\XINT_div_II_c #1#2#3#4%
1609 {%
1610     \expandafter\XINT_div_II_d\the\numexpr\XINT_div_xmini
1611     #1.#2!#3!#4!\{#1\}{#2}{#3!#4!}%
1612 }%
1613 \def\XINT_div_xmini #1%
1614 {%
1615     \xint_gob_til_one #1\XINT_div_xmini_a 1\XINT_div_mini #1%
1616 }%
1617 \def\XINT_div_xmini_a 1\XINT_div_mini 1#1%
1618 {%
1619     \xint_gob_til_zero #1\XINT_div_xmini_b 0\XINT_div_mini 1#1%
1620 }%
1621 \def\XINT_div_xmini_b 0\XINT_div_mini 10#1#2#3#4#5#6#7%
1622 {%
1623     \xint_gob_til_zero #7\XINT_div_xmini_c 0\XINT_div_mini 10#1#2#3#4#5#6#7%
1624 }%
```

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

```
1625 \def\XINT_div_xmini_c 0\XINT_div_mini 100000000.50000000!#1!#2!\{#1\}%
```

1 suivi de $q1$ sur huit chiffres! $\#2=x', \#3=y, \#4=\alpha.\#5=B, \{\{x'y\}, x, K, L\}, \alpha', B, \langle c \rangle \rightarrow$ nouvel $\alpha.x', y, B, q1, \{\{x'y\}, x, K, L\}, \alpha', B, \langle c \rangle$

```
1626 \def\XINT_div_II_d 1#1#2#3#4#5!#6#7#8.#9%
1627 {%
1628     \expandafter\XINT_div_II_e
1629     \romannumeral0\expandafter\XINT_div_sub\expandafter
1630     \{\romannumeral0\XINT_rev_nounsep \}{#8\R!\R!\R!\R!\R!\R!\R!\R!\W}%

```


3 Package *xintcore* implementation

```

1631      {\the\numexpr\XINT_div_smallmul_a 1000000000.#1#2#3#4.#5!#91\Z!}%
1632      .{\#6}{\#7}{\#9}{\#1#2#3#4#5}%
1633 }%

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

1634 \def\XINT_div_II_e 1#1!%
1635 {%
1636     \xint_gob_til_eightzeroes #1\XINT_div_II_skipf 000000000%
1637     \XINT_div_II_f 1#1!%
1638 }%

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»

1639 \def\XINT_div_II_skipf 000000000\XINT_div_II_f 1000000000!#1.#2#3#4#5#6%
1640 {%
1641     \XINT_div_II_k #6{\#1}{\#4}{\#5}%
1642 }%

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.

1643 \def\XINT_div_II_f #1!#2!#3.%
1644 {%
1645     \XINT_div_II_fa {\#1!#2!}{\#1!#2!#3}%
1646 }%
1647 \def\XINT_div_II_fa #1#2#3#4%
1648 {%
1649     \expandafter\XINT_div_II_g \the\numexpr\XINT_div_xmini #3.#4!#1{\#2}%
1650 }%

#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»

1651 \def\XINT_div_II_g 1#1#2#3#4#5!#6#7#8%
1652 {%
1653     \expandafter \XINT_div_II_h
1654     \the\numexpr 1#1#2#3#4#5+#8\expandafter\expandafter\expandafter
1655     .\expandafter\expandafter\expandafter
1656     {\expandafter\xint_gob_til_exclam
1657     \romannumeral0\expandafter\XINT_div_sub\expandafter
1658     {\romannumeral0\XINT_rev_nounsep {\#6\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1659     {\the\numexpr\XINT_div_smallmul_a 1000000000.#1#2#3#4.#5!#71\Z!}%}
1660     {\#7}%
1661 }%

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»

1662 \def\XINT_div_II_h 1#1.#2#3#4%
1663 {%
1664     \XINT_div_II_k #4{\#2}{\#3}{\#1}%
1665 }%

```

3 Package *xintcore* implementation

$\{x'y\}x, K, L$ à diminuer de 1, $\alpha, B\{q\}\alpha', B\langle c \rangle \rightarrow$ nouveau $L.K, x', y, x, \alpha.B, q, \alpha', B, \langle c \rangle \rightarrow \{LK\{x'y\}x\}, x, a, \alpha.B, q, \alpha', B, \langle c \rangle$

```
1666 \def\XINT_div_II_k #1#2#3#4#5%
1667 {%
1668   \expandafter\XINT_div_II_l \the\numexpr #4-\xint_c_i.{#3}#1{#2}#5.%
1669 }%
1670 \def\XINT_div_II_l #1.#2#3#4#51#6!%
1671 {%
1672   \XINT_div_II_m {{#1}{#2}}{{#3}{#4}}{{#5}}{{#5}{#6}1#6!%
1673 }%
```

$\{LK\{x'y\}x\}, x, a, \alpha.B\{q\}\alpha'B \rightarrow a, x, \alpha, B, q, L, K, \{x'y\}, x, \alpha', B, \langle c \rangle$

```
1674 \def\XINT_div_II_m #1#2#3#4.#5#6%
1675 {%
1676   \XINT_div_I_a {#3}{#2}{#4}{#5}{#6}#1%
1677 }%
```

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

```
1678 \def\XINT_div_minimulwc_a 1#1.#2.#3!#4#5#6#7#8.%
1679 {%
1680   \expandafter\XINT_div_minimulwc_b
1681   \the\numexpr \xint_c_x^ix+#1+#3*#8.#3*#4#5#6#7+#2*#8.#2*#4#5#6#7.%
1682 }%
1683 \def\XINT_div_minimulwc_b 1#1#2#3#4#5#6.#7.%
1684 {%
1685   \expandafter\XINT_div_minimulwc_c
1686   \the\numexpr \xint_c_x^ix+#1#2#3#4#5+#7.#6.%
1687 }%
1688 \def\XINT_div_minimulwc_c 1#1#2#3#4#5#6.#7.#8.%
1689 {%
1690   1#6#7\expandafter!%
1691   \the\numexpr\expandafter\XINT_div_smallmul_a
1692   \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8.%
1693 }%
1694 \def\XINT_div_smallmul_a #1.#2.#3!1#4!%
1695 {%
1696   \xint_gob_til_Z #4\XINT_div_smallmul_e\Z
1697   \XINT_div_minimulwc_a #1.#2.#3!#4.#2.#3!%
1698 }%
1699 \def\XINT_div_smallmul_e\Z\XINT_div_minimulwc_a 1#1.#2\Z #3!{1\relax #1!}%

```

Special very small multiplication for division. We only need to cater for multiplicands from 1 to 9. The ending is different from standard `verysmallmul`, a zero carry is not suppressed. And no final `1\Z!` is added. If multiplicand is just 1 let's not forget to add the zero carry 100000000! at the end.

```
1700 \def\XINT_div_verysmallmul #1%
1701   {\xint_gob_til_one #1\XINT_div_verysmallisone 1\XINT_div_verysmallmul_a 0.#1}%
1702 \def\XINT_div_verysmallisone 1\XINT_div_verysmallmul_a 0.1!1#11\Z!%
1703   {1\relax #1100000000!}%

```

3 Package *xintcore* implementation

```

1704 \def\XINT_div_verysmallmul_a #1.#2!1#3!%
1705 {%
1706   \xint_gob_til_Z #3\XINT_div_verysmallmul_e\Z
1707   \expandafter\XINT_div_verysmallmul_b
1708   \the\numexpr \xint_c_x^ix+#2*#3+#1.#2!%
1709 }%
1710 \def\XINT_div_verysmallmul_b 1#1#2.%
1711   {1#2\expandafter!\the\numexpr\XINT_div_verysmallmul_a #1.}%
1712 \def\XINT_div_verysmallmul_e\Z #1\Z +#2#3!{1\relax 0000000#2!}%

```

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

```

1713 \def\XINT_div_sub #1#2%
1714 {%
1715   \expandafter\XINT_div_sub_clean
1716   \the\numexpr\expandafter\XINT_div_sub_a\expandafter
1717   1#2\Z!\Z!\Z!\Z!\Z!\W #1\Z!\Z!\Z!\Z!\Z!\W
1718 }%
1719 \def\XINT_div_sub_clean #1-#2#3\W
1720 {%
1721   \if1#2\expandafter\XINT_rev_nounsep\else\expandafter\XINT_div_sub_neg\fi
1722   {#1\R!\R!\R!\R!\R!\R!\R!\R!\W
1723 }%
1724 \def\XINT_div_sub_neg #1\W { -}%
1725 \def\XINT_div_sub_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
1726 {%
1727   \XINT_div_sub_b #1!#6!#2!#7!#3!#8!#4!#9!#5\W
1728 }%
1729 \def\XINT_div_sub_b #1#2#3!#4!%
1730 {%
1731   \xint_gob_til_Z #4\XINT_div_sub_bi \Z
1732   \expandafter\XINT_div_sub_c\the\numexpr#1-#3+1#4-\xint_c_i.%
1733 }%
1734 \def\XINT_div_sub_c 1#1#2.%
1735 {%
1736   1#2\expandafter!\the\numexpr\XINT_div_sub_d #1%
1737 }%
1738 \def\XINT_div_sub_d #1#2#3!#4!%
1739 {%
1740   \xint_gob_til_Z #4\XINT_div_sub_di \Z
1741   \expandafter\XINT_div_sub_e\the\numexpr#1-#3+1#4-\xint_c_i.%
1742 }%
1743 \def\XINT_div_sub_e 1#1#2.%
1744 {%
1745   1#2\expandafter!\the\numexpr\XINT_div_sub_f #1%
1746 }%
1747 \def\XINT_div_sub_f #1#2#3!#4!%
1748 {%
1749   \xint_gob_til_Z #4\XINT_div_sub_fi \Z
1750   \expandafter\XINT_div_sub_g\the\numexpr#1-#3+1#4-\xint_c_i.%
1751 }%
1752 \def\XINT_div_sub_g 1#1#2.%

```

3 Package *xintcore* implementation

```

1753 {%
1754     1#2\expandafter!\the\numexpr\XINT_div_sub_h #1%
1755 }%
1756 \def\XINT_div_sub_h #1#2#3!#4!%
1757 {%
1758     \xint_gob_til_Z #4\XINT_div_sub_hi \Z
1759     \expandafter\XINT_div_sub_i\the\numexpr#1-#3+1#4-\xint_c_i.%
1760 }%
1761 \def\XINT_div_sub_i 1#1#2.%
1762 {%
1763     1#2\expandafter!\the\numexpr\XINT_div_sub_a #1%
1764 }%
1765 \def\XINT_div_sub_bi\Z
1766     \expandafter\XINT_div_sub_c\the\numexpr#1-#2+#3.#4!#5!#6!#7!#8!#9!\Z !\W
1767 {%
1768     \XINT_div_sub_l #1#2!#5!#7!#9!%
1769 }%
1770 \def\XINT_div_sub_di\Z
1771     \expandafter\XINT_div_sub_e\the\numexpr#1-#2+#3.#4!#5!#6!#7!#8\W
1772 {%
1773     \XINT_div_sub_l #1#2!#5!#7!%
1774 }%
1775 \def\XINT_div_sub_fi\Z
1776     \expandafter\XINT_div_sub_g\the\numexpr#1-#2+#3.#4!#5!#6\W
1777 {%
1778     \XINT_div_sub_l #1#2!#5!%
1779 }%
1780 \def\XINT_div_sub_hi\Z
1781     \expandafter\XINT_div_sub_i\the\numexpr#1-#2+#3.#4\W
1782 {%
1783     \XINT_div_sub_l #1#2!%
1784 }%
1785 \def\XINT_div_sub_l #1%
1786 {%
1787     \xint_UDzerofork
1788     #1{-2\relax}%
1789     0\XINT_div_sub_r
1790     \krof
1791 }%
1792 \def\XINT_div_sub_r #1!%
1793 {%
1794     -\ifnum 0#1=\xint_c_ 1\else2\fi\relax
1795 }%

```

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

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

```

1796 \def\XINT_sdiv_out #1\Z!#2!%
1797     {\expandafter

```

3 Package *xintcore* implementation

```
1798 {\romannumeral0\XINT_unsep_cuzsmall#11\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W}%
1799 {#2}}%
```

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

```
1800 \def\XINT_smallldivx_a #1.1#2!1#3!%
1801 {%
1802   \expandafter\XINT_smallldivx_b
1803   \the\numexpr (#3+#1)/#2-\xint_c_i!#1.#2!#3!%
1804 }%
1805 \def\XINT_smallldivx_b #1#2!%
1806 {%
1807   \if0#1\else
1808     \xint_c_x^viii+#1#2\xint_afterfi{\expandafter!\the\numexpr}\fi
1809   \XINT_smallldiv_c #1#2!%
1810 }%
1811 \def\XINT_smallldiv_c #1!#2.#3!#4!%
1812 {%
1813   \expandafter\XINT_smallldiv_d\the\numexpr #4-#1*#3!#2.#3!%
1814 }%
```

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

```
1815 \def\XINT_smallldiv_d #1!#2!1#3#4!%
1816 {%
1817   \xint_gob_til_Z #3\XINT_smallldiv_end \Z
1818   \XINT_smallldiv_e #1!#2!1#3#4!%
1819 }%
1820 \def\XINT_smallldiv_end\Z\XINT_smallldiv_e #1!#2!1\Z!{1!\Z!#1!}%
```

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

```
1821 \def\XINT_smallldiv_e #1!#2.#3!%
1822 {%
1823   \expandafter\XINT_smallldiv_f\the\numexpr
1824   \xint_c_xi_e_viii_mone+#1*\xint_c_x^viii/#3!#2.#3!#1!%
1825 }%
1826 \def\XINT_smallldiv_f 1#1#2#3#4#5#6!#7.#8!%
1827 {%
1828   \xint_gob_til_zero #1\XINT_smallldiv_fz 0%
1829   \expandafter\XINT_smallldiv_g
1830   \the\numexpr\XINT_minimul_a #2#3#4#5.#6!#8!#2#3#4#5#6!#7.#8!%
1831 }%
1832 \def\XINT_smallldiv_fz 0%
1833   \expandafter\XINT_smallldiv_g\the\numexpr\XINT_minimul_a
```

3 Package *xintcore* implementation

```

1834      9999.9999!#1!99999999!#2!0!1#3!%
1835 {%
1836      \XINT_smallldiv_i .#3!\xint_c_!#2!%
1837 }%
1838 \def\XINT_smallldiv_g 1#1!1#2!#3!#4!#5!#6!%
1839 {%
1840      \expandafter\XINT_smallldiv_h\the\numexpr 1#6-#1.#2!#5!#3!#4!%
1841 }%
1842 \def\XINT_smallldiv_h 1#1#2.#3!#4!%
1843 {%
1844      \expandafter\XINT_smallldiv_i\the\numexpr #4-#3+#1-\xint_c_i.#2!%
1845 }%
1846 \def\XINT_smallldiv_i #1.#2!#3!#4.#5!%
1847 {%
1848      \expandafter\XINT_smallldiv_j\the\numexpr (#1#2+#4)/#5-\xint_c_i!#3!#1#2!#4.#5!%
1849 }%
1850 \def\XINT_smallldiv_j #1!#2!%
1851 {%
1852      \xint_c_x^viii+#1+#2\expandafter!\the\numexpr\XINT_smallldiv_k
1853      #1!%
1854 }%

```

On boucle vers \XINT_smallldiv_d.

```

1855 \def\XINT_smallldiv_k #1!#2!#3.#4!%
1856 {%
1857      \expandafter\XINT_smallldiv_d\the\numexpr #2-#1*#4!#3.#4!%
1858 }%

```

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 $1<8\text{chiffres}>1<8\text{chiffres}>!$ avec plus significatif en premier.

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

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

```

1859 \def\XINT_div_mini #1.#2!1#3!%
1860 {%
1861      \expandafter\XINT_div_mini_a\the\numexpr
1862      \xint_c_xi_e_viii_mone+#3*\xint_c_x^viii/#1!#1.#2!#3!%
1863 }%

```

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.

```

1864 \def\XINT_div_mini_a 1#1#2#3#4#5#6!#7.#8!%
1865 {%
1866      \xint_gob_til_zero #1\XINT_div_mini_w 0%
1867      \expandafter\XINT_div_mini_b
1868      \the\numexpr\XINT_minimul_a #2#3#4#5.#6!#7!#2#3#4#5#6!#7.#8!%

```

```

1869 }%
1870 \def\XINT_div_mini_w 0%
1871   \expandafter\XINT_div_mini_b\the\numexpr\XINT_minimul_a
1872   9999.9999!#1!99999999!#2.#3!00000000!#4!%
1873 {%
1874   \xint_c_x^viii_mone+(#4+#3)/#2!%
1875 }%
1876 \def\XINT_div_mini_b 1#1!1#2!#3!#4!#5!#6!%
1877 {%
1878   \expandafter\XINT_div_mini_c
1879   \the\numexpr 1#6-#1.#2!#5!#3!#4!%
1880 }%
1881 \def\XINT_div_mini_c 1#1#2.#3!#4!%
1882 {%
1883   \expandafter\XINT_div_mini_d
1884   \the\numexpr #4-#3+#1-\xint_c_i.#2!%
1885 }%
1886 \def\XINT_div_mini_d #1.#2!#3!#4.#5!%
1887 {%
1888   \xint_c_x^viii_mone+#3+(#1#2+#5)/#4!%
1889 }%

```

3.28 \xintiDivRound, \xintiiDivRound

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

```

1890 \def\xintiDivRound {\romannumeral0\xintidivround }%
1891 \def\xintidivround #1%
1892   {\expandafter\XINT_idivround\romannumeral0\xintnum{#1}\Z }%
1893 \def\xintiiDivRound {\romannumeral0\xintiidivround }%
1894 \def\xintiidivround #1{\expandafter\XINT_iidivround \romannumeral`&&@#1\Z }%
1895 \def\XINT_idivround #1#2\Z #3%
1896   {\expandafter\XINT_iidivround_a\expandafter #1%
1897   \romannumeral0\xintnum{#3}\Z #2\Z }%
1898 \def\XINT_iidivround #1#2\Z #3%
1899   {\expandafter\XINT_iidivround_a\expandafter #1\romannumeral`&&@#3\Z #2\Z }%
1900 \def\XINT_iidivround_a #1#2% #1 de A, #2 de B.
1901 {%
1902   \if0#2\xint_dothis\XINT_iidivround_divbyzero\fi
1903   \if0#1\xint_dothis\XINT_iidivround_aiszero\fi
1904   \if-#2\xint_dothis{\XINT_iidivround_bneg #1}\fi
1905   \xint_orthat{\XINT_iidivround_bpos #1#2}%
1906 }%
1907 \def\XINT_iidivround_divbyzero #1\Z #2\Z {\xintError:DivisionByZero\space 0}%
1908 \def\XINT_iidivround_aiszero #1\Z #2\Z { 0}%
1909 \def\XINT_iidivround_bpos #1%
1910 {%
1911   \xint_UDsignfork
1912   #1{\xintiiopp\XINT_iidivround_pos {}}%
1913   -{\XINT_iidivround_pos #1}%
1914   \krof
1915 }%

```

```

1916 \def\XINT_iidivround_bneg #1%
1917 {%
1918   \xint_UDsignfork
1919     #1{\XINT_iidivround_pos {}}%
1920     -{\xintiioopp\XINT_iidivround_pos #1}%
1921   \krof
1922 }%
1923 \def\XINT_iidivround_pos #1#2\Z #3\Z
1924 {%
1925   \expandafter\expandafter\expandafter\XINT_dsrr
1926   \expandafter\xint_firstoftwo
1927   \romannumeral0\XINT_div_prepare {#2}{#1#30}%
1928   \xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax
1929 }%

```

3.29 \xintiDivTrunc, \xintiiDivTrunc

```

1930 \def\xintiDivTrunc   {\romannumeral0\xintidivtrunc }%
1931 \def\xintidivtrunc   #1{\expandafter\XINT_iidivtrunc\romannumeral0\xintnum{#1}\Z }%
1932 \def\xintiiDivTrunc  {\romannumeral0\xintiidivtrunc }%
1933 \def\xintiidivtrunc  #1{\expandafter\XINT_iidivtrunc \romannumeral`&&@#1\Z }%
1934 \def\XINT_iidivtrunc #1#2\Z #3{\expandafter\XINT_iidivtrunc_a\expandafter #1%
1935   \romannumeral`&&@#3\Z #2\Z }%
1936 \def\XINT_iidivtrunc_a #1#2\Z #1 de A, #2 de B.
1937 {%
1938   \if0#2\xint_dothis\XINT_iidivround_divbyzero\fi
1939   \if0#1\xint_dothis\XINT_iidivround_aiszero\fi
1940   \if-#2\xint_dothis{\XINT_iidivtrunc_bneg #1}\fi
1941   \xint_orthat{\XINT_iidivtrunc_bpos #1#2}%
1942 }%
1943 \def\XINT_iidivtrunc_bpos #1%
1944 {%
1945   \xint_UDsignfork
1946     #1{\xintiioopp\XINT_iidivtrunc_pos {}}%
1947     -{\XINT_iidivtrunc_pos #1}%
1948   \krof
1949 }%
1950 \def\XINT_iidivtrunc_bneg #1%
1951 {%
1952   \xint_UDsignfork
1953     #1{\XINT_iidivtrunc_pos {}}%
1954     -{\xintiioopp\XINT_iidivtrunc_pos #1}%
1955   \krof
1956 }%
1957 \def\XINT_iidivtrunc_pos #1#2\Z #3\Z%
1958   {\expandafter\xint_firstoftwo_thenstop
1959   \romannumeral0\XINT_div_prepare {#2}{#1#3}}%

```

3.30 \xintiMod, \xintiiMod

```

1960 \def\xintiMod   {\romannumeral0\xintimod }%
1961 \def\xintimod   #1{\expandafter\XINT_iimod\romannumeral0\xintnum{#1}\Z }%
1962 \def\xintiiMod  {\romannumeral0\xintiimod }%
1963 \def\xintiimod  #1{\expandafter\XINT_iimod \romannumeral`&&@#1\Z }%

```


3 Package *xintcore* implementation

```
1964 \def\XINT_iimod #1#2\Z #3{\expandafter\XINT_iimod_a\expandafter #1%
1965         \romannumeral`&&@#3\Z #2\Z }%
1966 \def\XINT_iimod_a #1#2% #1 de A, #2 de B.
1967 {%
1968     \if0#2\xint_dothis\XINT_iidivround_divbyzero\fi
1969     \if0#1\xint_dothis\XINT_iidivround_aiszero\fi
1970     \if-#2\xint_dothis{\XINT_iimod_bneg #1}\fi
1971     \xint_orthat{\XINT_iimod_bpos #1#2}%
1972 }%
1973 \def\XINT_iimod_bpos #1%
1974 {%
1975     \xint_UDsignfork
1976     #1{\xintiiopt\XINT_iimod_pos {}}%
1977     -{\XINT_iimod_pos #1}%
1978     \krof
1979 }%
1980 \def\XINT_iimod_bneg #1%
1981 {%
1982     \xint_UDsignfork
1983     #1{\xintiiopt\XINT_iimod_pos {}}%
1984     -{\XINT_iimod_pos #1}%
1985     \krof
1986 }%
1987 \def\XINT_iimod_pos #1#2\Z #3\Z%
1988     {\expandafter\xint_seconddoftwo_thenstop\romannumeral0\XINT_div_prepare
1989     {#2}{#1#3}}%
```

3.31 “Load *xintfrac*” macros

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

```
1990 \catcode`! 11
1991 \def\xintAbs {\Did_you_mean_iiAbs?or_load_xintfrac!}%
1992 \def\xintOpp {\Did_you_mean_iiOpp?or_load_xintfrac!}%
1993 \def\xintAdd {\Did_you_mean_iiAdd?or_load_xintfrac!}%
1994 \def\xintSub {\Did_you_mean_iiSub?or_load_xintfrac!}%
1995 \def\xintMul {\Did_you_mean_iiMul?or_load_xintfrac!}%
1996 \def\xintPow {\Did_you_mean_iiPow?or_load_xintfrac!}%
1997 \def\xintSqr {\Did_you_mean_iiSqr?or_load_xintfrac!}%
1998 \def\xintQuo {\Removed!use_xintiQuo_or_xintiiQuo!}%
1999 \def\xintRem {\Removed!use_xintiRem_or_xintiiRem!}%
2000 \XINT_restorecatcodes_endinput%
```

4 Package `xint` implementation

.1	Package identification	99	.24	<code>\xintAND</code> , <code>\xintOR</code> , <code>\xintXOR</code>	109
.2	More token management	99	.25	<code>\xintANDof</code>	109
.3	<code>\xintSgnFork</code>	99	.26	<code>\xintORof</code>	110
.4	<code>\xintIsOne</code> , <code>\xintiiIsOne</code>	100	.27	<code>\xintXORof</code>	110
.5	<code>\xintRev</code>	100	.28	<code>\xintGeq</code> , <code>\xintiiGeq</code>	110
.6	<code>\xintLen</code>	100	.29	<code>\xintiMax</code> , <code>\xintiiMax</code>	113
.7	<code>\xintBool</code> , <code>\xintToggle</code>	101	.30	<code>\xintiMaxof</code> , <code>\xintiiMaxof</code>	114
.8	<code>\xintifSgn</code> , <code>\xintiiifSgn</code>	101	.31	<code>\xintiMin</code> , <code>\xintiiMin</code>	115
.9	<code>\xintifZero</code> , <code>\xintifNotZero</code> , <code>\xintii-</code> <code>ifZero</code> , <code>\xintiiifNotZero</code>	101	.32	<code>\xintiMinof</code> , <code>\xintiiMinof</code>	116
.10	<code>\xintifOne</code> , <code>\xintiiifOne</code>	102	.33	<code>\xintiiSum</code>	117
.11	<code>\xintifTrueAelseB</code> , <code>\xintifFalseAelseB</code>	102	.34	<code>\xintiiPrd</code>	117
.12	<code>\xintifCmp</code> , <code>\xintiiifCmp</code>	103	.35	<code>\xintMON</code> , <code>\xintMMON</code> , <code>\xintiiMON</code> , <code>\xinti-</code> <code>immon</code>	117
.13	<code>\xintifEq</code> , <code>\xintiiifEq</code>	103	.36	<code>\xintOdd</code> , <code>\xintiiOdd</code> , <code>\xintEven</code> , <code>\xinti-</code> <code>iEven</code>	118
.14	<code>\xintifGt</code> , <code>\xintiiifGt</code>	103	.37	<code>\xintDSH</code> , <code>\xintDSHr</code>	119
.15	<code>\xintifLt</code> , <code>\xintiiifLt</code>	104	.38	<code>\xintDSx</code>	120
.16	<code>\xintifOdd</code> , <code>\xintiiifOdd</code>	104	.39	<code>\xintDecSplit</code> , <code>\xintDecSplitL</code> , <code>\xint-</code> <code>DecSplitR</code>	121
.17	<code>\xintCmp</code> , <code>\xintiiCmp</code>	105	.40	<code>\xintiiSqrt</code> , <code>\xintiiSqrtR</code> , <code>\xinti-</code> <code>iSquareRoot</code>	123
.18	<code>\xintEq</code> , <code>\xintGt</code> , <code>\xintLt</code>	108	.41	<code>\xintiiBinomial</code> , <code>\xintiBinomial</code>	129
.19	<code>\xintNeq</code> , <code>\xintGtorEq</code> , <code>\xintLtorEq</code>	108	.42	<code>\xintiiPFactorial</code> , <code>\xintiPFactorial</code>	135
.20	<code>\xintiiEq</code> , <code>\xintiiGt</code> , <code>\xintiiLt</code>	108	.43	<code>\xintiiE</code>	138
.21	<code>\xintiiNeq</code> , <code>\xintiiGtorEq</code> , <code>\xintiiLtorEq</code>	108	.44	“Load <code>xintfrac</code> ” macros	139
.22	<code>\xintIsZero</code> , <code>\xintIsNotZero</code> , <code>\xintii-</code> <code>IsZero</code> , <code>\xintiiIsNotZero</code>	109			
.23	<code>\xintIsTrue</code> , <code>\xintNot</code> , <code>\xintIsFalse</code>	109			

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

```

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

```

```

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

```

4.1 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xint}%
46 [2016/12/13 1.2i 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`.

`\XINT_isOne` rewritten for 1.2g. Works with expanded strict integers, positive or negative.

```

65 \def\xintiiIsOne {\romannumeral0\xintiiisone }%
66 \def\xintiiisone #1{\expandafter\XINT_isone\romannumeral`&&@#1XY}%
67 \def\xintIsOne {\romannumeral0\xintisone }%
68 \def\xintisone #1{\expandafter\XINT_isone\romannumeral0\xintnum{#1}XY}%
69 \def\XINT_isone #1#2#3Y%
70 {%
71   \unless\if#2X\xint_dothis{ 0}\fi
72   \unless\if#11\xint_dothis{ 0}\fi
73   \xint_orthat{ 1}%
74 }%
75 \def\XINT_isOne #1{\XINT_iSone#1XY}%
76 \def\XINT_iSone #1#2#3Y%
77 {%
78   \unless\if#2X\xint_dothis0\fi
79   \unless\if#11\xint_dothis0\fi
80   \xint_orthat1%
81 }%

```

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.

```

82 \let\xintRev\xintReverseDigits

```

4.6 `\xintLen`

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

```

83 \def\xintLen {\romannumeral0\xintlen }%
84 \def\xintlen #1%
85 {%
86   \expandafter\XINT_len_fork
87   \romannumeral0\xintnum{#1}\xint_relax\xint_relax\xint_relax\xint_relax
88   \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
89   \xint_c_i\xint_c_ii\xint_c_iii\xint_c_iv
90   \xint_c_v\xint_c_vi\xint_c_vii\xint_c_viii\xint_c_ix\xint_bye\relax
91 }%
92 \edef\XINT_len_fork #1%
93 {%
94   \noexpand\expandafter\space
95   \unexpanded{\the\numexpr\xint_c_ix\expandafter
96     \XINT_length_loop\xint_UDsignfork#1}{-}{#1}\krof}%
97 }%

```

4.7 `\xintBool`, `\xintToggle`

1.09c

```
98 \def\xintBool #1{\romannumeral`&&@%
99         \csname if#1\endcsname\expandafter1\else\expandafter0\fi }%
100 \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`)

```
101 \def\xintifSgn {\romannumeral0\xintifsgn }%
102 \def\xintifsgn #1%
103 {%
104     \ifcase \xintSgn{#1}
105         \expandafter\xint_secondofthree_thenstop
106         \or\expandafter\xint_thirdofthree_thenstop
107         \else\expandafter\xint_firstofthree_thenstop
108     \fi
109 }%
110 \def\xintiiifSgn {\romannumeral0\xintiiifsgn }%
111 \def\xintiiifsgn #1%
112 {%
113     \ifcase \xintiiSgn{#1}
114         \expandafter\xint_secondofthree_thenstop
115         \or\expandafter\xint_thirdofthree_thenstop
116         \else\expandafter\xint_firstofthree_thenstop
117     \fi
118 }%
```

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.

```
119 \def\xintifZero {\romannumeral0\xintifzero }%
120 \def\xintifzero #1%
121 {%
122     \if0\xintSgn{#1}%
123         \expandafter\xint_firstoftwo_thenstop
124     \else
125         \expandafter\xint_secondoftwo_thenstop
126     \fi
127 }%
128 \def\xintifNotZero {\romannumeral0\xintifnotzero }%
129 \def\xintifnotzero #1%
130 {%
```

```

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

```

4.10 `\xintifOne`, `\xintiiifOne`

added in 1.09i. 1.1a adds `\xintiiifOne`.

```

155 \def\xintiiifOne {\romannumeral0\xintiiifone }%
156 \def\xintiiifone #1%
157 {%
158 \if1\xintiiIsOne{#1}%
159 \expandafter\xint_firstoftwo_thenstop
160 \else
161 \expandafter\xint_secondoftwo_thenstop
162 \fi
163 }%
164 \def\xintifOne {\romannumeral0\xintifone }%
165 \def\xintifone #1%
166 {%
167 \if1\xintIsOne{#1}%
168 \expandafter\xint_firstoftwo_thenstop
169 \else
170 \expandafter\xint_secondoftwo_thenstop
171 \fi
172 }%

```

4.11 `\xintifTrueAelseB`, `\xintifFalseAelseB`

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

```

173 \let\xintifTrueAelseB\xintifNotZero

```

```
174 \let\xintifFalseAelseB\xintifZero
175 %%\let\xintifTrue\xintifNotZero      % now removed
176 %%\let\xintifTrueFalse\xintifNotZero % now removed
```

4.12 `\xintifCmp`, `\xintiiifCmp`

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

```
177 \def\xintifCmp {\romannumeral0\xintifcmp }%
178 \def\xintifcmp #1#2%
179 {%
180     \ifcase\xintCmp {#1}{#2}
181         \expandafter\xint_secondofthree_thenstop
182         \or\expandafter\xint_thirdofthree_thenstop
183         \else\expandafter\xint_firstofthree_thenstop
184     \fi
185 }%
186 \def\xintiiifCmp {\romannumeral0\xintiiifcmp }%
187 \def\xintiiifcmp #1#2%
188 {%
189     \ifcase\xintiiCmp {#1}{#2}
190         \expandafter\xint_secondofthree_thenstop
191         \or\expandafter\xint_thirdofthree_thenstop
192         \else\expandafter\xint_firstofthree_thenstop
193     \fi
194 }%
```

4.13 `\xintifEq`, `\xintiiifEq`

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

```
195 \def\xintifEq {\romannumeral0\xintifeq }%
196 \def\xintifeq #1#2%
197 {%
198     \if0\xintCmp{#1}{#2}%
199         \expandafter\xint_firstoftwo_thenstop
200         \else\expandafter\xint_secondoftwo_thenstop
201     \fi
202 }%
203 \def\xintiiifEq {\romannumeral0\xintiiifeq }%
204 \def\xintiiifeq #1#2%
205 {%
206     \if0\xintiiCmp{#1}{#2}%
207         \expandafter\xint_firstoftwo_thenstop
208         \else\expandafter\xint_secondoftwo_thenstop
209     \fi
210 }%
```

4.14 `\xintifGt`, `\xintiiifGt`

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

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

```

212 \def\xintifgt #1#2%
213 {%
214     \if1\xintCmp{#1}{#2}%
215         \expandafter\xint_firstoftwo_thenstop
216     \else\expandafter\xint_secondoftwo_thenstop
217     \fi
218 }%
219 \def\xintiiifGt {\romannumeral0\xintiiifgt }%
220 \def\xintiiifgt #1#2%
221 {%
222     \if1\xintiiCmp{#1}{#2}%
223         \expandafter\xint_firstoftwo_thenstop
224     \else\expandafter\xint_secondoftwo_thenstop
225     \fi
226 }%

```

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

```

227 \def\xintifLt {\romannumeral0\xintiflt }%
228 \def\xintiflt #1#2%
229 {%
230     \ifnum\xintCmp{#1}{#2}<\xint_c_
231         \expandafter\xint_firstoftwo_thenstop
232     \else \expandafter\xint_secondoftwo_thenstop
233     \fi
234 }%
235 \def\xintiiifLt {\romannumeral0\xintiiiflt }%
236 \def\xintiiiflt #1#2%
237 {%
238     \ifnum\xintiiCmp{#1}{#2}<\xint_c_
239         \expandafter\xint_firstoftwo_thenstop
240     \else \expandafter\xint_secondoftwo_thenstop
241     \fi
242 }%

```

4.16 \xintifOdd, \xintiiifOdd

1.09e. Restyled in 1.09i. 1.1a adds \xintiiifOdd.

```

243 \def\xintiiifOdd {\romannumeral0\xintiiifodd }%
244 \def\xintiiifodd #1%
245 {%
246     \if\xintiiOdd{#1}1%
247         \expandafter\xint_firstoftwo_thenstop
248     \else
249         \expandafter\xint_secondoftwo_thenstop
250     \fi
251 }%
252 \def\xintifOdd {\romannumeral0\xintifodd }%
253 \def\xintifodd #1%
254 {%

```



```

255 \if\xintOdd{#1}1%
256 \expandafter\xint_firstoftwo_thenstop
257 \else
258 \expandafter\xint_secondoftwo_thenstop
259 \fi
260 }%

```

4.17 *\xintCmp*, *\xintiicmp*

Faster than doing the full subtraction.

```

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

```

4 Package *xint* implementation

```

303 }%
304 \def\XINT_cmp_plusminus #1\Z #2\Z{ 1}%
305 \def\XINT_cmp_minusplus #1\Z #2\Z{ -1}%
306 \def\XINT_cmp_minusminus
307 --{\expandafter\XINT_opp\romannumeral0\XINT_cmp_plusplus {}{}}%
308 \def\XINT_cmp_plusplus #1#2#3\Z
309 {%
310 \expandafter\XINT_cmp_pp
311 \romannumeral0\expandafter\XINT_sepandrev_andcount
312 \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
313 #2#3\XINT_rsepbyviii_end_A 2345678%
314 \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_iii
315 \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
316 \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
317 \X #1%
318 }%
319 \def\XINT_cmp_pp #1.#2\X #3\Z
320 {%
321 \expandafter\XINT_cmp_checklengths
322 \the\numexpr #1\expandafter.%
323 \romannumeral0\expandafter\XINT_sepandrev_andcount
324 \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
325 #3\XINT_rsepbyviii_end_A 2345678%
326 \XINT_rsepbyviii_end_B 2345678\relax \xint_c_ii\xint_c_iii
327 \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
328 \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
329 \Z!\Z!\Z!\Z!\Z!\W #2\Z!\Z!\Z!\Z!\Z!\W
330 }%
331 \def\XINT_cmp_checklengths #1.#2.%
332 {%
333 \ifnum #1=#2
334 \expandafter\xint_firstoftwo
335 \else
336 \expandafter\xint_secondoftwo
337 \fi
338 \XINT_cmp_aa {\XINT_cmp_distinctlengths {#1}{#2}}%
339 }%
340 \def\XINT_cmp_distinctlengths #1#2#3\W #4\W
341 {%
342 \ifnum #1>#2
343 \expandafter\xint_firstoftwo
344 \else
345 \expandafter\xint_secondoftwo
346 \fi
347 { -1}{ 1}%
348 }%
349 \def\XINT_cmp_aa {\expandafter\XINT_cmp_w\the\numexpr\XINT_cmp_a \xint_c_i }%
350 \def\XINT_cmp_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
351 {%
352 \XINT_cmp_b #1!#6!#2!#7!#3!#8!#4!#9!#5\W
353 }%
354 \def\XINT_cmp_b #1#2#3!#4!%

```

4 Package *xint* implementation

```

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

```

```

407 {%
408   \XINT_cmp_k #1#2!%
409 }%
410 \def\XINT_cmp_k #1#2\W
411 {%
412   \xint_UDzerofork
413   #1{-1\relax \XINT_cmp_greater}%
414   0{-1\relax \XINT_cmp_lessequal}%
415   \krof
416 }%
417 \def\XINT_cmp_w #1-1#2{#2#11\Z!\W}%
418 \def\XINT_cmp_greater #1\Z!\W{ 1}%
419 \def\XINT_cmp_lessequal 1#1!%
420   {\xint_gob_til_Z #1\XINT_cmp_equal\Z
421    \xint_gob_til_eightzeroes #1\XINT_cmp_continue 00000000%
422    \XINT_cmp_less }%
423 \def\XINT_cmp_less #1\W { -1}%
424 \def\XINT_cmp_continue 00000000\XINT_cmp_less {\XINT_cmp_lessequal }%
425 \def\XINT_cmp_equal\Z\xint_gob_til_eightzeroes\Z\XINT_cmp_continue
426   00000000\XINT_cmp_less\W { 0}%

```

4.18 \xintEq, \xintGt, \xintLt

1.09a.

```

427 \def\xintEq {\romannumeral0\xinteq }\def\xinteq #1#2{\xintifeq{#1}{#2}{1}{0}}%
428 \def\xintGt {\romannumeral0\xintgt }\def\xintgt #1#2{\xintifgt{#1}{#2}{1}{0}}%
429 \def\xintLt {\romannumeral0\xintlft }\def\xintlft #1#2{\xintiflt{#1}{#2}{1}{0}}%

```

4.19 \xintNeq, \xintGtorEq, \xintLtorEq

1.1. Pour *xintexpr*. No lowercase macros

```

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

```

4.20 \xintiiEq, \xintiiGt, \xintiiLt

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

```

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

```

4.21 \xintiiNeq, \xintiiGtorEq, \xintiiLtorEq

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

```

436 \def\xintiiLtorEq #1#2{\romannumeral0\xintiiifgt {#1}{#2}{0}{1}}%
437 \def\xintiiGtorEq #1#2{\romannumeral0\xintiiiflt {#1}{#2}{0}{1}}%
438 \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`

```

439 \def\xintIsZero {\romannumeral0\xintiszero }%
440 \def\xintiszero #1{\if0\xintSgn{#1}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%
441 \def\xintIsNotZero {\romannumeral0\xintisnotzero }%
442 \def\xintisnotzero
443     #1{\if0\xintSgn{#1}\xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi}%
444 \def\xintiiIsZero {\romannumeral0\xintiiiszero }%
445 \def\xintiiiszero #1{\if0\xintiiSgn{#1}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%
446 \def\xintiiIsNotZero {\romannumeral0\xintiiisnotzero }%
447 \def\xintiiisnotzero
448     #1{\if0\xintiiSgn{#1}\xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi}%

```

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

1.09c

```

449 \let\xintIsTrue\xintIsNotZero
450 \let\xintNot\xintIsZero
451 \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

```

452 \def\xintAND {\romannumeral0\xintand }%
453 \def\xintand #1#2{\if0\xintSgn{#1}\expandafter\xint_firstoftwo
454     \else\expandafter\xint_secondoftwo\fi
455     { 0}{\xintisnotzero{#2}}}%
456 \def\xintOR {\romannumeral0\xintor }%
457 \def\xintor #1#2{\if0\xintSgn{#1}\expandafter\xint_firstoftwo
458     \else\expandafter\xint_secondoftwo\fi
459     {\xintisnotzero{#2}}{ 1}}%
460 \def\xintXOR {\romannumeral0\xintxor }%
461 \def\xintxor #1#2{\if\xintIsZero{#1}\xintIsZero{#2}%
462     \xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi }%

```

4.25 `\xintANDof`

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

```

463 \def\xintANDof {\romannumeral0\xintandof }%
464 \def\xintandof #1{\expandafter\xint_andof_a\romannumeral`&&#1\relax }%
465 \def\xint_andof_a #1{\expandafter\xint_andof_b\romannumeral`&&#1\Z }%
466 \def\xint_andof_b #1%
467     {\xint_gob_til_relax #1\xint_andof_e\relax\xint_andof_c #1}%
468 \def\xint_andof_c #1\Z
469     {\xintifTrueAelseB {#1}{\xint_andof_a}{\xint_andof_no}}%
470 \def\xint_andof_no #1\relax { 0}%
471 \def\xint_andof_e #1\Z { 1}%

```

4.26 `\xintORof`

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

```

472 \def\xintORof      {\romannumeral0\xintorof }%
473 \def\xintorof      #1{\expandafter\XINT_orof_a\romannumeral`&&@#1\relax }%
474 \def\XINT_orof_a #1{\expandafter\XINT_orof_b\romannumeral`&&@#1\Z }%
475 \def\XINT_orof_b #1%
476     {\xint_gob_til_relax #1\XINT_orof_e\relax\XINT_orof_c #1}%
477 \def\XINT_orof_c #1\Z
478     {\xintifTrueAelseB {#1}{\XINT_orof_yes}{\XINT_orof_a}}%
479 \def\XINT_orof_yes #1\relax { 1}%
480 \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

```

481 \def\xintXORof      {\romannumeral0\xintxorof }%
482 \def\xintxorof      #1{\expandafter\XINT_xorof_a\expandafter
483     0\romannumeral`&&@#1\relax }%
484 \def\XINT_xorof_a #1#2{\expandafter\XINT_xorof_b\romannumeral`&&@#2\Z #1}%
485 \def\XINT_xorof_b #1%
486     {\xint_gob_til_relax #1\XINT_xorof_e\relax\XINT_xorof_c #1}%
487 \def\XINT_xorof_c #1\Z #2%
488     {\xintifTrueAelseB {#1}{\if #20\xint_afterfi{\XINT_xorof_a 1}%
489     \else\xint_afterfi{\XINT_xorof_a 0}\fi}%
490     {\XINT_xorof_a #2}%
491     }%
492 \def\XINT_xorof_e #1\Z #2{ #2}%

```

4.28 `\xintGeq`, `\xintiGeq`

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

```

493 \def\xintGeq      {\romannumeral0\xintgeq }%
494 \def\xintgeq      #1{\expandafter\XINT_geq\romannumeral0\xintnum{#1}\Z }%
495 \def\xintiGeq      {\romannumeral0\xintiigeq }%
496 \def\xintiigeq #1{\expandafter\XINT_iigeq\romannumeral`&&@#1\Z }%
497 \def\XINT_iigeq #1#2\Z #3%
498 {%
499     \expandafter\XINT_geq_fork\expandafter #1\romannumeral`&&@#3\Z #2\Z
500 }%
501 \let\XINT_geq_pre \xintiigeq % TEMPORAIRE (oui, mais depuis quand ?)
502 \let\XINT_Geq \xintGeq      % TEMPORAIRE ATTENTION FAIT xintNum (et alors?)
503 \def\XINT_geq #1#2\Z #3%
504 {%
505     \expandafter\XINT_geq_fork\expandafter #1\romannumeral0\xintnum{#3}\Z #2\Z
506 }%
507 \def\XINT_geq_fork #1#2%
508 {%
509     \xint_UDzerofork
510     #1\XINT_geq_firstiszero

```

4 Package *xint* implementation

```

511      #2\XINT_geq_secondiszero
512      0{ }%
513      \krof
514      \xint_UDsignsfork
515          #1#2\XINT_geq_minusminus
516          #1-\XINT_geq_minusplus
517          #2-\XINT_geq_plusminus
518          --\XINT_geq_plusplus
519      \krof #1#2%
520 }%
521 \def\XINT_geq_firstiszero #1\krof 0#2#3\Z #4\Z
522      {\xint_UDzerofork #2{ 1}0{ 0}\krof }%
523 \def\XINT_geq_secondiszero #1\krof #20#3\Z #4\Z { 1}%
524 \def\XINT_geq_plusminus #1-{\XINT_geq_plusplus #1{ }}%
525 \def\XINT_geq_minusplus #-1{\XINT_geq_plusplus { }#1}%
526 \def\XINT_geq_minusminus --{\XINT_geq_plusplus { }{ }}%
527 \def\XINT_geq_plusplus #1#2#3\Z #4\Z {\XINT_geq_pp #1#4\Z #2#3\Z }%
528 \def\XINT_geq_pp #1\Z
529 {%
530     \expandafter\XINT_geq_pp_a
531     \romannumeral0\expandafter\XINT_sepandrev_andcount
532     \romannumeral0\XINT_zeroes_forviii #1\R\R\R\R\R\R\R\R{10}0000001\W
533     #1\XINT_rsepbyviii_end_A 2345678%
534     \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_iii
535     \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
536     \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
537     \X
538 }%
539 \def\XINT_geq_pp_a #1.#2\X #3\Z
540 {%
541     \expandafter\XINT_geq_checklengths
542     \the\numexpr #1\expandafter.%
543     \romannumeral0\expandafter\XINT_sepandrev_andcount
544     \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
545     #3\XINT_rsepbyviii_end_A 2345678%
546     \XINT_rsepbyviii_end_B 2345678\relax \xint_c_ii\xint_c_iii
547     \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
548     \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
549     \Z!\Z!\Z!\Z!\Z!\W #2\Z!\Z!\Z!\Z!\Z!\W
550 }%
551 \def\XINT_geq_checklengths #1.#2.%
552 {%
553     \ifnum #1=#2
554         \expandafter\xint_firstoftwo
555     \else
556         \expandafter\xint_secondoftwo
557     \fi
558     \XINT_geq_aa {\XINT_geq_distinctlengths {#1}{#2}}%
559 }%
560 \def\XINT_geq_distinctlengths #1#2#3\W #4\W
561 {%
562     \ifnum #1>#2

```

4 Package *xint* implementation

```

563     \expandafter\xint_firstoftwo
564   \else
565     \expandafter\xint_secondoftwo
566   \fi
567   { 1}{ 0}%
568 }%
569 \def\xint_geq_aa {\expandafter\xint_geq_w\the\numexpr\xint_geq_a \xint_c_i }%
570 \def\xint_geq_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
571 {%
572   \xint_geq_b #1!#6!#2!#7!#3!#8!#4!#9!#5\W
573 }%
574 \def\xint_geq_b #1#2#3!#4!%
575 {%
576   \xint_gob_til_Z #2\xint_geq_bi \Z
577   \expandafter\xint_geq_c\the\numexpr#1+1#4-#3-\xint_c_i.%
578 }%
579 \def\xint_geq_c 1#1#2.%
580 {%
581   1#2\expandafter!\the\numexpr\xint_geq_d #1%
582 }%
583 \def\xint_geq_d #1#2#3!#4!%
584 {%
585   \xint_gob_til_Z #2\xint_geq_di \Z
586   \expandafter\xint_geq_e\the\numexpr#1+1#4-#3-\xint_c_i.%
587 }%
588 \def\xint_geq_e 1#1#2.%
589 {%
590   1#2\expandafter!\the\numexpr\xint_geq_f #1%
591 }%
592 \def\xint_geq_f #1#2#3!#4!%
593 {%
594   \xint_gob_til_Z #2\xint_geq_fi \Z
595   \expandafter\xint_geq_g\the\numexpr#1+1#4-#3-\xint_c_i.%
596 }%
597 \def\xint_geq_g 1#1#2.%
598 {%
599   1#2\expandafter!\the\numexpr\xint_geq_h #1%
600 }%
601 \def\xint_geq_h #1#2#3!#4!%
602 {%
603   \xint_gob_til_Z #2\xint_geq_hi \Z
604   \expandafter\xint_geq_i\the\numexpr#1+1#4-#3-\xint_c_i.%
605 }%
606 \def\xint_geq_i 1#1#2.%
607 {%
608   1#2\expandafter!\the\numexpr\xint_geq_a #1%
609 }%
610 \def\xint_geq_bi\Z
611   \expandafter\xint_geq_c\the\numexpr#1+1#2-#3.#4!#5!#6!#7!#8!#9!\Z !\W
612 {%
613   \xint_geq_k #1#2!#5!#7!#9!%
614 }%

```



```

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

```

4.29 \xintiMax, \xintiiMax

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

```

638 \def\xintiMax {\romannumeral0\xintimax }%
639 \def\xintimax #1%
640 {%
641   \expandafter\xint_max\expandafter {\romannumeral0\xintnum{#1}}%
642 }%
643 \def\xint_max #1#2%
644 {%
645   \expandafter\XINT_max_pre\expandafter {\romannumeral0\xintnum{#2}}{#1}%
646 }%
647 \def\xintiiMax {\romannumeral0\xintiimax }%
648 \def\xintiimax #1%
649 {%
650   \expandafter\xint_iimax\expandafter {\romannumeral`&&@#1}%
651 }%
652 \def\xint_iimax #1#2%
653 {%
654   \expandafter\XINT_max_pre\expandafter {\romannumeral`&&@#2}{#1}%
655 }%
656 \def\XINT_max_pre #1#2{\XINT_max_fork #1\Z #2\Z {#2}{#1}}%
657 \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**

```

658 \def\XINT_max_fork #1#2\Z #3#4\Z
659 {%

```

```

660 \xint_UDsignsfork
661     #1#3\XINT_max_minusminus % A < 0, B < 0
662     #1-\XINT_max_minusplus % B < 0, A >= 0
663     #3-\XINT_max_plusminus % A < 0, B >= 0
664     --{\xint_UDzerosfork
665         #1#3\XINT_max_zerozero % A = B = 0
666         #10\XINT_max_zeroplus % B = 0, A > 0
667         #30\XINT_max_pluszero % A = 0, B > 0
668         00\XINT_max_plusplus % A, B > 0
669     \krof }%
670 \krof
671 {#2}{#4}#1#3%
672 }%

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

673 \def\XINT_max_zerozero #1#2#3#4{\xint_firstoftwo_thenstop }%
674 \def\XINT_max_zeroplus #1#2#3#4{\xint_firstoftwo_thenstop }%
675 \def\XINT_max_pluszero #1#2#3#4{\xint_secondoftwo_thenstop }%
676 \def\XINT_max_minusplus #1#2#3#4{\xint_firstoftwo_thenstop }%
677 \def\XINT_max_plusminus #1#2#3#4{\xint_secondoftwo_thenstop }%
678 \def\XINT_max_plusplus #1#2#3#4%
679 {%
680     \ifodd\XINT_Geq {#4#2}{#3#1}
681     \expandafter\xint_firstoftwo_thenstop
682     \else
683     \expandafter\xint_secondoftwo_thenstop
684     \fi
685 }%

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

686 \def\XINT_max_minusminus #1#2#3#4%
687 {%
688     \ifodd\XINT_Geq {#1}{#2}
689     \expandafter\xint_firstoftwo_thenstop
690     \else
691     \expandafter\xint_secondoftwo_thenstop
692     \fi
693 }%

```

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.

```

694 \def\xintiMaxof {\romannumeral0\xintimaxof }%
695 \def\xintimaxof #1{\expandafter\XINT_imaxof_a\romannumeral`&&@#1\relax }%
696 \def\XINT_imaxof_a #1{\expandafter\XINT_imaxof_b\romannumeral0\xintnum{#1}\Z }%
697 \def\XINT_imaxof_b #1\Z #2%
698     {\expandafter\XINT_imaxof_c\romannumeral`&&@#2\Z {#1}\Z}%
699 \def\XINT_imaxof_c #1%
700     {\xint_gob_til_relax #1\XINT_imaxof_e\relax\XINT_imaxof_d #1}%
701 \def\XINT_imaxof_d #1\Z

```

```

702      {\expandafter\XINT_imaxof_b\romannumeral0\xintimax {#1}}%
703 \def\XINT_imaxof_e #1\Z #2\Z { #2}%
704 \def\xintiMaxof      {\romannumeral0\xintiimaxof }%
705 \def\xintiimaxof     #1{\expandafter\XINT_iimaxof_a\romannumeral`&&@#1\relax }%
706 \def\XINT_iimaxof_a #1{\expandafter\XINT_iimaxof_b\romannumeral`&&@#1\Z }%
707 \def\XINT_iimaxof_b #1\Z #2%
708      {\expandafter\XINT_iimaxof_c\romannumeral`&&@#2\Z {#1}\Z}%
709 \def\XINT_iimaxof_c #1%
710      {\xint_gob_til_relax #1\XINT_iimaxof_e\relax\XINT_iimaxof_d #1}%
711 \def\XINT_iimaxof_d #1\Z
712      {\expandafter\XINT_iimaxof_b\romannumeral0\xintiimax {#1}}%
713 \def\XINT_iimaxof_e #1\Z #2\Z { #2}%

```

4.31 `\xintiMin`, `\xintiMin`

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

```

714 \def\xintiMin {\romannumeral0\xintimin }%
715 \def\xintimin #1%
716 {%
717   \expandafter\xint_min\expandafter {\romannumeral0\xintnum{#1}}%
718 }%
719 \def\xint_min #1#2%
720 {%
721   \expandafter\XINT_min_pre\expandafter {\romannumeral0\xintnum{#2}}{#1}%
722 }%
723 \def\xintiMin {\romannumeral0\xintiimin }%
724 \def\xintiimin #1%
725 {%
726   \expandafter\xint_iimin\expandafter {\romannumeral`&&@#1}%
727 }%
728 \def\xint_iimin #1#2%
729 {%
730   \expandafter\XINT_min_pre\expandafter {\romannumeral`&&@#2}{#1}%
731 }%
732 \def\XINT_min_pre #1#2{\XINT_min_fork #1\Z #2\Z {#2}{#1}}%
733 \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*`

```

734 \def\XINT_min_fork #1#2\Z #3#4\Z
735 {%
736   \xint_UDsignsfork
737     #1#3\XINT_min_minusminus % A < 0, B < 0
738     #1-\XINT_min_minusplus % B < 0, A >= 0
739     #3-\XINT_min_plusminus % A < 0, B >= 0
740     --{\xint_UDzerosfork
741       #1#3\XINT_min_zerozero % A = B = 0
742       #10\XINT_min_zeropplus % B = 0, A > 0
743       #30\XINT_min_pluszero % A = 0, B > 0
744       00\XINT_min_plusplus % A, B > 0

```

```

745          \krof }%
746      \krof
747      {#2}{#4}#1#3%
748 }%

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

749 \def\XINT_min_zerozero #1#2#3#4{\xint_firstoftwo_thenstop }%
750 \def\XINT_min_zeroplus #1#2#3#4{\xint_secondoftwo_thenstop }%
751 \def\XINT_min_pluszero #1#2#3#4{\xint_firstoftwo_thenstop }%
752 \def\XINT_min_minusplus #1#2#3#4{\xint_secondoftwo_thenstop }%
753 \def\XINT_min_plusminus #1#2#3#4{\xint_firstoftwo_thenstop }%
754 \def\XINT_min_plusplus #1#2#3#4%
755 {%
756     \ifodd\XINT_Geq {#4#2}{#3#1}
757     \expandafter\xint_secondoftwo_thenstop
758     \else
759     \expandafter\xint_firstoftwo_thenstop
760     \fi
761 }%

```

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

```

762 \def\XINT_min_minusminus #1#2#3#4%
763 {%
764     \ifodd\XINT_Geq {#1}{#2}
765     \expandafter\xint_secondoftwo_thenstop
766     \else
767     \expandafter\xint_firstoftwo_thenstop
768     \fi
769 }%

```

4.32 `\xintiMinof`, `\xintiiMinof`

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

```

770 \def\xintiMinof      {\romannumeral0\xintiminof }%
771 \def\xintiminof      #1{\expandafter\XINT_iminof_a\romannumeral`&&@#1\relax }%
772 \def\XINT_iminof_a #1{\expandafter\XINT_iminof_b\romannumeral0\xintnum{#1}\Z }%
773 \def\XINT_iminof_b #1\Z #2%
774     {\expandafter\XINT_iminof_c\romannumeral`&&@#2\Z {#1}\Z}%
775 \def\XINT_iminof_c #1%
776     {\xint_gob_til_relax #1\XINT_iminof_e\relax\XINT_iminof_d #1}%
777 \def\XINT_iminof_d #1\Z
778     {\expandafter\XINT_iminof_b\romannumeral0\xintimin {#1}}%
779 \def\XINT_iminof_e #1\Z #2\Z { #2}%
780 \def\xintiiMinof     {\romannumeral0\xintiiiminof }%
781 \def\xintiiiminof    #1{\expandafter\XINT_iiminof_a\romannumeral`&&@#1\relax }%
782 \def\XINT_iiminof_a #1{\expandafter\XINT_iiminof_b\romannumeral`&&@#1\Z }%
783 \def\XINT_iiminof_b #1\Z #2%
784     {\expandafter\XINT_iiminof_c\romannumeral`&&@#2\Z {#1}\Z}%
785 \def\XINT_iiminof_c #1%
786     {\xint_gob_til_relax #1\XINT_iiminof_e\relax\XINT_iiminof_d #1}%
787 \def\XINT_iiminof_d #1\Z

```

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

4.33 `\xintiiSum`

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

```
790 \def\xintiiSum {\romannumeral0\xintiisum }%
791 \def\xintiisum #1{\xintiisumexpr #1\relax }%
792 \def\xintiiSumExpr {\romannumeral0\xintiisumexpr }%
793 \def\xintiisumexpr {\expandafter\XINT_sumexpr\romannumeral`&&@}%
794 \def\XINT_sumexpr {\XINT_sum_loop_a 0\Z }%
795 \def\XINT_sum_loop_a #1\Z #2%
796     {\expandafter\XINT_sum_loop_b \romannumeral`&&@#2\Z #1\Z \Z}%
797 \def\XINT_sum_loop_b #1%
798     {\xint_gob_til_relax #1\XINT_sum_finished\relax\XINT_sum_loop_c #1}%
799 \def\XINT_sum_loop_c
800     {\expandafter\XINT_sum_loop_a\romannumeral0\XINT_add_fork }%
801 \def\XINT_sum_finished #1\Z #2\Z \Z { #2}%
```

4.34 `\xintiiPrd`

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

```
802 \def\xintiiPrd {\romannumeral0\xintiiprd }%
803 \def\xintiiprd #1{\xintiiprdexpr #1\relax }%
804 \def\xintiiPrdExpr {\romannumeral0\xintiiprdexpr }%
805 \def\xintiiprdexpr {\expandafter\XINT_prdexpr\romannumeral`&&@}%
806 \def\XINT_prdexpr {\XINT_prod_loop_a 1\Z }%
807 \def\XINT_prod_loop_a #1\Z #2%
808     {\expandafter\XINT_prod_loop_b \romannumeral`&&@#2\Z #1\Z \Z}%
809 \def\XINT_prod_loop_b #1%
810     {\xint_gob_til_relax #1\XINT_prod_finished\relax\XINT_prod_loop_c #1}%
811 \def\XINT_prod_loop_c
812     {\expandafter\XINT_prod_loop_a\romannumeral0\XINT_mul_fork }%
813 \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 needed by division macros) ODDNESS, MULTIPLICATION BY TEN, QUOTIENT BY TEN, (moved to xintcore 1.2i) QUOTIENT OR MULTIPLICATION BY POWER OF TEN, SPLIT OPERATION.

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

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

```
814 \def\xintiiMON {\romannumeral0\xintiimon }%
815 \def\xintiimon #1%
816 {%
817     \ifodd\xintiilDg {#1} %<- intentional space
818     \xint_afterfi{ -1}%
```

```

819   \else
820     \xint_afterfi{ 1}%
821   \fi
822 }%
823 \def\xintiMMON {\romannumeral0\xintiimmon }%
824 \def\xintiimmon #1%
825 {%
826   \ifodd\xintiLDg {#1} %<- intentional space
827     \xint_afterfi{ 1}%
828   \else
829     \xint_afterfi{ -1}%
830   \fi
831 }%
832 \def\xintMON {\romannumeral0\xintmon }%
833 \def\xintmon #1%
834 {%
835   \ifodd\xintLDg {#1} %<- intentional space
836     \xint_afterfi{ -1}%
837   \else
838     \xint_afterfi{ 1}%
839   \fi
840 }%
841 \def\xintMMON {\romannumeral0\xintmmon }%
842 \def\xintmmon #1%
843 {%
844   \ifodd\xintLDg {#1} %<- intentional space
845     \xint_afterfi{ 1}%
846   \else
847     \xint_afterfi{ -1}%
848   \fi
849 }%

```

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

```

850 \def\xintiiOdd {\romannumeral0\xintiiodd }%
851 \def\xintiiodd #1%
852 {%
853   \ifodd\xintiLDg{#1} %<- intentional space
854     \xint_afterfi{ 1}%
855   \else
856     \xint_afterfi{ 0}%
857   \fi
858 }%
859 \def\xintiiEven {\romannumeral0\xintiieven }%
860 \def\xintiieven #1%
861 {%
862   \ifodd\xintiLDg{#1} %<- intentional space
863     \xint_afterfi{ 0}%
864   \else
865     \xint_afterfi{ 1}%
866   \fi
867 }%
868 \def\xintOdd {\romannumeral0\xintodd }%

```

```

869 \def\xintodd #1%
870 {%
871   \ifodd\xintLDg{#1} %<- intentional space
872   \xint_afterfi{ 1}%
873   \else
874   \xint_afterfi{ 0}%
875   \fi
876 }%
877 \def\xintEven {\romannumeral0\xinteven }%
878 \def\xinteven #1%
879 {%
880   \ifodd\xintLDg{#1} %<- intentional space
881   \xint_afterfi{ 0}%
882   \else
883   \xint_afterfi{ 1}%
884   \fi
885 }%

```

4.37 `\xintDSH`, `\xintDSHr`

DECIMAL SHIFTS `\xintDSH {x}{A}`

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

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

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

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

Badly named macros.

Rewritten for 1.2i, this was old code and `\xintDSx` has changed interface.

```

886 \def\xintDSHr {\romannumeral0\xintdshr }%
887 \def\xintdshr #1#2%
888 {%
889   \expandafter\XINT_dshr_fork\the\numexpr#1\expandafter.\romannumeral`&&@#2;%
890 }%
891 \def\XINT_dshr_fork #1%
892 {%
893   \xint_UDzerominusfork
894   0#1\XINT_dshr_xzeroorneg
895   #1-\XINT_dshr_xzeroorneg
896   0-\XINT_dshr_xpositive
897   \krof #1%
898 }%
899 \def\XINT_dshr_xzeroorneg #1;{ 0}%
900 \def\XINT_dshr_xpositive
901 {%
902   \expandafter\xint_secondoftwo_thenstop\romannumeral0\XINT_dsx_xisPos
903 }%
904 \def\xintDSH {\romannumeral0\xintdsh }%
905 \def\xintdsh #1#2%
906 {%
907   \expandafter\XINT_dsh_fork\the\numexpr#1\expandafter.\romannumeral`&&@#2;%
908 }%
909 \def\XINT_dsh_fork #1%
910 {%
911   \xint_UDzerominusfork

```

```

912      #1-\XINT_dsh_xiszero
913      0#1\XINT_dsx_xisNeg_checkA
914      0-\XINT_dsh_xisPos #1}%
915      \krof
916 }%
917 \def\XINT_dsh_xiszero #1.#2;{ #2}%
918 \def\XINT_dsh_xisPos
919 {%
920      \expandafter\xint_firstoftwo_thenstop\romannumeral0\XINT_dsx_xisPos
921 }%

```

4.38 \xintDSx

--> Attention le cas $x=0$ est traité dans la même catégorie que $x > 0$ <--

si $x < 0$, fait $A \rightarrow A \cdot 10^{|x|}$

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

si $x \geq 0$, et $A < 0$, d'abord on calcule $\{ \text{quo}(-A, 10^x) \} \{ \text{rem}(-A, 10^x) \}$

puis, si le premier n'est pas nul on lui donne le signe -

si le premier est nul on donne le signe - au second.

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

Rewritten for 1.2i, this was old code.

```

922 \def\xintDSx {\romannumeral0\xintdsx}%
923 \def\xintdsx #1#2%
924 {%
925      \expandafter\XINT_dsx_fork\the\numexpr#1\expandafter.\romannumeral`&&@#2;%
926 }%
927 \def\XINT_dsx_fork #1%
928 {%
929      \xint_UDzerominusfork
930      #1-\XINT_dsx_xisZero
931      0#1\XINT_dsx_xisNeg_checkA
932      0-\XINT_dsx_xisPos #1}%
933      \krof
934 }%
935 \def\XINT_dsx_xisZero #1.#2;{{#2}{0}}%
936 \def\XINT_dsx_xisNeg_checkA #1.#2%
937 {%
938      \xint_gob_til_zero #2\XINT_dsx_xisNeg_Azero 0%
939      \expandafter\XINT_dsx_append\romannumeral\XINT_rep #1\endcsname 0.#2%
940 }%
941 \def\XINT_dsx_xisNeg_Azero #1;{ 0}%
942 \def\XINT_dsx_addzeros #1%
943      {\expandafter\XINT_dsx_append\romannumeral\XINT_rep#1\endcsname0.}%
944 \def\XINT_dsx_addzerosnofuss #1%
945      {\expandafter\XINT_dsx_append\romannumeral\xintreplicate{#1}0.}%
946 \def\XINT_dsx_append #1.#2;{ #2#1}%
947 \def\XINT_dsx_xisPos #1.#2%
948 {%
949      \xint_UDzerominusfork
950      #2-\XINT_dsx_AisZero
951      0#2\XINT_dsx_AisNeg

```


4 Package *xint* implementation

```

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

```

4.39 *\xintDecSplit*, *\xintDecSplitL*, *\xintDecSplitR*

DECIMAL SPLIT

The macro *\xintDecSplit* {x}{A} cuts A which is composed of digits (leading zeroes ok, but no sign) (*) into two (each possibly empty) pieces L and R. The concatenation LR always reproduces A.

The position of the cut is specified by the first argument x. If x is zero or positive the cut location is x slots to the left of the right end of the number. If x becomes equal to or larger than the length of the number then L becomes empty. If x is negative the location of the cut is |x| slots to the right of the left end of the number.

(*) versions earlier than 1.2i first replaced A with its absolute value. This is not the case anymore. This macro should NOT be used for A with a leading sign (+ or -).

Entirely rewritten for 1.2i (2016/12/11).

```

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

```

4 Package `xint` implementation

```

991 \expandafter\XINT_split_finish
992 \romannumeral0\expandafter\XINT_split_xfork
993 \the\numexpr #1\expandafter.\romannumeral`&&@#2%
994 \xint_bye2345678\xint_bye..%
995 }%
996 \def\xintDecSplitL {\romannumeral0\xintdecsplitl }%
997 \def\xintdecsplitl #1#2%
998 {%
999 \expandafter\XINT_splitl_finish
1000 \romannumeral0\expandafter\XINT_split_xfork
1001 \the\numexpr #1\expandafter.\romannumeral`&&@#2%
1002 \xint_bye2345678\xint_bye..%
1003 }%
1004 \def\xintDecSplitR {\romannumeral0\xintdecsplitr }%
1005 \def\xintdecsplitr #1#2%
1006 {%
1007 \expandafter\XINT_splitr_finish
1008 \romannumeral0\expandafter\XINT_split_xfork
1009 \the\numexpr #1\expandafter.\romannumeral`&&@#2%
1010 \xint_bye2345678\xint_bye..%
1011 }%
1012 \def\XINT_split_finish #1.#2.{{#1}{#2}}%
1013 \def\XINT_splitl_finish #1.#2.{ #1}%
1014 \def\XINT_splitr_finish #1.#2.{ #2}%
1015 \def\XINT_split_xfork #1%
1016 {%
1017 \xint_UDzerominusfork
1018 #1-\XINT_split_zerosplit
1019 0#1\XINT_split_fromleft
1020 0-{\XINT_split_fromright #1}%
1021 \krof
1022 }%
1023 \def\XINT_split_zerosplit .#1\xint_bye#2\xint_bye..{ #1..}%
1024 \def\XINT_split_fromleft
1025 {\expandafter\XINT_split_fromleft_a\the\numexpr\xint_c_viii-}%
1026 \def\XINT_split_fromleft_a #1%
1027 {%
1028 \xint_UDsignfork
1029 #1\XINT_split_fromleft_b
1030 -{\XINT_split_fromleft_end_a #1}%
1031 \krof
1032 }%
1033 \def\XINT_split_fromleft_b #1.#2#3#4#5#6#7#8#9%
1034 {%
1035 \expandafter\XINT_split_fromleft_clean
1036 \the\numexpr1#2#3#4#5#6#7#8#9\expandafter
1037 \XINT_split_fromleft_a\the\numexpr\xint_c_viii-#1.%
1038 }%
1039 \def\XINT_split_fromleft_end_a #1.%
1040 {%
1041 \expandafter\XINT_split_fromleft_clean
1042 \the\numexpr1\csname XINT_split_fromleft_end#1\endcsname

```

```

1043 }%
1044 \def\XINT_split_fromleft_clean 1{ }%
1045 \expandafter\def\csname XINT_split_fromleft_end7\endcsname #1%
1046   {#1\XINT_split_fromleft_end_b}%
1047 \expandafter\def\csname XINT_split_fromleft_end6\endcsname #1#2%
1048   {#1#2\XINT_split_fromleft_end_b}%
1049 \expandafter\def\csname XINT_split_fromleft_end5\endcsname #1#2#3%
1050   {#1#2#3\XINT_split_fromleft_end_b}%
1051 \expandafter\def\csname XINT_split_fromleft_end4\endcsname #1#2#3#4%
1052   {#1#2#3#4\XINT_split_fromleft_end_b}%
1053 \expandafter\def\csname XINT_split_fromleft_end3\endcsname #1#2#3#4#5%
1054   {#1#2#3#4#5\XINT_split_fromleft_end_b}%
1055 \expandafter\def\csname XINT_split_fromleft_end2\endcsname #1#2#3#4#5#6%
1056   {#1#2#3#4#5#6\XINT_split_fromleft_end_b}%
1057 \expandafter\def\csname XINT_split_fromleft_end1\endcsname #1#2#3#4#5#6#7%
1058   {#1#2#3#4#5#6#7\XINT_split_fromleft_end_b}%
1059 \expandafter\def\csname XINT_split_fromleft_end0\endcsname #1#2#3#4#5#6#7#8%
1060   {#1#2#3#4#5#6#7#8\XINT_split_fromleft_end_b}%
1061 \def\XINT_split_fromleft_end_b #1\xint_bye#2\xint_bye.{.#1}% puis .
1062 \def\XINT_split_fromright #1.#2\xint_bye
1063 {%
1064   \expandafter\XINT_split_fromright_a
1065   \the\numexpr#1-\numexpr\xint_c_ix\XINT_length_loop
1066   #2\xint_relax\xint_relax\xint_relax\xint_relax
1067   \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
1068   \xint_c_i\xint_c_ii\xint_c_iii\xint_c_iv
1069   \xint_c_v\xint_c_vi\xint_c_vii\xint_c_viii\xint_c_ix\xint_bye
1070   .#2\xint_bye
1071 }%
1072 \def\XINT_split_fromright_a #1%
1073 {%
1074   \xint_UDsignfork
1075   #1\XINT_split_fromleft
1076   -\XINT_split_fromright_Lempty
1077   \krof
1078 }%
1079 \def\XINT_split_fromright_Lempty #1.#2\xint_bye#3..{.#2.}%

```

4.40 *\xintiiSqrt*, *\xintiiSqrtR*, *\xintiiSquareRoot*

First done with 1.08.

1.1 added *\xintiiSquareRoot*.

1.1a added *\xintiiSqrtR*.

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

Actually there is some room for improvements, one could prepare better input X for the upcoming

4 Package *xint* implementation

treatment of fetching its digits by 16, then 32, then 64, etc...

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

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

Attention to impact here of some 1.2i changes to macros (`\xintDSx`, `\xintDecSplit` and their interfaces).

```

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

```

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

```

1089 \def\XINT_sqrtr_post #1#2{\xintiiiflt {#2}{#1}%
1090      { #1}{\XINT_dec #1\XINT_dec_bye234567890\xint_bye}}%
1091 \def\xintisquareroot #1%
1092     {\expandafter\XINT_sqrt_checkin\romannumeral0\xintnum{#1}\xint_relax }%
1093 \def\xintiisquareroot #1{\expandafter\XINT_sqrt_checkin\romannumeral0\xint_relax }%
1094 \def\XINT_sqrt_checkin #1%
1095 {%
1096     \xint_UDzerominusfork
1097     #1-\XINT_sqrt_iszero
1098     0#1\XINT_sqrt_isneg
1099     0-\XINT_sqrt #1}%
1100     \krof
1101 }%
1102 \def\XINT_sqrt_iszero #1\xint_relax { 11}%
1103 \edef\XINT_sqrt_isneg #1\xint_relax {\noexpand\xintError:RootOfNegative\space 11}%
1104 \def\XINT_sqrt #1\xint_relax
1105 {%
1106     \expandafter\XINT_sqrt_start\romannumeral0\xintlength {#1}.#1.%
1107 }%
1108 \def\XINT_sqrt_start #1.%
1109 {%
1110     \ifnum #1<\xint_c_x\xint_dothis\XINT_sqrt_small_a\fi
1111     \xint_orthat\XINT_sqrt_big_a #1.%
1112 }%
1113 \def\XINT_sqrt_small_a #1.{\XINT_sqrt_a #1.\XINT_sqrt_small_d }%
1114 \def\XINT_sqrt_big_a #1.{\XINT_sqrt_a #1.\XINT_sqrt_big_d }%
1115 \def\XINT_sqrt_a #1.%
1116 {%
1117     \ifodd #1
1118     \expandafter\XINT_sqrt_b0

```

4 Package `xint` implementation

```

1119 \else
1120 \expandafter\XINT_sqrt_bE
1121 \fi
1122 #1.%
1123 }%
1124 \def\XINT_sqrt_bE #1.#2#3#4%
1125 {%
1126 \XINT_sqrt_c {#3#4}#2{#1}#3#4%
1127 }%
1128 \def\XINT_sqrt_b0 #1.#2#3%
1129 {%
1130 \XINT_sqrt_c #3#2{#1}#3%
1131 }%
1132 \def\XINT_sqrt_c #1#2%
1133 {%
1134 \expandafter #2%
1135 \the\numexpr \ifnum #1>\xint_c_ii
1136 \ifnum #1>\xint_c_vi
1137 \ifnum #1>12 \ifnum #1>20 \ifnum #1>30
1138 \ifnum #1>42 \ifnum #1>56 \ifnum #1>72
1139 \ifnum #1>90
1140 10\else 9\fi \else 8\fi \else 7\fi \else 6\fi \else 5\fi
1141 \else 4\fi \else 3\fi \else 2\fi \else 1\fi .%
1142 }%
1143 \def\XINT_sqrt_small_d #1.#2%
1144 {%
1145 \expandafter\XINT_sqrt_small_e
1146 \the\numexpr #1\ifcase \numexpr #2/\xint_c_ii-\xint_c_i\relax
1147 \or 0\or 00\or 000\or 0000\fi .%
1148 }%
1149 \def\XINT_sqrt_small_e #1.#2.%
1150 {%
1151 \expandafter\XINT_sqrt_small_ea\the\numexpr #1*#1-#2.#1.%
1152 }%
1153 \def\XINT_sqrt_small_ea #1%
1154 {%
1155 \if0#1\xint_dothis\XINT_sqrt_small_ez\fi
1156 \if-#1\xint_dothis\XINT_sqrt_small_eb\fi
1157 \xint_orthat\XINT_sqrt_small_f #1%
1158 }%
1159 \def\XINT_sqrt_small_ez 0.#1.{\expandafter{\the\numexpr#1+\xint_c_i
1160 \expandafter}\expandafter{\the\numexpr #1*\xint_c_ii+\xint_c_i}}%
1161 \def\XINT_sqrt_small_eb -#1.#2.%
1162 {%
1163 \expandafter\XINT_sqrt_small_ec \the\numexpr
1164 (#1-\xint_c_i+#2)/(\xint_c_ii*#2).#1.#2.%
1165 }%
1166 \def\XINT_sqrt_small_ec #1.#2.#3.%
1167 {%
1168 \expandafter\XINT_sqrt_small_f \the\numexpr
1169 -#2+\xint_c_ii*#3*#1+#1*#1\expandafter.\the\numexpr #3+#1.%
1170 }%

```

4 Package `xint` implementation

```

1171 \def\XINT_sqrt_small_f #1.#2.%
1172 {%
1173   \expandafter\XINT_sqrt_small_g
1174   \the\numexpr (#1+#2)/(\xint_c_ii*#2)-\xint_c_i.#1.#2.%
1175 }%
1176 \def\XINT_sqrt_small_g #1#2.%
1177 {%
1178   \if 0#1%
1179     \expandafter\XINT_sqrt_small_end
1180   \else
1181     \expandafter\XINT_sqrt_small_h
1182   \fi
1183   #1#2.%
1184 }%
1185 \def\XINT_sqrt_small_h #1.#2.#3.%
1186 {%
1187   \expandafter\XINT_sqrt_small_f
1188   \the\numexpr #2-\xint_c_ii*#1*#3+#1*#1\expandafter.%
1189   \the\numexpr #3-#1.%
1190 }%
1191 \def\XINT_sqrt_small_end #1.#2.#3.{\{#3\}{#2\}}%
1192 \def\XINT_sqrt_big_d #1.#2%
1193 {%
1194   \ifodd #2 \xint_dothis{\expandafter\XINT_sqrt_big_e0}\fi
1195   \xint_orthat{\expandafter\XINT_sqrt_big_eE}%
1196   \the\numexpr (#2-\xint_c_i)/\xint_c_ii.#1;%
1197 }%
1198 \def\XINT_sqrt_big_eE #1;#2#3#4#5#6#7#8#9%
1199 {%
1200   \XINT_sqrt_big_eE_a #1;{\#2#3#4#5#6#7#8#9}%
1201 }%
1202 \def\XINT_sqrt_big_eE_a #1.#2;#3%
1203 {%
1204   \expandafter\XINT_sqrt_bigormed_f
1205   \romannumeral0\XINT_sqrt_small_e #2000.#3.#1;%
1206 }%
1207 \def\XINT_sqrt_big_e0 #1;#2#3#4#5#6#7#8#9%
1208 {%
1209   \XINT_sqrt_big_e0_a #1;{\#2#3#4#5#6#7#8#9}%
1210 }%
1211 \def\XINT_sqrt_big_e0_a #1.#2;#3#4%
1212 {%
1213   \expandafter\XINT_sqrt_bigormed_f
1214   \romannumeral0\XINT_sqrt_small_e #20000.#3#4.#1;%
1215 }%
1216 \def\XINT_sqrt_bigormed_f #1#2#3;%
1217 {%
1218   \ifnum#3<\xint_c_ix
1219     \xint_dothis {\csname XINT_sqrt_med_f\romannumeral#3\endcsname}%
1220   \fi
1221   \xint_orthat\XINT_sqrt_big_f #1.#2.#3;%
1222 }%

```

4 Package `xint` implementation

```

1223 \def\XINT_sqrt_med_fv {\XINT_sqrt_med_fa .}%
1224 \def\XINT_sqrt_med_fvi {\XINT_sqrt_med_fa 0.}%
1225 \def\XINT_sqrt_med_fvii {\XINT_sqrt_med_fa 00.}%
1226 \def\XINT_sqrt_med_fviii{\XINT_sqrt_med_fa 000.}%
1227 \def\XINT_sqrt_med_fa #1.#2.#3.#4;%
1228 {%
1229     \expandafter\XINT_sqrt_med_fb
1230     \the\numexpr (#30#1-5#1)/(\xint_c_ii*#2).#1.#2.#3.%
1231 }%
1232 \def\XINT_sqrt_med_fb #1.#2.#3.#4.#5.%
1233 {%
1234     \expandafter\XINT_sqrt_small_ea
1235     \the\numexpr (#40#2-\xint_c_ii*#3*#1)*10#2+(#1*#1-#5)\expandafter.%
1236     \the\numexpr #30#2-#1.%
1237 }%
1238 \def\XINT_sqrt_big_f #1;#2#3#4#5#6#7#8#9%
1239 {%
1240     \XINT_sqrt_big_fa #1;{#2#3#4#5#6#7#8#9}%
1241 }%
1242 \def\XINT_sqrt_big_fa #1.#2.#3;#4%
1243 {%
1244     \expandafter\XINT_sqrt_big_ga
1245     \the\numexpr #3-\xint_c_viii\expandafter.%
1246     \romannumeral0\XINT_sqrt_med_fa 000.#1.#2.;#4.%
1247 }%
1248 %
1249 \def\XINT_sqrt_big_ga #1.#2#3%
1250 {%
1251     \ifnum #1>\xint_c_viii
1252         \expandafter\XINT_sqrt_big_gb\else
1253         \expandafter\XINT_sqrt_big_ka
1254     \fi #1.#3.#2.%
1255 }%
1256 \def\XINT_sqrt_big_gb #1.#2.#3.%
1257 {%
1258     \expandafter\XINT_sqrt_big_gc
1259     \the\numexpr (\xint_c_ii*#2-\xint_c_i)*\xint_c_x^viii/(\xint_c_iv*#3).%
1260     #3.#2.#1;%
1261 }%
1262 \def\XINT_sqrt_big_gc #1.#2.#3.%
1263 {%
1264     \expandafter\XINT_sqrt_big_gd
1265     \romannumeral0\xintiiaadd
1266         {\xintiiSub {#3000000000}{\xintDouble{\xintiiMul{#2}{#1}}}{000000000}}%
1267         {\xintiiSqr {#1}}}%
1268     \romannumeral0\xintiisub{#2000000000}{#1}.%
1269 }%
1270 \def\XINT_sqrt_big_gd #1.#2.%
1271 {%
1272     \expandafter\XINT_sqrt_big_ge #2.#1.%
1273 }%
1274 \def\XINT_sqrt_big_ge #1;#2#3#4#5#6#7#8#9%

```

4 Package *xint* implementation

```

1275   {\XINT_sqrt_big_gf #1.#2#3#4#5#6#7#8#9;}%
1276 \def\XINT_sqrt_big_gf #1;#2#3#4#5#6#7#8#9%
1277   {\XINT_sqrt_big_gg #1#2#3#4#5#6#7#8#9.}%
1278 \def\XINT_sqrt_big_gg #1.#2.#3.#4.%
1279 {%
1280   \expandafter\XINT_sqrt_big_gloop
1281   \expandafter\xint_c_xvi\expandafter.%
1282   \the\numexpr #3-\xint_c_viii\expandafter.%
1283   \romannumeral0\xintiisub {#2}{\xintiNum{#4}}.#1.%
1284 }%
1285 \def\XINT_sqrt_big_gloop #1.#2.%
1286 {%
1287   \unless\ifnum #1<#2 \xint_dothis\XINT_sqrt_big_ka \fi
1288   \xint_orthat{\XINT_sqrt_big_gi #1.}#2.%
1289 }%
1290 \def\XINT_sqrt_big_gi #1.%
1291 {%
1292   \expandafter\XINT_sqrt_big_gj\romannumeral\xintreplicate{#1}0.#1.%
1293 }%
1294 \def\XINT_sqrt_big_gj #1.#2.#3.#4.#5.%
1295 {%
1296   \expandafter\XINT_sqrt_big_gk
1297   \romannumeral0\xintiidivision {#4#1}%
1298   {\XINT_dbl #5\xint_bye2345678\xint_bye*\xint_c_ii\relax}.%
1299   #1.#5.#2.#3.%
1300 }%
1301 \def\XINT_sqrt_big_gk #1#2.#3.#4.%
1302 {%
1303   \expandafter\XINT_sqrt_big_gl
1304   \romannumeral0\xintiiaadd {#2#3}{\xintiiSqr{#1}}.%
1305   \romannumeral0\xintiisub {#4#3}{#1}.%
1306 }%
1307 \def\XINT_sqrt_big_gl #1.#2.%
1308 {%
1309   \expandafter\XINT_sqrt_big_gm #2.#1.%
1310 }%
1311 \def\XINT_sqrt_big_gm #1.#2.#3.#4.#5.%
1312 {%
1313   \expandafter\XINT_sqrt_big_gn
1314   \romannumeral0\XINT_split_fromleft\xint_c_ii*#3.#5\xint_bye2345678\xint_bye..%
1315   #1.#2.#3.#4.%
1316 }%
1317 \def\XINT_sqrt_big_gn #1.#2.#3.#4.#5.#6.%
1318 {%
1319   \expandafter\XINT_sqrt_big_gloop
1320   \the\numexpr \xint_c_ii*#5\expandafter.%
1321   \the\numexpr #6-#5\expandafter.%
1322   \romannumeral0\xintiisub{#4}{\xintiNum{#1}}.#3.#2.%
1323 }%
1324 \def\XINT_sqrt_big_ka #1.#2.#3.#4.%
1325 {%
1326   \expandafter\XINT_sqrt_big_kb

```



```

1327 \romannumeral0\XINT_dsx_addzeros {#1}#3;.%
1328 \romannumeral0\xintiisub
1329 {\XINT_dsx_addzerosnofuss {\xint_c_ii*#1}#2;}%
1330 {\xintiNum{#4}}.%
1331 }%
1332 \def\XINT_sqrt_big_kb #1.#2.%
1333 {%
1334 \expandafter\XINT_sqrt_big_kc #2.#1.%
1335 }%
1336 \def\XINT_sqrt_big_kc #1%
1337 {%
1338 \if0#1\xint_dothis\XINT_sqrt_big_kz\fi
1339 \xint_orthat\XINT_sqrt_big_kloop #1%
1340 }%
1341 \def\XINT_sqrt_big_kz 0.#1.%
1342 {%
1343 \expandafter\XINT_sqrt_big_kend
1344 \romannumeral0%
1345 \xintinc{\XINT_dbl#1\xint_bye2345678\xint_bye*\xint_c_ii\relax}.#1.%
1346 }%
1347 \def\XINT_sqrt_big_kend #1.#2.%
1348 {%
1349 \expandafter{\romannumeral0\xintinc{#2}}{#1}%
1350 }%
1351 \def\XINT_sqrt_big_kloop #1.#2.%
1352 {%
1353 \expandafter\XINT_sqrt_big_ke
1354 \romannumeral0\xintiidivision{#1}%
1355 {\romannumeral0\XINT_dbl #2\xint_bye2345678\xint_bye*\xint_c_ii\relax}{#2}%
1356 }%
1357 \def\XINT_sqrt_big_ke #1%
1358 {%
1359 \if0\XINT_Sgn #1\Z
1360 \expandafter \XINT_sqrt_big_end
1361 \else \expandafter \XINT_sqrt_big_kf
1362 \fi {#1}%
1363 }%
1364 \def\XINT_sqrt_big_kf #1#2#3%
1365 {%
1366 \expandafter\XINT_sqrt_big_kg
1367 \romannumeral0\xintiisub {#3}{#1}.%
1368 \romannumeral0\xintiiadd {#2}{\xintiiSqr {#1}}.%
1369 }%
1370 \def\XINT_sqrt_big_kg #1.#2.%
1371 {%
1372 \expandafter\XINT_sqrt_big_kloop #2.#1.%
1373 }%
1374 \def\XINT_sqrt_big_end #1#2#3{{#3}{#2}}%

```

4.41 \xintiiBinomial, \xintiBinomial

2015/11/28-29 for 1.2f.

4 Package *xint* implementation

2016/11/19 for 1.2h: I truly can't understand why I hard-coded last year an error-message for arguments outside of the range for binomial formula. Naturally there should be no error but a rather a 0 return value for $\text{binomial}(x,y)$, if $y < 0$ or $x < y$!

I really lack some kind of infinity or NaN value.

```

1375 \def\xintiBinomial {\romannumeral0\xintiibinomial }%
1376 \def\xintiibinomial #1#2%
1377 {%
1378   \expandafter\XINT_binom_pre\the\numexpr #1\expandafter.\the\numexpr #2.%
1379 }%
1380 \def\XINT_binom_pre #1.#2.%
1381 {%
1382   \expandafter\XINT_binom_fork \the\numexpr#1-#2.#2.#1.%
1383 }%
1384 \def\xintiBinomial{\romannumeral0\xintiibinomial}%
1385 \let\xintiibinomial\xintiibinomial

```

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

```

1386 \def\XINT_binom_fork #1#2.#3#4.#5#6.%
1387 {%
1388   \if-#5\xint_dothis{\xintError:OutOfRangeBinomial\space 0}\fi
1389   \if-#1\xint_dothis{ 0}\fi
1390   \if-#3\xint_dothis{ 0}\fi
1391   \if0#1\xint_dothis{ 1}\fi
1392   \if0#3\xint_dothis{ 1}\fi
1393   \ifnum #5#6>\xint_c_x^viii_mone\xint_dothis{\xintError:OutOfRangeBinomial\space 0}\fi
1394   \ifnum #1#2>#3#4 \xint_dothis{\XINT_binom_a #1#2.#3#4.}\fi
1395   \xint_orthat{\XINT_binom_a #3#4.#1#2.}%
1396 }%

```

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

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

```

1397 \def\XINT_binom_a #1.#2.%
1398 {%
1399   \expandafter\XINT_binom_b\the\numexpr \xint_c_i+#1.1.#2.10000000011!\Z!0!%
1400 }%

```

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

```

1401 \def\XINT_binom_b #1.%
1402 {%
1403   \ifnum #1>9999 \xint_dothis\XINT_binom_vbigloop \fi

```

4 Package *xint* implementation

```

1404 \ifnum #1>463 \xint_dothis\XINT_binom_bigloop \fi
1405 \ifnum #1>98 \xint_dothis\XINT_binom_medloop \fi
1406 \ifnum #1>29 \xint_dothis\XINT_binom_smallloop \fi
1407 \xint_orthat\XINT_binom_vsmallloop #1.%
1408 }%

```

y.j.k. Au départ on avait $x-k+1.1.k$. Ensuite on a des blocs $1<8d>!$ donnant le résultat intermédiaire, dans l'ordre, et à la fin on a $1!1\backslash Z!0!$. Dans `smallloop` on peut prendre 4 par 4.

```

1409 \def\XINT_binom_smallloop #1.#2.#3.%
1410 {%
1411 \ifcase\numexpr #3-#2\relax
1412 \expandafter\XINT_binom_end_
1413 \or \expandafter\XINT_binom_end_i
1414 \or \expandafter\XINT_binom_end_ii
1415 \or \expandafter\XINT_binom_end_iii
1416 \else\expandafter\XINT_binom_smallloop_a
1417 \fi #1.#2.#3.%
1418 }%

```

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

```

1419 \def\XINT_binom_smallloop_a #1.#2.#3.%
1420 {%
1421 \expandafter\XINT_binom_smallloop_b
1422 \the\numexpr #1+\xint_c_iv\expandafter.%
1423 \the\numexpr #2+\xint_c_iv\expandafter.%
1424 \the\numexpr #3\expandafter.%
1425 \the\numexpr\expandafter\XINT_binom_div
1426 \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1427 !\romannumeral0\expandafter\XINT_binom_mul
1428 \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1429 }%
1430 \def\XINT_binom_smallloop_b #1.%
1431 {%
1432 \ifnum #1>98 \expandafter\XINT_binom_medloop \else
1433 \expandafter\XINT_binom_smallloop \fi #1.%
1434 }%

```

Ici on prend trois par trois.

```

1435 \def\XINT_binom_medloop #1.#2.#3.%
1436 {%
1437 \ifcase\numexpr #3-#2\relax
1438 \expandafter\XINT_binom_end_
1439 \or \expandafter\XINT_binom_end_i
1440 \or \expandafter\XINT_binom_end_ii
1441 \else\expandafter\XINT_binom_medloop_a
1442 \fi #1.#2.#3.%
1443 }%
1444 \def\XINT_binom_medloop_a #1.#2.#3.%
1445 {%
1446 \expandafter\XINT_binom_medloop_b

```

4 Package *xint* implementation

```
1447 \the\numexpr #1+\xint_c_iii\expandafter.%
1448 \the\numexpr #2+\xint_c_iii\expandafter.%
1449 \the\numexpr #3\expandafter.%
1450 \the\numexpr\expandafter\XINT_binom_div
1451 \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
1452 !\romannumeral0\expandafter\XINT_binom_mul
1453 \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1454 }%
1455 \def\XINT_binom_medloop_b #1.%
1456 {%
1457 \ifnum #1>463 \expandafter\XINT_binom_bigloop \else
1458 \expandafter\XINT_binom_medloop \fi #1.%
1459 }%
```

Ici on prend deux par deux.

```
1460 \def\XINT_binom_bigloop #1.#2.#3.%
1461 {%
1462 \ifcase\numexpr #3-#2\relax
1463 \expandafter\XINT_binom_end_
1464 \or \expandafter\XINT_binom_end_i
1465 \else\expandafter\XINT_binom_bigloop_a
1466 \fi #1.#2.#3.%
1467 }%
1468 \def\XINT_binom_bigloop_a #1.#2.#3.%
1469 {%
1470 \expandafter\XINT_binom_bigloop_b
1471 \the\numexpr #1+\xint_c_ii\expandafter.%
1472 \the\numexpr #2+\xint_c_ii\expandafter.%
1473 \the\numexpr #3\expandafter.%
1474 \the\numexpr\expandafter\XINT_binom_div
1475 \the\numexpr #2*(#2+\xint_c_i)\expandafter
1476 !\romannumeral0\expandafter\XINT_binom_mul
1477 \the\numexpr #1*(#1+\xint_c_i)!%
1478 }%
1479 \def\XINT_binom_bigloop_b #1.%
1480 {%
1481 \ifnum #1>9999 \expandafter\XINT_binom_vbigloop \else
1482 \expandafter\XINT_binom_bigloop \fi #1.%
1483 }%
```

Et finalement un par un.

```
1484 \def\XINT_binom_vbigloop #1.#2.#3.%
1485 {%
1486 \ifnum #3=#2
1487 \expandafter\XINT_binom_end_
1488 \else\expandafter\XINT_binom_vbigloop_a
1489 \fi #1.#2.#3.%
1490 }%
1491 \def\XINT_binom_vbigloop_a #1.#2.#3.%
1492 {%
1493 \expandafter\XINT_binom_vbigloop
1494 \the\numexpr #1+\xint_c_i\expandafter.%
```

4 Package *xint* implementation

```

1495 \the\numexpr #2+\xint_c_i\expandafter.%
1496 \the\numexpr #3\expandafter.%
1497 \the\numexpr\expandafter\XINT_binom_div\the\numexpr #2\expandafter
1498 !\romannumeral0\XINT_binom_mul #1!%
1499 }%

```

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

```

1500 \def\XINT_binom_vsmallloop #1.#2.#3.%
1501 {%
1502   \ifcase\numexpr #3-#2\relax
1503     \expandafter\XINT_binom_vsmallend_
1504   \or \expandafter\XINT_binom_vsmallend_i
1505   \or \expandafter\XINT_binom_vsmallend_ii
1506   \or \expandafter\XINT_binom_vsmallend_iii
1507   \else\expandafter\XINT_binom_vsmallloop_a
1508   \fi #1.#2.#3.%
1509 }%
1510 \def\XINT_binom_vsmallloop_a #1.%
1511 {%
1512   \ifnum #1>26 \expandafter\XINT_binom_smallloop_a \else
1513     \expandafter\XINT_binom_vsmallloop_b \fi #1.%
1514 }%
1515 \def\XINT_binom_vsmallloop_b #1.#2.#3.%
1516 {%
1517   \expandafter\XINT_binom_vsmallloop
1518   \the\numexpr #1+\xint_c_iv\expandafter.%
1519   \the\numexpr #2+\xint_c_iv\expandafter.%
1520   \the\numexpr #3\expandafter.%
1521   \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1522   \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1523   !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1524 }%
1525 \def\XINT_binom_mul #1!#21!\Z!0!%
1526 {%
1527   \expandafter\XINT_rev_nounsep\expandafter{\expandafter}%
1528   \the\numexpr\expandafter\XINT_smallmul
1529   \the\numexpr\xint_c_x^viii+#1\expandafter
1530   !\romannumeral0\XINT_rev_nounsep {}1\Z!#2%
1531   \R!\R!\R!\R!\R!\R!\R!\R!\W
1532   \R!\R!\R!\R!\R!\R!\R!\R!\W
1533   1\Z!%
1534 }%
1535 \def\XINT_binom_div #1!1\Z!%
1536 {%
1537   \expandafter\XINT_smalldivx_a
1538   \the\numexpr #1/\xint_c_ii\expandafter.%
1539   \the\numexpr \xint_c_x^viii+#1!%
1540 }%

```

4 Package *xint* implementation

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

```
1541 \def\XINT_binom_vsmallmuldiv #1!#2!1#3!{\xint_c_x^viii+#2*#3/#1!}%
```

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

```
1542 \def\XINT_binom_end_iii #1.#2.#3.%
1543 {%
1544   \expandafter\XINT_binom_finish
1545   \the\numexpr\expandafter\XINT_binom_div
1546     \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1547     !\romannumeral0\expandafter\XINT_binom_mul
1548     \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1549 }%
1550 \def\XINT_binom_end_ii #1.#2.#3.%
1551 {%
1552   \expandafter\XINT_binom_finish
1553   \the\numexpr\expandafter\XINT_binom_div
1554     \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
1555     !\romannumeral0\expandafter\XINT_binom_mul
1556     \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1557 }%
1558 \def\XINT_binom_end_i #1.#2.#3.%
1559 {%
1560   \expandafter\XINT_binom_finish
1561   \the\numexpr\expandafter\XINT_binom_div
1562     \the\numexpr #2*(#2+\xint_c_i)\expandafter
1563     !\romannumeral0\expandafter\XINT_binom_mul
1564     \the\numexpr #1*(#1+\xint_c_i)!%
1565 }%
1566 \def\XINT_binom_end_ #1.#2.#3.%
1567 {%
1568   \expandafter\XINT_binom_finish
1569   \the\numexpr\expandafter\XINT_binom_div\the\numexpr #2\expandafter
1570   !\romannumeral0\XINT_binom_mul #1!%
1571 }%
1572 \def\XINT_binom_finish #1\Z!0!%
1573 {\XINT_unsep_cuzsmall #11\R!1\R!1\R!1\R!1\R!1\R!1\R!\W }%
```

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

```
1574 \def\XINT_binom_vsmallend_iii #1.%
1575 {%
1576   \ifnum #1>26 \expandafter\XINT_binom_end_iii \else
1577     \expandafter\XINT_binom_vsmallend_iiib \fi #1.%
1578 }%
1579 \def\XINT_binom_vsmallend_iiib #1.#2.#3.%
1580 {%
1581   \expandafter\XINT_binom_vsmallfinish
1582   \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1583   \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1584   !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
```

4 Package `xint` implementation

```

1585 }%
1586 \def\XINT_binom_vsmallend_ii #1.%
1587 {%
1588     \ifnum #1>27 \expandafter\XINT_binom_end_ii \else
1589         \expandafter\XINT_binom_vsmallend_iib \fi #1.%
1590 }%
1591 \def\XINT_binom_vsmallend_iib #1.#2.#3.%
1592 {%
1593     \expandafter\XINT_binom_vsmallfinish
1594     \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1595     \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
1596     !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1597 }%
1598 \def\XINT_binom_vsmallend_i #1.%
1599 {%
1600     \ifnum #1>28 \expandafter\XINT_binom_end_i \else
1601         \expandafter\XINT_binom_vsmallend_ib \fi #1.%
1602 }%
1603 \def\XINT_binom_vsmallend_ib #1.#2.#3.%
1604 {%
1605     \expandafter\XINT_binom_vsmallfinish
1606     \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1607     \the\numexpr #2*(#2+\xint_c_i)\expandafter
1608     !\the\numexpr #1*(#1+\xint_c_i)!%
1609 }%
1610 \def\XINT_binom_vsmallend_ #1.%
1611 {%
1612     \ifnum #1>29 \expandafter\XINT_binom_end_ \else
1613         \expandafter\XINT_binom_vsmallend_b \fi #1.%
1614 }%
1615 \def\XINT_binom_vsmallend_b #1.#2.#3.%
1616 {%
1617     \expandafter\XINT_binom_vsmallfinish
1618     \the\numexpr\XINT_binom_vsmallmuldiv #2!#1!%
1619 }%
1620 \edef\XINT_binom_vsmallfinish 1#1!1!\Z!0!%
1621 { \noexpand\expandafter\space\noexpand\the\numexpr #1\relax}%

```

4.42 `\xintiPFactorial`, `\xintiPFactorial`

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

1.2h (2016/11/20) removes the non-negativity condition. It was a bit unfortunate that the code raised `\xintError:OutOfRangePFac` if $0\leq a\leq b<10^8$ was violated. The rule now applied is to interpret $\text{pfac}(a,b)$ as the product for $a<j\leq b$ (not as a ratio of Gamma function), hence if $a\geq b$, return 1 because of an empty product. If $a<b$: if $a<0$, return 0 for $b\geq 0$ and $(-1)^{(b-a)}$ times $|b|\dots(|a|-1)$ for $b<0$. But only for the range $0\leq a\leq b<10^8$ is the macro result to be considered as stable.

```

1622 \def\xintiPFactorial {\romannumeral0\xintiipfactorial }%
1623 \def\xintiipfactorial #1#2%
1624 {%
1625     \expandafter\XINT_pfac_fork\the\numexpr#1\expandafter.\the\numexpr #2.%
1626 }%

```

4 Package *xint* implementation

```
1627 \def\xintiPFactorial{\romannumeral0\xintipfactorial}%
1628 \let\xintipfactorial\xintiipfactorial
```

Code is a simplified version of the one for `\xintiiBinomial`, with no attempt at implementing a "very small" branch.

```
1629 \def\XINT_pfac_fork #1#2.#3#4.%
1630 {%
1631   \unless\ifnum #1#2<#3#4 \xint_dothis\XINT_pfac_one\fi
1632   \if-#3\xint_dothis\XINT_pfac_neg\fi
1633   \if-#1\xint_dothis\XINT_pfac_zero\fi
1634   \ifnum #3#4>\xint_c_x^viii_mone\xint_dothis\XINT_pfac_outofrange\fi
1635   \xint_orthat \XINT_pfac_a #1#2.#3#4.%
1636 }%
1637 \def\XINT_pfac_outofrange #1.#2.{\xintError:OutOfRangePFac\space 0}%
1638 \def\XINT_pfac_one #1.#2.{ 1}%
1639 \def\XINT_pfac_zero #1.#2.{ 0}%
1640 \def\XINT_pfac_neg -#1.-#2.%
1641 {%
1642   \ifnum #1>\xint_c_x^viii\xint_dothis\XINT_pfac_outofrange\fi
1643   \xint_orthat
1644   {\ifodd\numexpr#2-#1\relax\xint_afterfi{\expandafter-\romannumeral`&&@}\fi
1645   \expandafter\XINT_pfac_a }%
1646   \the\numexpr #2-\xint_c_i\expandafter.\the\numexpr#1-\xint_c_i.%
1647 }%
1648 \def\XINT_pfac_a #1.#2.%
1649 {%
1650   \expandafter\XINT_pfac_b\the\numexpr \xint_c_i+#1.#2.10000000111\Z!%
1651   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1652 }%
1653 \def\XINT_pfac_b #1.%
1654 {%
1655   \ifnum #1>9999 \xint_dothis\XINT_pfac_vbigloop \fi
1656   \ifnum #1>463 \xint_dothis\XINT_pfac_bigloop \fi
1657   \ifnum #1>98 \xint_dothis\XINT_pfac_medloop \fi
1658   \xint_orthat\XINT_pfac_smallloop #1.%
1659 }%
1660 \def\XINT_pfac_smallloop #1.#2.%
1661 {%
1662   \ifcase\numexpr #2-#1\relax
1663     \expandafter\XINT_pfac_end_
1664     \or \expandafter\XINT_pfac_end_i
1665     \or \expandafter\XINT_pfac_end_ii
1666     \or \expandafter\XINT_pfac_end_iii
1667     \else\expandafter\XINT_pfac_smallloop_a
1668     \fi #1.#2.%
1669 }%
1670 \def\XINT_pfac_smallloop_a #1.#2.%
1671 {%
1672   \expandafter\XINT_pfac_smallloop_b
1673   \the\numexpr #1+\xint_c_iv\expandafter.%
1674   \the\numexpr #2\expandafter.%
1675   \the\numexpr\expandafter\XINT_smallmul
```


4 Package *xint* implementation

```

1676 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1677 }%
1678 \def\xint_pfac_smallloop_b #1.%
1679 {%
1680 \ifnum #1>98 \expandafter\xint_pfac_medloop \else
1681 \expandafter\xint_pfac_smallloop \fi #1.%
1682 }%
1683 \def\xint_pfac_medloop #1.#2.%
1684 {%
1685 \ifcase\numexpr #2-#1\relax
1686 \expandafter\xint_pfac_end_
1687 \or \expandafter\xint_pfac_end_i
1688 \or \expandafter\xint_pfac_end_ii
1689 \else\expandafter\xint_pfac_medloop_a
1690 \fi #1.#2.%
1691 }%
1692 \def\xint_pfac_medloop_a #1.#2.%
1693 {%
1694 \expandafter\xint_pfac_medloop_b
1695 \the\numexpr #1+\xint_c_iii\expandafter.%
1696 \the\numexpr #2\expandafter.%
1697 \the\numexpr\expandafter\xint_smallmul
1698 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1699 }%
1700 \def\xint_pfac_medloop_b #1.%
1701 {%
1702 \ifnum #1>463 \expandafter\xint_pfac_bigloop \else
1703 \expandafter\xint_pfac_medloop \fi #1.%
1704 }%
1705 \def\xint_pfac_bigloop #1.#2.%
1706 {%
1707 \ifcase\numexpr #2-#1\relax
1708 \expandafter\xint_pfac_end_
1709 \or \expandafter\xint_pfac_end_i
1710 \else\expandafter\xint_pfac_bigloop_a
1711 \fi #1.#2.%
1712 }%
1713 \def\xint_pfac_bigloop_a #1.#2.%
1714 {%
1715 \expandafter\xint_pfac_bigloop_b
1716 \the\numexpr #1+\xint_c_ii\expandafter.%
1717 \the\numexpr #2\expandafter.%
1718 \the\numexpr\expandafter
1719 \xint_smallmul\the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
1720 }%
1721 \def\xint_pfac_bigloop_b #1.%
1722 {%
1723 \ifnum #1>9999 \expandafter\xint_pfac_vbigloop \else
1724 \expandafter\xint_pfac_bigloop \fi #1.%
1725 }%
1726 \def\xint_pfac_vbigloop #1.#2.%
1727 {%

```

```

1728 \ifnum #2=#1
1729 \expandafter\XINT_pfac_end_
1730 \else\expandafter\XINT_pfac_vbigloop_a
1731 \fi #1.#2.%
1732 }%
1733 \def\XINT_pfac_vbigloop_a #1.#2.%
1734 {%
1735 \expandafter\XINT_pfac_vbigloop
1736 \the\numexpr #1+\xint_c_i\expandafter.%
1737 \the\numexpr #2\expandafter.%
1738 \the\numexpr\expandafter\XINT_smallmul\the\numexpr\xint_c_x^viii+#1!%
1739 }%
1740 \def\XINT_pfac_end_iii #1.#2.%
1741 {%
1742 \expandafter\XINT_mul_out
1743 \the\numexpr\expandafter\XINT_smallmul
1744 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1745 }%
1746 \def\XINT_pfac_end_ii #1.#2.%
1747 {%
1748 \expandafter\XINT_mul_out
1749 \the\numexpr\expandafter\XINT_smallmul
1750 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1751 }%
1752 \def\XINT_pfac_end_i #1.#2.%
1753 {%
1754 \expandafter\XINT_mul_out
1755 \the\numexpr\expandafter\XINT_smallmul
1756 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
1757 }%
1758 \def\XINT_pfac_end_ #1.#2.%
1759 {%
1760 \expandafter\XINT_mul_out
1761 \the\numexpr\expandafter\XINT_smallmul\the\numexpr \xint_c_x^viii+#1!%
1762 }%

```

4.43 `\xintiie`

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

```

1763 \def\xintiieE {\romannumeral0\xintiie }% used in \xintMod.
1764 \def\xintiie #1#2%
1765 {\expandafter\XINT_ii_fork\the\numexpr #2\expandafter.\romannumeral`&&@#1;}%
1766 \def\XINT_ii_fork #1%
1767 {%
1768 \xint_UDsignfork
1769 #1\XINT_ii_neg
1770 -\XINT_ii_a
1771 \krof #1%
1772 }%
1773 \def\XINT_ii_a #1.%
1774 {\expandafter\XINT_dsx_append\romannumeral\XINT_rep #1\endcsname 0.}%
1775 \def\XINT_ii_neg #1.#2;{ #2}%

```

4.44 “Load `xintfrac`” macros

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

```

1776 \catcode`! 11
1777 \def\xintMax {\Did_you_mean_iiMax?or_load_xintfrac!}%
1778 \def\xintMin {\Did_you_mean_iiMin?or_load_xintfrac!}%
1779 \def\xintMaxof {\Did_you_mean_iMaxof?or_load_xintfrac!}%
1780 \def\xintMinof {\Did_you_mean_iMinof?or_load_xintfrac!}%
1781 \def\xintSum {\Did_you_mean_iiSum?or_load_xintfrac!}%
1782 \def\xintPrd {\Did_you_mean_iiPrd?or_load_xintfrac!}%
1783 \def\xintPrdExpr {\Did_you_mean_iiPrdExpr?or_load_xintfrac!}%
1784 \def\xintSumExpr {\Did_you_mean_iiSumExpr?or_load_xintfrac!}%
1785 \XINT_restorecatcodes_endinput%
```

5 Package *xintbinhex* implementation

.1	Catcodes, ε -TeX and reload detection . .	140	.6	<code>\xintHexToDec</code>	146
.2	Package identification	141	.7	<code>\xintBinToDec</code>	148
.3	Constants, etc...	141	.8	<code>\xintBinToHex</code>	150
.4	<code>\XINT_OQ</code>	143	.9	<code>\xintHexToBin</code>	151
.5	<code>\xintDecToHex</code> , <code>\xintDecToBin</code>	143	.10	<code>\xintCHexToBin</code>	152

The commenting is currently (2016/12/13) 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 [2016/12/13 1.2i Expandable binary and hexadecimal conversions (JFB)]%

```

5.3 Constants, etc...

1.08

```

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

```

5 Package *xintbinhex* implementation

```

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

```

```

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_OOQ

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

```

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

```

5.5 \xintDecToHex, \xintDecToBin

1.08

```

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

```

5 Package *xintbinhex* implementation

```

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

```


5 Package *xintbinhex* implementation

```

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

```

5 Package *xintbinhex* implementation

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

5.6 *\xintHexToDec*

1.08

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

5 Package *xintbinhex* implementation

```

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

```

5 Package *xintbinhex* implementation

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

5.7 *\xintBinToDec*

1.08

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

5 Package *xintbinhex* implementation

```

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

```

```

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

```

5.8 \xintBinToHex

1.08

```

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

```

```

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

```

5.9 \xintHexToBin

1.08

```

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

```

```

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

```

5.10 \xintCHexToBin

1.08

```

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

```


5 Package *xintbinhex* implementation

```
627     \XINT_num_loop
628 }%
629 \XINT_restorecatcodes_endinput%
```

6 Package *xintgcd* implementation

.1	Catcodes, ε -TeX and reload detection . .	154	.7	<code>\xintBezoutAlgorithm</code>	161
.2	Package identification	155	.8	<code>\xintGCDof</code>	163
.3	<code>\xintGCD</code> , <code>\xintiiGCD</code>	155	.9	<code>\xintLCMof</code>	163
.4	<code>\xintLCM</code> , <code>\xintiiLCM</code>	156	.10	<code>\xintTypesetEuclideanAlgorithm</code>	164
.5	<code>\xintBezout</code>	156	.11	<code>\xintTypesetBezoutAlgorithm</code>	164
.6	<code>\xintEuclideanAlgorithm</code>	160			

The commenting is currently (2016/12/13) very sparse. Release 1.09h has modified a bit the `\xintTypesetEuclideanAlgorithm` and `\xintTypesetBezoutAlgorithm` layout with respect to line indentation in particular. And they use the `xinttools` `\xintloop` rather than the Plain TeX or \mathbb{M} TeX'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 -TeX and reload detection

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

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

```

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

```

33      % variable is initialized, but \ProvidesPackage not yet seen
34      \ifx\w\relax % xintcore.sty not yet loaded.
35      \def\z{\endgroup\RequirePackage{xintcore}}%
36      \fi
37      \else
38      \aftergroup\endinput % xintgcd already loaded.
39      \fi
40      \fi
41      \fi
42 \z%
43 \XINTsetupcatcodes% defined in xintkernel.sty

```

6.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintgcd}%
46 [2016/12/13 1.2i Euclide algorithm with xint package (JFB)]%

```

6.3 *\xintGCD*, *\xintiGCD*

```

47 \def\xintGCD {\romannumeral0\xintgcd}%
48 \def\xintgcd #1%
49 {%
50   \expandafter\XINT_gcd\expandafter{\romannumeral0\xintiabs {#1}}%
51 }%
52 \def\XINT_gcd #1#2%
53 {%
54   \expandafter\XINT_gcd_fork\romannumeral0\xintiabs {#2}\Z #1\Z
55 }%
56 \def\xintiGCD {\romannumeral0\xintiigcd}%
57 \def\xintiigcd #1%
58 {%
59   \expandafter\XINT_iigcd\expandafter{\romannumeral0\xintiiabs {#1}}%
60 }%
61 \def\XINT_iigcd #1#2%
62 {%
63   \expandafter\XINT_gcd_fork\romannumeral0\xintiiabs {#2}\Z #1\Z
64 }%

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

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

```

```

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

#1=B, #2=A

81 \def\xINT_gcd_loop #1#2%
82 {%
83   \expandafter\expandafter\expandafter
84     \XINT_gcd_CheckRem
85   \expandafter\xint_seconddoftwo
86   \romannumeral0\xINT_div_prepare {#1}{#2}\Z
87   {#1}%
88 }%

```

6.4 \xintLCM, \xintiilCM

```

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

```

6.5 \xintBezout

```

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

```

```

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

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

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

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

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

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

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

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

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

```

```

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

```

minusplus $A > 0, B < 0$

```

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

```

plusminus $A < 0, B > 0$

```

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

```

plusplus

```

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

```

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

```

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

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

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

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

223 \def\XINT_bezout_loop_b #1#2#3#4#5#6#7#8%
224 {%
225     \expandafter \XINT_bezout_loop_c \expandafter
226     {\romannumeral0\xintiiadd{\XINT_mul_fork #5\Z #2\Z}{#7}}%
227     {\romannumeral0\xintiiadd{\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}}...{A}{B}}\Z
a(n) = r(n-1). Pour n=0 on a juste {0}{B}{A}{A}{B}}\Z
\XINT_div_prepare {u}{v} divise v par u
```



```

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

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

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

r(n+1)\Z {n+1}{r(n+1)}{r(n)}{{q(n+1)}{r(n+1)}}{{qn}{rn}}...
Test si r(n+1) est nul.

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

{n+1}{r(n+1)}{r(n)}{{q(n+1)}{r(n+1)}}...{\Z Ici r(n+1) = 0. On arrête on se prépare à inverser
{n+1}{0}{r(n)}{{q(n+1)}{r(n+1)}}....{{q1}{r1}}{{A}{B}}{\Z
On veut renvoyer: {N=n+1}{A}{D=r(n)}{B}{q1}{r1}{q2}{r2}{q3}{r3}....{qN}{rN=0}}

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

```

6.7 \xintBezoutAlgorithm

Pour Bezout: objectif, renvoyer

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

```

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

```

```

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

```

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

```

320 \def\XINT_bezalg_fork #1#2\Z #3#4\Z
321 {%
322     \xint_UDzerofork
323     #1\XINT_bezalg_BisZero
324     #3\XINT_bezalg_AisZero
325     0\XINT_bezalg_a
326     \krof
327     0{#1#2}{#3#4}1001{{#3#4}{#1#2}}{}}\Z
328 }%
329 \def\XINT_bezalg_AisZero #1#2#3\Z{ {1}{0}{0}{1}{#2}{#2}{1}{0}{0}{0}{0}{1}}%
330 \def\XINT_bezalg_BisZero #1#2#3#4\Z{ {1}{0}{0}{1}{#3}{#3}{1}{0}{0}{0}{0}{1}}%

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

```

r(n+1)\Z {n+1}{r(n+1)}{r(n)}{alpha(n+1)}{beta(n+1)}
{alpha(n)}{beta(n)}{q,r,alpha,beta(n+1)}
Test si r(n+1) est nul.

```

```

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

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

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

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

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

6.8 \xintGCDof

```

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

6.9 \xintLCMof

New with 1.09a

```

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

6.10 `\xintTypesetEuclideanAlgorithm`

TYPESETTING

Organisation:

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

$\backslash U1 = N = \text{nombre d'étapes}$, $\backslash U3 = \text{PGCD}$, $\backslash U2 = A$, $\backslash U4=B$ $q_1 = \backslash U5$, $q_2 = \backslash U7 \rightarrow q_n = \backslash U<2n+3>$, $r_n = \backslash U<2n+4>$ $bn = rn$. $B = r0$. $A=r(-1)$

$r(n-2) = q(n)r(n-1)+r(n)$ (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   \beginingroup
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 -T _E X and reload detection . . .	166	.39	<code>\xintiBinomial</code>	196
.2	Package identification	167	.40	<code>\xintiPFactorial</code>	196
.3	<code>\XINT_cntSgnFork</code>	167	.41	<code>\xintPrd</code>	196
.4	<code>\xintLen</code>	168	.42	<code>\xintDiv</code>	197
.5	<code>\XINT_outfrac</code>	168	.43	<code>\xintDivFloor</code>	197
.6	<code>\XINT_inFrac</code>	169	.44	<code>\xintDivTrunc</code>	197
.7	<code>\XINT_frac_gen</code>	170	.45	<code>\xintDivRound</code>	197
.8	<code>\XINT_factoritens</code> , <code>\XINT_cuz_cnt</code> . . .	172	.46	<code>\xintMod</code>	197
.9	<code>\xintRaw</code>	174	.47	<code>\xintIsOne</code>	198
.10	<code>\xintPRaw</code>	174	.48	<code>\xintGeq</code>	198
.11	<code>\xintRawWithZeros</code>	175	.49	<code>\xintMax</code>	199
.12	<code>\xintFloor</code> , <code>\xintiFloor</code>	175	.50	<code>\xintMaxof</code>	200
.13	<code>\xintCeil</code> , <code>\xintiCeil</code>	176	.51	<code>\xintMin</code>	200
.14	<code>\xintNumerator</code>	176	.52	<code>\xintMinof</code>	201
.15	<code>\xintDenominator</code>	176	.53	<code>\xintCmp</code>	201
.16	<code>\xintFrac</code>	177	.54	<code>\xintAbs</code>	203
.17	<code>\xintSignedFrac</code>	177	.55	<code>\xintOpp</code>	203
.18	<code>\xintFwOver</code>	178	.56	<code>\xintSgn</code>	203
.19	<code>\xintSignedFwOver</code>	178	.57	Floating point macros	203
.20	<code>\xintREZ</code>	179	.58	<code>\xintFloat</code>	203
.21	<code>\xintE</code>	179	.59	<code>\XINTinFloat</code>	209
.22	<code>\xintIrr</code>	180	.60	<code>\xintPFloat</code>	212
.23	<code>\xintifInt</code>	181	.61	<code>\XINTinFloatFracdigits</code>	214
.24	<code>\xintJrr</code>	181	.62	<code>\xintFloatAdd</code> , <code>\XINTinFloatAdd</code> . . .	214
.25	<code>\xintTFrac</code>	182	.63	<code>\xintFloatSub</code> , <code>\XINTinFloatSub</code> . . .	215
.26	<code>\xintTrunc</code> , <code>\xintiTrunc</code>	183	.64	<code>\xintFloatMul</code> , <code>\XINTinFloatMul</code> . . .	216
.27	<code>\xintTTrunc</code>	185	.65	<code>\xintFloatDiv</code> , <code>\XINTinFloatDiv</code> . . .	217
.28	<code>\xintNum</code>	185	.66	<code>\xintFloatPow</code> , <code>\XINTinFloatPow</code> . . .	217
.29	<code>\xintRound</code> , <code>\xintiRound</code>	185	.67	<code>\xintFloatPower</code> , <code>\XINTinFloatPower</code> . .	221
.30	<code>\xintXTrunc</code>	186	.68	<code>\xintFloatFac</code> , <code>\XINTFloatFac</code>	224
.31	<code>\xintDigits</code>	191	.69	<code>\xintFloatPFactorial</code> , <code>\XINTinFloatP-</code> <code>Factorial</code>	229
.32	<code>\xintAdd</code>	191	.70	<code>\xintFloatBinomial</code> , <code>\XINTinFloatBino-</code> <code>mial</code>	232
.33	<code>\xintSub</code>	193	.71	<code>\xintFloatSqrt</code> , <code>\XINTinFloatSqrt</code> . . .	234
.34	<code>\xintSum</code>	193	.72	<code>\xintFloatE</code> , <code>\XINTinFloatE</code>	236
.35	<code>\xintMul</code>	194	.73	<code>\XINTinFloatMod</code>	236
.36	<code>\xintSqr</code>	194			
.37	<code>\xintPow</code>	195			
.38	<code>\xintiFac</code>	196			

The commenting is currently (2016/12/13) very sparse.

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

```

7.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintfrac}%
46 [2016/12/13 1.2i Expandable operations on fractions (JFB)]%

```

7.3 *\XINT_cntSgnFork*

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

```

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

```

```

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

```

7.4 `\xintLen`

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

```

54 \def\xintLen {\romannumeral0\xintlen }%
55 \def\xintlen #1%
56 {%
57   \expandafter\XINT_flen\romannumeral0\XINT_infrac {#1}%
58 }%
59 \def\XINT_flen #1#2#3%
60 {%
61   \expandafter\space
62   \the\numexpr \XINT_abs#1+\xint_c_viii
63   \XINT_length_loop
64   #2#3\xint_relax\xint_relax\xint_relax\xint_relax
65   \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
66   \xint_c_i\xint_c_ii\xint_c_iii\xint_c_iv
67   \xint_c_v\xint_c_vi\xint_c_vii\xint_c_viii\xint_c_ix\xint_bye
68   \relax
69 }%

```

7.5 `\XINT_outfrac`

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

```

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

```



```

90 \def\XINT_outfrac_N #1#2%
91 {%
92   \expandafter\XINT_outfrac_N_a\expandafter
93   {\romannumeral0\XINT_opp #2}{\romannumeral0\XINT_opp #1}%
94 }%
95 \def\XINT_outfrac_N_a #1#2%
96 {%
97   \expandafter\XINT_outfrac_P\expandafter {#2}{#1}%
98 }%

```

7.6 \XINT_inFrac

Parses fraction, scientific notation, etc... and produces $\{n\}{A}{B}$ corresponding to A/B times 10^n . No reduction to smallest terms.

Extended in 1.07 to accept scientific notation on input. With lowercase e only. The `\xintexpr` parser does accept uppercase E also. Ah, by the way, perhaps I should at least say what this macro does? (belated addition 2014/10/22...), before I forget! It prepares the fraction in the internal format `{exponent}{Numerator}{Denominator}` where Denominator is at least 1.

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

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

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

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

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

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

Regarding calling the macro with an argument `A[<expression>]`, a `/` in the expression must be suitably hidden for example in `\firstofone` type constructs.

Note: when the numerator is found to be zero `\XINT_infrac` *always* returns `{0}{0}{1}`. This behaviour must not change because 1.2g `\xintFloat` and `XINTinFloat` (for example) rely upon it: if the denominator on output is not 1, then `\xintFloat` assumes that the numerator is not zero.

As described in the manual, if the input contains a (final) $[N]$ part, it is assumed that it is in the shape $A[N]$ or $A/B[N]$ with A (and B) not containing neither decimal mark nor scientific part, moreover B must be positive and A have at most one minus sign (and no plus sign). Else there will be errors, for example `-0/2[0]` would not be recognized as being zero at this stage and this could cause issues afterwards. When there is no ending $[N]$ part, both numerator and denominator will be parsed for the more general format allowing decimal digits and scientific part and possibly multiple leading signs.

```

99 \def\XINT_inFrac {\romannumeral0\XINT_infrac }%

```

```

100 \def\XINT_infrac #1%
101 {%
102   \expandafter\XINT_infrac_fork\romannumeral`&&@#1/\XINT_W[\XINT_W\XINT_T
103 }%
104 \def\XINT_infrac_fork #1[#2%
105 {%
106   \xint_UDXINTWfork
107   #2\XINT_frac_gen
108   \XINT_W\XINT_infrac_res_a % strict A[N] or A/B[N] input
109   \krof
110   #1[#2%
111 }%
112 \def\XINT_infrac_res_a #1%
113 {%
114   \xint_gob_til_zero #1\XINT_infrac_res_zero 0\XINT_infrac_res_b #1%
115 }%
116 \def\XINT_infrac_res_zero 0\XINT_infrac_res_b #1\XINT_T {{0}{0}{1}}%
117 \def\XINT_infrac_res_b #1/#2%
118 {%
119   \xint_UDXINTWfork
120   #2\XINT_infrac_res_ca
121   \XINT_W\XINT_infrac_res_cb
122   \krof
123   #1/#2%
124 }%

```

An empty [] is not allowed. (this was authorized in 1.2, removed in 1.2f). As nobody reads xint documentation, no one will have noticed the fleeting possibility.

```

125 \def\XINT_infrac_res_ca #1[#2]/\XINT_W[\XINT_W\XINT_T
126   {\expandafter{\the\numexpr #2}{#1}{1}}%
127 \def\XINT_infrac_res_cb #1/#2[%
128   {\expandafter\XINT_infrac_res_cc\romannumeral`&&@#2~#1[%
129 \def\XINT_infrac_res_cc #1~#2[#3]/\XINT_W[\XINT_W\XINT_T
130   {\expandafter{\the\numexpr #3}{#2}{#1}}%

```

7.7 \XINT_frac_gen

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

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

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

```

131 \def\XINT_frac_gen #1/#2%
132 {%
133   \xint_UDXINTWfork
134   #2\XINT_frac_gen_A
135   \XINT_W\XINT_frac_gen_B
136   \krof
137   #1/#2%

```

7 Package *xintfrac* implementation

```

138 }%
139 \def\XINT_frac_gen_A #1/\XINT_W [\XINT_W {\XINT_frac_gen_C 0~1!#1ee.\XINT_W }%
140 \def\XINT_frac_gen_B #1/#2/\XINT_W[%\XINT_W
141 {%
142     \expandafter\XINT_frac_gen_Ba
143     \romannumeral`&&@#2ee.\XINT_W\XINT_Z #1ee.%\XINT_W
144 }%
145 \def\XINT_frac_gen_Ba #1.#2%
146 {%
147     \xint_UDXINTWfork
148     #2\XINT_frac_gen_Bb
149     \XINT_W\XINT_frac_gen_Bc
150     \krof
151     #1.#2%
152 }%
153 \def\XINT_frac_gen_Bb #1e#2e#3\XINT_Z
154     {\expandafter\XINT_frac_gen_C\the\numexpr #2+\xint_c_~#1!}%
155 \def\XINT_frac_gen_Bc #1.#2e%
156 {%
157     \expandafter\XINT_frac_gen_Bd\romannumeral`&&@#2.#1e%
158 }%
159 \def\XINT_frac_gen_Bd #1.#2e#3e#4\XINT_Z
160 {%
161     \expandafter\XINT_frac_gen_C\the\numexpr #3-%
162     \numexpr\xint_c_ix\XINT_length_loop
163     #1\xint_relax\xint_relax\xint_relax\xint_relax
164     \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
165     \xint_c_i\xint_c_ii\xint_c_iii\xint_c_iv
166     \xint_c_v\xint_c_vi\xint_c_vii\xint_c_viii\xint_c_ix\xint_bye
167     ~#2#1!%
168 }%
169 \def\XINT_frac_gen_C #1!#2.#3%
170 {%
171     \xint_UDXINTWfork
172     #3\XINT_frac_gen_Ca
173     \XINT_W\XINT_frac_gen_Cb
174     \krof
175     #1!#2.#3%
176 }%
177 \def\XINT_frac_gen_Ca #1~#2!#3e#4e#5\XINT_T
178 {%
179     \expandafter\XINT_frac_gen_F\the\numexpr #4-#1\expandafter
180     ~\romannumeral0\XINT_num_loop
181     #2\xint_relax\xint_relax\xint_relax\xint_relax
182     \xint_relax\xint_relax\xint_relax\xint_relax\Z~#3~%
183 }%
184 \def\XINT_frac_gen_Cb #1.#2e%
185 {%
186     \expandafter\XINT_frac_gen_Cc\romannumeral`&&@#2.#1e%
187 }%
188 \def\XINT_frac_gen_Cc #1.#2~#3!#4e#5e#6\XINT_T
189 {%

```

```

190 \expandafter\XINT_frac_gen_F\the\numexpr #5-#2-%
191 \numexpr\xint_c_ix\XINT_length_loop
192 #1\xint_relax\xint_relax\xint_relax\xint_relax
193 \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
194 \xint_c_i\xint_c_ii\xint_c_iii\xint_c_iv
195 \xint_c_v\xint_c_vi\xint_c_vii\xint_c_viii\xint_c_ix\xint_bye
196 \relax\expandafter~\romannumeral0\XINT_num_loop
197 #3\xint_relax\xint_relax\xint_relax\xint_relax
198 \xint_relax\xint_relax\xint_relax\xint_relax\Z
199 ~#4#1~%
200 }%
201 \def\XINT_frac_gen_F #1~#2%
202 {%
203 \xint_UDzerominusfork
204 #2-\XINT_frac_gen_Gdivbyzero
205 0#2{\XINT_frac_gen_G -{}}%
206 0-{\XINT_frac_gen_G {}#2}%
207 \krof #1~%
208 }%
209 \def\XINT_frac_gen_Gdivbyzero #1~~#2~%
210 {%
211 \expandafter\XINT_frac_gen_Gdivbyzero_a
212 \romannumeral0\XINT_num_loop
213 #2\xint_relax\xint_relax\xint_relax\xint_relax
214 \xint_relax\xint_relax\xint_relax\xint_relax\Z~#1~%
215 }%
216 \def\XINT_frac_gen_Gdivbyzero_a #1~#2~%
217 {%
218 \xintError:DivisionByZero {#2}{#1}{0}%
219 }%
220 \def\XINT_frac_gen_G #1#2#3~#4~#5~%
221 {%
222 \expandafter\XINT_frac_gen_Ga
223 \romannumeral0\XINT_num_loop
224 #1#5\xint_relax\xint_relax\xint_relax\xint_relax
225 \xint_relax\xint_relax\xint_relax\xint_relax\Z~#3~{#2#4}%
226 }%
227 \def\XINT_frac_gen_Ga #1#2~#3~%
228 {%
229 \xint_gob_til_zero #1\XINT_frac_gen_zero 0%
230 {#3}{#1#2}%
231 }%
232 \def\XINT_frac_gen_zero 0#1#2#3{{0}{0}{1}}%

```

7.8 \XINT_factortens, \XINT_cuz_cnt

Old routines.

```

233 \def\XINT_factortens #1%
234 {%
235 \expandafter\XINT_cuz_cnt_loop\expandafter
236 {\expandafter}\romannumeral0\XINT_rord_main {}#1%
237 \xint_relax

```

```

238      \xint_bye\xint_bye\xint_bye\xint_bye
239      \xint_bye\xint_bye\xint_bye\xint_bye
240      \xint_relax
241      \R\R\R\R\R\R\R\R\Z
242 }%
243 \def\xint_cuz_cnt #1%
244 {%
245     \XINT_cuz_cnt_loop {}#1\R\R\R\R\R\R\R\R\Z
246 }%
247 \def\xint_cuz_cnt_loop #1#2#3#4#5#6#7#8#9%
248 {%
249     \xint_gob_til_R #9\xint_cuz_cnt_toofara \R
250     \expandafter\xint_cuz_cnt_checka\expandafter
251     {\the\numexpr #1+8\relax}{#2#3#4#5#6#7#8#9}%
252 }%
253 \def\xint_cuz_cnt_toofara\R
254     \expandafter\xint_cuz_cnt_checka\expandafter #1#2%
255 {%
256     \XINT_cuz_cnt_toofarb {}#1#2%
257 }%
258 \def\xint_cuz_cnt_toofarb #1#2\Z {\XINT_cuz_cnt_toofarc #2\Z {}#1}%
259 \def\xint_cuz_cnt_toofarc #1#2#3#4#5#6#7#8%
260 {%
261     \xint_gob_til_R #2\xint_cuz_cnt_toofard 7%
262         #3\xint_cuz_cnt_toofard 6%
263         #4\xint_cuz_cnt_toofard 5%
264         #5\xint_cuz_cnt_toofard 4%
265         #6\xint_cuz_cnt_toofard 3%
266         #7\xint_cuz_cnt_toofard 2%
267         #8\xint_cuz_cnt_toofard 1%
268         \Z #1#2#3#4#5#6#7#8%
269 }%
270 \def\xint_cuz_cnt_toofard #1#2\Z #3\R #4\Z #5%
271 {%
272     \expandafter\xint_cuz_cnt_toofare
273     \the\numexpr #3\relax \R\R\R\R\R\R\R\R\Z
274     {\the\numexpr #5-#1\relax}\R\Z
275 }%
276 \def\xint_cuz_cnt_toofare #1#2#3#4#5#6#7#8%
277 {%
278     \xint_gob_til_R #2\xint_cuz_cnt_stopc 1%
279         #3\xint_cuz_cnt_stopc 2%
280         #4\xint_cuz_cnt_stopc 3%
281         #5\xint_cuz_cnt_stopc 4%
282         #6\xint_cuz_cnt_stopc 5%
283         #7\xint_cuz_cnt_stopc 6%
284         #8\xint_cuz_cnt_stopc 7%
285         \Z #1#2#3#4#5#6#7#8%
286 }%
287 \def\xint_cuz_cnt_checka #1#2%
288 {%
289     \expandafter\xint_cuz_cnt_checkb\the\numexpr #2\relax \Z {}#1}%

```

```

290 }%
291 \def\XINT_cuz_cnt_checkb #1%
292 {%
293   \xint_gob_til_zero #1\expandafter\XINT_cuz_cnt_loop\xint_gob_til_Z
294   0\XINT_cuz_cnt_stopa #1%
295 }%
296 \def\XINT_cuz_cnt_stopa #1\Z
297 {%
298   \XINT_cuz_cnt_stopb #1\R\R\R\R\R\R\R\Z %
299 }%
300 \def\XINT_cuz_cnt_stopb #1#2#3#4#5#6#7#8#9%
301 {%
302   \xint_gob_til_R #2\XINT_cuz_cnt_stopc 1%
303       #3\XINT_cuz_cnt_stopc 2%
304       #4\XINT_cuz_cnt_stopc 3%
305       #5\XINT_cuz_cnt_stopc 4%
306       #6\XINT_cuz_cnt_stopc 5%
307       #7\XINT_cuz_cnt_stopc 6%
308       #8\XINT_cuz_cnt_stopc 7%
309       #9\XINT_cuz_cnt_stopc 8%
310       \Z #1#2#3#4#5#6#7#8#9%
311 }%
312 \def\XINT_cuz_cnt_stopc #1#2\Z #3\R #4\Z #5%
313 {%
314   \expandafter\XINT_cuz_cnt_stopd\expandafter
315   {\the\numexpr #5-#1}\#3%
316 }%
317 \def\XINT_cuz_cnt_stopd #1#2\R #3\Z
318 {%
319   \expandafter\space\expandafter
320   {\romannumeral0\XINT_rord_main {}}#2%
321   \xint_relax
322   \xint_bye\xint_bye\xint_bye\xint_bye
323   \xint_bye\xint_bye\xint_bye\xint_bye
324   \xint_relax }{\#1}%
325 }%

```

7.9 \xintRaw

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

```

326 \def\xintRaw {\romannumeral0\xintra #1}%
327 \def\xintra #1%
328 {%
329   \expandafter\XINT_raw\romannumeral0\XINT_infrac
330 }%
331 \def\XINT_raw #1#2#3{ #2/#3[#1]}%

```

7.10 \xintPraw

1.09b

```

332 \def\xintPraw {\romannumeral0\xintpraw }%
333 \def\xintpraw
334 {%
335   \expandafter\XINT_praw\romannumeral0\XINT_infrac
336 }%
337 \def\XINT_praw #1%
338 {%
339   \ifnum #1=\xint_c_ \expandafter\XINT_praw_a\fi \XINT_praw_A {#1}%
340 }%
341 \def\XINT_praw_A #1#2#3%
342 {%
343   \if\XINT_isOne{#3}1\expandafter\xint_firstoftwo
344     \else\expandafter\xint_secondoftwo
345   \fi { #2[#1]}{ #2/#3[#1]}%
346 }%
347 \def\XINT_praw_a\XINT_praw_A #1#2#3%
348 {%
349   \if\XINT_isOne{#3}1\expandafter\xint_firstoftwo
350     \else\expandafter\xint_secondoftwo
351   \fi { #2}{ #2/#3}%
352 }%

```

7.11 \xintRawWithZeros

This was called \xintRaw in versions earlier than 1.07

```

353 \def\xintRawWithZeros {\romannumeral0\xintrawwithzeros }%
354 \def\xintrawwithzeros
355 {%
356   \expandafter\XINT_rawz_fork\romannumeral0\XINT_infrac
357 }%
358 \def\XINT_rawz_fork #1%
359 {%
360   \ifnum#1<\xint_c_
361     \expandafter\XINT_rawz_Ba
362   \else
363     \expandafter\XINT_rawz_A
364   \fi
365   #1.%
366 }%
367 \def\XINT_rawz_A #1.#2#3{\XINT_dsx_addzeros{#1}#2;/#3}%
368 \def\XINT_rawz_Ba -#1.#2#3{\expandafter\XINT_rawz_Bb
369   \expandafter{\romannumeral0\XINT_dsx_addzeros{#1}#3;}{#2}}%
370 \def\XINT_rawz_Bb #1#2{ #2/#1}%

```

7.12 \xintFloor, \xintiFloor

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

```

371 \def\xintFloor {\romannumeral0\xintfloor }%
372 \def\xintfloor #1% devrais-je faire \xintREZ?
373   {\expandafter\XINT_ifloor \romannumeral0\xintrawwithzeros {#1}./1[0]}%

```

```

374 \def\xintiFloor {\romannumeral0\xintifloor }%
375 \def\xintifloor #1%
376   {\expandafter\XINT_ifloor \romannumeral0\xintraffwithzeros {#1}.}%
377 \def\XINT_ifloor #1/#2.{\xintiifquo {#1}{#2}}%

```

7.13 \xintCeil, \xintiCeil

1.09a

```

378 \def\xintCeil {\romannumeral0\xintceil }%
379 \def\xintceil #1{\xintiifopp {\xintFloor {\xintOpp{#1}}}}%
380 \def\xintiCeil {\romannumeral0\xinticeil }%
381 \def\xinticeil #1{\xintiifopp {\xintiFloor {\xintOpp{#1}}}}%

```

7.14 \xintNumerator

```

382 \def\xintNumerator {\romannumeral0\xintnumerator }%
383 \def\xintnumerator
384 {%
385   \expandafter\XINT_numer\romannumeral0\XINT_infrac
386 }%
387 \def\XINT_numer #1%
388 {%
389   \ifcase\XINT_cntSgn #1\Z
390     \expandafter\XINT_numer_B
391   \or
392     \expandafter\XINT_numer_A
393   \else
394     \expandafter\XINT_numer_B
395   \fi
396   {#1}%
397 }%
398 \def\XINT_numer_A #1#2#3{\XINT_dsx_addzeros{#1}#2;}%
399 \def\XINT_numer_B #1#2#3{ #2}%

```

7.15 \xintDenominator

```

400 \def\xintDenominator {\romannumeral0\xintdenominator }%
401 \def\xintdenominator
402 {%
403   \expandafter\XINT_denom_fork\romannumeral0\XINT_infrac
404 }%
405 \def\XINT_denom_fork #1%
406 {%
407   \ifnum#1<\xint_c_
408     \expandafter\XINT_denom_B
409   \else
410     \expandafter\XINT_denom_A
411   \fi
412   #1.%
413 }%
414 \def\XINT_denom_A #1.#2#3{ #3}%
415 \def\XINT_denom_B -#1.#2#3{\XINT_dsx_addzeros{#1}#3;}%

```


7.16 `\xintFrac`

Useless typesetting macro.

```

416 \def\xintFrac {\romannumeral0\xintfrac }%
417 \def\xintfrac #1%
418 {%
419   \expandafter\XINT_fracfrac_A\romannumeral0\XINT_infrac {#1}%
420 }%
421 \def\XINT_fracfrac_A #1{\XINT_fracfrac_B #1\Z }%
422 \catcode`^=7
423 \def\XINT_fracfrac_B #1#2\Z
424 {%
425   \xint_gob_til_zero #1\XINT_fracfrac_C 0\XINT_fracfrac_D {10^{#1#2}}%
426 }%
427 \def\XINT_fracfrac_C 0\XINT_fracfrac_D #1#2#3%
428 {%
429   \if1\XINT_isOne {#3}%
430     \xint_afterfi {\expandafter\xint_firstoftwo_thenstop\xint_gobble_ii }%
431   \fi
432   \space
433   \frac {#2}{#3}%
434 }%
435 \def\XINT_fracfrac_D #1#2#3%
436 {%
437   \if1\XINT_isOne {#3}\XINT_fracfrac_E\fi
438   \space
439   \frac {#2}{#3}#1%
440 }%
441 \def\XINT_fracfrac_E \fi\space\frac #1#2{\fi \space #1\cdot }%

```

7.17 `\xintSignedFrac`

```

442 \def\xintSignedFrac {\romannumeral0\xintsignedfrac }%
443 \def\xintsignedfrac #1%
444 {%
445   \expandafter\XINT_sgnfrac_a\romannumeral0\XINT_infrac {#1}%
446 }%
447 \def\XINT_sgnfrac_a #1#2%
448 {%
449   \XINT_sgnfrac_b #2\Z {#1}%
450 }%
451 \def\XINT_sgnfrac_b #1%
452 {%
453   \xint_UDsignfork
454     #1\XINT_sgnfrac_N
455     -{\XINT_sgnfrac_P #1}%
456   \krof
457 }%
458 \def\XINT_sgnfrac_P #1\Z #2%
459 {%
460   \XINT_fracfrac_A {#2}{#1}%
461 }%

```

```

462 \def\XINT_sgnfrac_N
463 {%
464   \expandafter\romannumeral0\XINT_sgnfrac_P
465 }%

```

7.18 \xintFwOver

```

466 \def\xintFwOver {\romannumeral0\xintfwover }%
467 \def\xintfwover #1%
468 {%
469   \expandafter\XINT_fwover_A\romannumeral0\XINT_infrac {#1}%
470 }%
471 \def\XINT_fwover_A #1{\XINT_fwover_B #1\Z }%
472 \def\XINT_fwover_B #1#2\Z
473 {%
474   \xint_gob_til_zero #1\XINT_fwover_C 0\XINT_fwover_D {10^{#1#2}}%
475 }%
476 \catcode\^=11
477 \def\XINT_fwover_C #1#2#3#4#5%
478 {%
479   \if0\XINT_isOne {#5}\xint_afterfi { {#4\over #5}}%
480   \else\xint_afterfi { #4}%
481   \fi
482 }%
483 \def\XINT_fwover_D #1#2#3%
484 {%
485   \if0\XINT_isOne {#3}\xint_afterfi { {#2\over #3}}%
486   \else\xint_afterfi { #2\cdot }%
487   \fi
488   #1%
489 }%

```

7.19 \xintSignedFwOver

```

490 \def\xintSignedFwOver {\romannumeral0\xintsignedfwover }%
491 \def\xintsignedfwover #1%
492 {%
493   \expandafter\XINT_sgnfwover_a\romannumeral0\XINT_infrac {#1}%
494 }%
495 \def\XINT_sgnfwover_a #1#2%
496 {%
497   \XINT_sgnfwover_b #2\Z {#1}%
498 }%
499 \def\XINT_sgnfwover_b #1%
500 {%
501   \xint_UDsignfork
502   #1\XINT_sgnfwover_N
503   -{\XINT_sgnfwover_P #1}%
504   \krof
505 }%
506 \def\XINT_sgnfwover_P #1\Z #2%
507 {%
508   \XINT_fwover_A {#2}{#1}%
509 }%
510 \def\XINT_sgnfwover_N

```

```

511 {%
512   \expandafter-\romannumeral0\XINT_sgnfwover_P
513 }%

```

7.20 \xintREZ

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

```

514 \def\xintREZ {\romannumeral0\xintrez }%
515 \def\xintrez
516 {%
517   \expandafter\XINT_rez_A\romannumeral0\XINT_infrac
518 }%
519 \def\XINT_rez_A #1#2%
520 {%
521   \XINT_rez_AB #2\Z {#1}%
522 }%
523 \def\XINT_rez_AB #1%
524 {%
525   \xint_UDzerominusfork
526   #1-\XINT_rez_zero
527   0#1\XINT_rez_neg
528   0-{\XINT_rez_B #1}%
529   \krof
530 }%
531 \def\XINT_rez_zero #1\Z #2#3{ 0/1[0]}%
532 \def\XINT_rez_neg {\expandafter-\romannumeral0\XINT_rez_B }%
533 \def\XINT_rez_B #1\Z
534 {%
535   \expandafter\XINT_rez_C\romannumeral0\XINT_factortens {#1}%
536 }%
537 \def\XINT_rez_C #1#2#3#4%
538 {%
539   \expandafter\XINT_rez_D\romannumeral0\XINT_factortens {#4}{#3}{#2}{#1}%
540 }%
541 \def\XINT_rez_D #1#2#3#4#5%
542 {%
543   \expandafter\XINT_rez_E\expandafter
544   {\the\numexpr #3+#4-#2}{#1}{#5}%
545 }%
546 \def\XINT_rez_E #1#2#3{ #3/#2[#1]}%

```

7.21 \xintE

1.07: The fraction is the first argument contrarily to \xintTrunc and \xintRound.
 1.1 modifies and moves \xintiiE to xint.sty.

```

547 \def\xintE {\romannumeral0\xinte }%
548 \def\xinte #1%
549 {%
550   \expandafter\XINT_e \romannumeral0\XINT_infrac {#1}%
551 }%
552 \def\XINT_e #1#2#3#4%

```

```

553 {%
554   \expandafter\XINT_e_end\the\numexpr #1+#4.{#2}{#3}%
555 }%
556 \def\XINT_e_end #1.#2#3{ #2/#3[#1]}%

```

7.22 \xintIrr

```

557 \def\xintIrr {\romannumeral0\xintirr }%
558 \def\xintirr #1%
559 {%
560   \expandafter\XINT_irr_start\romannumeral0\xintrawithzeros {#1}\Z
561 }%
562 \def\XINT_irr_start #1#2/#3\Z
563 {%
564   \if0\XINT_isOne {#3}%
565     \xint_afterfi
566       {\xint_UDsignfork
567         #1\XINT_irr_negative
568         -{\XINT_irr_nonneg #1}%
569       \krof}%
570   \else
571     \xint_afterfi{\XINT_irr_denomisine #1}%
572   \fi
573   #2\Z {#3}%
574 }%
575 \def\XINT_irr_denomisine #1\Z #2{ #1/1}% changed in 1.08
576 \def\XINT_irr_negative #1\Z #2{\XINT_irr_D #1\Z #2\Z -}%
577 \def\XINT_irr_nonneg #1\Z #2{\XINT_irr_D #1\Z #2\Z \space}%
578 \def\XINT_irr_D #1#2\Z #3#4\Z
579 {%
580   \xint_UDzerosfork
581     #3#1\XINT_irr_indeterminate
582     #30\XINT_irr_divisionbyzero
583     #10\XINT_irr_zero
584     00\XINT_irr_loop_a
585   \krof
586   {#3#4}{#1#2}{#3#4}{#1#2}%
587 }%
588 \def\XINT_irr_indeterminate #1#2#3#4#5{\xintError:NaN\space 0/0}%
589 \def\XINT_irr_divisionbyzero #1#2#3#4#5{\xintError:DivisionByZero #5#2/0}%
590 \def\XINT_irr_zero #1#2#3#4#5{ 0/1}% changed in 1.08
591 \def\XINT_irr_loop_a #1#2%
592 {%
593   \expandafter\XINT_irr_loop_d
594   \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
595 }%
596 \def\XINT_irr_loop_d #1#2%
597 {%
598   \XINT_irr_loop_e #2\Z
599 }%
600 \def\XINT_irr_loop_e #1#2\Z
601 {%
602   \xint_gob_til_zero #1\XINT_irr_loop_exit0\XINT_irr_loop_a {#1#2}%

```

```

603 }%
604 \def\XINT_irr_loop_exit0\XINT_irr_loop_a #1#2#3#4%
605 {%
606   \expandafter\XINT_irr_loop_exitb\expandafter
607   {\romannumeral0\xintiigo {#3}{#2}}%
608   {\romannumeral0\xintiigo {#4}{#2}}%
609 }%
610 \def\XINT_irr_loop_exitb #1#2%
611 {%
612   \expandafter\XINT_irr_finish\expandafter {#2}{#1}%
613 }%
614 \def\XINT_irr_finish #1#2#3{#3#1/#2}% changed in 1.08

```

7.23 \xintifInt

```

615 \def\xintifInt {\romannumeral0\xintifint }%
616 \def\xintifint #1{\expandafter\XINT_ifint\romannumeral0\xintraawithzeros {#1}.}%
617 \def\XINT_ifint #1/#2.%
618 {%
619   \if 0\xintiiRem {#1}{#2}%
620   \expandafter\xint_firstoftwo_thenstop
621   \else
622   \expandafter\xint_secondoftwo_thenstop
623   \fi
624 }%

```

7.24 \xintJrr

```

625 \def\xintJrr {\romannumeral0\xintjrr }%
626 \def\xintjrr #1%
627 {%
628   \expandafter\XINT_jrr_start\romannumeral0\xintraawithzeros {#1}\Z
629 }%
630 \def\XINT_jrr_start #1#2/#3\Z
631 {%
632   \if0\XINT_isOne {#3}\xint_afterfi
633     {\xint_UDsignfork
634       #1\XINT_jrr_negative
635       -{\XINT_jrr_nonneg #1}%
636       \krof}%
637   \else
638     \xint_afterfi{\XINT_jrr_denomiseone #1}%
639   \fi
640   #2\Z {#3}%
641 }%
642 \def\XINT_jrr_denomiseone #1\Z #2{ #1/1}% changed in 1.08
643 \def\XINT_jrr_negative #1\Z #2{\XINT_jrr_D #1\Z #2\Z -}%
644 \def\XINT_jrr_nonneg #1\Z #2{\XINT_jrr_D #1\Z #2\Z \space}%
645 \def\XINT_jrr_D #1#2\Z #3#4\Z
646 {%
647   \xint_UDzerosfork
648     #3#1\XINT_jrr_indeterminate
649     #30\XINT_jrr_divisionbyzero
650     #10\XINT_jrr_zero
651     00\XINT_jrr_loop_a

```

```

652 \krof
653 {#3#4}{#1#2}1001%
654 }%
655 \def\XINT_jrr_indeterminate #1#2#3#4#5#6#7{\xintError:NaN\space 0/0}%
656 \def\XINT_jrr_divisionbyzero #1#2#3#4#5#6#7{\xintError:DivisionByZero #7#2/0}%
657 \def\XINT_jrr_zero #1#2#3#4#5#6#7{ 0/1}% changed in 1.08
658 \def\XINT_jrr_loop_a #1#2%
659 {%
660 \expandafter\XINT_jrr_loop_b
661 \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
662 }%
663 \def\XINT_jrr_loop_b #1#2#3#4#5#6#7%
664 {%
665 \expandafter \XINT_jrr_loop_c \expandafter
666 {\romannumeral0\xinttiadd{\XINT_mul_fork #4\Z #1\Z}{#6}}%
667 {\romannumeral0\xinttiadd{\XINT_mul_fork #5\Z #1\Z}{#7}}%
668 {#2}{#3}{#4}{#5}%
669 }%
670 \def\XINT_jrr_loop_c #1#2%
671 {%
672 \expandafter \XINT_jrr_loop_d \expandafter{#2}{#1}%
673 }%
674 \def\XINT_jrr_loop_d #1#2#3#4%
675 {%
676 \XINT_jrr_loop_e #3\Z {#4}{#2}{#1}%
677 }%
678 \def\XINT_jrr_loop_e #1#2\Z
679 {%
680 \xint_gob_til_zero #1\XINT_jrr_loop_exit0\XINT_jrr_loop_a {#1#2}%
681 }%
682 \def\XINT_jrr_loop_exit0\XINT_jrr_loop_a #1#2#3#4#5#6%
683 {%
684 \XINT_irr_finish {#3}{#4}%
685 }%

```

7.25 \xintTFrac

1.09i, for frac in \xintexpr. And \xintFrac is already assigned. T for truncation. However, potentially not very efficient with numbers in scientific notations, with big exponents. Will have to think it again some day. I hesitated how to call the macro. Same convention as in maple, but some people reserve fractional part to $x - \text{floor}(x)$. 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.

```

686 \def\xintTFrac {\romannumeral0\xinttfrac }%
687 \def\xinttfrac #1{\expandafter\XINT_tfrac_fork\romannumeral0\xintrawithzeros {#1}\Z }%
688 \def\XINT_tfrac_fork #1%
689 {%
690 \xint_UDzerominusfork
691 #1-\XINT_tfrac_zero
692 0#1{\xinttiopp\XINT_tfrac_P }%
693 0-{\XINT_tfrac_P #1}%
694 \krof
695 }%

```

```

696 \def\XINT_tfrac_zero #1\Z { 0/1[0]}%
697 \def\XINT_tfrac_P #1/#2\Z {\expandafter\XINT_rez_AB
698 \romannumeral0\xintiirem{#1}{#2}\Z {0}{#2}}%

```

7.26 \xintTrunc, \xintiTrunc

1.2i release notes: ever since its inception this macro was stupid for a decimal input: it did not handle it separately from the general fraction case $A/B[N]$ with $B>1$, hence ended up doing divisions by powers of ten. But this meant that nesting `\xintTrunc` with itself was very inefficient.

1.2i version is better. However it still handles $B>1$, $N<0$ via adding zeros to B and dividing with this extended B . A possibly more efficient approach is implemented in `\xintXTrunc`, but its logic is more complicated, the code is quite longer and making it f-expandable would not shorten it... I decided for the time being to not complicate things here.

```

699 \def\xintTrunc {\romannumeral0\xinttrunc}%
700 \def\xintiTrunc {\romannumeral0\xintitrunc}%
701 \def\xinttrunc #1{\expandafter\XINT_trunc\the\numexpr#1.\XINT_trunc_G}%
702 \def\xintitrunc #1{\expandafter\XINT_trunc\the\numexpr#1.\XINT_itrunc_G}%
703 \def\XINT_trunc #1.#2#3%
704 {%
705   \expandafter\XINT_trunc_a\romannumeral0\XINT_infrac{#3}{#1.#2}%
706 }%
707 \def\XINT_trunc_a #1#2#3#4.#5%
708 {%
709   \if0\XINT_Sgn#2\Z\xint_dothis\XINT_trunc_zero\fi
710   \if1\XINT_iSone#3XY\xint_dothis\XINT_trunc_sp_b\fi
711   \xint_orthat\XINT_trunc_b #1+#4.{#2}{#3}{#5#4.}%
712 }%
713 \def\XINT_trunc_zero #1.#2.{ 0}%
714 \def\XINT_trunc_b {\expandafter\XINT_trunc_B\the\numexpr}%
715 \def\XINT_trunc_sp_b {\expandafter\XINT_trunc_sp_B\the\numexpr}%
716 \def\XINT_trunc_B #1%
717 {%
718   \xint_UDsignfork
719   #1\XINT_trunc_C
720   -\XINT_trunc_D
721   \krof #1%
722 }%
723 \def\XINT_trunc_sp_B #1%
724 {%
725   \xint_UDsignfork
726   #1\XINT_trunc_sp_C
727   -\XINT_trunc_sp_D
728   \krof #1%
729 }%
730 \def\XINT_trunc_C -#1.#2#3%
731 {%
732   \expandafter\XINT_trunc_CE
733   \romannumeral0\XINT_dsx_addzeros{#1}{#3};.{#2}%
734 }%
735 \def\XINT_trunc_CE #1.#2{\XINT_trunc_E #2.{#1}}%
736 \def\XINT_trunc_sp_C -#1.#2#3{\XINT_trunc_sp_Ca #2.#1.}%
737 \def\XINT_trunc_sp_Ca #1%

```

```

738 {%
739   \xint_UDsignfork
740     #1{\XINT_trunc_sp_Cb -}%
741     -{\XINT_trunc_sp_Cb \space#1}%
742   \krof
743 }%
744 \def\XINT_trunc_sp_Cb #1#2.#3.%
745 {%
746   \expandafter\XINT_trunc_sp_Cc
747   \romannumeral0\expandafter\XINT_split_fromright_a
748   \the\numexpr#3-\numexpr\xint_c_ix\XINT_length_loop
749   #2\xint_relax\xint_relax\xint_relax\xint_relax
750   \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
751   \xint_c_i\xint_c_ii\xint_c_iii\xint_c_iv
752   \xint_c_v\xint_c_vi\xint_c_vii\xint_c_viii\xint_c_ix\xint_bye
753   .#2\xint_bye2345678\xint_bye..#1%
754 }%
755 \def\XINT_trunc_sp_Cc #1%
756 {%
757   \if.#1\xint_dothis{\XINT_trunc_sp_Cd 0.}\fi
758   \xint_orthat {\XINT_trunc_sp_Cd #1}%
759 }%
760 \def\XINT_trunc_sp_Cd #1.#2.#3%
761 {%
762   \XINT_trunc_sp_F #3#1.%
763 }%
764 \def\XINT_trunc_D #1.#2%
765 {%
766   \expandafter\XINT_trunc_E
767   \romannumeral0\XINT_dsx_addzeros {#1}#2;.%
768 }%
769 \def\XINT_trunc_sp_D #1.#2#3%
770 {%
771   \expandafter\XINT_trunc_sp_E
772   \romannumeral0\XINT_dsx_addzeros {#1}#2;.%
773 }%
774 \def\XINT_trunc_E #1%
775 {%
776   \xint_UDsignfork
777     #1{\XINT_trunc_F -}%
778     -{\XINT_trunc_F \space#1}%
779   \krof
780 }%
781 \def\XINT_trunc_sp_E #1%
782 {%
783   \xint_UDsignfork
784     #1{\XINT_trunc_sp_F -}%
785     -{\XINT_trunc_sp_F\space#1}%
786   \krof
787 }%
788 \def\XINT_trunc_F #1#2.#3#4%
789   {\expandafter#4\romannumeral`&&\expandafter\xint_firstoftwo

```



```

790          \romannumeral0\XINT_div_prepare {#3}{#2}.#1}%
791 \def\XINT_trunc_sp_F #1#2.#3{#3#2.#1}%
792 \def\XINT_itrunc_G #1#2.#3#4.{\if#10\xint_dothis{ 0}\fi\xint_orthat{#3#1}#2}%
793 \def\XINT_trunc_G #1.#2#3.%
794 {%
795   \expandafter\XINT_trunc_H
796   \the\numexpr\romannumeral0\xintlength {#1}-#3.#3.{#1}#2%
797 }%
798 \def\XINT_trunc_H #1.#2.%
799 {%
800   \ifnum #1 > \xint_c_
801     \xint_afterfi {\XINT_trunc_Ha {#2}}%
802   \else
803     \xint_afterfi {\XINT_trunc_Hb {-#1}}% -0,--1,--2, ....
804   \fi
805 }%
806 \def\XINT_trunc_Ha{\expandafter\XINT_trunc_Haa\romannumeral0\xintdecsplit}%
807 \def\XINT_trunc_Haa #1#2#3{#3#1.#2}%
808 \def\XINT_trunc_Hb #1#2#3%
809 {%
810   \expandafter #3\expandafter0\expandafter.%
811   \romannumeral\xintreplicate{#1}0#2%
812 }%

```

7.27 \xintTTrunc

1.1. Modified in 1.2i, it does simply \xintiTrunc0 with no shortcut (the latter having been modified)

```

813 \def\xintTTrunc {\romannumeral0\xintttrunc }%
814 \def\xintttrunc {\xintitrunc\xint_c_}%

```

7.28 \xintNum

```

815 \let\xintNum \xintTTrunc
816 \let\xintnum \xintttrunc

```

7.29 \xintRound, \xintiRound

Modified in 1.2i.

It benefits first of all from the faster \xintTrunc, particularly when the input is already a decimal number (denominator B=1).

And the rounding is now done in 1.2 style (with much delay, sorry), like of the rewritten \xintInc and \xintDec.

```

817 \def\xintRound {\romannumeral0\xintround }%
818 \def\xintiRound {\romannumeral0\xintiround }%
819 \def\xintround #1{\expandafter\XINT_round\the\numexpr #1.\XINT_round_A}%
820 \def\xintiround #1{\expandafter\XINT_round\the\numexpr #1.\XINT_iround_A}%
821 \def\XINT_round #1.{\expandafter\XINT_round_aa\the\numexpr #1+\xint_c_i.#1}%
822 \def\XINT_round_aa #1.#2.#3#4%
823 {%
824   \expandafter\XINT_round_a\romannumeral0\XINT_infrac{#4}#1.#3#2.%
825 }%

```

```

826 \def\XINT_round_a #1#2#3#4.%
827 {%
828   \if0\XINT_Sgn#2\Z\xint_dothis\XINT_trunc_zero\fi
829   \if1\XINT_iSone#3XY\xint_dothis\XINT_trunc_sp_b\fi
830   \xint_orthat\XINT_trunc_b #1+#4.{#2}{#3}%
831 }%
832 \def\XINT_round_A{\expandafter\XINT_trunc_G\romannumeral0\XINT_round_B}%
833 \def\XINT_iround_A{\expandafter\XINT_itrunc_G\romannumeral0\XINT_round_B}%
834 \def\XINT_round_B #1.%
835   {\XINT_dsrr #1\xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax.}%

```

7.30 \xintXTrunc

1.09j [2014/01/06] This is completely expandable but not f-expandable. Rewritten for 1.2i (2016/12/04):

- no more use of \xintloop from xinttools.sty (replaced by \xintreplicate... from xintkernel.sty),
 - no more use in 0>N>-D case of a dummy control sequence name via \csname...\endcsname
 - handles better the case of an input already a decimal number
- Need to transfer code comments into public dtx.

```

836 \def\xintXTrunc #1#2%
837 {%
838   \expandafter\XINT_xtrunc_a
839   \the\numexpr #1\expandafter.\romannumeral0\xintraw
840 }%
841 \def\XINT_xtrunc_a #1.% ?? faire autre chose
842 {%
843   \expandafter\XINT_xtrunc_b\the\numexpr\ifnum#1<\xint_c_i \xint_c_i-\fi #1.%
844 }%
845 \def\XINT_xtrunc_b #1.#2{\XINT_xtrunc_c #2{#1}}%
846 \def\XINT_xtrunc_c #1%
847 {%
848   \xint_UDzerominusfork
849   #1-\XINT_xtrunc_zero
850   0#1{-\XINT_xtrunc_d {}}%
851   0-{\XINT_xtrunc_d #1}%
852   \krof
853 }%[
854 \def\XINT_xtrunc_zero #1#2]{0.\romannumeral\xintreplicate{#1}0}%
855 \def\XINT_xtrunc_d #1#2#3/#4[#5]%
856 {%
857   \XINT_xtrunc_prepare_a#4\R\R\R\R\R\R\R {10}0000001\W !{#4};{#5}{#2}{#1#3}%
858 }%
859 \def\XINT_xtrunc_prepare_a #1#2#3#4#5#6#7#8#9%
860 {%
861   \xint_gob_til_R #9\XINT_xtrunc_prepare_small\R
862   \XINT_xtrunc_prepare_b #9%
863 }%
864 \def\XINT_xtrunc_prepare_small\R #1!#2;%
865 {%
866   \ifcase #2
867   \or\expandafter\XINT_xtrunc_BisOne

```

```

868 \or\expandafter\XINT_xtrunc_BisTwo
869 \or
870 \or\expandafter\XINT_xtrunc_BisFour
871 \or\expandafter\XINT_xtrunc_BisFive
872 \or
873 \or
874 \or\expandafter\XINT_xtrunc_BisEight
875 \fi\XINT_xtrunc_BisSmall {#2}%
876 }%
877 \def\XINT_xtrunc_BisOne\XINT_xtrunc_BisSmall #1#2#3#4%
878 {\XINT_xtrunc_sp_e {#2}{#4}{#3}}%
879 \def\XINT_xtrunc_BisTwo\XINT_xtrunc_BisSmall #1#2#3#4%
880 {%
881 \expandafter\XINT_xtrunc_sp_e\expandafter
882 {\the\numexpr #2-\xint_c_i\expandafter}\expandafter
883 {\romannumeral0\xintiimul 5{#4}{#3}}%
884 }%
885 \def\XINT_xtrunc_BisFour\XINT_xtrunc_BisSmall #1#2#3#4%
886 {%
887 \expandafter\XINT_xtrunc_sp_e\expandafter
888 {\the\numexpr #2-\xint_c_ii\expandafter}\expandafter
889 {\romannumeral0\xintiimul {25}{#4}{#3}}%
890 }%
891 \def\XINT_xtrunc_BisFive\XINT_xtrunc_BisSmall #1#2#3#4%
892 {%
893 \expandafter\XINT_xtrunc_sp_e\expandafter
894 {\the\numexpr #2-\xint_c_i\expandafter}\expandafter
895 {\romannumeral0\xintdouble {#4}{#3}}%
896 }%
897 \def\XINT_xtrunc_BisEight\XINT_xtrunc_BisSmall #1#2#3#4%
898 {%
899 \expandafter\XINT_xtrunc_sp_e\expandafter
900 {\the\numexpr #2-\xint_c_iii\expandafter}\expandafter
901 {\romannumeral0\xintiimul {125}{#4}{#3}}%
902 }%
903 \def\XINT_xtrunc_BisSmall #1%
904 {%
905 \expandafter\XINT_xtrunc_e\expandafter
906 {\expandafter\XINT_xtrunc_small_a
907 \the\numexpr #1/\xint_c_ii\expandafter.\the\numexpr \xint_c_x^viii+#1!}%
908 }%
909 \def\XINT_xtrunc_small_a #1.#2!#3%
910 {%
911 \expandafter\XINT_div_small_b\the\numexpr #1\expandafter
912 .\the\numexpr #2\expandafter!%
913 \romannumeral0\XINT_div_small_ba #3\R\R\R\R\R\R\R\R{10}0000001\W
914 #3\XINT_sepbyviii_Z_end 2345678\relax
915 }%
916 \def\XINT_xtrunc_prepare_b
917 {\expandafter\XINT_xtrunc_prepare_c\romannumeral0\XINT_zeroes_forviii }%
918 \def\XINT_xtrunc_prepare_c #1!%
919 {%

```

7 Package *xintfrac* implementation

```

920 \XINT_xtrunc_prepare_d #1.00000000!{#1}%
921 }%
922 \def\XINT_xtrunc_prepare_d #1#2#3#4#5#6#7#8#9%
923 {%
924 \expandafter\XINT_xtrunc_prepare_e\xint_gob_til_dot #1#2#3#4#5#6#7#8#9!%
925 }%
926 \def\XINT_xtrunc_prepare_e #1!#2!#3#4%
927 {%
928 \XINT_xtrunc_prepare_f #4#3\X {#1}{#3}%
929 }%
930 \def\XINT_xtrunc_prepare_f #1#2#3#4#5#6#7#8#9\X
931 {%
932 \expandafter\XINT_xtrunc_prepare_g\expandafter
933 \XINT_div_prepare_g
934 \the\numexpr #1#2#3#4#5#6#7#8+\xint_c_i\expandafter
935 .\the\numexpr (#1#2#3#4#5#6#7#8+\xint_c_i)/\xint_c_ii\expandafter
936 .\the\numexpr #1#2#3#4#5#6#7#8\expandafter
937 .\romannumeral0\XINT_sepandrev_andcount
938 #1#2#3#4#5#6#7#8#9\XINT_rsepbyviii_end_A 2345678%
939 \XINT_rsepbyviii_end_B 2345678%
940 \relax\xint_c_ii\xint_c_iii
941 \R.\xint_c_vi\R.\xint_c_v\R.\xint_c_iv\R.\xint_c_iii
942 \R.\xint_c_ii\R.\xint_c_i\R.\xint_c_\W
943 \X
944 }%
945 \def\XINT_xtrunc_prepare_g #1;{\XINT_xtrunc_e {#1}}%
946 \def\XINT_xtrunc_e #1#2%
947 {%
948 \ifnum #2<\xint_c_
949 \expandafter\XINT_xtrunc_I
950 \else
951 \expandafter\XINT_xtrunc_II
952 \fi #2.{#1}%
953 }%
954 \def\XINT_xtrunc_I -#1.#2#3#4%
955 {%
956 \expandafter\XINT_xtrunc_I_a\romannumeral0#2{#4}{#2}{#1}{#3}%
957 }%
958 \def\XINT_xtrunc_I_a #1#2#3#4#5%
959 {%
960 \expandafter\XINT_xtrunc_I_b\the\numexpr #4-#5.#4.{#5}{#2}{#3}{#1}%
961 }%
962 \def\XINT_xtrunc_I_b #1%
963 {%
964 \xint_UDsignfork
965 #1\XINT_xtrunc_IA_c
966 -\XINT_xtrunc_IB_c
967 \krof #1%
968 }%
969 \def\XINT_xtrunc_IA_c -#1.#2.#3#4#5#6%
970 {%
971 \expandafter\XINT_xtrunc_IA_d

```

7 Package *xintfrac* implementation

```

972 \the\numexpr#2-\xintLength{#6}.{#6}%
973 \expandafter\XINT_xtrunc_IA_xd
974 \the\numexpr (#1+\xint_c_ii^v)/\xint_c_ii^vi-\xint_c_i.#1.{#5}{#4}%
975 }%
976 \def\XINT_xtrunc_IA_d #1%
977 {%
978 \xint_UDsignfork
979 #1\XINT_xtrunc_IAA_e
980 -\XINT_xtrunc_IAB_e
981 \krof #1%
982 }%
983 \def\XINT_xtrunc_IAA_e -#1.#2%
984 {%
985 \romannumeral0\XINT_split_fromleft
986 #1.#2\xint_gobble_i\xint_bye2345678\xint_bye..%
987 }%
988 \def\XINT_xtrunc_IAB_e #1.#2%
989 {%
990 0.\romannumeral\XINT_rep#1\endcsname0#2%
991 }%
992 \def\XINT_xtrunc_IA_xd #1.#2.%
993 {%
994 \expandafter\XINT_xtrunc_IA_xe\the\numexpr #2-\xint_c_ii^vi*#1.#1.%
995 }%
996 \def\XINT_xtrunc_IA_xe #1.#2.#3#4%
997 {%
998 \XINT_xtrunc_loop {#2}{#4}{#3}{#1}%
999 }%
1000 \def\XINT_xtrunc_IB_c #1.#2.#3#4#5#6%
1001 {%
1002 \expandafter\XINT_xtrunc_IB_d
1003 \romannumeral0\XINT_split_xfork #1.#6\xint_bye2345678\xint_bye..{#3}%
1004 }%
1005 \def\XINT_xtrunc_IB_d #1.#2.#3%
1006 {%
1007 \expandafter\XINT_xtrunc_IA_d\the\numexpr#3-\xintLength {#1}.{#1}%
1008 }%
1009 \def\XINT_xtrunc_II #1.%
1010 {%
1011 \expandafter\XINT_xtrunc_II_a\romannumeral\xintreplicate{#1}0.%
1012 }%
1013 \def\XINT_xtrunc_II_a #1.#2#3#4%
1014 {%
1015 \expandafter\XINT_xtrunc_II_b
1016 \the\numexpr (#3+\xint_c_ii^v)/\xint_c_ii^vi-\xint_c_i\expandafter.%
1017 \the\numexpr #3\expandafter.\romannumeral0#2{#4#1}{#2}%
1018 }%
1019 \def\XINT_xtrunc_II_b #1.#2.%
1020 {%
1021 \expandafter\XINT_xtrunc_II_c\the\numexpr #2-\xint_c_ii^vi*#1.#1.%
1022 }%
1023 \def\XINT_xtrunc_II_c #1.#2.#3#4#5%

```

[illegible]

```

1076 }%
1077 \def\XINT_xtrunc_sp_IA_b -#1.#2.#3#4%
1078 {%
1079   \expandafter\XINT_xtrunc_sp_IA_c
1080   \the\numexpr#2-\xintLength{#4}.{#4}\romannumeral\XINT_rep#1\endcsname0%
1081 }%
1082 \def\XINT_xtrunc_sp_IA_c #1%
1083 {%
1084   \xint_UDsignfork
1085   #1\XINT_xtrunc_sp_IAA
1086   -\XINT_xtrunc_sp_IAB
1087   \krof #1%
1088 }%
1089 \def\XINT_xtrunc_sp_IAA -#1.#2%
1090 {%
1091   \romannumeral0\XINT_split_fromleft
1092   #1.#2\xint_gobble_i\xint_bye2345678\xint_bye.>%
1093 }%
1094 \def\XINT_xtrunc_sp_IAB #1.#2%
1095 {%
1096   0.\romannumeral\XINT_rep#1\endcsname0#2%
1097 }%
1098 \def\XINT_xtrunc_sp_IB_b #1.#2.#3#4%
1099 {%
1100   \expandafter\XINT_xtrunc_sp_IB_c
1101   \romannumeral0\XINT_split_xfork #1.#4\xint_bye2345678\xint_bye..{#3}%
1102 }%
1103 \def\XINT_xtrunc_sp_IB_c #1.#2.#3%
1104 {%
1105   \expandafter\XINT_xtrunc_sp_IA_c\the\numexpr#3-\xintLength {#1}.{#1}%
1106 }%
1107 \def\XINT_xtrunc_sp_II #1.#2#3%
1108 {%
1109   #2\romannumeral\XINT_rep#1\endcsname0.\romannumeral\XINT_rep#3\endcsname0%
1110 }%

```

7.31 \xintDigits

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

```

1111 \mathchardef\XINTdigits 16
1112 \def\xintDigits #1#2%
1113   {\afterassignment \xint_gobble_i \mathchardef\XINTdigits=}%
1114 \def\xinttheDigits {\number\XINTdigits }%

```

7.32 \xintAdd

```

1115 \def\xintAdd {\romannumeral0\xintadd }%
1116 \def\xintadd #1{\expandafter\XINT_fadd\romannumeral0\xintra #1}%
1117 \def\XINT_fadd #1{\xint_gob_til_zero #1\XINT_fadd_Azero 0\XINT_fadd_a #1}%
1118 \def\XINT_fadd_Azero #1{\xintra #1}%
1119 \def\XINT_fadd_a #1/#2[#3]#4%

```

```

1120   {\expandafter\XINT_fadd_b\romannumeral0\xintra {#4}{#3}{#1}{#2}}%
1121 \def\XINT_fadd_b #1{\xint_gob_til_zero #1\XINT_fadd_Bzero 0\XINT_fadd_c #1}%
1122 \def\XINT_fadd_Bzero #1]#2#3#4{ #3/#4[#2]}%
1123 \def\XINT_fadd_c #1/#2[#3]#4%
1124 {%
1125   \expandafter\XINT_fadd_Aa\the\numexpr #4-#3.{#3}{#4}{#1}{#2}%
1126 }%
1127 \def\XINT_fadd_Aa #1%
1128 {%
1129   \xint_UDzerominusfork
1130     #1-\XINT_fadd_B
1131     0#1\XINT_fadd_Bb
1132     0-\XINT_fadd_Ba
1133   \krof #1%
1134 }%
1135 \def\XINT_fadd_B #1.#2#3#4#5#6#7{\XINT_fadd_C {#4}{#5}{#7}{#6}[#3]}%
1136 \def\XINT_fadd_Ba #1.#2#3#4#5#6#7%
1137 {%
1138   \expandafter\XINT_fadd_C\expandafter
1139     {\romannumeral0\XINT_dsx_addzeros {#1}#6;}%
1140   {#7}{#5}{#4}[#2]}%
1141 }%
1142 \def\XINT_fadd_Bb -#1.#2#3#4#5#6#7%
1143 {%
1144   \expandafter\XINT_fadd_C\expandafter
1145     {\romannumeral0\XINT_dsx_addzeros {#1}#4;}%
1146   {#5}{#7}{#6}[#3]}%
1147 }%
1148 \def\XINT_fadd_C #1#2#3%
1149 {%
1150   \ifcase\romannumeral0\xintiicmp {#2}{#3} %<- intentional space here.
1151     \expandafter\XINT_fadd_eq
1152   \or\expandafter\XINT_fadd_D
1153   \else\expandafter\XINT_fadd_Da
1154   \fi {#2}{#3}{#1}%
1155 }%
1156 \def\XINT_fadd_eq #1#2#3#4#5%
1157 {%
1158   \expandafter\XINT_fadd_G
1159   \romannumeral0\xintiiadd {#3}{#4}/#1[#5]%
1160 }%
1161 \def\XINT_fadd_D #1#2%
1162 {%
1163   \expandafter\XINT_fadd_E\romannumeral0\XINT_div_prepare {#2}{#1}{#1}{#2}%
1164 }%
1165 \def\XINT_fadd_E #1#2%
1166 {%
1167   \if0\XINT_Sgn #2\Z
1168     \expandafter\XINT_fadd_F
1169   \else\expandafter\XINT_fadd_K
1170   \fi {#1}%
1171 }%

```



```

1172 \def\XINT_fadd_F #1#2#3#4#5#6%
1173 {%
1174   \expandafter\XINT_fadd_G
1175   \romannumeral0\xintiiadd {\xintiiMul {#5}{#1}}{#4}/#2%[#6]%
1176 }%
1177 \def\XINT_fadd_Da #1#2%
1178 {%
1179   \expandafter\XINT_fadd_Ea\romannumeral0\XINT_div_prepare {#1}{#2}{#1}{#2}%
1180 }%
1181 \def\XINT_fadd_Ea #1#2%
1182 {%
1183   \if0\XINT_Sgn #2\Z
1184     \expandafter\XINT_fadd_Fa
1185   \else\expandafter\XINT_fadd_K
1186   \fi {#1}%
1187 }%
1188 \def\XINT_fadd_Fa #1#2#3#4#5#6%
1189 {%
1190   \expandafter\XINT_fadd_G
1191   \romannumeral0\xintiiadd {\xintiiMul {#4}{#1}}{#5}/#3%[#6]%
1192 }%
1193 \def\XINT_fadd_G #1{\if0#1\XINT_fadd_iszero\fi\space #1}%
1194 \def\XINT_fadd_K #1#2#3#4#5%
1195 {%
1196   \expandafter\XINT_fadd_L
1197   \romannumeral0\xintiiadd {\xintiiMul {#2}{#5}}{\xintiiMul {#3}{#4}}.%
1198   {{#2}{#3}}}%
1199 }%
1200 \def\XINT_fadd_L #1{\if0#1\XINT_fadd_iszero\fi \XINT_fadd_M #1}%
1201 \def\XINT_fadd_M #1.#2{\expandafter\XINT_fadd_N \expandafter
1202   {\romannumeral0\xintiimul #2}{#1}}%
1203 \def\XINT_fadd_N #1#2{ #2/#1}%
1204 \edef\XINT_fadd_iszero\fi #1[#2]{\noexpand\fi\space 0/1[0]}% ou [#2] originel?

```

7.33 *\xintSub*

```

1205 \def\xintSub {\romannumeral0\xintsub }%
1206 \def\xintsub #1{\expandafter\XINT_fsub\romannumeral0\xintraw {#1}}%
1207 \def\XINT_fsub #1{\xint_gob_til_zero #1\XINT_fsub_Azero 0\XINT_fsub_a #1}%
1208 \def\XINT_fsub_Azero #1]{\xintopp }%
1209 \def\XINT_fsub_a #1/#2[#3]#4%
1210   {\expandafter\XINT_fsub_b\romannumeral0\xintraw {#4}{#3}{#1}{#2}}%
1211 \def\XINT_fsub_b #1{\xint_UDzerominusfork
1212   #1-\XINT_fadd_Bzero
1213   0#1\XINT_fadd_c
1214   0-\XINT_fadd_c -#1}%
1215   \krof }%

```

7.34 *\xintSum*

```

1216 \def\xintSum {\romannumeral0\xintsum }%
1217 \def\xintsum #1{\xintsumexpr #1\relax }%
1218 \def\xintSumExpr {\romannumeral0\xintsumexpr }%
1219 \def\xintsumexpr {\expandafter\XINT_fsumexpr\romannumeral`&&@}%
1220 \def\XINT_fsumexpr {\XINT_fsum_loop_a {0/1[0]}}%

```

```

1221 \def\XINT_fsum_loop_a #1#2%
1222 {%
1223   \expandafter\XINT_fsum_loop_b \romannumeral`&&#2\Z {#1}%
1224 }%
1225 \def\XINT_fsum_loop_b #1%
1226 {%
1227   \xint_gob_til_relax #1\XINT_fsum_finished\relax
1228   \XINT_fsum_loop_c #1%
1229 }%
1230 \def\XINT_fsum_loop_c #1\Z #2%
1231 {%
1232   \expandafter\XINT_fsum_loop_a\expandafter{\romannumeral0\xintadd {#2}{#1}}%
1233 }%
1234 \def\XINT_fsum_finished #1\Z #2{ #2}%

```

7.35 *\xintMul*

```

1235 \def\xintMul {\romannumeral0\xintmul }%
1236 \def\xintmul #1{\expandafter\XINT_fmulo\romannumeral0\xintraw {#1}.}%
1237 \def\XINT_fmulo #1{\xint_gob_til_zero #1\XINT_fmulo_zero 0\XINT_fmulo_a #1}%
1238 \def\XINT_fmulo_a #1[#2].#3%
1239   {\expandafter\XINT_fmulo_b\romannumeral0\xintraw {#3}#1[#2.]}%
1240 \def\XINT_fmulo_b #1{\xint_gob_til_zero #1\XINT_fmulo_zero 0\XINT_fmulo_c #1}%
1241 \def\XINT_fmulo_c #1/#2[#3]#4/#5[#6.]%
1242 {%
1243   \expandafter\XINT_fmulo_d
1244   \expandafter{\the\numexpr #3+#6\expandafter}%
1245   \expandafter{\romannumeral0\xintiimul {#5}{#2}}%
1246   {\romannumeral0\xintiimul {#4}{#1}}%
1247 }%
1248 \def\XINT_fmulo_d #1#2#3%
1249 {%
1250   \expandafter \XINT_fmulo_e \expandafter{#3}{#1}{#2}%
1251 }%
1252 \def\XINT_fmulo_e #1#2{\XINT_outfrac {#2}{#1}}%
1253 \def\XINT_fmulo_zero #1.#2{ 0/1[0]}%

```

7.36 *\xintSqr*

1.1 modifs comme xintMul.

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

```

1254 \def\xintSqr {\romannumeral0\xintsqr }%
1255 \def\xintsqr #1{\expandafter\XINT_fsqr\romannumeral0\xintraw {#1}}%
1256 \def\XINT_fsqr #1{\xint_gob_til_zero #1\XINT_fsqr_zero 0\XINT_fsqr_a #1}%
1257 \def\XINT_fsqr_a #1/#2[#3]%
1258 {%
1259   \expandafter\XINT_fsqr_b
1260   \expandafter{\the\numexpr #3+#3\expandafter}%
1261   \expandafter{\romannumeral0\xintiisqr {#2}}%
1262   {\romannumeral0\xintiisqr {#1}}%
1263 }%

```

```

1264 \def\XINT_fsqr_b #1#2#3{\expandafter \XINT_fmulo_e \expandafter{#3}{#1}{#2}}%
1265 \def\XINT_fsqr_zero #1{\ 0/1[0]}%

```

7.37 \xintPow

1.2f: to be coherent with the "i" convention \xintiPow should parse also its exponent via \xintNum when xintfrac.sty is loaded. This was not the case so far. Cependant le problème est que le fait d'appliquer \xintNum rend impossible certains inputs qui auraient pu être gérés par \numexpr. Le \numexpr externe est ici pour intercepter trop grand input.

```

1266 \def\xintipow #1#2%
1267 {%
1268   \expandafter\xint_pow\the\numexpr \xintNum{#2}\expandafter.\romannumeral0\xintnum{#1}\Z%
1269 }%
1270 \def\xintPow {\romannumeral0\xintpow }%
1271 \def\xintpow #1%
1272 {%
1273   \expandafter\XINT_fpow\expandafter {\romannumeral0\XINT_infrac {#1}}%
1274 }%
1275 \def\XINT_fpow #1#2%
1276 {%
1277   \expandafter\XINT_fpow_fork\the\numexpr \xintNum{#2}\relax\Z #1%
1278 }%
1279 \def\XINT_fpow_fork #1#2\Z
1280 {%
1281   \xint_UDzerominusfork
1282   #1-\XINT_fpow_zero
1283   0#1\XINT_fpow_neg
1284   0-{\XINT_fpow_pos #1}%
1285   \krof
1286   {#2}%
1287 }%
1288 \def\XINT_fpow_zero #1#2#3#4{ 1/1[0]}%
1289 \def\XINT_fpow_pos #1#2#3#4#5%
1290 {%
1291   \expandafter\XINT_fpow_pos_A\expandafter
1292   {\the\numexpr #1#2*#3\expandafter}\expandafter
1293   {\romannumeral0\xintiipow {#5}{#1#2}}%
1294   {\romannumeral0\xintiipow {#4}{#1#2}}%
1295 }%
1296 \def\XINT_fpow_neg #1#2#3#4%
1297 {%
1298   \expandafter\XINT_fpow_pos_A\expandafter
1299   {\the\numexpr -#1*#2\expandafter}\expandafter
1300   {\romannumeral0\xintiipow {#3}{#1}}%
1301   {\romannumeral0\xintiipow {#4}{#1}}%
1302 }%
1303 \def\XINT_fpow_pos_A #1#2#3%
1304 {%
1305   \expandafter\XINT_fpow_pos_B\expandafter {#3}{#1}{#2}%
1306 }%
1307 \def\XINT_fpow_pos_B #1#2{\XINT_outfrac {#2}{#1}}%

```

7.38 `\xintiFac`

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

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

```
1308 \def\xintifac #1{\expandafter\XINT_fac_fork\the\numexpr \xintNum{#1}.}%
```

7.39 `\xintiBinomial`

1.2f. Binomial coefficients.

```
1309 \def\xintibinomial #1#2%
1310 {%
1311   \expandafter\XINT_binom_pre
1312   \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.%
1313 }%
```

7.40 `\xintiPFactorial`

1.2f. Partial factorial.

```
1314 \def\xintipfactorial #1#2%
1315 {%
1316   \expandafter\XINT_pfac_fork
1317   \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.%
1318 }%
```

7.41 `\xintPrd`

```
1319 \def\xintPrd {\romannumeral0\xintprd }%
1320 \def\xintprd #1{\xintprdexpr #1\relax }%
1321 \def\xintPrdExpr {\romannumeral0\xintprdexpr }%
1322 \def\xintprdexpr {\expandafter\XINT_fprdexpr \romannumeral`&&@}%
1323 \def\XINT_fprdexpr {\XINT_fprod_loop_a {1/1[0]}}%
1324 \def\XINT_fprod_loop_a #1#2%
1325 {%
1326   \expandafter\XINT_fprod_loop_b \romannumeral`&&@#2\Z {#1}%
1327 }%
1328 \def\XINT_fprod_loop_b #1%
1329 {%
1330   \xint_gob_til_relax #1\XINT_fprod_finished\relax
1331   \XINT_fprod_loop_c #1%
1332 }%
1333 \def\XINT_fprod_loop_c #1\Z #2%
1334 {%
1335   \expandafter\XINT_fprod_loop_a\expandafter{\romannumeral0\xintmul {#1}{#2}}%
1336 }%
1337 \def\XINT_fprod_finished #1\Z #2{ #2}%
```

7.42 `\xintDiv`

```

1338 \def\xintDiv {\romannumeral0\xintdiv }%
1339 \def\xintdiv #1%
1340 {%
1341   \expandafter\XINT_fdiv\expandafter {\romannumeral0\XINT_infrac {#1}}%
1342 }%
1343 \def\XINT_fdiv #1#2%
1344   {\expandafter\XINT_fdiv_A\romannumeral0\XINT_infrac {#2}#1}%
1345 \def\XINT_fdiv_A #1#2#3#4#5#6%
1346 {%
1347   \expandafter\XINT_fdiv_B
1348   \expandafter{\the\numexpr #4-#1\expandafter}%
1349   \expandafter{\romannumeral0\xintiimul {#2}{#6}}%
1350   {\romannumeral0\xintiimul {#3}{#5}}%
1351 }%
1352 \def\XINT_fdiv_B #1#2#3%
1353 {%
1354   \expandafter\XINT_fdiv_C
1355   \expandafter{#3}{#1}{#2}%
1356 }%
1357 \def\XINT_fdiv_C #1#2{\XINT_outfrac {#2}{#1}}%

```

7.43 `\xintDivFloor`

1.1

```

1358 \def\xintDivFloor {\romannumeral0\xintdivfloor }%
1359 \def\xintdivfloor #1#2{\xintfloor{\xintDiv {#1}{#2}}}%

```

7.44 `\xintDivTrunc`

1.1. `\xintttrunc` rather than `\xintitrunc0` in 1.1a

```

1360 \def\xintDivTrunc {\romannumeral0\xintdivtrunc }%
1361 \def\xintdivtrunc #1#2{\xintttrunc {\xintDiv {#1}{#2}}}%

```

7.45 `\xintDivRound`

1.1

```

1362 \def\xintDivRound {\romannumeral0\xintdivround }%
1363 \def\xintdivround #1#2{\xintiround 0{\xintDiv {#1}{#2}}}%

```

7.46 `\xintMod`

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

```

1364 \def\xintMod {\romannumeral0\xintmod }%
1365 \def\xintmod #1{\expandafter\XINT_mod_a\romannumeral0\xintra{#1}.}%
1366 \def\XINT_mod_a #1#2.#3%
1367   {\expandafter\XINT_mod_b\expandafter #1\romannumeral0\xintra{#3}#2.}%

```

```

1368 \def\XINT_mod_b #1#2% #1 de A, #2 de B.
1369 {%
1370   \if0#2\xint_dothis\XINT_mod_divbyzero\fi
1371   \if0#1\xint_dothis\XINT_mod_aiszero\fi
1372   \if-#2\xint_dothis{\XINT_mod_bneg #1}\fi
1373   \xint_orthat{\XINT_mod_bpos #1#2}%
1374 }%
1375 \def\XINT_mod_bpos #1%
1376 {%
1377   \xint_UDsignfork
1378   #1{\xintiiopp\XINT_mod_pos {}}%
1379   -{\XINT_mod_pos #1}%
1380   \krof
1381 }%
1382 \def\XINT_mod_bneg #1%
1383 {%
1384   \xint_UDsignfork
1385   #1{\xintiiopp\XINT_mod_pos {}}%
1386   -{\XINT_mod_pos #1}%
1387   \krof
1388 }%
1389 \def\XINT_mod_divbyzero #1.{\xintError:DivisionByZero\space 0/1[0]}%
1390 \def\XINT_mod_aiszero #1.{ 0/1[0]}%
1391 \def\XINT_mod_pos #1#2/#3[#4]#5/#6[#7].%
1392 {%
1393   \expandafter\XINT_mod_pos_a
1394   \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.\expandafter
1395   {\romannumeral0\xintiimul {#6}{#3}}%      n fois u
1396   {\xintiiE{\xintiiMul {#1#5}{#3}}{#7-#4}}% m fois u
1397   {\xintiiE{\xintiiMul {#2}{#6}}{#4-#7}}%    t fois n
1398 }%
1399 \def\XINT_mod_pos_a #1.#2#3#4{\xintiirem {#3}{#4}/#2[#1]}%

```

7.47 \xintIsOne

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

```

1400 \def\xintIsOne {\romannumeral0\xintisone }%
1401 \def\xintisone #1{\expandafter\XINT_fracisone
1402   \romannumeral0\xintrawwithzeros{#1}\Z }%
1403 \def\XINT_fracisone #1/#2\Z
1404   {\if0\XINT_Cmp {#1}{#2}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%

```

7.48 \xintGeq

```

1405 \def\xintGeq {\romannumeral0\xintgeq }%
1406 \def\xintgeq #1%
1407 {%
1408   \expandafter\XINT_fgeq\expandafter {\romannumeral0\xintabs {#1}}%
1409 }%
1410 \def\XINT_fgeq #1#2%
1411 {%

```

```

1412 \expandafter\XINT_fgeq_A \romannumeral0\xintabs {#2}#1%
1413 }%
1414 \def\XINT_fgeq_A #1%
1415 {%
1416 \xint_gob_til_zero #1\XINT_fgeq_Zii 0%
1417 \XINT_fgeq_B #1%
1418 }%
1419 \def\XINT_fgeq_Zii 0\XINT_fgeq_B #1[#2]#3[#4]{ 1}%
1420 \def\XINT_fgeq_B #1/#2[#3]#4#5/#6[#7]%
1421 {%
1422 \xint_gob_til_zero #4\XINT_fgeq_Zi 0%
1423 \expandafter\XINT_fgeq_C\expandafter
1424 {\the\numexpr #7-#3\expandafter}\expandafter
1425 {\romannumeral0\xintiimul {#4#5}{#2}}%
1426 {\romannumeral0\xintiimul {#6}{#1}}%
1427 }%
1428 \def\XINT_fgeq_Zi 0#1#2#3#4#5#6#7{ 0}%
1429 \def\XINT_fgeq_C #1#2#3%
1430 {%
1431 \expandafter\XINT_fgeq_D\expandafter
1432 {#3}{#1}{#2}%
1433 }%
1434 \def\XINT_fgeq_D #1#2#3%
1435 {%
1436 \expandafter\XINT_cntSgnFork\romannumeral`&&\expandafter\XINT_cntSgn
1437 \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\Z
1438 { 0}{\XINT_fgeq_E #2\Z {#3}{#1}}{ 1}%
1439 }%
1440 \def\XINT_fgeq_E #1%
1441 {%
1442 \xint_UDsignfork
1443 #1\XINT_fgeq_Fd
1444 -{\XINT_fgeq_Fn #1}%
1445 \krof
1446 }%
1447 \def\XINT_fgeq_Fd #1\Z #2#3%
1448 {%
1449 \expandafter\XINT_fgeq_Fe\expandafter
1450 {\romannumeral0\XINT_dsx_addzeros {#1}#3;}{#2}%
1451 }%
1452 \def\XINT_fgeq_Fe #1#2{\XINT_geq_pre {#2}{#1}}%
1453 \def\XINT_fgeq_Fn #1\Z #2#3%
1454 {%
1455 \expandafter\XINT_geq_pre\expandafter
1456 {\romannumeral0\XINT_dsx_addzeros {#1}#2;}{#3}%
1457 }%

```

7.49 \xintMax

```

1458 \def\xintMax {\romannumeral0\xintmax }%
1459 \def\xintmax #1%
1460 {%
1461 \expandafter\XINT_fmax\expandafter {\romannumeral0\xintraw {#1}}%
1462 }%

```

```

1463 \def\XINT_fmax #1#2%
1464 {%
1465   \expandafter\XINT_fmax_A\romannumeral0\xintra {#2}#1%
1466 }%
1467 \def\XINT_fmax_A #1#2/#3[#4]#5#6/#7[#8]%
1468 {%
1469   \xint_UDsignsfork
1470     #1#5\XINT_fmax_minusminus
1471     -#5\XINT_fmax_firstneg
1472     #1-\XINT_fmax_secondneg
1473     --\XINT_fmax_nonneg_a
1474   \krof
1475   #1#5{#2/#3[#4]}{#6/#7[#8]}%
1476 }%
1477 \def\XINT_fmax_minusminus --%
1478   {\expandafter-\romannumeral0\XINT_fmin_nonneg_b }%
1479 \def\XINT_fmax_firstneg #1-#2#3{ #1#2}%
1480 \def\XINT_fmax_secondneg -#1#2#3{ #1#3}%
1481 \def\XINT_fmax_nonneg_a #1#2#3#4%
1482 {%
1483   \XINT_fmax_nonneg_b {#1#3}{#2#4}%
1484 }%
1485 \def\XINT_fmax_nonneg_b #1#2%
1486 {%
1487   \if0\romannumeral0\XINT_fgeq_A #1#2%
1488     \xint_afterfi{ #1}%
1489   \else \xint_afterfi{ #2}%
1490   \fi
1491 }%

```

7.50 \xintMaxof

```

1492 \def\xintMaxof {\romannumeral0\xintmaxof }%
1493 \def\xintmaxof #1{\expandafter\XINT_maxof_a\romannumeral`&&@#1\relax }%
1494 \def\XINT_maxof_a #1{\expandafter\XINT_maxof_b\romannumeral0\xintra{#1}\Z }%
1495 \def\XINT_maxof_b #1\Z #2%
1496   {\expandafter\XINT_maxof_c\romannumeral`&&@#2\Z {#1}\Z}%
1497 \def\XINT_maxof_c #1%
1498   {\xint_gob_til_relax #1\XINT_maxof_e\relax\XINT_maxof_d #1}%
1499 \def\XINT_maxof_d #1\Z
1500   {\expandafter\XINT_maxof_b\romannumeral0\xintmax {#1}}%
1501 \def\XINT_maxof_e #1\Z #2\Z { #2}%

```

7.51 \xintMin

```

1502 \def\xintMin {\romannumeral0\xintmin }%
1503 \def\xintmin #1%
1504 {%
1505   \expandafter\XINT_fmin\expandafter {\romannumeral0\xintra {#1}}%
1506 }%
1507 \def\XINT_fmin #1#2%
1508 {%
1509   \expandafter\XINT_fmin_A\romannumeral0\xintra {#2}#1%
1510 }%
1511 \def\XINT_fmin_A #1#2/#3[#4]#5#6/#7[#8]%

```



```

1512 {%
1513   \xint_UDsignsfork
1514     #1#5\XINT_fmin_minusminus
1515     -#5\XINT_fmin_firstneg
1516     #1-\XINT_fmin_secondneg
1517     --\XINT_fmin_nonneg_a
1518   \krof
1519   #1#5{#2/#3[#4]}{#6/#7[#8]}%
1520 }%
1521 \def\XINT_fmin_minusminus --%
1522   {\expandafter\romannumeral0\XINT_fmax_nonneg_b }%
1523 \def\XINT_fmin_firstneg #1-#2#3{ -#3}%
1524 \def\XINT_fmin_secondneg -#1#2#3{ -#2}%
1525 \def\XINT_fmin_nonneg_a #1#2#3#4%
1526 {%
1527   \XINT_fmin_nonneg_b {#1#3}{#2#4}%
1528 }%
1529 \def\XINT_fmin_nonneg_b #1#2%
1530 {%
1531   \if0\romannumeral0\XINT_fgeq_A #1#2%
1532     \xint_afterfi{ #2}%
1533   \else \xint_afterfi{ #1}%
1534   \fi
1535 }%

```

7.52 \xintMinof

```

1536 \def\xintMinof   {\romannumeral0\xintminof }%
1537 \def\xintminof   #1{\expandafter\XINT_minof_a\romannumeral`&&@#1\relax }%
1538 \def\XINT_minof_a #1{\expandafter\XINT_minof_b\romannumeral0\xintra{#1}\Z }%
1539 \def\XINT_minof_b #1\Z #2%
1540   {\expandafter\XINT_minof_c\romannumeral`&&@#2\Z {#1}\Z}%
1541 \def\XINT_minof_c #1%
1542   {\xint_gob_til_relax #1\XINT_minof_e\relax\XINT_minof_d #1}%
1543 \def\XINT_minof_d #1\Z
1544   {\expandafter\XINT_minof_b\romannumeral0\xintmin {#1}}%
1545 \def\XINT_minof_e #1\Z #2\Z { #2}%

```

7.53 \xintCmp

```

1546 %\def\xintCmp {\romannumeral0\xintcmp }%
1547 \def\xintcmp #1%
1548 {%
1549   \expandafter\XINT_fcmp\expandafter {\romannumeral0\xintra{#1}}%
1550 }%
1551 \def\XINT_fcmp #1#2%
1552 {%
1553   \expandafter\XINT_fcmp_A\romannumeral0\xintra{#2}#1%
1554 }%
1555 \def\XINT_fcmp_A #1#2/#3[#4]#5#6/#7[#8]%
1556 {%
1557   \xint_UDsignsfork
1558     #1#5\XINT_fcmp_minusminus
1559     -#5\XINT_fcmp_firstneg
1560     #1-\XINT_fcmp_secondneg

```

7 Package *xintfrac* implementation

```

1561      --\XINT_fcmp_nonneg_a
1562      \krof
1563      #1#5{#2/#3[#4]}{#6/#7[#8]}%
1564 }%
1565 \def\XINT_fcmp_minusminus --#1#2{\XINT_fcmp_B #2#1}%
1566 \def\XINT_fcmp_firstneg #1-#2#3{ -1}%
1567 \def\XINT_fcmp_secondneg -#1#2#3{ 1}%
1568 \def\XINT_fcmp_nonneg_a #1#2%
1569 {%
1570      \xint_UDzerosfork
1571      #1#2\XINT_fcmp_zerozero
1572      0#2\XINT_fcmp_firstzero
1573      #10\XINT_fcmp_secondzero
1574      00\XINT_fcmp_pos
1575      \krof
1576      #1#2%
1577 }%
1578 \def\XINT_fcmp_zerozero #1#2#3#4{ 0}% 1.08b had some [ and ] here!!!
1579 \def\XINT_fcmp_firstzero #1#2#3#4{ -1}% incredibly I never saw that until
1580 \def\XINT_fcmp_secondzero #1#2#3#4{ 1}% preparing 1.09a.
1581 \def\XINT_fcmp_pos #1#2#3#4%
1582 {%
1583      \XINT_fcmp_B #1#3#2#4%
1584 }%
1585 \def\XINT_fcmp_B #1/#2[#3]#4/#5[#6]%
1586 {%
1587      \expandafter\XINT_fcmp_C\expandafter
1588      {\the\numexpr #6-#3\expandafter}\expandafter
1589      {\romannumeral0\xintiimul {#4}{#2}}%
1590      {\romannumeral0\xintiimul {#5}{#1}}%
1591 }%
1592 \def\XINT_fcmp_C #1#2#3%
1593 {%
1594      \expandafter\XINT_fcmp_D\expandafter
1595      {#3}{#1}{#2}%
1596 }%
1597 \def\XINT_fcmp_D #1#2#3%
1598 {%
1599      \expandafter\XINT_cntSgnFork\romannumeral`&&\expandafter\XINT_cntSgn
1600      \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\Z
1601      { -1}{\XINT_fcmp_E #2\Z {#3}{#1}}{ 1}%
1602 }%
1603 \def\XINT_fcmp_E #1%
1604 {%
1605      \xint_UDsignfork
1606      #1\XINT_fcmp_Fd
1607      -{\XINT_fcmp_Fn #1}%
1608      \krof
1609 }%
1610 \def\XINT_fcmp_Fd #1\Z #2#3%
1611 {%
1612      \expandafter\XINT_fcmp_Fe\expandafter

```

```

1613     {\romannumeral0\XINT_dsx_addzeros {#1}#3;}{#2}%
1614 }%
1615 \def\XINT_fcmp_Fe #1#2{\xintiicmp {#2}{#1}}%
1616 \def\XINT_fcmp_Fn #1\Z #2#3%
1617 {%
1618     \expandafter\xintiicmp\expandafter
1619     {\romannumeral0\XINT_dsx_addzeros {#1}#2;}{#3}%
1620 }%

```

7.54 `\xintAbs`

```

1621 \def\xintAbs    {\romannumeral0\xintabs }%
1622 \def\xintabs #1{\expandafter\XINT_abs\romannumeral0\xintraw {#1}}%

```

7.55 `\xintOpp`

```

1623 \def\xintOpp    {\romannumeral0\xintopp }%
1624 \def\xintopp #1{\expandafter\XINT_opp\romannumeral0\xintraw {#1}}%

```

7.56 `\xintSgn`

```

1625 \def\xintSgn    {\romannumeral0\xintsgn }%
1626 \def\xintsgn #1{\expandafter\XINT_sgn\romannumeral0\xintraw {#1}\Z }%

```

7.57 Floating point macros

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

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

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

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

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

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

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

7.58 `\xintFloat`

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

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

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

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

7 Package *xintfrac* implementation

`digits**`: to round $A/B \cdot 10^n$ to P digits of precision the routine first truncates A and B to $P+2$ digits and after that does the exact rounding of A/B . Naturally this means that it may then not necessarily compute the correct rounding of A/B .

Example : `\xintFloat {1/17597472569900621233}`
with `xintfrac 1.07`: $5.682634230727187e-20$
with `xintfrac 1.08b` or later: $5.682634230727188e-20$
and the exact value is $5.682634230727187499924124\dots e-20$.

1.07 computed the exact rounding for all inputs but as explained from 1.08 on, `\xintFloat` first truncates the denominator to $16+2=18$ digits, and here this increases $1/x$ enough to make the final rounding produce a result *lulp* higher.

In fact already dropping the last digit is enough to make the quotient cross the border:
 $1/17597472569900621233=5.682634230727187499924124\dots e-20$
 $1/1759747256990062123=56.82634230727187500892894\dots e-20$

1.2f did some minor improvements to the code, there was in particular a never-used branch. And it tries to handle more swiftly the case of inputs which are not fractions. (improved again in 1.2g)

This routine uses old macros `\XINT_addm_A` and `\XINT_lenrord_loop`. This could now be penalizing for P exceeding a few dozens, compared to doing it the 1.2 way.

I have always hesitated about the policy of printing $10.00\dots 0$ in case of rounding upwards towards next power of ten. It does make sense because it tells the (higher) precision of the rounding and moreover it is not too hard to test on output, although it is a bit cumbersome not to be certain to have exactly P digits.

Since 1.2f only `\xintFloat` but not `\XINTinFloat` may have $P+1$ digits in the mantissa on output. `\XINTinFloat` is the inner macro used for example by all operations in float expression to output their result, hence most of the time receive it in $A[N]$ shape also, which is quickly parsed --- apart of course that each time length of A is computed again.

1.2g adds `\XINTinFloatS` which may output $A[N]$ with A having $<P$ digits. This is for situations with large P 's (say in the hundreds) and has a dramatic speed-up effect on things like $2x$ or $x/3$. Addition and subtraction will still use `\XINTinFloat` on inputs; anyway this will have to be changed again when inner structure will carry up front at least the length of mantissa as a data not to recompute.

1.2g has also re-written both `\xintFloat` and `\XINTinFloat` start code to intercept more quickly and more satisfactorily $B=1$ case.

1.2i simplifies code via use of `\xintDSRr`. No more use of `\XINT_lenrord_loop` and of `\XINT_addm_A`, which have now been deleted from the `xint` sources.

```

1627 \def\xintFloat    {\romannumeral0\xintfloat}%
1628 \def\xintfloat #1{\XINT_float_chkopt #1\xint_relax}%
1629 \def\XINT_float_chkopt #1%
1630 {%
1631   \ifx [#1\expandafter\XINT_float_opt
1632     \else\expandafter\XINT_float_noopt
1633   \fi #1%
1634}%
1635 \def\XINT_float_noopt #1\xint_relax
1636 {%
1637   \expandafter\XINT_float_a\expandafter\XINTdigits\expandafter.%
1638   \romannumeral0\XINT_infrac {#1}\XINT_float_Q
1639}%
1640 \def\XINT_float_opt [\xint_relax #1]#2%
1641 {%
1642   \expandafter\XINT_float_a\the\numexpr #1\expandafter.%
1643   \romannumeral0\XINT_infrac {#2}\XINT_float_Q
1644}%

```

7 Package *xintfrac* implementation

Note 2015/12/02. Le but de ce code de 1.08 (2013), jusqu'à l'exécution de `\XINT_float_Q`, est simplement de tronquer numérateur et dénominateur à au plus P+2 chiffres en ajustant la partie décimale "n".

2016/03/19. First thing to do is to intercept denominator=1 case.

```
1645 \def\XINT_float_a #1.#2#3#4%
1646 {%
1647   \if1\XINT_iSone#4XY\expandafter\XINT_float_sp
1648   \else\expandafter\XINT_float_fork\fi #3.{#1}{#2}{#4}%
1649 }%
```

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

```
1650 \def\XINT_float_sp #1%
1651 {%
1652   \xint_UDzerominusfork
1653   #1-\XINT_float_spzero
1654   0#1\XINT_float_spneg
1655   0-\XINT_float_sppos #1}%
1656   \krof
1657 }%
1658 \def\XINT_float_spzero .#1#2#3#4{ 0.e0}%
1659 \def\XINT_float_spneg {\expandafter-\romannumeral0\XINT_float_sppos}%
1660 \def\XINT_float_sppos #1.#2#3#4#5%
1661 {%
1662   \expandafter\XINT_float_sp_b\the\numexpr#2-\xintLength{#1}.#1.#2.#3.%
1663 }%
1664 \def\XINT_float_sp_b #1%
1665 {%
1666   \xint_UDzerominusfork
1667   #1-\XINT_float_sp_quick
1668   0#1\XINT_float_sp_c
1669   0-\XINT_float_sp_addzeros #1}%
1670   \krof
1671 }%
1672 \def\XINT_float_sp_quick .#1.#2.#3.%
1673 {%
1674   \expandafter\XINT_float_sp_done\the\numexpr #3+#2-\xint_c_i.#1;%
1675 }%
1676 \def\XINT_float_sp_addzeros #1.#2.#3.#4.%
1677 {%
1678   \expandafter\XINT_float_sp_done
1679   \the\numexpr #4-#1+#3-\xint_c_i\expandafter.%
1680   \romannumeral0\XINT_dsx_addzeros {#1}#2;;%
1681 }%
1682 \def\XINT_float_sp_done #1.#2#3;{ #2.#3e#1}%
1683 \def\XINT_float_sp_c #1.#2%
1684 {%
1685   \if #29\xint_dothis {\XINT_float_sp_d\XINT_float_Wb }\fi
1686   \xint_orthat {\XINT_float_sp_d\XINT_float_Wa }#1.#2%
1687 }%
```

7 Package *xintfrac* implementation

```

1688 \def\XINT_float_sp_d #1#2.#3.#4.#5.%
1689 {%
1690   \expandafter\XINT_float_sp_e
1691   \the\numexpr #5+#2+#4-\xint_c_i\expandafter.%
1692   \romannumeral0\XINT_split_fromleft
1693   (\xint_c_i+#4).#3\xint_bye2345678\xint_bye..#1%
1694 }%

```

1.2i uses the \xintDSRr quick \numexpr loop for faster treatment.

#1=exposant final.#2=P+1chiffres de A.#3=junk.#4=\XINT_float_Wb ou \XINT_float_Wa

```

1695 \def\XINT_float_sp_e #1.#2.#3.#4%
1696 {%
1697   \expandafter#4\romannumeral0\XINT_dsrr#2%
1698   \xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax e#1%
1699 }%

```

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

```

1700 \def\XINT_float_fork #1%
1701 {%
1702   \xint_UDsignfork
1703   #1\XINT_float_J
1704   -{\XINT_float_K #1}%
1705   \krof
1706 }%
1707 \def\XINT_float_J {\expandafter-\romannumeral0\XINT_float_K }%
1708 \def\XINT_float_K #1.#2%
1709 {%
1710   \expandafter\XINT_float_L
1711   \the\numexpr\xintLength{#1}\expandafter.\the\numexpr #2+\xint_c_ii.{#1}{#2}%
1712 }%

```

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

```

1713 \def\XINT_float_L #1.#2.%
1714 {%
1715   \ifnum #1>#2
1716     \expandafter\XINT_float_Ma
1717   \else
1718     \expandafter\XINT_float_Mb
1719   \fi #1.#2.%
1720 }%

```

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

```

1721 \def\XINT_float_Ma #1.#2.#3%
1722 {%
1723   \expandafter\XINT_float_MatoN
1724   \the\numexpr #1-#2\expandafter.%
1725   \romannumeral0\XINT_split_fromleft#2.#3\xint_bye2345678\xint_bye..{#2}%
1726 }%

```

7 Package *xintfrac* implementation

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

```
1727 \def\XINT_float_MatoN #1.#2.#3.#4#5#6#7%
1728 {%
1729   \expandafter\XINT_float_N
1730   \the\numexpr\xintLength{#7}\expandafter.\the\numexpr #1+#6.#4.%
1731   {#7}{#4}{#2}{#5}%
1732 }%
```

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

```
1733 \def\XINT_float_Mb #1.#2.#3#4#5#6%
1734 {%
1735   \expandafter\XINT_float_N
1736   \romannumeral0\xintlength{#6}.#5.#2.{#6}{#1}{#3}{#4}%
1737 }%
```

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

```
1738 \def\XINT_float_N #1.#2.#3.%
1739 {%
1740   \ifnum #1>#3
1741     \expandafter\XINT_float_N_Blong
1742   \else\expandafter\XINT_float_P\fi
1743   #1.#2.#3.%
1744 }%
```

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

```
1745 \def\XINT_float_N_Blong #1.#2.#3.#4%
1746 {%
1747   \expandafter\XINT_float_NaP
1748   \the\numexpr #2-#1+#3\expandafter.%
1749   \romannumeral0\XINT_split_fromleft#3.#4\xint_bye2345678\xint_bye..{#3}%
1750 }%
```

$n' = n' - (|B| - (P+2)) \cdot \{B'\} \cdot \{\text{junk}\} \cdot \{P+2\} \rightarrow 0 \cdot P+2 \cdot n' \cdot P+2 \cdot \{B'\}$

```
1751 \def\XINT_float_NaP #1.#2.#3.#4{\XINT_float_P #4.#1.#4.{#2}}%
```

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

```
1752 \def\XINT_float_P #1.#2.#3.#4#5#6#7#8%
1753 {%
1754   \expandafter #8\the\numexpr #1-#5+#3-\xint_c_i.{#6}{#4}{#7}{#2}%
1755 }%
```

7 Package *xintfrac* implementation

On arrive ici avec $|B|-|A|+P+1$. $\{A\}\{B\}\{P\}\{n\}$, les A et B étant ceux d'origine tronqués à au plus P+2 chiffres, le n a été ajusté si besoin.

On calcule maintenant le quotient euclidien de $A \cdot 10^{|B|-|A|+P+1}$ (qui a P+1 chiffres de plus que B) par B. Ce quotient Q aura P+1 ou P+2 chiffres.

```
1756 \def\XINT_float_Q #1.#2#3%
1757 {%
1758   \expandafter\XINT_float_Sa
1759   \romannumeral0\xintiifquo{\XINT_dsx_addzeros {#1}#2;}{#3}\Z {#1}%
1760 }%
```

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

```
1761 \def\XINT_float_Sa #1%
1762 {%
1763   \if #19\xint_dothis {\XINT_float_Sb\XINT_float_Wb }\fi
1764   \xint_orthat {\XINT_float_Sb\XINT_float_Wa }#1%
1765 }%
```

En entrée $\XINT_float_W(a \text{ ou } b) \cdot Z^{|B|-|A|+P+1}\{P\}\{n\}$.

Refait pour 1.2i, plus de reverse, mais emploi de \xintDSRr . Mais on doit savoir si Q est de longueur $L=P+1$ ou $P+2$.

#1=Wa ou Wb, #2=Q, #3= $|B|-|A|+P+1$, #4=P, #5=n. On va calculer L-P.

```
1766 \def\XINT_float_Sb #1#2\Z #3#4%
1767 {%
1768   \expandafter\XINT_float_T
1769   \the\numexpr\xint_c_ix\XINT_length_loop
1770   #2\xint_relax\xint_relax\xint_relax\xint_relax
1771   \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
1772   \xint_c_i\xint_c_ii\xint_c_iii\xint_c_iv
1773   \xint_c_v\xint_c_vi\xint_c_vii\xint_c_viii\xint_c_ix\xint_bye-#4.%
1774   {#2}#1{#3}{#4}%
1775 }%
```

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

```
1776 \def\XINT_float_T #1.%
1777 {%
1778   \if2#1\expandafter\XINT_float_Ub\else\expandafter\XINT_float_Ua\fi
1779 }%
1780 \def\XINT_float_Ua #1#2%
1781 {%
1782   \expandafter\XINT_float_Xa
1783   \romannumeral0\expandafter#2%
1784   \romannumeral0\XINT_dsrr
1785   #1\xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax;%
1786 }%
1787 \def\XINT_float_Ub #1#2%
```



```

1788 {%
1789   \expandafter\XINT_float_Xb
1790   \romannumeral0\expandafter#2%
1791   \romannumeral0\expandafter\XINT_dsrr
1792   \romannumeral0\XINT_dsr
1793   #1\xint_bye\xint_Bye3456789\xint_bye+\xint_c_v)/\xint_c_x-\xint_c_i\relax
1794   \xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax;%
1795 }%

```

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

```

1796 \def\XINT_float_Wa #1{ #1.}%
1797 \def\XINT_float_Wb #1#2{\if #11\xint_dothis{ 10.}\fi\xint_orthat{ #1.#2}}%

```

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

```

1798 \def\XINT_float_Xa #1;#2#3#4%
1799 {%
1800   \expandafter\XINT_float_Y\the\numexpr #3+#4-#2.{#1}%
1801 }%
1802 \def\XINT_float_Xb #1;#2#3#4%
1803 {%
1804   \expandafter\XINT_float_Y\the\numexpr #3+#4+\xint_c_i-#2.{#1}%
1805 }%
1806 \def\XINT_float_Y #1.#2{ #2e#1}%

```

7.59 \XINTinFloat

1.07.

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

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

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

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

Each time `\XINTinFloat` is called it at least computes a length. Naturally if we had some format for floats that would be dispensed of...

Something like <letterP><length of mantissa>.mantissa.exponent, etc...

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

7 Package *xintfrac* implementation

2016/03/18. 1.2g adds a variant `\XINTinFloatS` which allows its output to be shorter than the current precision. Indeed `\XINTinFloat` is used everywhere but for example it is silly to convert 2 into 2<499zeroes>[-499] if we need to do some multiplication or division...

2016/03/19. 1.2g uses non-patched `\XINT_infrac` but then immediately filters out all denominator=1 cases (only the A[N] cases were identified by 1.2f).

2016/12/11. 1.2i simplifies the coding via use of `\xintDSRr`.

```

1807 \def\XINTinFloat {\romannumeral0\XINTinfloat}%
1808 \def\XINTinfloat
1809   {\expandafter\XINT_infloat_clean\romannumeral0\XINT_infloat}%
1810 \def\XINT_infloat_clean #1%
1811   {\if #1!\xint_dothis\XINT_infloat_clean_a\fi\xint_orthat{ }#1}%
1812 \def\XINT_infloat_clean_a !#1.#2[#3]%
1813 {%
1814   \expandafter\XINT_infloat_done
1815   \the\numexpr #3-#1\expandafter.%
1816   \romannumeral0\XINT_dsx_addzeros {#1}#2;;%
1817}%
1818 \def\XINT_infloat_done #1.#2;{ #2[#1]}%
1819 \def\XINTinFloatS {\romannumeral0\XINTinfloatS}%
1820 \def\XINTinfloatS
1821   {\expandafter\XINT_infloatS_clean\romannumeral0\XINT_infloat}%
1822 \def\XINT_infloatS_clean #1%
1823   {\if #1!\xint_dothis\XINT_infloatS_clean_a\fi\xint_orthat{ }#1}%
1824 \def\XINT_infloatS_clean_a !#1.{ }%
1825 \def\XINT_infloat [#1]#2%
1826 {%
1827   \expandafter\XINT_infloat_a\the\numexpr #1\expandafter.%
1828   \romannumeral0\XINT_infrac {#2}\XINT_infloat_Q
1829}%
1830 \def\XINT_infloat_a #1.#2#3#4%
1831 {%
1832   \if1\XINT_iSone#4XY\expandafter\XINT_infloat_sp
1833   \else\expandafter\XINT_float_fork\fi #3.{#1}{#2}{#4}%
1834}%

```

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

```

1835 \def\XINT_infloat_sp #1%
1836 {%
1837   \xint_UDzerominusfork
1838   #1-\XINT_infloat_spzero
1839   0#1\XINT_infloat_spneg
1840   0-\XINT_infloat_sppos #1}%
1841 \krof
1842}%
1843 \def\XINT_infloat_spzero .#1#2#3#4{ 0[0]}%
1844 \def\XINT_infloat_spneg
1845   {\expandafter\XINT_infloat_spnegend\romannumeral0\XINT_infloat_sppos}%
1846 \def\XINT_infloat_spnegend #1%
1847   {\if#1!\expandafter\XINT_infloat_spneg_needzeros\fi -#1}%
1848 \def\XINT_infloat_spneg_needzeros -!#1.{!#1.-}%
1849 \def\XINT_infloat_sppos #1.#2#3#4#5%

```

7 Package *xintfrac* implementation

```

1850 {%
1851   \expandafter\XINT_infloat_sp_b\the\numexpr#2-\xintLength{#1}.#1.#2.#3.%
1852 }%
1853 \def\XINT_infloat_sp_b #1%
1854 {%
1855   \xint_UDzerominusfork
1856   #1-\XINT_infloat_sp_quick
1857   0#1\XINT_infloat_sp_c
1858   0-\XINT_infloat_sp_needzeros #1}%
1859   \krof
1860 }%
1861 \def\XINT_infloat_sp_quick .#1.#2.#3.{ #1[#3]}%
1862 \def\XINT_infloat_sp_needzeros #1.#2.#3.#4.{!#1.#2[#4]}%

```

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

```

1863 \def\XINT_infloat_sp_c #1.#2%
1864 {%
1865   \if #29\xint_dothis {\XINT_infloat_sp_d\XINT_infloat_Wb }\fi
1866   \xint_orthat {\XINT_infloat_sp_d .}#1.#2%
1867 }%
1868 \def\XINT_infloat_sp_d #1#2.#3.#4.##5.%
1869 {%
1870   \expandafter\XINT_infloat_sp_e
1871   \romannumeral0\XINT_split_fromleft
1872   (\xint_c_i+#4).#3\xint_bye2345678\xint_bye..#1#2.%
1873 }%

```

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

```

1874 \def\XINT_infloat_sp_e #1.#2.#3#4.##5.%
1875 {%
1876   \expandafter\XINT_infloat_done
1877   \the\numexpr#4+#5\expandafter#3\romannumeral0\XINT_dsrr
1878   #1\xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax;%
1879 }%

```

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

```

1880 \def\XINT_infloat_Q #1.#2#3%
1881 {%
1882   \expandafter\XINT_infloat_Sa
1883   \romannumeral0\xintiiquo{\XINT_dsx_addzeros {#1}#2;}{#3}\Z {#1}%
1884 }%
1885 \def\XINT_infloat_Sa #1%
1886 {%
1887   \if #19\xint_dothis {\XINT_infloat_Sb\XINT_infloat_Wb }\fi
1888   \xint_orthat {\XINT_infloat_Sb .}#1%
1889 }%
1890 % \lverb?Refait pour 1.2i: calcul de longueur, puis \xintDSRr.

```

```

1891 %
1892 % En entrée \XINT_infloat_Wb ou «.»  $Q \setminus Z \{|B| - |A| + P + 1\} \{P\} \{n\}$ . On
1893 % commence par évaluer longueur de  $Q - P$  pour savoir si 1 ou 2. Plus besoin de
1894 %  $P$  après. L'exposant final sera  $1 + n - (|B| - |A| + P + 1)$  ou  $2 + n - (|B| - |A| + P + 1)$ .
1895 % modulo ajustement avec encore un +1 éventuel de  $Wb$ .
1896 % ?
1897 \def\XINT_infloat_Sb #1#2\Z #3#4%
1898 {%
1899   \expandafter\XINT_infloat_T
1900   \the\numexpr\xint_c_ix\XINT_length_loop
1901   #2\xint_relax\xint_relax\xint_relax\xint_relax
1902   \xint_relax\xint_relax\xint_relax\xint_relax\xint_relax
1903   \xint_c_i\xint_c_ii\xint_c_iii\xint_c_iv
1904   \xint_c_v\xint_c_vi\xint_c_vii\xint_c_viii\xint_c_ix\xint_bye-#4.%
1905   {#2}#1{#3}%
1906}%

```

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

```

1907 \def\XINT_infloat_T #1.%
1908 {%
1909   \if2#1\expandafter\XINT_infloat_Ub\else\expandafter\XINT_infloat_Ua\fi
1910}%
1911 \def\XINT_infloat_Ua #1#2#3#4%
1912 {%
1913   \expandafter\XINT_infloat_done
1914   \the\numexpr\xint_c_i+#4-#3\romannumeral0\expandafter#2%
1915   \romannumeral0\XINT_dsrr
1916   #1\xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax;%
1917}%
1918 \def\XINT_infloat_Ub #1#2#3#4%
1919 {%
1920   \expandafter\XINT_infloat_done
1921   \the\numexpr\xint_c_ii+#4-#3\romannumeral0\expandafter#2%
1922   \romannumeral0\expandafter\XINT_dsrr
1923   \romannumeral0\XINT_dsr
1924   #1\xint_bye\xint_Bye3456789\xint_bye+\xint_c_v)/\xint_c_x-\xint_c_i\relax
1925   \xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax;%
1926}%
1927 \def\XINT_infloat_Wb #1#2%
1928   {\if #11\xint_dothis{+\xint_c_i.1}\fi\xint_orthat{.#1#2}}%

```

7.60 \xintPFloat

1.1. This is a prettifying printing macro for floats.

2016/03/07.

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

7 Package *xintfrac* implementation

The new rule is simply that $x.yz...eN$ will drop scientific notation in favor of pure decimal notation if $-5 \leq N \leq 5$. This is the default behaviour of Maple. The N here is as produced on output by `\xintFloat`. There is the exceptional cases where $x=10$, and $yz..=0000....$ from rounding up to the next power of ten.

```

1929 \def\xintPFloat    {\romannumeral0\xintpfloat }%
1930 \def\xintpfloat #1{\XINT_pfloat_chkopt #1\xint_relax }%
1931 \def\XINT_pfloat_chkopt #1%
1932 {%
1933     \ifx [#1\expandafter\XINT_pfloat_opt
1934         \else\expandafter\XINT_pfloat_noopt
1935     \fi #1%
1936 }%
1937 \def\XINT_pfloat_noopt #1\xint_relax
1938 {%
1939     \expandafter\XINT_pfloat_a
1940     \romannumeral0\xintfloat [\XINTdigits]{#1};\XINTdigits.%
1941 }%
1942 \def\XINT_pfloat_opt [\xint_relax #1]#2%
1943 {%
1944     \expandafter\XINT_pfloat_opt_a \the\numexpr #1.%
1945 }%
1946 \def\XINT_pfloat_opt_a #1.#2%
1947 {%
1948     \expandafter\XINT_pfloat_a\romannumeral0\xintfloat [#1]{#2};#1.%
1949 }%
1950 \def\XINT_pfloat_a #1%
1951 {%
1952     \xint_UDzerominusfork
1953     #1-\XINT_pfloat_zero
1954     0#1\XINT_pfloat_neg
1955     0-{\XINT_pfloat_pos #1}%
1956     \krof
1957 }%
1958 \def\XINT_pfloat_zero #1;#2.{ 0.}%
1959 \def\XINT_pfloat_neg {\expandafter-\romannumeral0\XINT_pfloat_pos }%
1960 \def\XINT_pfloat_pos #1e#2;#3.%
1961 {%
1962     \ifnum #2>\xint_c_v \xint_dothis\XINT_pfloat_no\fi
1963     \ifnum #2<-\xint_c_v \xint_dothis\XINT_pfloat_no\fi
1964     \ifnum #2<\xint_c_ \xint_dothis\XINT_pfloat_N\fi
1965     \ifnum #2>\numexpr #3-\xint_c_i\relax \xint_dothis\XINT_pfloat_Ps\fi
1966     \xint_orthat\XINT_pfloat_P #1e#2;%
1967 }%
1968 \def\XINT_pfloat_no #1;{ #1}%
1969 \def\XINT_pfloat_N #1#2.#3e#4;%
1970 {%
1971     \csname XINT_pfloat_N\romannumeral-#4\endcsname #2#10.#3;%
1972 }%
1973 \def\XINT_pfloat_Ni #1#2#3.#4;{ #2.#1#4}%
1974 \def\XINT_pfloat_Nii #1#2#3.#4;{ 0.#2#1#4}%
1975 \def\XINT_pfloat_Niii#1#2#3.#4;{ 0.0#2#1#4}%
1976 \def\XINT_pfloat_Niv #1#2#3.#4;{ 0.00#2#1#4}%

```

```

1977 \def\XINT_pfloat_Nv #1#2#3.#4;{ 0.000#2#1#4}%
1978 \def\XINT_pfloat_P #1#2.#3e#4;%
1979 {%
1980   \csname XINT_pfloat_P_\romannumeral#4\endcsname #3.#1#2;%
1981 }%
1982 \def\XINT_pfloat_P_ #1.#2;{ #2.#1}%
1983 \def\XINT_pfloat_P_i #1#2.#3;{ #3#1.#2}%
1984 \def\XINT_pfloat_P_ii #1#2#3.#4;{ #4#1#2.#3}%
1985 \def\XINT_pfloat_P_iii#1#2#3#4.#5;{ #5#1#2#3.#4}%
1986 \def\XINT_pfloat_P_iv #1#2#3#4#5.#6;{ #6#1#2#3#4.#5}%
1987 \def\XINT_pfloat_P_v #1#2#3#4#5#6.#7;{ #7#1#2#3#4#5.#6}%
1988 \def\XINT_pfloat_Ps #1#2.#3e#4;%
1989 {%
1990   \csname XINT_pfloat_Ps_\romannumeral#4\endcsname #300000.#1#2;%
1991 }%
1992 \def\XINT_pfloat_Psi #1#2.#3;{ #3#1.%}
1993 \def\XINT_pfloat_Psii #1#2#3.#4;{ #4#1#2.%}
1994 \def\XINT_pfloat_Psiii#1#2#3#4.#5;{ #5#1#2#3.%}
1995 \def\XINT_pfloat_Psiv #1#2#3#4#5.#6;{ #6#1#2#3#4.%}
1996 \def\XINT_pfloat_Psv #1#2#3#4#5#6.#7;{ #7#1#2#3#4#5.%}

```

7.61 \XINTinFloatFracdigits

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

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

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

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

```

1997 \def\XINTinFloatFracdigits {\romannumeral0\xintinfloatfracdigits}%
1998 \def\xintinfloatfracdigits #1%
1999 {%
2000   \expandafter\XINT_infloatfracdg_a\expandafter {\romannumeral0\xintfrac{#1}}%
2001 }%
2002 \def\XINT_infloatfracdg_a {\XINTinfloat [\XINTdigits]}%

```

7.62 \xintFloatAdd, \XINTinFloatAdd

First included in release 1.07.

1.09ka improved a bit the efficiency. However the `add`, `sub`, `mul`, `div` routines were provisory and supposed to be revised soon.

Which didn't happen until 1.2f. Now, the inputs are first rounded to `P` digits, not `P+2` as earlier.

```

2003 \def\xintFloatAdd {\romannumeral0\xintfloatadd}%
2004 \def\xintfloatadd #1{\XINT_fladd_chkopt \xintfloat #1\xint_relax}%
2005 \def\XINTinFloatAdd {\romannumeral0\XINTinfloatadd}%
2006 \def\XINTinfloatadd #1{\XINT_fladd_chkopt \XINTinfloatS #1\xint_relax}%
2007 \def\XINT_fladd_chkopt #1#2%
2008 {%

```

```

2009 \ifx [#2\expandafter\XINT_fladd_opt
2010 \else\expandafter\XINT_fladd_noopt
2011 \fi #1#2%
2012 }%
2013 \def\XINT_fladd_noopt #1#2\xint_relax #3%
2014 {%
2015 #1[\XINTdigits]%
2016 {\expandafter\XINT_FL_add_a
2017 \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{#3}}%
2018 }%
2019 \def\XINT_fladd_opt #1[\xint_relax #2]%#3#4%
2020 {%
2021 \expandafter\XINT_fladd_opt_a\the\numexpr #2.#1%
2022 }%
2023 \def\XINT_fladd_opt_a #1.#2#3#4%
2024 {%
2025 #2[#1]{\expandafter\XINT_FL_add_a\romannumeral0\XINTinfloat[#1]{#3}#1.{#4}}%
2026 }%
2027 \def\XINT_FL_add_a #1%
2028 {%
2029 \xint_gob_til_zero #1\XINT_FL_add_zero 0\XINT_FL_add_b #1%
2030 }%
2031 \def\XINT_FL_add_zero #1.#2{#2}%
2032 \def\XINT_FL_add_b #1]#2.#3%
2033 {%
2034 \expandafter\XINT_FL_add_c\romannumeral0\XINTinfloat[#2]{#3}#2.#1%
2035 }%
2036 \def\XINT_FL_add_c #1%
2037 {%
2038 \xint_gob_til_zero #1\XINT_FL_add_zero 0\XINT_FL_add_d #1%
2039 }%
2040 \def\XINT_FL_add_d #1[#2]#3.#4[#5]%
2041 {%
2042 \ifnum\numexpr #2-#3-#5>\xint_c\xint_dothis\xint_firstoftwo\fi
2043 \ifnum\numexpr #5-#3-#2>\xint_c\xint_dothis\xint_secondoftwo\fi
2044 \xint_orthat\xintAdd {#1[#2]}{#4[#5]}%
2045 }%

```

7.63 \xintFloatSub, \XINTinFloatSub

First done 1.07.

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

```

2046 \def\xintFloatSub {\romannumeral0\xintfloatsub }%
2047 \def\xintfloatsub #1{\XINT_fsub_chkopt \xintfloat #1\xint_relax }%
2048 \def\XINTinFloatSub {\romannumeral0\XINTinfloatsub }%
2049 \def\XINTinfloatsub #1{\XINT_fsub_chkopt \XINTinfloatS #1\xint_relax }%
2050 \def\XINT_fsub_chkopt #1#2%
2051 {%
2052 \ifx [#2\expandafter\XINT_fsub_opt
2053 \else\expandafter\XINT_fsub_noopt
2054 \fi #1#2%
2055 }%

```

```

2056 \def\XINT_flsb_noopt #1#2\xint_relax #3%
2057 {%
2058   #1[\XINTdigits]%
2059   {\expandafter\XINT_FL_add_a
2060     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{\xintOpp{#3}}}%
2061 }%
2062 \def\XINT_flsb_opt #1[\xint_relax #2]%#3#4%
2063 {%
2064   \expandafter\XINT_flsb_opt_a\the\numexpr #2.#1%
2065 }%
2066 \def\XINT_flsb_opt_a #1.#2#3#4%
2067 {%
2068   #2[#1]{\expandafter\XINT_FL_add_a\romannumeral0\XINTinfloat[#1]{#3}#1.{\xintOpp{#4}}}%
2069 }%

```

7.64 \xintFloatMul, \XINTinFloatMul

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

1.07.

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

1.2g handles the inputs via \XINTinFloatS which will be more efficient when the precision is large and the input is for example a small constant like 2.

```

2070 \def\xintFloatMul {\romannumeral0\xintfloatmul }%
2071 \def\xintfloatmul #1{\XINT_flmul_chkopt \xintfloat #1\xint_relax }%
2072 \def\XINTinFloatMul {\romannumeral0\XINTinfloatmul }%
2073 \def\XINTinfloatmul #1{\XINT_flmul_chkopt \XINTinfloatS #1\xint_relax }%
2074 \def\XINT_flmul_chkopt #1#2%
2075 {%
2076   \ifx [#2\expandafter\XINT_flmul_opt
2077     \else\expandafter\XINT_flmul_noopt
2078   \fi #1#2%
2079 }%
2080 \def\XINT_flmul_noopt #1#2\xint_relax #3%
2081 {%
2082   #1[\XINTdigits]%
2083   {\expandafter\XINT_FL_mul_a
2084     \romannumeral0\XINTinfloatS[\XINTdigits]{#2}\XINTdigits.{#3}}%
2085 }%
2086 \def\XINT_flmul_opt #1[\xint_relax #2]%#3#4%
2087 {%
2088   \expandafter\XINT_flmul_opt_a\the\numexpr #2.#1%
2089 }%
2090 \def\XINT_flmul_opt_a #1.#2#3#4%
2091 {%
2092   #2[#1]{\expandafter\XINT_FL_mul_a\romannumeral0\XINTinfloatS[#1]{#3}#1.{#4}}%
2093 }%
2094 \def\XINT_FL_mul_a #1[#2]#3.#4%
2095 {%

```



```

2096 \expandafter\XINT_FL_mul_b\romannumeral0\XINTinfloatS[#3]{#4}#1[#2]%
2097 }%
2098 \def\XINT_FL_mul_b #1[#2]#3[#4]{\xintE{\xintiiMul {#3}{#1}}{#4+#2}}%

```

7.65 \xintFloatDiv, \XINTinFloatDiv

1.07.

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

1.2g handles the inputs via \XINTinFloatS which will be more efficient when the precision is large and the input is for example a small constant like 2.

```

2099 \def\xintFloatDiv {\romannumeral0\xintfloatdiv }%
2100 \def\xintfloatdiv #1{\XINT_fldiv_chkopt \xintfloat #1\xint_relax }%
2101 \def\XINTinFloatDiv {\romannumeral0\XINTinfloatdiv }%
2102 \def\XINTinfloatdiv #1{\XINT_fldiv_chkopt \XINTinfloatS #1\xint_relax }%
2103 \def\XINT_fldiv_chkopt #1#2%
2104 {%
2105 \ifx [#2\expandafter\XINT_fldiv_opt
2106 \else\expandafter\XINT_fldiv_noopt
2107 \fi #1#2%
2108 }%
2109 \def\XINT_fldiv_noopt #1#2\xint_relax #3%
2110 {%
2111 #1[\XINTdigits]%
2112 {\expandafter\XINT_FL_div_a
2113 \romannumeral0\XINTinfloatS[\XINTdigits]{#2}\XINTdigits.{#3}}%
2114 }%
2115 \def\XINT_fldiv_opt #1[\xint_relax #2]%#3#4%
2116 {%
2117 \expandafter\XINT_fldiv_opt_a\the\numexpr #2.#1%
2118 }%
2119 \def\XINT_fldiv_opt_a #1.#2#3#4%
2120 {%
2121 #2[#1]{\expandafter\XINT_FL_div_a\romannumeral0\XINTinfloatS[#1]{#3}#1.#4}%
2122 }%
2123 \def\XINT_FL_div_a #1[#2]#3.#4%
2124 {%
2125 \expandafter\XINT_FL_div_b\romannumeral0\XINTinfloatS[#3]{#4}#1[#2]%
2126 }%
2127 \def\XINT_FL_div_b #1[#2]#3[#4]{\xintE{#3/#1}{#4-#2}}%

```

7.66 \xintFloatPow, \XINTinFloatPow

1.07: initial version. 1.09j has re-organized the core loop.

2015/12/07. I have hesitated to maintain the mapping of ^ in expressions to \xintFloatPow rather than \xintFloatPower. But for 1.234567890123456 to the power 2145678912 with P=16, using Pow rather than Power seems to bring only about 5% gain.

This routine requires the exponent x to be compatible with \numexpr parsing.

1.2f has rewritten the code for better efficiency. Also, now the argument A for A^x is first rounded to P digits before switching to the increased working precision (which depends upon x).

```

2128 \def\xintFloatPow {\romannumeral0\xintfloatpow}%
2129 \def\xintfloatpow #1{\XINT_flpow_chkopt \xintfloat #1\xint_relax }%

```

```

2130 \def\XINTinFloatPow {\romannumeral0\XINTinfloatpow }%
2131 \def\XINTinfloatpow #1{\XINT_flpow_chkopt \XINTinfloatS #1\xint_relax }%
2132 \def\XINT_flpow_chkopt #1#2%
2133 {%
2134   \ifx [#2\expandafter\XINT_flpow_opt
2135     \else\expandafter\XINT_flpow_noopt
2136   \fi
2137   #1#2%
2138 }%
2139 \def\XINT_flpow_noopt #1#2\xint_relax #3%
2140 {%
2141   \expandafter\XINT_flpow_checkB_a
2142   \the\numexpr #3.\XINTdigits.{#2}{#1[\XINTdigits]]}%
2143 }%
2144 \def\XINT_flpow_opt #1[\xint_relax #2]%
2145 {%
2146   \expandafter\XINT_flpow_opt_a\the\numexpr #2.#1%
2147 }%
2148 \def\XINT_flpow_opt_a #1.#2#3#4%
2149 {%
2150   \expandafter\XINT_flpow_checkB_a\the\numexpr #4.#1.{#3}{#2[#1]}%
2151 }%
2152 \def\XINT_flpow_checkB_a #1%
2153 {%
2154   \xint_UDzerominusfork
2155   #1-\XINT_flpow_BisZero
2156   0#1{\XINT_flpow_checkB_b -}%
2157   0-{\XINT_flpow_checkB_b }{#1}%
2158   \krof
2159 }%
2160 \def\XINT_flpow_BisZero .#1.#2#3{#3{1[0]}}%
2161 \def\XINT_flpow_checkB_b #1#2.#3.%
2162 {%
2163   \expandafter\XINT_flpow_checkB_c
2164   \the\numexpr\xintLength{#2}+\xint_c_iii.#3.#2.{#1}%
2165 }%
2166 \def\XINT_flpow_checkB_c #1.#2.%
2167 {%
2168   \expandafter\XINT_flpow_checkB_d\the\numexpr#1+#2.#1.#2.%
2169 }%

```

1.2f rounds input to P digits, first.

```

2170 \def\XINT_flpow_checkB_d #1.#2.#3.#4.#5#6%
2171 {%
2172   \expandafter \XINT_flpow_aa
2173   \romannumeral0\XINTinfloat [#3]{#6}{#2}{#1}{#4}{#5}%
2174 }%
2175 \def\XINT_flpow_aa #1[#2]#3%
2176 {%
2177   \expandafter\XINT_flpow_ab\the\numexpr #2-#3\expandafter.%
2178   \romannumeral\XINT_rep #3\endcsname0.#1.%
2179 }%

```

7 Package *xintfrac* implementation

```

2180 \def\XINT_flpow_ab #1.#2.#3.{\XINT_flpow_a #3#2[#1]}%
2181 \def\XINT_flpow_a #1%
2182 {%
2183   \xint_UDzerominusfork
2184   #1-\XINT_flpow_zero
2185   0#1{\XINT_flpow_b \iftrue}%
2186   0-{\XINT_flpow_b \iffalse#1}%
2187   \krof
2188 }%
2189 \def\XINT_flpow_zero #1[#2]#3#4#5#6%
2190 {%
2191   \if 1#51\xint_afterfi {#6{0[0]}}\else
2192   \xint_afterfi {\xintError:DivisionByZero #6{1[2147483648]}}\fi
2193 }%
2194 \def\XINT_flpow_b #1#2[#3]#4#5%
2195 {%
2196   \XINT_flpow_loopI #5.#3.#2.#4.{#1\ifodd #5 \xint_c_i\fi\fi}%
2197 }%
2198 \def\XINT_flpow_truncate #1.#2.#3.%
2199 {%
2200   \expandafter\XINT_flpow_truncate_a
2201   \romannumeral0\XINT_split_fromleft
2202   #3.#2\xint_bye2345678\xint_bye..#1.#3.%
2203 }%
2204 \def\XINT_flpow_truncate_a #1.#2.#3.{#3+\xintLength{#2}.#1.}%
2205 \def\XINT_flpow_loopI #1.%
2206 {%
2207   \ifnum #1=\xint_c_i\expandafter\XINT_flpow_ItoIII\fi
2208   \ifodd #1
2209     \expandafter\XINT_flpow_loopI_odd
2210   \else
2211     \expandafter\XINT_flpow_loopI_even
2212   \fi
2213   #1.%
2214 }%
2215 \def\XINT_flpow_ItoIII\ifodd #1\fi #2.#3.#4.#5.#6%
2216 {%
2217   \expandafter\XINT_flpow_III\the\numexpr #6+\xint_c_.#3.#4.#5.%
2218 }%
2219 \def\XINT_flpow_loopI_even #1.#2.#3.%#4.%
2220 {%
2221   \expandafter\XINT_flpow_loopI
2222   \the\numexpr #1/\xint_c_ii\expandafter.%
2223   \the\numexpr\expandafter\XINT_flpow_truncate
2224   \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.%
2225 }%
2226 \def\XINT_flpow_loopI_odd #1.#2.#3.#4.%
2227 {%
2228   \expandafter\XINT_flpow_loopII
2229   \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter.%
2230   \the\numexpr\expandafter\XINT_flpow_truncate
2231   \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#2.#3.%

```

7 Package *xintfrac* implementation

```

2232 }%
2233 \def\XINT_flpow_loopII #1.%
2234 {%
2235     \ifnum #1 = \xint_c_i\expandafter\XINT_flpow_IItoIII\fi
2236     \ifodd #1
2237         \expandafter\XINT_flpow_loopII_odd
2238     \else
2239         \expandafter\XINT_flpow_loopII_even
2240     \fi
2241     #1.%
2242 }%
2243 \def\XINT_flpow_loopII_even #1.#2.#3.##4.%
2244 {%
2245     \expandafter\XINT_flpow_loopII
2246     \the\numexpr #1/\xint_c_ii\expandafter.%
2247     \the\numexpr\expandafter\XINT_flpow_truncate
2248     \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.%
2249 }%
2250 \def\XINT_flpow_loopII_odd #1.#2.#3.#4.#5.#6.%
2251 {%
2252     \expandafter\XINT_flpow_loopII_odda
2253     \the\numexpr\expandafter\XINT_flpow_truncate
2254     \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2255     #1.#2.#3.%
2256 }%
2257 \def\XINT_flpow_loopII_odda #1.#2.#3.#4.#5.#6.%
2258 {%
2259     \expandafter\XINT_flpow_loopII
2260     \the\numexpr #4/\xint_c_ii-\xint_c_i\expandafter.%
2261     \the\numexpr\expandafter\XINT_flpow_truncate
2262     \the\numexpr\xint_c_ii*#5\expandafter.\romannumeral0\xintiisqr{#6}.#3.%
2263     #1.#2.%
2264 }%
2265 \def\XINT_flpow_IItoIII\ifodd #1\fi #2.#3.#4.#5.#6.#7.#8%
2266 {%
2267     \expandafter\XINT_flpow_III\the\numexpr #8+\xint_c_\expandafter.%
2268     \the\numexpr\expandafter\XINT_flpow_truncate
2269     \the\numexpr#3+#6\expandafter.\romannumeral0\xintiimul{#4}{#7}.#5.%
2270 }%

```

This ending is common with `\xintFloatPower`. In the case of negative exponent we will inverse the Q-digits mantissa, keeping Q significant digits (exceptionally 10^Q) before the final rounding to P digits. Here Q is working precision. Releases prior to 1.2f trusted the final inverse to `\xintFloat` on output but this worked only with P+2 digits on denominator. Enough for 0.1 ulp extra error, but as our goal is to get <0.6ulp, and there is already 0.5ulp from rounding error, this was not enough. When `\xintFloat` will achieve correct rounding for arbitrary fractions, the step here will not be needed.

```

2271 \def\XINT_flpow_III #1.#2.#3.#4.#5%
2272 {%
2273     \expandafter\XINT_flpow_IIIend
2274     \xint_UDsignfork
2275     #5{\xintNum{1/#3[\xint_c_ii*#4-\xint_c_i]][\xint_c_i-\xint_c_ii*#4-#2]}}%

```

```

2276      -{{#3[#2]}}}%
2277      \krof #1%
2278 }%
2279 \def\XINT_flpow_IIIend #1#2#3%
2280      {#3{\if#21\xint_afterfi{\expandafter-\romannumeral`&&}\fi#1}}%

```

7.67 \xintFloatPower, \XINTinFloatPower

1.07. The core loop has been re-organized in 1.09j for some slight efficiency gain. The exponent B is given to \xintNum. The ^ in expressions is mapped to this routine.

Same modifications as in \xintFloatPow for 1.2f.

1.2f adds a special macro for allowing half-integral exponents for use with ^ within \xintfloat-texpr. The exponent will be first truncated to either an integer or an half-integer.

```

2281 \def\xintFloatPower {\romannumeral0\xintfloatpower}%
2282 \def\xintfloatpower #1{\XINT_flpower_chkopt \xintfloat #1\xint_relax }%
2283 \def\XINTinFloatPower {\romannumeral0\XINTinfloatpower }%
2284 \def\XINTinfloatpower #1{\XINT_flpower_chkopt \XINTinfloatS #1\xint_relax }%
2285 \def\XINTinFloatPowerH {\romannumeral0\XINTinfloatpowerH }%
2286 \def\XINTinfloatpowerH #1#2%
2287     {\expandafter\XINT_flpowerh_a \romannumeral0\xinttrunc 1{#2}.0;%
2288     \XINTdigits.{#1}{\XINTinfloatS[\XINTdigits]}}%
2289 \def\XINT_flpowerh_a #1.#2%
2290 {%
2291     \ifnum#2>\xint_c_iv\xint_dothis\XINT_flpowerh_b\fi
2292     \xint_orthat\XINT_flpowerh_i #1.#2%
2293 }%
2294 \def\XINT_flpowerh_i #1.#2;%
2295     {\expandafter\XINT_flpower_checkB_a\romannumeral0\xintinum{#1}.}%
2296 \def\XINT_flpowerh_b #1%
2297 {%
2298     \if#1-\xint_dothis\XINT_flpowerh_bneg\fi
2299     \xint_orthat{\XINT_flpowerh_bpos #1}%
2300 }%
2301 \def\XINT_flpowerh_bpos #1.#2;\XINTdigits.#3#4%
2302 {%
2303     \expandafter\XINT_flpower_checkB_a
2304     \romannumeral0\xintinc{\xintDouble{#1}}.%
2305     \XINTdigits.{#3}{\XINTinfloatsqrt[\XINTdigits]}}%
2306 }%
2307 \def\XINT_flpowerh_bneg #1.#2;\XINTdigits.#3#4%
2308 {%
2309     \expandafter\XINT_flpower_checkB_a
2310     \expandafter-\romannumeral0\xintinc{\xintDouble{#1}}.%
2311     \XINTdigits.{#3}{\XINTinfloatsqrt[\XINTdigits]}}%
2312 }%
2313 \def\XINT_flpower_chkopt #1#2%
2314 {%
2315     \ifx [#2\expandafter\XINT_flpower_opt
2316     \else\expandafter\XINT_flpower_noopt
2317     \fi
2318     #1#2%
2319 }%

```

7 Package *xintfrac* implementation

```

2320 \def\XINT_flpower_noopt #1#2\xint_relax #3%
2321 {%
2322   \expandafter\XINT_flpower_checkB_a
2323   \romannumeral0\xintnum{#3}.\XINTdigits.{#2}{#1[\XINTdigits]}%
2324 }%
2325 \def\XINT_flpower_opt #1[\xint_relax #2]%
2326 {%
2327   \expandafter\XINT_flpower_opt_a\the\numexpr #2.#1%
2328 }%
2329 \def\XINT_flpower_opt_a #1.#2#3#4%
2330 {%
2331   \expandafter\XINT_flpower_checkB_a
2332   \romannumeral0\xintnum{#4}.#1.{#3}{#2[#1]}%
2333 }%
2334 \def\XINT_flpower_checkB_a #1%
2335 {%
2336   \xint_UDzerominusfork
2337   #1-\XINT_flpower_BisZero
2338   0#1{\XINT_flpower_checkB_b -}%
2339   0-{\XINT_flpower_checkB_b }{#1}%
2340   \krof
2341 }%
2342 \def\XINT_flpower_BisZero .#1.#2#3{#3{1[0]}}%
2343 \def\XINT_flpower_checkB_b #1#2.#3.%
2344 {%
2345   \expandafter\XINT_flpower_checkB_c
2346   \the\numexpr\xintLength{#2}+\xint_c_iii.#3.#2.{#1}%
2347 }%
2348 \def\XINT_flpower_checkB_c #1.#2.%
2349 {%
2350   \expandafter\XINT_flpower_checkB_d\the\numexpr#1+#2.#1.#2.%
2351 }%
2352 \def\XINT_flpower_checkB_d #1.#2.#3.#4.#5#6%
2353 {%
2354   \expandafter \XINT_flpower_aa
2355   \romannumeral0\XINTinfloat [#3]{#6}{#2}{#1}{#4}{#5}%
2356 }%
2357 \def\XINT_flpower_aa #1[#2]#3%
2358 {%
2359   \expandafter\XINT_flpower_ab\the\numexpr #2-#3\expandafter.%
2360   \romannumeral\XINT_rep #3\endcsname0.#1.%
2361 }%
2362 \def\XINT_flpower_ab #1.#2.#3.{\XINT_flpower_a #3#2[#1]}%
2363 \def\XINT_flpower_a #1%
2364 {%
2365   \xint_UDzerominusfork
2366   #1-\XINT_flpow_zero
2367   0#1{\XINT_flpower_b \iftrue}%
2368   0-{\XINT_flpower_b \iffalse#1}%
2369   \krof
2370 }%
2371 \def\XINT_flpower_b #1#2[#3]#4#5%

```

```

2372 {%
2373   \XINT_flpower_loopI #5.#3.#2.#4.{#1\xintiiOdd{#5}\fi}%
2374 }%
2375 \def\XINT_flpower_loopI #1.%
2376 {%
2377   \if1\XINT_isOne {#1}\xint_dothis\XINT_flpower_ItoIII\fi
2378   \if1\xintiiOdd {#1}\xint_dothis{\expandafter\XINT_flpower_loopI_odd}\fi
2379   \xint_orthat{\expandafter\XINT_flpower_loopI_even}%
2380   \romannumeral0\XINT_half
2381   #1\xint_bye\xint_Bye345678\xint_bye
2382   *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax.%
2383 }%
2384 \def\XINT_flpower_ItoIII #1.#2.#3.#4.#5%
2385 {%
2386   \expandafter\XINT_flpow_III\the\numexpr #5+\xint_c_.#2.#3.#4.%
2387 }%
2388 \def\XINT_flpower_loopI_even #1.#2.#3.#4.%
2389 {%
2390   \expandafter\XINT_flpower_toloopI
2391   \the\numexpr\expandafter\XINT_flpow_truncate
2392   \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#1.%
2393 }%
2394 \def\XINT_flpower_toloopI #1.#2.#3.#4.{\XINT_flpower_loopI #4.#1.#2.#3.}%
2395 \def\XINT_flpower_loopI_odd #1.#2.#3.#4.%
2396 {%
2397   \expandafter\XINT_flpower_toloopII
2398   \the\numexpr\expandafter\XINT_flpow_truncate
2399   \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.%
2400   #1.#2.#3.%
2401 }%
2402 \def\XINT_flpower_toloopII #1.#2.#3.#4.{\XINT_flpower_loopII #4.#1.#2.#3.}%
2403 \def\XINT_flpower_loopII #1.%
2404 {%
2405   \if1\XINT_isOne{#1}\xint_dothis\XINT_flpower_ItoIII\fi
2406   \if1\xintiiOdd{#1}\xint_dothis{\expandafter\XINT_flpower_loopII_odd}\fi
2407   \xint_orthat{\expandafter\XINT_flpower_loopII_even}%
2408   \romannumeral0\XINT_half#1\xint_bye\xint_Bye345678\xint_bye
2409   *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax.%
2410 }%
2411 \def\XINT_flpower_loopII_even #1.#2.#3.#4.%
2412 {%
2413   \expandafter\XINT_flpower_toloopII
2414   \the\numexpr\expandafter\XINT_flpow_truncate
2415   \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#1.%
2416 }%
2417 \def\XINT_flpower_loopII_odd #1.#2.#3.#4.#5.#6.%
2418 {%
2419   \expandafter\XINT_flpower_loopII_odda
2420   \the\numexpr\expandafter\XINT_flpow_truncate
2421   \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2422   #1.#2.#3.%
2423 }%

```

```

2424 \def\XINT_flpower_loopII_odda #1.#2.#3.#4.#5.#6.%
2425 {%
2426   \expandafter\XINT_flpower_toloopII
2427   \the\numexpr\expandafter\XINT_flpow_truncate
2428   \the\numexpr\xint_c_ii*#5\expandafter.\romannumeral0\xintiisqr{#6}.#3.%
2429   #4.#1.#2.%
2430 }%
2431 \def\XINT_flpower_IItoIII #1.#2.#3.#4.#5.#6.#7%
2432 {%
2433   \expandafter\XINT_flpow_III\the\numexpr #7+\xint_c_\expandafter.%
2434   \the\numexpr\expandafter\XINT_flpow_truncate
2435   \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2436 }%

```

7.68 \xintFloatFac, \XINTFloatFac

```

2437 \def\xintFloatFac {\romannumeral0\xintfloatfac}%
2438 \def\xintfloatfac #1{\XINT_flfac_chkopt \xintfloat #1\xint_relax }%
2439 \def\XINTinFloatFac {\romannumeral0\XINTinfloatfac }%
2440 \def\XINTinfloatfac #1{\XINT_flfac_chkopt \XINTinfloat #1\xint_relax }%
2441 \def\XINT_flfac_chkopt #1#2%
2442 {%
2443   \ifx [#2\expandafter\XINT_flfac_opt
2444   \else\expandafter\XINT_flfac_noopt
2445   \fi
2446   #1#2%
2447 }%
2448 \def\XINT_flfac_noopt #1#2\xint_relax
2449 {%
2450   \expandafter\XINT_FL_fac_fork_a
2451   \the\numexpr \xintNum{#2}.\xint_c_i \XINTdigits\XINT_FL_fac_out{#1[\XINTdigits]]}%
2452 }%
2453 \def\XINT_flfac_opt #1[\xint_relax #2]%
2454 {%
2455   \expandafter\XINT_flfac_opt_a\the\numexpr #2.#1%
2456 }%
2457 \def\XINT_flfac_opt_a #1.#2#3%
2458 {%
2459   \expandafter\XINT_FL_fac_fork_a\the\numexpr \xintNum{#3}.\xint_c_i {#1}\XINT_FL_fac_out{#2[#1]]}%
2460 }%
2461 \def\XINT_FL_fac_fork_a #1%
2462 {%
2463   \xint_UDzerominusfork
2464   #1-\XINT_FL_fac_iszero
2465   0#1\XINT_FL_fac_isneg
2466   0-{\XINT_FL_fac_fork_b #1}%
2467   \krof
2468 }%
2469 \def\XINT_FL_fac_iszero #1.#2#3#4#5{#5{1[0]}}%

```

1.2f XINT_FL_fac_isneg returns 0, earlier versions used 1 here.

```

2470 \def\XINT_FL_fac_isneg #1.#2#3#4#5{\expandafter\xintError:FactorialOfNegative #5{0[0]}}%
2471 \def\XINT_FL_fac_fork_b #1.%

```



```

2472 {%
2473   \ifnum #1>\xint_c_x^viii_mone\xint_dothis\XINT_FL_fac_toobig\fi
2474   \ifnum #1>\xint_c_x^iv\xint_dothis\XINT_FL_fac_vbig \fi
2475   \ifnum #1>465 \xint_dothis\XINT_FL_fac_big\fi
2476   \ifnum #1>101 \xint_dothis\XINT_FL_fac_med\fi
2477   \xint_orthat\XINT_FL_fac_small
2478   #1.%
2479 }%
2480 \def\XINT_FL_fac_toobig #1.#2#3#4#5{\expandafter\xintError:TooBigFactorial #5{1[0]}}%

```

Computations are done with Q blocks of eight digits. When a multiplication has a carry, hence creates $Q+1$ blocks, the least significant one is dropped. The goal is to compute an approximate value X' to the exact value X , such that the final relative error $(X-X')/X$ will be at most 10^{-P-1} with P the desired precision. Then, when we round X' to X'' with P significant digits, we can prove that the absolute error $|X-X''|$ is bounded (strictly) by $0.6 \text{ ulp}(X'')$. (ulp= unit in the last (significant) place). Let N be the number of such operations, the formula for Q deduces from the previous explanations is that $8Q$ should be at least $P+9+k$, with k the number of digits of N (in base 10). Note that 1.2 version used $P+10+k$, for 1.2f I reduced to $P+9+k$. Also, k should be the number of digits of the number N of multiplications done, hence for $n \leq 10000$ we can take $N=n/2$, or $N/3$, or $N/4$. This is rounded above by `numexpr` and always an overestimate of the actual number of approximate multiplications done (the first ones are exact). (vérifier ce que je raconte, j'ai la flemme là).

We then want $\text{ceil}((P+k+n)/8)$. Using `\numexpr` rounding division (ARRRRRGGGHHHH), if m is a positive integer, $\text{ceil}(m/8)$ can be computed as $(m+3)/8$. Thus with $m=P+10+k$, this gives $Q < -(P+13+k)/8$. The routine actually computes $8(Q-1)$ for use in `\XINT_FL_fac_addzeros`.

With 1.2f the formula is $m=P+9+k$, $Q < -(P+12+k)/8$, and we use now $4=12-8$ rather than the earlier $5=13-8$. Whatever happens, the value computed in `\XINT_FL_fac_increaseP` is at least 8. There will always be an extra block.

Note: with Digits:=32; Maple gives for 200!:

```
> factorial(200.);
```

375

```
0.78865786736479050355236321393218 10
```

My 1.2f routine (and also 1.2) outputs:

```
7.8865786736479050355236321393219e374
```

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

```
7.886578673647905035523632139321850622951e374
```

Maple's result (contrarily to `xint`) is thus not the correct rounding but still it is less than 0.6 ulp wrong.

```

2481 \def\XINT_FL_fac_vbig
2482   {\expandafter\XINT_FL_fac_vbigloop_a
2483   \the\numexpr \XINT_FL_fac_increaseP \xint_c_i }%
2484 \def\XINT_FL_fac_big
2485   {\expandafter\XINT_FL_fac_bigloop_a
2486   \the\numexpr \XINT_FL_fac_increaseP \xint_c_ii }%
2487 \def\XINT_FL_fac_med
2488   {\expandafter\XINT_FL_fac_medloop_a
2489   \the\numexpr \XINT_FL_fac_increaseP \xint_c_iii }%
2490 \def\XINT_FL_fac_small
2491   {\expandafter\XINT_FL_fac_smallloop_a
2492   \the\numexpr \XINT_FL_fac_increaseP \xint_c_iv }%
2493 \def\XINT_FL_fac_increaseP #1#2.#3#4%
2494 {%
2495   #2\expandafter.\the\numexpr\xint_c_viii*%

```

7 Package *xintfrac* implementation

```

2496      ((\xint_c_iv+#4+\expandafter\XINT_FL_fac_countdigits
2497          \the\numexpr #2/(#1*#3)\relax 87654321\Z)/\xint_c_viii).%
2498 }%
2499 \def\XINT_FL_fac_countdigits #1#2#3#4#5#6#7#8{\XINT_FL_fac_countdone }%
2500 \def\XINT_FL_fac_countdone #1#2\Z {#1}%
2501 \def\XINT_FL_fac_out #1\Z![#2]#3{#3{\romannumeral0\XINT_mul_out
2502      #1\Z!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W [#2]}}%
2503 \def\XINT_FL_fac_vbigloop_a #1.#2.%
2504 {%
2505     \XINT_FL_fac_bigloop_a \xint_c_x^iv.#2.%
2506     {\expandafter\XINT_FL_fac_vbigloop_loop\the\numexpr 100010001\expandafter.%
2507         \the\numexpr \xint_c_x^viii+#1.}%
2508 }%
2509 \def\XINT_FL_fac_vbigloop_loop #1.#2.%
2510 {%
2511     \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2512     \expandafter\XINT_FL_fac_vbigloop_loop
2513     \the\numexpr #1+\xint_c_i\expandafter.%
2514     \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_mul #1!%
2515 }%
2516 \def\XINT_FL_fac_bigloop_a #1.%
2517 {%
2518     \expandafter\XINT_FL_fac_bigloop_b \the\numexpr
2519     #1+\xint_c_i-\xint_c_ii*((#1-464)/\xint_c_ii).#1.%
2520 }%
2521 \def\XINT_FL_fac_bigloop_b #1.#2.#3.%
2522 {%
2523     \expandafter\XINT_FL_fac_medloop_a
2524     \the\numexpr #1-\xint_c_i.#3.{\XINT_FL_fac_bigloop_loop #1.#2.}%
2525 }%
2526 \def\XINT_FL_fac_bigloop_loop #1.#2.%
2527 {%
2528     \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2529     \expandafter\XINT_FL_fac_bigloop_loop
2530     \the\numexpr #1+\xint_c_ii\expandafter.%
2531     \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_bigloop_mul #1!%
2532 }%
2533 \def\XINT_FL_fac_bigloop_mul #1!%
2534 {%
2535     \expandafter\XINT_FL_fac_mul
2536     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2537 }%
2538 \def\XINT_FL_fac_medloop_a #1.%
2539 {%
2540     \expandafter\XINT_FL_fac_medloop_b
2541     \the\numexpr #1+\xint_c_i-\xint_c_iii*((#1-100)/\xint_c_iii).#1.%
2542 }%
2543 \def\XINT_FL_fac_medloop_b #1.#2.#3.%
2544 {%
2545     \expandafter\XINT_FL_fac_smallloop_a
2546     \the\numexpr #1-\xint_c_i.#3.{\XINT_FL_fac_medloop_loop #1.#2.}%
2547 }%

```

7 Package *xintfrac* implementation

```

2548 \def\XINT_FL_fac_medloop_loop #1.#2.%
2549 {%
2550   \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2551   \expandafter\XINT_FL_fac_medloop_loop
2552   \the\numexpr #1+\xint_c_iii\expandafter.%
2553   \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_medloop_mul #1!%
2554 }%
2555 \def\XINT_FL_fac_medloop_mul #1!%
2556 {%
2557   \expandafter\XINT_FL_fac_mul
2558   \the\numexpr
2559     \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2560 }%
2561 \def\XINT_FL_fac_smallloop_a #1.%
2562 {%
2563   \csname
2564     XINT_FL_fac_smallloop_\the\numexpr #1-\xint_c_iv*(#1/\xint_c_iv)\relax
2565   \endcsname #1.%
2566 }%
2567 \expandafter\def\csname XINT_FL_fac_smallloop_1\endcsname #1.#2.%
2568 {%
2569   \XINT_FL_fac_addzeros #2.100000001!.{2.#1.}{#2}%
2570 }%
2571 \expandafter\def\csname XINT_FL_fac_smallloop_-2\endcsname #1.#2.%
2572 {%
2573   \XINT_FL_fac_addzeros #2.100000002!.{3.#1.}{#2}%
2574 }%
2575 \expandafter\def\csname XINT_FL_fac_smallloop_-1\endcsname #1.#2.%
2576 {%
2577   \XINT_FL_fac_addzeros #2.100000006!.{4.#1.}{#2}%
2578 }%
2579 \expandafter\def\csname XINT_FL_fac_smallloop_0\endcsname #1.#2.%
2580 {%
2581   \XINT_FL_fac_addzeros #2.100000024!.{5.#1.}{#2}%
2582 }%
2583 \def\XINT_FL_fac_addzeros #1.%
2584 {%
2585   \ifnum #1=\xint_c_viii \expandafter\XINT_FL_fac_addzeros_exit\fi
2586   \expandafter\XINT_FL_fac_addzeros\the\numexpr #1-\xint_c_viii.100000000!%
2587 }%

```

We will manipulate by successive **small** multiplications *Q* blocks *1<8d>!*, terminated by *1\Z!*. We need a custom small multiplication which tells us when it has create a new block, and the least significant one should be dropped.

```

2588 \def\XINT_FL_fac_addzeros_exit #1.#2.#3#4{\XINT_FL_fac_smallloop_loop #3#21\Z![-#4]}%
2589 \def\XINT_FL_fac_smallloop_loop #1.#2.%
2590 {%
2591   \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2592   \expandafter\XINT_FL_fac_smallloop_loop
2593   \the\numexpr #1+\xint_c_iv\expandafter.%
2594   \the\numexpr #2\expandafter.\romannumeral0\XINT_FL_fac_smallloop_mul #1!%
2595 }%
2596 \def\XINT_FL_fac_smallloop_mul #1!%

```

7 Package *xintfrac* implementation

```

2597 {%
2598   \expandafter\XINT_FL_fac_mul
2599   \the\numexpr
2600     \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2601 }%[[
2602 \def\XINT_FL_fac_loop_exit #1!#2]#3{#3#2}}%
2603 \def\XINT_FL_fac_mul 1#1!%
2604   {\expandafter\XINT_FL_fac_mul_a\the\numexpr\XINT_FL_fac_smallmul 10!{#1}}%
2605 \def\XINT_FL_fac_mul_a #1-#2%
2606 {%
2607   \if#21\xint_afterfi{\expandafter\space\xint_gob_til_exclam}\else
2608   \expandafter\space\fi #11\Z!%
2609 }%
2610 \def\XINT_FL_fac_minimulwc_a #1#2#3#4#5!#6#7#8#9%
2611 {%
2612   \XINT_FL_fac_minimulwc_b {#1#2#3#4}{#5}{#6#7#8#9}%
2613 }%
2614 \def\XINT_FL_fac_minimulwc_b #1#2#3#4!#5%
2615 {%
2616   \expandafter\XINT_FL_fac_minimulwc_c
2617   \the\numexpr \xint_c_x^ix+#5+#2*#4.{#1}{#2}{#3}{#4}}%
2618 }%
2619 \def\XINT_FL_fac_minimulwc_c 1#1#2#3#4#5#6.#7%
2620 {%
2621   \expandafter\XINT_FL_fac_minimulwc_d {#1#2#3#4#5}#7{#6}%
2622 }%
2623 \def\XINT_FL_fac_minimulwc_d #1#2#3#4#5%
2624 {%
2625   \expandafter\XINT_FL_fac_minimulwc_e
2626   \the\numexpr \xint_c_x^ix+#1+#2*#5+#3*#4.{#2}{#4}%
2627 }%
2628 \def\XINT_FL_fac_minimulwc_e 1#1#2#3#4#5#6.#7#8#9%
2629 {%
2630   1#6#9\expandafter!%
2631   \the\numexpr\expandafter\XINT_FL_fac_smallmul
2632   \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#7*#8!%
2633 }%
2634 \def\XINT_FL_fac_smallmul 1#1!#21#3!%
2635 {%
2636   \xint_gob_til_Z #3\XINT_FL_fac_smallmul_end\Z
2637   \XINT_FL_fac_minimulwc_a #2!#3!{#1}{#2}%
2638 }%

```

This is the crucial ending. I note that I used here an `\ifnum` test rather than the `gob_til_eightzeroes` thing. Actually for eight digits there is much less difference than for only four.

The "carry" situation is marked by a final `!-1` rather than `!-2` for no-carry. (a `\numexpr` must be stopped, and leaving a `-` as delimiter is good as it will not arise earlier.)

```

2639 \def\XINT_FL_fac_smallmul_end\Z\XINT_FL_fac_minimulwc_a #1!\Z!#2#3[#4]%
2640 {%
2641   \ifnum #2=\xint_c_
2642     \expandafter\xint_firstoftwo\else
2643     \expandafter\xint_secondoftwo
2644   \fi

```

```

2645 {-2\relax[#4]}%
2646 {1#2\expandafter!\expandafter-\expandafter1\expandafter
2647 [\the\numexpr #4+\xint_c_viii]}%
2648 }%

```

7.69 \xintFloatPFactorial, \XINTinFloatPFactorial

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

1.2h (2016/11/20) now avoids raising \xintError:OutOfRangePFac if the condition $0 \leq a \leq b < 10^8$ is violated. Same as for \xintiiPFactorial.

```

2649 \def\xintFloatPFactorial {\romannumeral0\xintfloatpfactorial}%
2650 \def\xintfloatpfactorial #1{\XINT_flpfac_chkopt \xintfloat #1\xint_relax}%
2651 \def\XINTinFloatPFactorial {\romannumeral0\XINTinfloatpfactorial}%
2652 \def\XINTinfloatpfactorial #1{\XINT_flpfac_chkopt \XINTinfloat #1\xint_relax}%
2653 \def\XINT_flpfac_chkopt #1#2%
2654 {%
2655   \ifx [#2\expandafter\XINT_flpfac_opt
2656   \else\expandafter\XINT_flpfac_noopt
2657   \fi
2658   #1#2%
2659 }%
2660 \def\XINT_flpfac_noopt #1#2\xint_relax #3%
2661 {%
2662   \expandafter\XINT_FL_pfac_fork
2663   \the\numexpr \xintNum{#2}\expandafter.%
2664   \the\numexpr \xintNum{#3}.\xint_c_i{\XINTdigits}{#1[\XINTdigits]}%
2665 }%
2666 \def\XINT_flpfac_opt #1[\xint_relax #2]%
2667 {%
2668   \expandafter\XINT_flpfac_opt_b\the\numexpr #2.#1%
2669 }%
2670 \def\XINT_flpfac_opt_b #1.#2#3#4%
2671 {%
2672   \expandafter\XINT_FL_pfac_fork
2673   \the\numexpr \xintNum{#3}\expandafter.%
2674   \the\numexpr \xintNum{#4}.\xint_c_i{#1}{#2[#1]}%
2675 }%
2676 \def\XINT_FL_pfac_fork #1#2.#3#4.%
2677 {%
2678   \unless\ifnum #1#2<#3#4 \xint_dothis\XINT_FL_pfac_one\fi
2679   \if-#3\xint_dothis\XINT_FL_pfac_neg \fi
2680   \if-#1\xint_dothis\XINT_FL_pfac_zero\fi
2681   \ifnum #3#4>\xint_c_x^viii_mone\xint_dothis\XINT_FL_pfac_outofrange\fi
2682   \xint_orthat \XINT_FL_pfac_increaseP #1#2.#3#4.%
2683 }%
2684 \def\XINT_FL_pfac_outofrange #1.#2.#3#4#5{\xintError:OutOfRangePFac #5{0[0]}}%
2685 \def\XINT_FL_pfac_one #1.#2.#3#4#5{#5{1[0]}}%
2686 \def\XINT_FL_pfac_zero #1.#2.#3#4#5{#5{0[0]}}%
2687 \def\XINT_FL_pfac_neg -#1.-#2.%
2688 {%
2689   \ifnum #1>\xint_c_x^viii\xint_dothis\XINT_FL_pfac_outofrange\fi
2690   \xint_orthat {%

```

7 Package *xintfrac* implementation

```

2691 \ifodd\numexpr#2-#1\relax\xint_afterfi{\expandafter-\romannumeral`&&@\fi
2692 \expandafter\XINT_FL_pfac_increaseP}%
2693 \the\numexpr #2-\xint_c_i\expandafter.\the\numexpr#1-\xint_c_i.%
2694 }%

```

See the comments for `\XINT_FL_pfac_increaseP`. Case of $b=a+1$ should be filtered out perhaps. We only needed here to copy the `\xintPFactorial` macros and re-use `\XINT_FL_fac_mul/\XINT_FL_fac_out`. Had to modify a bit `\XINT_FL_pfac_addzeroes`. We can enter here directly with #3 equal to specify the precision (the calculated value before final rounding has a relative error less than $\#3 \cdot 10^{\{-\#4-1\}}$), and #5 would hold the macro doing the final rounding (or truncating, if I make a `FloatTrunc` available) to a given number of digits, possibly not #4. By default the #3 is 1, but `FloatBinomial` calls it with #3=4.

```

2695 \def\XINT_FL_pfac_increaseP #1.#2.#3#4%
2696 {%
2697   \expandafter\XINT_FL_pfac_a
2698   \the\numexpr \xint_c_viii*((\xint_c_iv+#4+\expandafter
2699     \XINT_FL_fac_countdigits\the\numexpr (#2-#1-\xint_c_i)%
2700     /\ifnum #2>\xint_c_x^iv #3\else(#3*\xint_c_ii)\fi\relax
2701     87654321\Z)/\xint_c_viii).#1.#2.%
2702 }%
2703 \def\XINT_FL_pfac_a #1.#2.#3.%
2704 {%
2705   \expandafter\XINT_FL_pfac_b\the\numexpr \xint_c_i+#2\expandafter.%
2706   \the\numexpr#3\expandafter.%
2707   \romannumeral0\XINT_FL_pfac_addzeroes #1.1000000001!1\Z![-#1]%
2708 }%
2709 \def\XINT_FL_pfac_addzeroes #1.%
2710 {%
2711   \ifnum #1=\xint_c_viii \expandafter\XINT_FL_pfac_addzeroes_exit\fi
2712   \expandafter\XINT_FL_pfac_addzeroes\the\numexpr #1-\xint_c_viii.100000000!%
2713 }%
2714 \def\XINT_FL_pfac_addzeroes_exit #1.{ }%
2715 \def\XINT_FL_pfac_b #1.%
2716 {%
2717   \ifnum #1>9999 \xint_dothis\XINT_FL_pfac_vbigloop \fi
2718   \ifnum #1>463 \xint_dothis\XINT_FL_pfac_bigloop \fi
2719   \ifnum #1>98 \xint_dothis\XINT_FL_pfac_medloop \fi
2720   \xint_orthat\XINT_FL_pfac_smallloop #1.%
2721 }%
2722 \def\XINT_FL_pfac_smallloop #1.#2.%
2723 {%
2724   \ifcase\numexpr #2-#1\relax
2725     \expandafter\XINT_FL_pfac_end_
2726   \or \expandafter\XINT_FL_pfac_end_i
2727   \or \expandafter\XINT_FL_pfac_end_ii
2728   \or \expandafter\XINT_FL_pfac_end_iii
2729   \else\expandafter\XINT_FL_pfac_smallloop_a
2730   \fi #1.#2.%
2731 }%
2732 \def\XINT_FL_pfac_smallloop_a #1.#2.%
2733 {%
2734   \expandafter\XINT_FL_pfac_smallloop_b

```

7 Package *xintfrac* implementation

```

2735 \the\numexpr #1+\xint_c_iv\expandafter.%
2736 \the\numexpr #2\expandafter.%
2737 \romannumeral0\expandafter\XINT_FL_fac_mul
2738 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2739 }%
2740 \def\XINT_FL_pfac_smallloop_b #1.%
2741 {%
2742 \ifnum #1>98 \expandafter\XINT_FL_pfac_medloop \else
2743 \expandafter\XINT_FL_pfac_smallloop \fi #1.%
2744 }%
2745 \def\XINT_FL_pfac_medloop #1.#2.%
2746 {%
2747 \ifcase\numexpr #2-#1\relax
2748 \expandafter\XINT_FL_pfac_end_
2749 \or \expandafter\XINT_FL_pfac_end_i
2750 \or \expandafter\XINT_FL_pfac_end_ii
2751 \else\expandafter\XINT_FL_pfac_medloop_a
2752 \fi #1.#2.%
2753 }%
2754 \def\XINT_FL_pfac_medloop_a #1.#2.%
2755 {%
2756 \expandafter\XINT_FL_pfac_medloop_b
2757 \the\numexpr #1+\xint_c_iii\expandafter.%
2758 \the\numexpr #2\expandafter.%
2759 \romannumeral0\expandafter\XINT_FL_fac_mul
2760 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2761 }%
2762 \def\XINT_FL_pfac_medloop_b #1.%
2763 {%
2764 \ifnum #1>463 \expandafter\XINT_FL_pfac_bigloop \else
2765 \expandafter\XINT_FL_pfac_medloop \fi #1.%
2766 }%
2767 \def\XINT_FL_pfac_bigloop #1.#2.%
2768 {%
2769 \ifcase\numexpr #2-#1\relax
2770 \expandafter\XINT_FL_pfac_end_
2771 \or \expandafter\XINT_FL_pfac_end_i
2772 \else\expandafter\XINT_FL_pfac_bigloop_a
2773 \fi #1.#2.%
2774 }%
2775 \def\XINT_FL_pfac_bigloop_a #1.#2.%
2776 {%
2777 \expandafter\XINT_FL_pfac_bigloop_b
2778 \the\numexpr #1+\xint_c_ii\expandafter.%
2779 \the\numexpr #2\expandafter.%
2780 \romannumeral0\expandafter\XINT_FL_fac_mul
2781 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2782 }%
2783 \def\XINT_FL_pfac_bigloop_b #1.%
2784 {%
2785 \ifnum #1>9999 \expandafter\XINT_FL_pfac_vbigloop \else
2786 \expandafter\XINT_FL_pfac_bigloop \fi #1.%

```

```

2787 }%
2788 \def\XINT_FL_pfac_vbigloop #1.#2.%
2789 {%
2790   \ifnum #2=#1
2791     \expandafter\XINT_FL_pfac_end_
2792   \else\expandafter\XINT_FL_pfac_vbigloop_a
2793   \fi #1.#2.%
2794 }%
2795 \def\XINT_FL_pfac_vbigloop_a #1.#2.%
2796 {%
2797   \expandafter\XINT_FL_pfac_vbigloop
2798   \the\numexpr #1+\xint_c_i\expandafter.%
2799   \the\numexpr #2\expandafter.%
2800   \romannumeral0\expandafter\XINT_FL_fac_mul
2801   \the\numexpr\xint_c_x^viii+#1!%
2802 }%
2803 \def\XINT_FL_pfac_end_iii #1.#2.%
2804 {%
2805   \expandafter\XINT_FL_fac_out
2806   \romannumeral0\expandafter\XINT_FL_fac_mul
2807   \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2808 }%
2809 \def\XINT_FL_pfac_end_ii #1.#2.%
2810 {%
2811   \expandafter\XINT_FL_fac_out
2812   \romannumeral0\expandafter\XINT_FL_fac_mul
2813   \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2814 }%
2815 \def\XINT_FL_pfac_end_i #1.#2.%
2816 {%
2817   \expandafter\XINT_FL_fac_out
2818   \romannumeral0\expandafter\XINT_FL_fac_mul
2819   \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2820 }%
2821 \def\XINT_FL_pfac_end_ #1.#2.%
2822 {%
2823   \expandafter\XINT_FL_fac_out
2824   \romannumeral0\expandafter\XINT_FL_fac_mul
2825   \the\numexpr \xint_c_x^viii+#1!%
2826 }%

```

7.70 \xintFloatBinomial, \XINTinFloatBinomial

1.2f. We compute $\text{binomial}(x,y)$ as $\text{pfac}(x-y,x)/y!$, where the numerator and denominator are computed with a relative error at most $4 \cdot 10^{-P-2}$, then rounded (once I have a float truncation, I will use truncation rather) to $P+3$ digits, and finally the quotient is correctly rounded to P digits. This will guarantee that the exact value X differs from the computed one Y by at most $0.6 \text{ ulp}(Y)$. (2015/12/01).

2016/11/19 for 1.2h. As for `\xintiiBinomial`, hard to understand why last year I coded this to raise an error if $y < 0$ or $y > x$! The question of the Gamma function is for another occasion, here x and y must be (small) integers.

```

2827 \def\xintFloatBinomial    {\romannumeral0\xintfloatbinomial}%

```


7 Package *xintfrac* implementation

```

2828 \def\xintfloatbinomial #1{\XINT_flbinom_chkopt \xintfloat #1\xint_relax }%
2829 \def\XINTinFloatBinomial {\romannumeral0\XINTinfloatbinomial }%
2830 \def\XINTinfloatbinomial #1{\XINT_flbinom_chkopt \XINTinfloat #1\xint_relax }%
2831 \def\XINT_flbinom_chkopt #1#2%
2832 {%
2833   \ifx [#2\expandafter\XINT_flbinom_opt
2834     \else\expandafter\XINT_flbinom_noopt
2835   \fi #1#2%
2836 }%
2837 \def\XINT_flbinom_noopt #1#2\xint_relax #3%
2838 {%
2839   \expandafter\XINT_FL_binom_a
2840   \the\numexpr\xintNum{#2}\expandafter.\the\numexpr\xintNum{#3}.\XINTdigits.#1%
2841 }%
2842 \def\XINT_flbinom_opt #1[\xint_relax #2]#3#4%
2843 {%
2844   \expandafter\XINT_FL_binom_a
2845   \the\numexpr\xintNum{#3}\expandafter.\the\numexpr\xintNum{#4}\expandafter.%
2846   \the\numexpr #2.#1%
2847 }%
2848 \def\XINT_FL_binom_a #1.#2.%
2849 {%
2850   \expandafter\XINT_FL_binom_fork \the\numexpr #1-#2.#2.#1.%
2851 }%
2852 \def\XINT_FL_binom_fork #1#2.#3#4.#5#6.%
2853 {%
2854   \if-#5\xint_dothis \XINT_FL_binom_outofrange\fi
2855   \if-#1\xint_dothis \XINT_FL_binom_zero\fi
2856   \if-#3\xint_dothis \XINT_FL_binom_zero\fi
2857   \if0#1\xint_dothis \XINT_FL_binom_one\fi
2858   \if0#3\xint_dothis \XINT_FL_binom_one\fi
2859   \ifnum #5#6>\xint_c_x^viii_mone \xint_dothis\XINT_FL_binom_outofrange\fi
2860   \ifnum #1#2>#3#4 \xint_dothis\XINT_FL_binom_ab \fi
2861   \xint_orthat\XINT_FL_binom_aa
2862   #1#2.#3#4.#5#6.%
2863 }%
2864 \def\XINT_FL_binom_outofrange #1.#2.#3.#4.#5%
2865   {\xintError:OutOfRangeBinomial #5[#4]{0[0]}}%
2866 \def\XINT_FL_binom_one #1.#2.#3.#4.#5{#5[#4]{1[0]}}%
2867 \def\XINT_FL_binom_zero #1.#2.#3.#4.#5{#5[#4]{0[0]}}%
2868 \def\XINT_FL_binom_aa #1.#2.#3.#4.#5%
2869 {%
2870   #5[#4]{\xintDiv{\XINT_FL_pfac_increaseP
2871     #2.#3.\xint_c_iv{#4+\xint_c_i}{\XINTinfloat[#4+\xint_c_iii]}}%
2872     {\XINT_FL_fac_fork_b
2873     #1.\xint_c_iv{#4+\xint_c_i}{\XINT_FL_fac_out{\XINTinfloat[#4+\xint_c_iii]}}}%
2874 }%
2875 \def\XINT_FL_binom_ab #1.#2.#3.#4.#5%
2876 {%
2877   #5[#4]{\xintDiv{\XINT_FL_pfac_increaseP
2878     #1.#3.\xint_c_iv{#4+\xint_c_i}{\XINTinfloat[#4+\xint_c_iii]}}%
2879     {\XINT_FL_fac_fork_b

```

```

2880      #2.\xint_c_iv{#4+\xint_c_i}\XINT_FL_fac_out{\XINTinfloat[#4+\xint_c_iii]}}}%
2881 }%

```

7.71 \xintFloatSqrt, \XINTinFloatSqrt

First done for 1.08.

The float version was developed at the same time as the integer one and even a bit earlier. As a result the integer variant had some sub-optimal parts. Anyway, for 1.2f I have rewritten the integer variant, and the float variant delegates all preparatory work for it until the last step. In particular the very low precisions are not penalized anymore from doing computations for at least 17 or 18 digits. Both the large and small precisions give quite shorter computation times.

Also, after examining more closely the achieved precision I decided to extend the float version in order for it to obtain the correct rounding (for inputs already of at most P digits with P the precision) of the theoretical exact value.

Beyond about 500 digits of precision the efficiency decreases swiftly, as is the case generally speaking with xintcore/xint/xintfrac arithmetic macros.

Final note: with 1.2f the input is always first rounded to P significant places.

```

2882 \def\xintFloatSqrt      {\romannumeral0\xintfloatsqrt }%
2883 \def\xintfloatsqrt      #1{\XINT_flsqrt_chkopt \xintfloat #1\xint_relax }%
2884 \def\XINTinFloatSqrt    {\romannumeral0\XINTinfloatsqrt }%
2885 \def\XINTinfloatsqrt    #1{\XINT_flsqrt_chkopt \XINTinfloat #1\xint_relax }%
2886 \def\XINT_flsqrt_chkopt #1#2%
2887 {%
2888   \ifx [#2\expandafter\XINT_flsqrt_opt
2889     \else\expandafter\XINT_flsqrt_noopt
2890   \fi  #1#2%
2891 }%
2892 \def\XINT_flsqrt_noopt #1#2\xint_relax
2893 {%
2894   \expandafter\XINT_FL_sqrt_a
2895     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.#1%
2896 }%
2897 \def\XINT_flsqrt_opt #1[\xint_relax #2]%#3%
2898 {%
2899   \expandafter\XINT_flsqrt_opt_a\the\numexpr #2.#1%
2900 }%
2901 \def\XINT_flsqrt_opt_a #1.#2#3%
2902 {%
2903   \expandafter\XINT_FL_sqrt_a\romannumeral0\XINTinfloat[#1]{#3}#1.#2%
2904 }%
2905 \def\XINT_FL_sqrt_a #1%
2906 {%
2907   \xint_UDzerominusfork
2908   #1-\XINT_FL_sqrt_iszero
2909   0#1\XINT_FL_sqrt_isneg
2910   0-\XINT_FL_sqrt_pos #1}%
2911   \krof
2912 }%[
2913 \def\XINT_FL_sqrt_iszero #1]#2.#3{#3[#2]{0[0]}}%
2914 \def\XINT_FL_sqrt_isneg #1]#2.#3{\xintError:RootOfNegative #3[#2]{0[0]}}%
2915 \def\XINT_FL_sqrt_pos #1[#2]#3.%
2916 {%

```

7 Package *xintfrac* implementation

```

2917 \expandafter\XINT_flsqrt
2918 \the\numexpr #3\ifodd #2 \xint_dothis {+\xint_c_iii.(#2+\xint_c_i).0}\fi
2919 \xint_orthat {+\xint_c_ii.#2.{}#100.#3.%
2920 }%
2921 \def\XINT_flsqrt #1.#2.%
2922 {%
2923 \expandafter\XINT_flsqrt_a
2924 \the\numexpr #2/\xint_c_ii-(#1-\xint_c_i)/\xint_c_ii.#1.%
2925 }%
2926 \def\XINT_flsqrt_a #1.#2.#3#4.#5.%
2927 {%
2928 \expandafter\XINT_flsqrt_b
2929 \the\numexpr (#2-\xint_c_i)/\xint_c_ii\expandafter.%
2930 \romannumeral0\XINT_sqrt_start #2.#4#3.#5.#2.#4#3.#5.#1.%
2931 }%
2932 \def\XINT_flsqrt_b #1.#2#3%
2933 {%
2934 \expandafter\XINT_flsqrt_c
2935 \romannumeral0\xintiisub
2936 {\XINT_dsx_addzeros {#1}#2;}%
2937 {\xintiiDivRound{\XINT_dsx_addzeros {#1}#3;}%
2938 {\XINT_dbl#2\xint_bye2345678\xint_bye*\xint_c_ii\relax}}.%
2939 }%
2940 \def\XINT_flsqrt_c #1.#2.%
2941 {%
2942 \expandafter\XINT_flsqrt_d
2943 \romannumeral0\XINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..%
2944 }%
2945 \def\XINT_flsqrt_d #1.#2#3.%
2946 {%
2947 \ifnum #2=\xint_c_v
2948 \expandafter\XINT_flsqrt_f\else\expandafter\XINT_flsqrt_finish\fi
2949 #2#3.#1.%
2950 }%
2951 \def\XINT_flsqrt_finish #1#2.#3.#4.#5.#6.#7.#8{#8[#6]{#3#1[#7]}}%
2952 \def\XINT_flsqrt_f 5#1.%
2953 {\expandafter\XINT_flsqrt_g\romannumeral0\xintinum{#1}\relax.%
2954 \def\XINT_flsqrt_g #1#2#3.{\if\relax#2\xint_dothis{\XINT_flsqrt_h #1}\fi
2955 \xint_orthat{\XINT_flsqrt_finish 5.}}%
2956 \def\XINT_flsqrt_h #1{\ifnum #1<\xint_c_iii\xint_dothis{\XINT_flsqrt_again}\fi
2957 \xint_orthat{\XINT_flsqrt_finish 5.}}%
2958 \def\XINT_flsqrt_again #1.#2.%
2959 {%
2960 \expandafter\XINT_flsqrt_again_a\the\numexpr #2+\xint_c_viii.%
2961 }%
2962 \def\XINT_flsqrt_again_a #1.#2.#3.%
2963 {%
2964 \expandafter\XINT_flsqrt_b
2965 \the\numexpr (#1-\xint_c_i)/\xint_c_ii\expandafter.%
2966 \romannumeral0\XINT_sqrt_start #1.#2000000000.#3.%
2967 #1.#2000000000.#3.%
2968 }%

```

7.72 \backslash xintFloatE, \backslash XINTinFloatE

```

2969 \def\xintFloatE    {\romannumeral0\xintfloate }%
2970 \def\xintfloate #1{\XINT_floate_chkopt #1\xint_relax }%
2971 \def\XINT_floate_chkopt #1%
2972 {%
2973     \ifx [#1\expandafter\XINT_floate_opt
2974         \else\expandafter\XINT_floate_noopt
2975     \fi #1%
2976 }%
2977 \def\XINT_floate_noopt #1\xint_relax
2978 {%
2979     \expandafter\XINT_floate_a\expandafter\XINTdigits
2980         \romannumeral0\XINT_infrac {#1}%
2981 }%
2982 \def\XINT_floate_opt [\xint_relax #1]#2%
2983 {%
2984     \expandafter\XINT_floate_a\expandafter
2985     {\the\numexpr #1\expandafter}\romannumeral0\XINT_infrac {#2}%
2986 }%
2987 \def\XINT_floate_a #1#2#3#4#5%
2988 {%
2989     \expandafter\XINT_float_a\the\numexpr#1\expandafter.%
2990     \expandafter{\the\numexpr #2+#5}{#3}{#4}\XINT_float_Q
2991 }%
2992 \def\XINTinFloatE {\romannumeral0\XINTinfloate }%
2993 \def\XINTinfloate
2994     {\expandafter\XINT_infloate\romannumeral0\XINTinfloat [\XINTdigits]]%
2995 \def\XINT_infloate #1[#2]#3%
2996     {\expandafter\XINT_infloate_end\the\numexpr #3+#2.{#1}}%
2997 \def\XINT_infloate_end #1.#2{ #2[#1]}%

```

7.73 \backslash XINTinFloatMod

```

2998 \def\XINTinFloatMod {\romannumeral0\XINTinfloatmod [\XINTdigits]]%
2999 \def\XINTinfloatmod [#1]#2#3{\expandafter\XINT_infloatmod\expandafter
3000     {\romannumeral0\XINTinfloat[#1]{#2}}%
3001     {\romannumeral0\XINTinfloat[#1]{#3}}{#1}}%
3002 \def\XINT_infloatmod #1#2{\expandafter\XINT_infloatmod_a\expandafter {#2}{#1}}%
3003 \def\XINT_infloatmod_a #1#2#3{\XINTinfloat [#3]{\xintMod {#2}{#1}}}%
3004 \XINT_restorecatcodes_endinput%

```

8 Package *xintseries* implementation

.1	Catcodes, ε -TeX and reload detection . . .	237	.7	<code>\xintRationalSeries</code>	240
.2	Package identification	238	.8	<code>\xintRationalSeriesX</code>	241
.3	<code>\xintSeries</code>	238	.9	<code>\xintFxpPtPowerSeries</code>	242
.4	<code>\xintiSeries</code>	238	.10	<code>\xintFxpPtPowerSeriesX</code>	243
.5	<code>\xintPowerSeries</code>	239	.11	<code>\xintFloatPowerSeries</code>	243
.6	<code>\xintPowerSeriesX</code>	240	.12	<code>\xintFloatPowerSeriesX</code>	245

The commenting is currently (2016/12/13) 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 [2016/12/13 1.2i Expandable partial sums with xint package (JFB)]%

```

8.3 *\xintSeries*

```

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

```

8.4 *\xintiSeries*

```

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

```

```

84     \xint_afterfi {\XINT_iseries_loop {#1}{0}{#2}{#3}}%
85   \fi
86 }%
87 \def\XINT_iseries_loop #1#2#3#4%
88 {%
89   \ifnum #3>#1 \else \XINT_iseries_exit \fi
90   \expandafter\XINT_iseries_loop\expandafter
91   {\the\numexpr #1+1\expandafter }\expandafter
92   {\romannumeral0\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\xintiiadd {#4}{\xintiTrunc {#7}{\xintMul {#6{#2}}{#1}}}%
251   {#1}{#3}{#5}{#6}{#7}%
252 }%
253 \def\xINT_fppowseries_exit_i\fi\expandafter\xINT_fppowseries_loop_ii
254   {\fi \expandafter\xINT_fppowseries_exit_ii }%
255 \def\xINT_fppowseries_exit_ii #1#2#3#4#5#6#7%
256 {%
257   \xinttrunc {#7}
258   {\xintiiadd {#4}{\xintiTrunc {#7}{\xintMul {#6{#2}}{#1}}}{-#7}}%
259 }%

```

8.10 \xintFxFtPowerSeriesX

a,b,coeff,x,D

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

```

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

```

8.11 \xintFloatPowerSeries

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

```

283 \def\xintFloatPowerSeries {\romannumeral0\xintfloatpowerseries }%
284 \def\xintfloatpowerseries #1{\XINT_flpowseries_chkopt #1\xint_relax }%
285 \def\xINT_flpowseries_chkopt #1%
286 {%
287   \ifx [#1\expandafter\xINT_flpowseries_opt

```

```

288     \else\expandafter\XINT_flpowseries_noopt
289     \fi
290     #1%
291 }%
292 \def\XINT_flpowseries_noopt #1\xint_relax #2%
293 {%
294     \expandafter\XINT_flpowseries\expandafter
295     {\the\numexpr #1\expandafter}\expandafter
296     {\the\numexpr #2}\XINTdigits
297 }%
298 \def\XINT_flpowseries_opt [\xint_relax #1]#2#3%
299 {%
300     \expandafter\XINT_flpowseries\expandafter
301     {\the\numexpr #2\expandafter}\expandafter
302     {\the\numexpr #3\expandafter}{\the\numexpr #1}%
303 }%
304 \def\XINT_flpowseries #1#2#3#4#5%
305 {%
306     \ifnum #2<#1
307         \xint_afterfi { 0.e0}%
308     \else
309         \xint_afterfi
310         {\expandafter\XINT_flpowseries_loop_pre\expandafter
311          {\romannumeral0\XINTinfloatpow [#3]{#5}{#1}}%
312          {#1}{#5}{#2}{#4}{#3}%
313         }%
314     \fi
315 }%
316 \def\XINT_flpowseries_loop_pre #1#2#3#4#5#6%
317 {%
318     \ifnum #4>#2 \else\XINT_flpowseries_dont_i \fi
319     \expandafter\XINT_flpowseries_loop_i\expandafter
320     {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
321     {\romannumeral0\XINTinfloatmul [#6]{#5}{#2}}{#1}%
322     {#1}{#3}{#4}{#5}{#6}%
323 }%
324 \def\XINT_flpowseries_dont_i \fi\expandafter\XINT_flpowseries_loop_i
325     {\fi \expandafter\XINT_flpowseries_dont_ii }%
326 \def\XINT_flpowseries_dont_ii #1#2#3#4#5#6#7{\xintfloat [#7]{#2}}%
327 \def\XINT_flpowseries_loop_i #1#2#3#4#5#6#7%
328 {%
329     \ifnum #5>#1 \else \XINT_flpowseries_exit_i \fi
330     \expandafter\XINT_flpowseries_loop_ii\expandafter
331     {\romannumeral0\XINTinfloatmul [#7]{#3}{#4}}%
332     {#1}{#4}{#2}{#5}{#6}{#7}%
333 }%
334 \def\XINT_flpowseries_loop_ii #1#2#3#4#5#6#7%
335 {%
336     \expandafter\XINT_flpowseries_loop_i\expandafter
337     {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
338     {\romannumeral0\XINTinfloatadd [#7]{#4}%
339      {\XINTinfloatmul [#7]{#6}{#2}}{#1}}}%

```

```

340      {#1}{#3}{#5}{#6}{#7}%
341 }%
342 \def\XINT_flpowseries_exit_i\fi\expandafter\XINT_flpowseries_loop_ii
343   {\fi \expandafter\XINT_flpowseries_exit_ii }%
344 \def\XINT_flpowseries_exit_ii #1#2#3#4#5#6#7%
345 {%
346   \xintfloatadd [#7]{#4}{\XINTinfloatmul [#7]{#6{#2}}{#1}}%
347 }%

```

8.12 \xintFloatPowerSeriesX

1.08a

```

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

```

9 Package `xintcfrac` implementation

.1	Catcodes, ε -TeX and reload detection . . .	246	.16	<code>\xintiGctoF</code>	258
.2	Package identification	247	.17	<code>\xintCtoCv</code> , <code>\xintCstoCv</code>	259
.3	<code>\xintCFrac</code>	247	.18	<code>\xintiCstoCv</code>	260
.4	<code>\xintGCFrac</code>	248	.19	<code>\xintGctoCv</code>	260
.5	<code>\xintGGCFrac</code>	250	.20	<code>\xintiGctoCv</code>	262
.6	<code>\xintGctoGCx</code>	251	.21	<code>\xintFtoCv</code>	263
.7	<code>\xintFtoCs</code>	251	.22	<code>\xintFtoCCv</code>	263
.8	<code>\xintFtoCx</code>	252	.23	<code>\xintCntoF</code>	263
.9	<code>\xintFtoC</code>	252	.24	<code>\xintGcntoF</code>	264
.10	<code>\xintFtoGC</code>	253	.25	<code>\xintCntoCs</code>	265
.11	<code>\xintFGtoC</code>	253	.26	<code>\xintCntoGC</code>	265
.12	<code>\xintFtoCC</code>	254	.27	<code>\xintGcntoGC</code>	266
.13	<code>\xintCtoF</code> , <code>\xintCstoF</code>	255	.28	<code>\xintCstoGC</code>	267
.14	<code>\xintiCstoF</code>	256	.29	<code>\xintGctoGC</code>	267
.15	<code>\xintGctoF</code>	256			

The commenting is currently (2016/12/13) 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 [2016/12/13 1.2i Expandable continued fractions with xint package (JFB)]%

```

9.3 *\xintCFrac*

```

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

```

```

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

```

9.4 *\xintGCFrac*

```

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

```



```

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

```

9.5 *\xintGGCFrac*

New with 1.09m

```

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

```

```

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

```

9.6 \xintGctoGCx

```

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

```

9.7 \xintFtoCs

Modified in 1.09m: a space is added after the inserted commas.

```

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

```

```

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

```

9.8 \xintFtoCx

```

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

```

9.9 \xintFtoC

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

```

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

```

9.10 `\xintFtoGC`

```

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

```

9.11 `\xintFGtoC`

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

```

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

```

```

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

```

9.12 \xintFtoCC

```

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

```

```

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

```

9.13 `\xintCtoF`, `\xintCstof`

1.09m uses `\xintCSVtoList` on the argument of `\xintCstof` to allow spaces also before the commas. And the original `\xintCstof` code became the one of the new `\xintCtoF` dealing with a braced rather than comma separated list.

```

434 \def\xintCstof {\romannumeral0\xintcstof }%
435 \def\xintcstof #1%
436 {%
437   \expandafter\XINT_ctf_prep \romannumeral0\xintcsvtoList{#1}\xint_relax
438 }%
439 \def\xintCtoF {\romannumeral0\xintctof }%
440 \def\xintctof #1%
441 {%
442   \expandafter\XINT_ctf_prep \romannumeral`&&@#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	Old comments	269
10.2	Catcodes, ε -TeX and reload detection	269
10.3	Package identification	271
10.4	Locking and unlocking	271
10.5	<code>\XINT_expr_wrap</code> , <code>\XINT_iiexpr_wrap</code>	271
10.6	<code>\XINT_protectii</code> , <code>\XINT_expr_usethe</code>	271
10.7	<code>\XINT_expr_print</code> , <code>\XINT_iiexpr_print</code> , <code>\XINT_boolexpr_print</code>	272
10.8	<code>\xintexpr</code> , <code>\xintiexpr</code> , <code>\xintfloatexpr</code> , <code>\xintiiepr</code>	272
10.9	<code>\xinttheexpr</code> , <code>\xinttheiexpr</code> , <code>\xintthefloatexpr</code> , <code>\xinttheiiepr</code>	272
10.10	<code>\xintthe</code>	272
10.11	<code>\thexintexpr</code> , <code>\thexintiexpr</code> , <code>\thexintfloatexpr</code> , <code>\thexintiiepr</code>	272
10.12	<code>\xintthecoords</code>	272
10.13	<code>\xintbareeval</code> , <code>\xintbarefloateval</code> , <code>\xintbareiieval</code>	273
10.14	<code>\xintthebareeval</code> , <code>\xintthebarefloateval</code> , <code>\xintthebareiieval</code>	273
10.15	<code>\xinteval</code> , <code>\xintiieval</code>	273
10.16	<code>\xintieval</code> , <code>\XINT_iexpr_wrap</code>	273
10.17	<code>\xintfloateval</code> , <code>\XINT_flexpr_wrap</code> , <code>\XINT_flexpr_print</code>	273
10.18	<code>\xintboolexpr</code> , <code>\xinttheboolexpr</code> , <code>\thexintboolexpr</code>	274
10.19	<code>\xintifboolexpr</code> , <code>\xintifboolfloatexpr</code> , <code>\xintifbooliiepr</code>	274
10.20	Macros handling csv lists on output (for <code>\XINT_expr_print</code> et al. routines)	274
10.21	<code>\XINT_expr_getnext</code> : fetching some number then an operator	276
10.22	The integer or decimal number or hexa-decimal number or function name or variable name or special hacky things big parser	277
10.23	<code>\XINT_expr_getop</code> : finding the next operator or closing parenthesis or end of expression	284
10.24	Expansion spanning; opening and closing parentheses	285
10.25	<code> </code> , <code> </code> , <code>&</code> , <code>&&</code> , <code><</code> , <code>></code> , <code>=</code> , <code>==</code> , <code><=</code> , <code>>=</code> , <code>!=</code> , <code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>^</code> , <code>**</code> , <code>//</code> , <code>/:</code> , <code>...</code> , <code>..[</code> , <code>]</code> , <code>]</code> , <code>]</code> , <code>]</code> , <code>]</code> , and <code>++</code> operators	286
10.26	Macros for list selectors: <code>[list][N]</code> , <code>[list][:b]</code> , <code>[list][a:]</code> , <code>[list][a:b]</code>	293
10.27	Macros for a..b list generation	296
10.28	Macros for a..[d]..b list generation	298
10.29	The comma as binary operator	299
10.30	The minus as prefix operator of variable precedence level	300
10.31	<code>?</code> as two-way and <code>??</code> as three-way conditionals with braced branches	301
10.32	<code>!</code> as postfix factorial operator	301
10.33	The A/B[N] mechanism	302
10.34	<code>\XINT_expr_op_`</code> for recognizing functions	302
10.35	The bool, togl, protect pseudo “functions”	303
10.36	The break function	303
10.37	The qint, qfrac, qfloat “functions”	303
10.38	<code>\XINT_expr_op__</code> for recognizing variables	303
10.39	User defined variables: <code>\xintdefvar</code> , <code>\xintdefiivar</code> , <code>\xintdeffloatvar</code>	304
10.40	<code>\xintunassignvar</code>	305
10.41	seq and the implementation of dummy variables	305
10.42	add, mul	311
10.43	subs	312
10.44	rseq	312
10.45	iter	314

10.46	<code>rrseq</code>	316
10.47	<code>iterr</code>	318
10.48	Macros handling csv lists for functions with multiple comma separated arguments in expressions	320
10.49	The num, reduce, abs, sgn, frac, floor, ceil, sqr, sqrt, sqrtr, float, round, trunc, mod, quo, rem, gcd, lcm, max, min, <code>`+`</code> , <code>`*`</code> , <code>?</code> , <code>!</code> , not, all, any, xor, if, ifsgn, even, odd, first, last, len, reversed, factorial and binomial functions	323
10.50	f-expandable versions of the <code>\xintSeqB::csv</code> and alike routines, for <code>\xintNewExpr</code>	330
10.51	User defined functions: <code>\xintdeffunc</code> , <code>\xintdefiifunc</code> , <code>\xintdeffloatfunc</code>	332
10.52	<code>\xintNewFunction</code>	333
10.53	<code>\xintNewExpr</code> , <code>\xintNewIExpr</code> , <code>\xintNewFloatExpr</code> , <code>\xintNewIExpr</code>	334

This is release 1.2i of [2016/12/13].

10.1 Old comments

These general comments were last updated at the end of the 1.09x series in 2014. The principles remain in place to this day but refer to [CHANGES.html](#) for some significant evolutions since.

The first version was released in June 2013. I was greatly helped in this task of writing an expandable parser of infix operations by the comments provided in `l3fp-parse.dtx` (in its version as available in April-May 2013). One will recognize in particular the idea of the ``until`` macros; I have not looked into the actual `l3fp` code beyond the very useful comments provided in its documentation.

A main worry was that my data has no a priori bound on its size; to keep the code reasonably efficient, I experimented with a technique of storing and retrieving data expandably as *names* of control sequences. Intermediate computation results are stored as control sequences `\.=a/b[n]`.

Roughly speaking, the parser mechanism is as follows: at any given time the last found ``operator`` has its associated `until` macro awaiting some news from the token flow; first `getnext` expands forward in the hope to construct some number, which may come from a parenthesized sub-expression, from some braced material, or from a digit by digit scan. After this number has been formed the next operator is looked for by the `getop` macro. Once `getop` has finished its job, `until` is presented with three tokens: the first one is the precedence level of the new found operator (which may be an end of expression marker), the second is the operator character token (earlier versions had here already some macro name, but in order to keep as much common code to `expr` and `floatexpr` common as possible, this was modified) of the new found operator, and the third one is the newly found number (which was encountered just before the new operator).

The `until` macro of the earlier operator examines the precedence level of the new found one, and either executes the earlier operator (in the case of a binary operation, with the found number and a previously stored one) or it delays execution, giving the hand to the `until` macro of the operator having been found of higher precedence.

A minus sign acting as prefix gets converted into a (unary) operator inheriting the precedence level of the previous operator.

Once the end of the expression is found (it has to be marked by a `\relax`) the final result is output as four tokens (five tokens since 1.09j) the first one a catcode 11 exclamation mark, the second one an error generating macro, the third one is a protection mechanism, the fourth one a printing macro and the fifth is `\.=a/b[n]`. The prefix `\xintthe` makes the output printable by killing the first three tokens.

10.2 Catcodes, ε -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 \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%
```

10.3 Package identification

```
55 \XINT_providespackage
56 \ProvidesPackage{xintexpr}%
57 [2016/12/13 1.2i Expandable expression parser (JFB)]%
58 \catcode\! 11
```

10.4 Locking and unlocking

Some renaming and modifications here with release 1.2 to switch from using chains of `\romannumeral-`0` in order to gather numbers, possibly hexadecimal, to using a `\csname` governed expansion. In this way no more limit at 5000 digits, and besides this is a logical move because the `\xintexpr` parser is already based on `\csname...\endcsname` storage of numbers as one token.

The limitation at 5000 digits didn't worry me too much because it was not very realistic to launch computations with thousands of digits... such computations are still slow with 1.2 but less so now. Chains or `\romannumeral` are still used for the gathering of function names and other stuff which I have half-forgotten because the parser does many things.

In the earlier versions we used the `lockscan` macro after a chain of `\romannumeral-`0` had ended gathering digits; this uses has been replaced by direct processing inside a `\csname...\endcsname` and the macro is kept only for matters of dummy variables.

Currently, the parsing of hexadecimal numbers needs two nested `\csname...\endcsname`, first to gather the letters (possibly with a hexadecimal fractional part), and in a second stage to apply `\xintHexToDec` to do the actual conversion. This should be faster than updating on the fly the number (which would be hard for the fraction part...). The macro `\xintHexToDec` could probably be made faster by using techniques similar as the ones 1.2 uses in `xintcore.sty`.

```
59 \def\xint_gob_til_! #1!{% catcode 11 ! default in xintexpr.sty code.
60 \edef\XINT_expr_lockscan#1!% not used for decimal numbers in xintexpr 1.2
61   {noexpand\expandafter\space\noexpand\csname .=#1\endcsname }%
62 \edef\XINT_expr_lockit
63   #1{\noexpand\expandafter\space\noexpand\csname .=#1\endcsname }%
64 \def\XINT_expr_unlock_hex_in #1% expanded inside \csname..\endcsname
65   {\expandafter\XINT_expr_inhex\romannumeral`&&@\XINT_expr_unlock#1;}%
66 \def\XINT_expr_inhex #1.#2#3;% expanded inside \csname..\endcsname
67 {%
68   \if#2>\xintHexToDec{#1}%
69   \else
70     \xintiiMul{\xintiiPow{625}{\xintLength{#3}}}{\xintHexToDec{#1#3}}%
71     [\the\numexpr-4*\xintLength{#3}]%
72   \fi
73 }%
74 \def\XINT_expr_unlock {\expandafter\XINT_expr_unlock_a\string}%
75 \def\XINT_expr_unlock_a #1.={}%
76 \def\XINT_expr_unexpectedtoken {\xintError:ignored}%
77 \let\XINT_expr_done\space
```

10.5 `\XINT_expr_wrap`, `\XINT_iiexpr_wrap`

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

10.6 `\XINT_protectii`, `\XINT_expr_usethe`

```
80 \def\XINT_protectii #1{\noexpand\XINT_protectii\noexpand #1\noexpand }%
81 \protected\def\XINT_expr_usethe\XINT_protectii {\xintError:missing_xintthe!}%
```

10.7 \XINT_expr_print, \XINT_iexpr_print, \XINT_boolexpr_print

See also the \XINT_flexpr_print which is special, below.

```
82 \def\XINT_expr_print      #1{\xintSPraw::csv  {\XINT_expr_unlock #1}}%
83 \def\XINT_iexpr_print    #1{\xintCSV::csv    {\XINT_expr_unlock #1}}%
84 \def\XINT_boolexpr_print #1{\xintIsTrue::csv {\XINT_expr_unlock #1}}%
```

10.8 \xintexpr, \xintiexpr, \xintfloatexpr, \xintiiepr

```
85 \def\xintexpr      {\romannumeral0\xinteval      }%
86 \def\xintiexpr     {\romannumeral0\xintieval     }%
87 \def\xintfloatexpr {\romannumeral0\xintfloateval }%
88 \def\xintiiepr     {\romannumeral0\xintiieval    }%
```

10.9 \xinttheexpr, \xinttheiexpr, \xintthefloatexpr, \xinttheiiepr

```
89 \def\xinttheexpr
90   {\romannumeral`&&\expandafter\XINT_expr_print\romannumeral0\xintbareeval }%
91 \def\xinttheiexpr   {\romannumeral`&&\xintthe\xintiexpr }%
92 \def\xintthefloatexpr {\romannumeral`&&\xintthe\xintfloatexpr }%
93 \def\xinttheiiepr
94   {\romannumeral`&&\expandafter\XINT_iexpr_print\romannumeral0\xintbareiieval }%
```

10.10 \xintthe

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

10.11 \thexintexpr, \thexintiexpr, \thexintfloatexpr, \thexintiiepr

New with 1.2h. I have been three years long very strict in terms of prefixing macros, but well.

```
96 \let\thexintexpr      \xinttheexpr
97 \let\thexintiexpr     \xinttheiexpr
98 \let\thexintfloatexpr\xintthefloatexpr
99 \let\thexintiiepr     \xinttheiiepr
```

10.12 \xintthecoords

1.1 Wraps up an even number of comma separated items into pairs of TikZ coordinates; for use in the following way:

```
coordinates {\xintthecoords\xintfloatexpr ... \relax}
```

The crazyness with the \csname and unlock is due to TikZ somewhat STRANGE control of the TOTAL number of expansions which should not exceed the very low value of 100 !! As we implemented \XINT_thecoords_b in an "inline" style for efficiency, we need to hide its expansions.

Not to be used as \xintthecoords\xintthefloatexpr, only as \xintthecoords\xintfloatexpr (or \xintiexpr etc...). Perhaps \xintthecoords could make an extra check, but one should not accustom users to too loose requirements!

```
100 \def\xintthecoords #1{\romannumeral`&&\expandafter\expandafter\expandafter
101      \XINT_thecoords_a
102      \expandafter\xint_gobble_iii\romannumeral0#1}%
103 \def\XINT_thecoords_a #1#2% #1=print macro, indispensable for scientific notation
```



```

104   {\expandafter\XINT_expr_unlock\csname.=\expandafter\XINT_thecoords_b
105       \romannumeral`&&@#1#2,!,!,^%\endcsname }%
106 \def\XINT_thecoords_b #1#2,#3#4,%
107   {\xint_gob_til_! #3\XINT_thecoords_c ! (#1#2, #3#4)\XINT_thecoords_b }%
108 \def\XINT_thecoords_c #1^{}%

```

10.13 `\xintbareeval`, `\xintbarefloateval`, `\xintbareiieval`

```

109 \def\xintbareeval
110   {\expandafter\XINT_expr_until_end_a\romannumeral`&&\XINT_expr_getnext }%
111 \def\xintbarefloateval
112   {\expandafter\XINT_flexpr_until_end_a\romannumeral`&&\XINT_expr_getnext }%
113 \def\xintbareiieval
114   {\expandafter\XINT_iexpr_until_end_a\romannumeral`&&\XINT_expr_getnext }%

```

10.14 `\xintthebareeval`, `\xintthebarefloateval`, `\xintthebareiieval`

```

115 \def\xintthebareeval      {\expandafter\XINT_expr_unlock\romannumeral0\xintbareeval}%
116 \def\xintthebarefloateval {\expandafter\XINT_expr_unlock\romannumeral0\xintbarefloateval}%
117 \def\xintthebareiieval    {\expandafter\XINT_expr_unlock\romannumeral0\xintbareiieval}%

```

10.15 `\xinteval`, `\xintiieval`

```

118 \def\xinteval    {\expandafter\XINT_expr_wrap\romannumeral0\xintbareeval }%
119 \def\xintiieval  {\expandafter\XINT_iexpr_wrap\romannumeral0\xintbareiieval }%

```

10.16 `\xintieval`, `\XINT_iexpr_wrap`

Optional argument since 1.1.

```

120 \def\xintieval #1%
121   {\ifx [#1\expandafter\XINT_iexpr_withopt\else\expandafter\XINT_iexpr_noopt \fi #1}%
122 \def\XINT_iexpr_noopt
123   {\expandafter\XINT_iexpr_wrap \expandafter 0\romannumeral0\xintbareeval }%
124 \def\XINT_iexpr_withopt [#1]%
125 {%
126   \expandafter\XINT_iexpr_wrap\expandafter
127   {\the\numexpr \xint_zapspace #1 \xint_gobble_i\expandafter}%
128   \romannumeral0\xintbareeval
129 }%
130 \def\XINT_iexpr_wrap #1#2%
131 {%
132   \expandafter\XINT_expr_wrap
133   \csname .=\xintRound::csv {#1}{\XINT_expr_unlock #2}\endcsname
134 }%

```

10.17 `\xintfloateval`, `\XINT_flexpr_wrap`, `\XINT_flexpr_print`

Optional argument since 1.1

```

135 \def\xintfloateval #1%
136 {%
137   \ifx [#1\expandafter\XINT_flexpr_withopt_a\else\expandafter\XINT_flexpr_noopt
138   \fi #1%
139 }%

```

```

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

```

10.18 *\xintboolexpr*, *\xinttheboolexpr*, *\thexintboolexpr*

```

165 \def\xintboolexpr      {\romannumeral0\expandafter\expandafter\expandafter
166   \XINT_boolexpr_done \expandafter\xint_gobble_iv\romannumeral0\xinteval }%
167 \def\xinttheboolexpr   {\romannumeral`&&\expandafter\expandafter\expandafter
168   \XINT_boolexpr_print\expandafter\xint_gobble_iv\romannumeral0\xinteval }%
169 \let\thexintboolexpr\xinttheboolexpr
170 \def\XINT_boolexpr_done { !\XINT_expr_usethe\XINT_protectii\XINT_boolexpr_print }%

```

10.19 *\xintifboolexpr*, *\xintifboolfloatexpr*, *\xintifbooliiexpr*

Do not work with comma separated expressions.

```

171 \def\xintifboolexpr    #1{\romannumeral0\xintifnotzero {\xinttheexpr #1\relax}}%
172 \def\xintifboolfloatexpr #1{\romannumeral0\xintifnotzero {\xintthefloatexpr #1\relax}}%
173 \def\xintifbooliiexpr  #1{\romannumeral0\xintifnotzero {\xinttheiiexpr #1\relax}}%

```

10.20 Macros handling csv lists on output (for *\XINT_expr_print* et al. routines)

10.20.1	<i>\XINT::_end</i>	275
10.20.2	<i>\xintCSV::csv</i>	275
10.20.3	<i>\xintSPRaw, \xintSPRaw::csv</i>	275
10.20.4	<i>\xintIsTrue::csv</i>	275
10.20.5	<i>\xintRound::csv</i>	275
10.20.6	<i>\XINTinFloat::csv</i>	275
10.20.7	<i>\xintPFloat::csv</i>	276

Changed completely for 1.1, which adds the optional arguments to *\xintexpr* and *\xintfloatexpr*.

10.20.1 \XINT_::_end

Le mécanisme est le suivant, #2 est dans des accolades et commence par ,<sp>. Donc le gobble se débarrasse du, et le <sp> après brace stripping arrête un \romannumeral0 ou \romannumeral-`0

```
174 \def\XINT_::_end #1,#2{\xint_gobble_i #2}%
```

10.20.2 \xintCSV::csv

```
175 \def\xintCSV::csv #1{\expandafter\XINT_csv::_a\romannumeral`&&@#1,^}%
176 \def\XINT_csv::_a {\XINT_csv::_b {}}%
177 \def\XINT_csv::_b #1#2,{\expandafter\XINT_csv::_c \romannumeral`&&@#2,{#1}}%
178 \def\XINT_csv::_c #1{\if ^#1\expandafter\XINT_::_end\fi\XINT_csv::_d #1}%
179 \def\XINT_csv::_d #1,#2{\XINT_csv::_b {#2, #1}}% possibly, item #1 is empty.
```

10.20.3 \xintSPRaw, \xintSPRaw::csv

```
180 \def\xintSPRaw {\romannumeral0\xintspraw }%
181 \def\xintspraw #1{\expandafter\XINT_spraw\romannumeral`&&@#1[\W]}%
182 \def\XINT_spraw #1[#2#3]{\xint_gob_til_W #2\XINT_spraw_a\W\XINT_spraw_p #1[#2#3]}%
183 \def\XINT_spraw_a\W\XINT_spraw_p #1[\W]{ #1}%
184 \def\XINT_spraw_p #1[\W]{\xintpraw {#1}}%
185 \def\xintSPRaw::csv #1{\romannumeral0\expandafter\XINT_spraw::_a\romannumeral`&&@#1,^}%
186 \def\XINT_spraw::_a {\XINT_spraw::_b {}}%
187 \def\XINT_spraw::_b #1#2,{\expandafter\XINT_spraw::_c \romannumeral`&&@#2,{#1}}%
188 \def\XINT_spraw::_c #1{\if ,#1\xint_dothis\XINT_spraw::_e\fi
189     \if ^#1\xint_dothis\XINT_::_end\fi
190     \xint_orthat\XINT_spraw::_d #1}%
191 \def\XINT_spraw::_d #1,{\expandafter\XINT_spraw::_e\romannumeral0\XINT_spraw #1[\W],}%
192 \def\XINT_spraw::_e #1,#2{\XINT_spraw::_b {#2, #1}}%
```

10.20.4 \xintIsTrue::csv

```
193 \def\xintIsTrue::csv #1{\romannumeral0\expandafter\XINT_istrue::_a\romannumeral`&&@#1,^}%
194 \def\XINT_istrue::_a {\XINT_istrue::_b {}}%
195 \def\XINT_istrue::_b #1#2,{\expandafter\XINT_istrue::_c \romannumeral`&&@#2,{#1}}%
196 \def\XINT_istrue::_c #1{\if ,#1\xint_dothis\XINT_istrue::_e\fi
197     \if ^#1\xint_dothis\XINT_::_end\fi
198     \xint_orthat\XINT_istrue::_d #1}%
199 \def\XINT_istrue::_d #1,{\expandafter\XINT_istrue::_e\romannumeral0\xintisnotzero {#1},}%
200 \def\XINT_istrue::_e #1,#2{\XINT_istrue::_b {#2, #1}}%
```

10.20.5 \xintRound::csv

```
201 \def\XINT_:::_end #1,#2#3{\xint_gobble_i #3}%
202 \def\xintRound::csv #1#2{\romannumeral0\expandafter\XINT_round::_b\expandafter
203     {\the\numexpr#1\expandafter}\expandafter{\expandafter}\romannumeral`&&@#2,^}%
204 \def\XINT_round::_b #1#2#3,{\expandafter\XINT_round::_c \romannumeral`&&@#3,{#1}{#2}}%
205 \def\XINT_round::_c #1{\if ,#1\xint_dothis\XINT_round::_e\fi
206     \if ^#1\xint_dothis\XINT_:::_end\fi
207     \xint_orthat\XINT_round::_d #1}%
208 \def\XINT_round::_d #1,#2{%
209     \expandafter\XINT_round::_e\romannumeral0\ifnum#2>\xint_c_
210     \expandafter\xintround\else\expandafter\xintround\fi {#2}{#1},{#2}}%
211 \def\XINT_round::_e #1,#2#3{\XINT_round::_b {#2}{#3, #1}}%
```

10.20.6 \XINTinFloat::csv

```

212 \def\XINTinFloat::csv #1#2{\romannumeral0\expandafter\XINT_infloat::_b\expandafter
213   \the\numexpr #1\expandafter}\expandafter{\expandafter}\romannumeral`&&#2,^,%}
214 \def\XINT_infloat::_b #1#2#3,{\XINT_infloat::_c #3,{#1}{#2}}%
215 \def\XINT_infloat::_c #1{\if ,#1\xint_dothis\XINT_infloat::_e\fi
216   \if ^#1\xint_dothis\XINT_::_end\fi
217   \xint_orthat\XINT_infloat::_d #1}%
218 \def\XINT_infloat::_d #1,#2%
219   {\expandafter\XINT_infloat::_e\romannumeral0\XINTinfloat [#2]{#1},{#2}}%
220 \def\XINT_infloat::_e #1,#2#3{\XINT_infloat::_b {#2}{#3, #1}}%

```

10.20.7 \xintPFloat::csv

```

221 \def\xintPFloat::csv #1#2{\romannumeral0\expandafter\XINT_pfloat::_b\expandafter
222   \the\numexpr #1\expandafter}\expandafter{\expandafter}\romannumeral`&&@#2,^,%
223 \def\XINT_pfloat::_b #1#2#3,{\expandafter\XINT_pfloat::_c \romannumeral`&&@#3,{#1}{#2}}%
224 \def\XINT_pfloat::_c #1{\if ,#1\xint_dothis\XINT_pfloat::_e\fi
225   \if ^#1\xint_dothis\XINT_::_end\fi
226   \xint_orthat\XINT_pfloat::_d #1}%
227 \def\XINT_pfloat::_d #1,#2%
228   {\expandafter\XINT_pfloat::_e\romannumeral0\XINT_pfloat_opt [\xint_relax #2]{#1},{#2}}%
229 \def\XINT_pfloat::_e #1.#2#3{\XINT_pfloat::_b {#2}{#3, #1}}%

```

10.21 `\XINT_expr_getnext`: fetching some number then an operator

Big change in 1.1, no attempt to detect braced stuff anymore as the [N] notation is implemented otherwise. Now, braces should not be used at all; one level removed, then \romannumeral-`0 expansion.

```

230 \def\XINT_expr_getnext #1%
231 {%
232   \expandafter\XINT_expr_getnext_a\romannumeral`&&@#1%
233 }%
234 \def\XINT_expr_getnext_a #1%
235 {% screens out sub-expressions and \count or \dimen registers/variables
236   \xint_gob_til_! #1\XINT_expr_subexpr !% recall this ! has catcode 11
237   \ifcat\relax#1% \count or \numexpr etc... token or count, dimen, skip cs
238     \expandafter\XINT_expr_countetc
239   \else
240     \expandafter\expandafter\expandafter\XINT_expr_getnextfork\expandafter\string
241     \fi
242     #1%
243 }%
244 \def\XINT_expr_subexpr !#1\fi !{\expandafter\XINT_expr_getop\xint_gobble_iii }%

```

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

```

245 \def\XINT_expr_countetc #1%
246 {%
247     \ifx\count#1\else\ifx\dimen#1\else\ifx\numexpr#1\else\ifx\dimexpr#1\else
248     \ifx\skip#1\else\ifx\glueexpr#1\else\ifx\fontdimen#1\else\ifx\ht#1\else
249     \ifx\dp#1\else\ifx\wd#1\else\ifx\fontcharht#1\else\ifx\fontcharwd#1\else
250     \ifx\fontchardp#1\else\ifx\fontcharic#1\else
251         \XINT_expr_unpackvar
252     \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
253     \expandafter\XINT_expr_getnext\number #1%

```

```

254 }%
255 \def\XINT_expr_unpackvar\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
256   \expandafter\XINT_expr_getnext\number #1%
257   {\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
258   \expandafter\XINT_expr_getop\csname .=\number#1\endcsname }%
259 \begingroup
260 \lccode`*=`#
261 \lowercase{\endgroup
262 \def\XINT_expr_getnextfork #1{%
263   \if#1*\xint_dothis {\XINT_expr_scan_macropar *}\fi
264   \if#1[\xint_dothis {\xint_c_xviii ({} }\fi
265   \if#1+\xint_dothis \XINT_expr_getnext \fi
266   \if#1.\xint_dothis {\XINT_expr_startdec}\fi
267   \if#1-\xint_dothis -\fi
268   \if#1(\xint_dothis {\xint_c_xviii ({} }\fi
269   \xint_orthat {\XINT_expr_scan_nbr_or_func #1}%
270 }}%
271 \def\XINT_expr_scan_macropar #1#2{\expandafter\XINT_expr_getop\csname .=#1#2\endcsname }%

```

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

10.22.1	Integral part (skipping zeroes)	278
10.22.2	Fractional part	279
10.22.3	Scientific notation	280
10.22.4	Hexadecimal numbers	281
10.22.5	Parsing names of functions and variables	282

1.2 release has replaced chains of `\romannumeral-`0` by `\csname` governed expansion. Thus there is no more the limit at about 5000 digits for parsed numbers.

In order to avoid having to lock and unlock in succession to handle the scientific part and adjust the exponent according to the number of digits of the decimal part, the parsing of this decimal part counts on the fly the number of digits it encounters.

There is some slight annoyance with `\xintiexpr` which should never be given a `[n]` inside its `\csname.<digits>\endcsname` storage of numbers (because its arithmetic uses the `ii` macros which know nothing about the `[N]` notation). Hence if the parser has only seen digits when hitting something else than the dot or `e` (or `E`), it will not insert a `[0]`. Thus we very slightly compromise the efficiency of `\xintexpr` and `\xintfloatexpr` in order to be able to share the same code with `\xintiexpr`.

Indeed, the parser at this location is completely common to all, it does not know if it is working inside `\xintexpr` or `\xintiexpr`. On the other hand if a dot or a `e` (or `E`) is met, then the (common) parser has no scruples ending this number with a `[n]`, this will provoke an error later if that was within an `\xintiexpr`, as soon as an arithmetic macro is used.

As the gathered numbers have no spaces, no pluses, no minuses, the only remaining issue is with leading zeroes, which are discarded on the fly. The hexadecimal numbers leading zeroes are stripped in a second stage by the `\xintHexToDec` macro.

With 1.2, `\xinttheexpr . \relax` does not work anymore (it did in earlier releases). There must be digits either before or after the decimal mark. Thus both `\xinttheexpr 1.\relax` and `\xinttheexpr .1\relax` are legal.

The ``` syntax is here used for special constructs like ``+`(..)`, ``*`(..)` where `+` or `*` will be treated as functions. Current implementation pick only one token (could have been braced stuff), thus here it will be `+` or `*`, and via `\XINT_expr_op_`` this into becomes a suitable

`\XINT_{expr|iiexpr|fexpr}_func_+` (or `*`). Documentation of 1.1 said to use ``+`(...)`, but ``+`(...)` is also valid. The opening parenthesis must be there, it is not allowed to come from expansion.

```

272 \catcode96 11 % `
273 \def\XINT_expr_scan_nbr_or_func #1% this #1 has necessarily here catcode 12
274 {%
275   \if "#1\xint_dothis \XINT_expr_scanhex_I\fi
276   \if `#1\xint_dothis {\XINT_expr_onlitteral_}\fi
277   \ifnum \xint_c_ix<1#1 \xint_dothis \XINT_expr_startint\fi
278   \xint_orthat \XINT_expr_scanfunc #1%
279 }%
280 \def\XINT_expr_onlitteral_` #1#2#3({\xint_c_xviii `{#2}}%
281 \catcode96 12 % `
282 \def\XINT_expr_startint #1%
283 {%
284   \if #10\expandafter\XINT_expr_gobz_a\else\XINT_expr_scanint_a\fi #1%
285 }%
286 \def\XINT_expr_scanint_a #1#2%
287   {\expandafter\XINT_expr_getop\csname.=#1%
288     \expandafter\XINT_expr_scanint_b\romannumeral`&&@#2}%
289 \def\XINT_expr_gobz_a #1%
290   {\expandafter\XINT_expr_getop\csname.=#1%
291     \expandafter\XINT_expr_gobz_scanint_b\romannumeral`&&@#1}%
292 \def\XINT_expr_startdec #1%
293   {\expandafter\XINT_expr_getop\csname.=#1%
294     \expandafter\XINT_expr_scandec_a\romannumeral`&&@#1}%

```

10.22.1 Integral part (skipping zeroes)

1.2 has modified the code to give highest priority to digits, the accelerating impact is non-negligeable. I don't think the doubled `\string` is a serious penalty.

```

295 \def\XINT_expr_scanint_b #1%
296 {%
297   \ifcat \relax #1\expandafter\XINT_expr_scanint_endbycs\expandafter #1\fi
298   \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanint_c\fi
299   \string#1\XINT_expr_scanint_d
300 }%
301 \def\XINT_expr_scanint_d #1%
302 {%
303   \expandafter\XINT_expr_scanint_b\romannumeral`&&@#1%
304 }%
305 \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.

```

306 \def\XINT_expr_scanint_c\string #1\XINT_expr_scanint_d
307 {%
308   \if e#1\xint_dothis{[\the\numexpr0\XINT_expr_scanexp_a +}\fi
309   \if E#1\xint_dothis{[\the\numexpr0\XINT_expr_scanexp_a +}\fi
310   \if .#1\xint_dothis{\XINT_expr_startdec_a .}\fi
311   \xint_orthat {\endcsname #1}%
312 }%
313 \def\XINT_expr_startdec_a .#1%
314 {%
315   \expandafter\XINT_expr_scandec_a\romannumeral`&&@#1%
316 }%
317 \def\XINT_expr_scandec_a #1%
318 {%
319   \if .#1\xint_dothis{\endcsname..}\fi
320   \xint_orthat {\XINT_expr_scandec_b 0.#1}%
321 }%
322 \def\XINT_expr_gobz_scanint_b #1%
323 {%
324   \ifcat \relax #1\expandafter\XINT_expr_gobz_scanint_endbycs\expandafter #1\fi
325   \ifnum\xint_c_x<1\string#1 \else\expandafter\XINT_expr_gobz_scanint_c\fi
326   \string#1\XINT_expr_scanint_d
327 }%
328 \def\XINT_expr_gobz_scanint_endbycs#1#2\XINT_expr_scanint_d{0\endcsname #1}%
329 \def\XINT_expr_gobz_scanint_c\string #1\XINT_expr_scanint_d
330 {%
331   \if e#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
332   \if E#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
333   \if .#1\xint_dothis{\XINT_expr_gobz_startdec_a .}\fi
334   \if 0#1\xint_dothis\XINT_expr_gobz_scanint_d\fi
335   \xint_orthat {0\endcsname #1}%
336 }%
337 \def\XINT_expr_gobz_scanint_d #1%
338 {%
339   \expandafter\XINT_expr_gobz_scanint_b\romannumeral`&&@#1%
340 }%
341 \def\XINT_expr_gobz_startdec_a .#1%
342 {%
343   \expandafter\XINT_expr_gobz_scandec_a\romannumeral`&&@#1%
344 }%
345 \def\XINT_expr_gobz_scandec_a #1%
346 {%
347   \if .#1\xint_dothis{0\endcsname..}\fi
348   \xint_orthat {\XINT_expr_gobz_scandec_b 0.#1}%
349 }%

```

10.22.2 Fractional part

Annoying duplication of code to allow 0. as input.

1.2a corrects a very bad bug in 1.2 \XINT_expr_gobz_scandec_b which should have stripped leading zeroes in the fractional part but didn't; as a result \xinttheexpr 0.01\relax returned 0 =:-(((Thanks to Kroum Tzanev who reported the issue. Does it improve things if I say the bug was introduced in 1.2, it wasn't present before ?

```

350 \def\XINT_expr_scandec_b #1.#2%
351 {%
352   \ifcat \relax #2\expandafter\XINT_expr_scandec_endbycs\expandafter#2\fi
353   \ifnum\xint_c_ix<1\string#2 \else\expandafter\XINT_expr_scandec_c\fi
354   \string#2\expandafter\XINT_expr_scandec_d\the\numexpr #1-\xint_c_i.%
355 }%
356 \def\XINT_expr_scandec_endbycs #1#2\XINT_expr_scandec_d
357   \the\numexpr#3-\xint_c_i.{[#3]\endcsname #1}%
358 \def\XINT_expr_scandec_d #1.#2%
359 {%
360   \expandafter\XINT_expr_scandec_b
361   \the\numexpr #1\expandafter.\romannumeral`&&@#2%
362 }%
363 \def\XINT_expr_scandec_c\string #1#2\the\numexpr#3-\xint_c_i.%
364 {%
365   \if e#1\xint_dothis{[\the\numexpr#3\XINT_expr_scanexp_a +}\fi
366   \if E#1\xint_dothis{[\the\numexpr#3\XINT_expr_scanexp_a +}\fi
367   \xint_orthat {[#3]\endcsname #1}%
368 }%
369 \def\XINT_expr_gobz_scandec_b #1.#2%
370 {%
371   \ifcat \relax #2\expandafter\XINT_expr_gobz_scandec_endbycs\expandafter#2\fi
372   \ifnum\xint_c_ix<1\string#2 \else\expandafter\XINT_expr_gobz_scandec_c\fi
373   \if0#2\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
374   {\expandafter\XINT_expr_gobz_scandec_b}%
375   {\string#2\expandafter\XINT_expr_scandec_d}\the\numexpr#1-\xint_c_i.%
376 }%
377 \def\XINT_expr_gobz_scandec_endbycs #1#2\xint_c_i.{0[0]\endcsname #1}%
378 \def\XINT_expr_gobz_scandec_c\if0#1#2\fi #3\xint_c_i.%
379 {%
380   \if e#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
381   \if E#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
382   \xint_orthat {0[0]\endcsname #1}%
383 }%

```

10.22.3 Scientific notation

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

```

384 \def\XINT_expr_scanexp_a #1#2%
385 {%
386   #1\expandafter\XINT_expr_scanexp_b\romannumeral`&&@#2%
387 }%
388 \def\XINT_expr_scanexp_b #1%
389 {%
390   \ifcat \relax #1\expandafter\XINT_expr_scanexp_endbycs\expandafter #1\fi
391   \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanexp_c\fi
392   \string#1\XINT_expr_scanexp_d
393 }%
394 \def\XINT_expr_scanexp_endbycs#1#2\XINT_expr_scanexp_d []\endcsname #1}%
395 \def\XINT_expr_scanexp_d #1%
396 {%

```



```

397 \expandafter\XINT_expr_scanexp_bb\romannumeral`&&@#1%
398 }%
399 \def\XINT_expr_scanexp_c\string #1\XINT_expr_scanexp_d
400 {%
401 \if +#1\xint_dothis {\XINT_expr_scanexp_a +}\fi
402 \if -#1\xint_dothis {\XINT_expr_scanexp_a -}\fi
403 \xint_orthat {}]\endcsname #1}%
404 }%
405 \def\XINT_expr_scanexp_bb #1%
406 {%
407 \ifcat \relax #1\expandafter\XINT_expr_scanexp_endbycs_b\expandafter #1\fi
408 \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanexp_cb\fi
409 \string#1\XINT_expr_scanexp_db
410 }%
411 \def\XINT_expr_scanexp_endbycs_b#1#2\XINT_expr_scanexp_db {}]\endcsname #1}%
412 \def\XINT_expr_scanexp_db #1%
413 {%
414 \expandafter\XINT_expr_scanexp_bb\romannumeral`&&@#1%
415 }%
416 \def\XINT_expr_scanexp_cb\string #1\XINT_expr_scanexp_db {}]\endcsname #1}%

```

10.22.4 Hexadecimal numbers

1.2d has moved most of the handling of tacit multiplication to `\XINT_expr_getop`, but we have to do some of it here, because we apply `\string` before calling `\XINT_expr_scanhexI_aa`. I do not insert the `*` in `\XINT_expr_scanhexI_a`, because it is its higher precedence variant which will 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`).

```

417 \def\XINT_expr_scanhex_I #1% #1="
418 {%
419 \expandafter\XINT_expr_getop\csname.=\expandafter
420 \XINT_expr_unlock_hex_in\csname.=\XINT_expr_scanhexI_a
421 }%
422 \def\XINT_expr_scanhexI_a #1%
423 {%
424 \ifcat #1\relax\xint_dothis{.>\endcsname\endcsname #1}\fi
425 \ifx !#1\xint_dothis{.>\endcsname\endcsname !}\fi
426 \xint_orthat {\expandafter\XINT_expr_scanhexI_aa\string #1}%
427 }%
428 \def\XINT_expr_scanhexI_aa #1%
429 {%
430 \if\ifnum`#1>`/
431 \ifnum`#1>`9
432 \ifnum`#1>`@
433 \ifnum`#1>`F
434 0\else1\fi\else0\fi\else1\fi\else0\fi 1%
435 \expandafter\XINT_expr_scanhexI_b
436 \else
437 \if .#1%
438 \expandafter\xint_firstoftwo
439 \else % gather what we got so far, leave catcode 12 #1 in stream
440 \expandafter\xint_secondoftwo

```

```

441     \fi
442     {\expandafter\XINT_expr_scanhex_transition}%
443     {\xint_afterfi {.>\endcsname\endcsname}}}%
444     \fi
445     #1%
446 }%
447 \def\XINT_expr_scanhexI_b #1#2%
448 {%
449     #1\expandafter\XINT_expr_scanhexI_a\romannumeral`&&@#2%
450 }%
451 \def\XINT_expr_scanhex_transition .#1%
452 {%
453     \expandafter.\expandafter.\expandafter
454     \XINT_expr_scanhexII_a\romannumeral`&&@#1%
455 }%
456 \def\XINT_expr_scanhexII_a #1%
457 {%
458     \ifcat #1\relax\xint_dothis{\endcsname\endcsname#1}\fi
459     \ifx !#1\xint_dothis{\endcsname\endcsname !}\fi
460     \xint_orthat {\expandafter\XINT_expr_scanhexII_aa\string #1}%
461 }%
462 \def\XINT_expr_scanhexII_aa #1%
463 {%
464     \if\ifnum`#1>`/
465         \ifnum`#1>`9
466             \ifnum`#1>`@
467                 \ifnum`#1>`F
468                     0\else1\fi\else0\fi\else1\fi\else0\fi 1%
469                 \expandafter\XINT_expr_scanhexII_b
470             \else
471                 \xint_afterfi {\endcsname\endcsname}%
472             \fi
473             #1%
474 }%
475 \def\XINT_expr_scanhexII_b #1#2%
476 {%
477     #1\expandafter\XINT_expr_scanhexII_a\romannumeral`&&@#2%
478 }%

```

10.22.5 Parsing names of functions and variables

```

479 \def\XINT_expr_scanfunc
480 {%
481     \expandafter\XINT_expr_func\romannumeral`&&@\XINT_expr_scanfunc_a
482 }%
483 \def\XINT_expr_scanfunc_a #1#2%
484 {%
485     \expandafter #1\romannumeral`&&@\expandafter\XINT_expr_scanfunc_b\romannumeral`&&@#2%
486 }%

```

This handles: 1) (indirectly) tacit multiplication by a variable in front a of sub-expression, 2) (indirectly) tacit multiplication in front of a \count etc..., 3) functions which are recognized via an encountered opening parenthesis (but later this must be disambiguated from variables

with tacit multiplication) 4) 5) 6) 7) acceptable components of a variable or function names: @, underscore, digits, letters (or chars of category code letter.)

The short lived 1.2d which followed the even shorter lived 1.2c managed to introduce a bug here as it removed the check for catcode 11 !, which must be recognized if ! is not to be taken as part of a variable name. Don't know what I was thinking, it was the time when I was moving the handling of tacit multiplication entirely to the \XINT_expr_getop side. Fixed in 1.2e.

I almost decided to remove the \ifcat\relax test whose rôle is to avoid the \string#1 to do something bad is the escape char is a digit! Perhaps I will remove it at some point ! I truly almost did it, but also the case of no escape char is a problem (\string\0, if \0 is a count ...)

The (indirectly) above means that via \XINT_expr_func then \XINT_expr_op__ one goes back to \XINT_expr_getop then \XINT_expr_getop_b which is the location where tacit multiplication is now centralized. This makes the treatment of tacit multiplication for situations such as <variable>\count or <variable>\xintexpr..\relax, perhaps a bit sub-optimal, but first the variable name must be gathered, second the variable must expand to its value.

```

487 \def\XINT_expr_scanfunc_b #1%
488 {%
489   \ifx !#1\xint_dothis{(_)\fi
490   \ifcat \relax#1\xint_dothis{(_)\fi
491   \if (#1\xint_dothis{\xint_firstoftwo{(`)}\fi
492   \if @#1\xint_dothis \XINT_expr_scanfunc_a \fi
493   \if _#1\xint_dothis \XINT_expr_scanfunc_a \fi
494   \ifnum \xint_c_ix<1\string#1 \xint_dothis \XINT_expr_scanfunc_a \fi
495   \ifcat a#1\xint_dothis \XINT_expr_scanfunc_a \fi
496   \xint_orthat {(_}%
497   #1%
498 }%

```

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|floatexpr}_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.

```

499 \def\XINT_expr_func #1(#2{\if _#2\xint_dothis\XINT_expr_op__\fi
500   \xint_orthat{\xint_c_xviii #2}{#1}}%

```

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

Release 1.1 implements multi-character operators.

1.2d adds tacit 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.

```

501 \def\XINT_expr_getop #1#2% this #1 is the current locked computed value
502 {%
503   \expandafter\XINT_expr_getop_a\expandafter #1\romannumeral`&&@#2%
504 }%
505 \catcode`* 11
506 \def\XINT_expr_getop_a #1#2%
507 {%
508   \ifx   \relax #2\xint_dothis\xint_firstofthree\fi
509   \ifcat \relax #2\xint_dothis\xint_secondofthree\fi
510   \if   _#2\xint_dothis      \xint_secondofthree\fi
511   \if   @#2\xint_dothis      \xint_secondofthree\fi
512   \if   (#2\xint_dothis      \xint_secondofthree\fi
513   \ifcat a#2\xint_dothis      \xint_secondofthree\fi
514   \xint_orthat \xint_thirdofthree
515   {\XINT_expr_foundend #1}%
516   {\XINT_expr_precedence_*** **#1#2}% tacit multiplication with higher precedence
517   {\expandafter\XINT_expr_getop_b \string#2#1}%
518 }%
519 \catcode`* 12
520 \def\XINT_expr_foundend {\xint_c_ \relax }% \relax is a place holder here.

```

? is a very special operator with top precedence which will check if the next token is another ?, while avoiding removing a brace pair from token stream due to its syntax. Pre 1.1 releases used : rather than ??, but we need : for Python like slices of lists.

```

521 \def\XINT_expr_getop_b #1%
522 {%
523   \if '#1\xint_dothis{\XINT_expr_binopwrd }\fi
524   \if ?#1\xint_dothis{\XINT_expr_precedence_? ?}\fi
525   \xint_orthat      {\XINT_expr_scanop_a #1}%
526 }%
527 \def\XINT_expr_binopwrd #1#2'\{\expandafter\XINT_expr_foundop_a
528   \csname XINT_expr_itself\xint_zapspace #2 \xint_gobble_i\endcsname #1}%
529 \def\XINT_expr_scanop_a #1#2#3%
530   {\expandafter\XINT_expr_scanop_b\expandafter #1\expandafter #2\romannumeral`&&@#3}%
531 \def\XINT_expr_scanop_b #1#2#3%

```

```

532 {%
533   \ifcat#3\relax\xint_dothis{\XINT_expr_foundop_a #1#2#3}\fi
534   \ifcsname XINT_expr_itself_#1#3\endcsname
535   \xint_dothis
536       {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#3\endcsname #2}\fi
537   \xint_orthat {\XINT_expr_foundop_a #1#2#3}%
538 }%
539 \def\XINT_expr_scanop_c #1#2#3%
540 {%
541   \expandafter\XINT_expr_scanop_d\expandafter #1\expandafter #2\romannumeral`&&@#3%
542 }%
543 \def\XINT_expr_scanop_d #1#2#3%
544 {%
545   \ifcat#3\relax \xint_dothis{\XINT_expr_foundop #1#2#3}\fi
546   \ifcsname XINT_expr_itself_#1#3\endcsname
547   \xint_dothis
548       {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#3\endcsname #2}\fi
549   \xint_orthat {\csname XINT_expr_precedence_#1\endcsname #1#2#3}%
550 }%
551 \def\XINT_expr_foundop_a #1%
552 {%
553   \ifcsname XINT_expr_precedence_#1\endcsname
554       \csname XINT_expr_precedence_#1\endcsname\expandafter\endcsname
555       \expandafter #1%
556   \else
557       \xint_afterfi{\XINT_expr_unknown_operator {#1}\XINT_expr_getop}%
558   \fi
559 }%
560 \def\XINT_expr_unknown_operator #1{\xintError:removed \xint_gobble_i {#1}}%
561 \def\XINT_expr_foundop #1{\csname XINT_expr_precedence_#1\endcsname #1}%

```

10.24 Expansion spanning; opening and closing parentheses

Version 1.1 had a hack inside the until macros for handling the omit and abort in iterations over dummy variables. This has been removed by 1.2c, see the subsection where omit and abort are discussed.

```

562 \catcode`) 11
563 \def\XINT_tmpa #1#2#3#4%
564 {%
565   \def#1##1%
566   {%
567     \xint_UDsignfork
568         ##1{\expandafter#1\romannumeral`&&@#3}%
569         -{#2##1}%
570     \krof
571   }%
572   \def#2##1##2%
573   {%
574     \ifcase ##1\expandafter\XINT_expr_done
575     \or\xint_afterfi{\XINT_expr_extra_)
576         \expandafter #1\romannumeral`&&@\XINT_expr_getop }%
577     \else

```

```

578      \xint_afterfi{\expandafter#1\romannumeral`&&\csname XINT_#4_op_##2\endcsname }%
579      \fi
580  }%
581 }%
582 \def\xintexpr_extra_ {\xintError:removed }%
583 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
584   \expandafter\xint_tmpa
585   \csname XINT_#1_until_end_a\expandafter\endcsname
586   \csname XINT_#1_until_end_b\expandafter\endcsname
587   \csname XINT_#1_op_-vi\endcsname
588   {#1}%
589 }%
590 \def\xint_tmpa #1#2#3#4#5#6%
591 {%
592   \def #1##1{\expandafter #3\romannumeral`&&\xintexpr_getnext }%
593   \def #2{\expandafter #3\romannumeral`&&\xintexpr_getnext }%
594   \def #3##1{\xint_UDsignfork
595     ##1{\expandafter #3\romannumeral`&&#5}%
596     -{#4##1}%
597     \kroft }%
598   \def #4##1##2{\ifcase ##1\expandafter\xintexpr_missing_
599     \or \csname XINT_#6_op_##2\expandafter\endcsname
600     \else
601       \xint_afterfi{\expandafter #3\romannumeral`&&\csname XINT_#6_op_##2\endcsname }%
602       \fi
603   }%
604 }%
605 \def\xintexpr_missing_ {\xintError:inserted \xint_c_ \xintexpr_done }%

```

We should be using `until_` notation to stay synchronous with `until_+`, `until_*` etc..., but I found that `until_` was more telling.

```

606 \catcode` ) 12
607 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
608   \expandafter\xint_tmpa
609   \csname XINT_#1_op_(\expandafter\endcsname
610   \csname XINT_#1_oparen\expandafter\endcsname
611   \csname XINT_#1_until_)_a\expandafter\endcsname
612   \csname XINT_#1_until_)_b\expandafter\endcsname
613   \csname XINT_#1_op_-vi\endcsname
614   {#1}%
615 }%
616 \expandafter\let\csname XINTexpr_precedence_\endcsname\xint_c_i

```

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

- 10.25.1 Square brackets for lists, the !? for omit and abort, and the ++ postfix construct 287
- 10.25.2 The |, &, xor, <, >, =, <=, >=, !=, //, /:, .., +, -, *, /, ^, ..[, and].. operators for expr, floatexpr and iiexpr operators 288
- 10.25.3 The]+,]-,]*,]/,]^,]+,]-,]*,]/, and]^ list operators 290
- 10.25.4 The 'and', 'or', 'xor', and 'mod' as infix operator words 292
- 10.25.5 The ||, &&, **, **[,]** operators as synonyms 292

10.25.1 Square brackets for lists, the !? for omit and abort, and the ++ postfix construct

This is all very clever and only need setting some suitable precedence levels, if only I could understand what I did in 2014... just joking. Notice that `op_)` macros are defined here in the `\xintFor` loop.

There is some clever business going on here with the letter `a` for handling constructs such as `[3..5]*2` (I think...).

1.2c has replaced 1.1's private dealings with "`^C`" (which was done before dummy variables got implemented) by use of "`!?`". See discussion of omit and abort.

```
617 \expandafter\let\csname XINT_expr_precedence_]\endcsname\xint_c_i
618 \expandafter\let\csname XINT_expr_precedence_;\endcsname\xint_c_i
619 \let\XINT_expr_precedence_a \xint_c_xviii
620 \let\XINT_expr_precedence_!? \xint_c_ii
621 \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.)

```
622 \catcode`. 11 \catcode`= 11 \catcode`+ 11
623 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
624   \expandafter\let\csname XINT_#1_op_)\endcsname \XINT_expr_getop
625   \expandafter\let\csname XINT_#1_op_;\endcsname \space
626   \expandafter\def\csname XINT_#1_op_]\endcsname ##1{\XINT_expr_scanop_a a##1}%
627   \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.

```
628   \expandafter\def\csname XINT_#1_op_++)\endcsname ##1##2\relax
629   {\expandafter\XINT_expr_foundedend \expandafter
630     {\expandafter\.=+\csname .=\xintiCeil{\XINT_expr_unlock ##1}\endcsname }}%
631 }%
632 \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.

```

633 \catcode`& 12
634 \xintFor* #1 in {{==}{<=}{>=}{!=}{&&}{||}{**}{//}{/:}{..}{...}{.}{.}{.}}%
635      {+[-]{*[/]{**[^]{a+}{a-}{a*}{a/}{a**}{a^}%
636      {][}{[:]{:}{!}{?}{++}{++}}}%{***}}
637   \do {\expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}}%
638 \catcode`& 7
639 \expandafter\let\csname XINT_expr_precedence_***\endcsname \xint_c_viii

```

10.25.2 The `|`, `&`, `xor`, `<`, `>`, `=`, `<=`, `>=`, `!=`, `//`, `/:`, `..`, `+`, `-`, `*`, `/`, `^`, `..[`, and `]`.. operators for `expr`, `floatexpr` and `iiexpr` operators

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

```

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

```



```

677 \XINT_expr_defbin_b {expr} {xor}{iii}{vi} {xintXOR}%
678 \XINT_expr_defbin_b {flexpr}{xor}{iii}{vi} {xintXOR}%
679 \XINT_expr_defbin_b {iiexpr}{xor}{iii}{vi} {xintXOR}%
680 \XINT_expr_defbin_b {expr} < {v}{vi} {xintLt}%
681 \XINT_expr_defbin_b {flexpr} < {v}{vi} {xintLt}%
682 \XINT_expr_defbin_b {iiexpr} < {v}{vi} {xintiiLt}%
683 \XINT_expr_defbin_b {expr} > {v}{vi} {xintGt}%
684 \XINT_expr_defbin_b {flexpr} > {v}{vi} {xintGt}%
685 \XINT_expr_defbin_b {iiexpr} > {v}{vi} {xintiiGt}%
686 \XINT_expr_defbin_b {expr} = {v}{vi} {xintEq}%
687 \XINT_expr_defbin_b {flexpr} = {v}{vi} {xintEq}%
688 \XINT_expr_defbin_b {iiexpr} = {v}{vi} {xintiiEq}%
689 \XINT_expr_defbin_b {expr} {<=} {v}{vi} {xintLtorEq}%
690 \XINT_expr_defbin_b {flexpr}{<=} {v}{vi} {xintLtorEq}%
691 \XINT_expr_defbin_b {iiexpr}{<=} {v}{vi} {xintiiLtorEq}%
692 \XINT_expr_defbin_b {expr} {>=} {v}{vi} {xintGtorEq}%
693 \XINT_expr_defbin_b {flexpr}{>=} {v}{vi} {xintGtorEq}%
694 \XINT_expr_defbin_b {iiexpr}{>=} {v}{vi} {xintiiGtorEq}%
695 \XINT_expr_defbin_b {expr} {!=} {v}{vi} {xintNeq}%
696 \XINT_expr_defbin_b {flexpr}{!=} {v}{vi} {xintNeq}%
697 \XINT_expr_defbin_b {iiexpr}{!=} {v}{vi} {xintiiNeq}%
698 \XINT_expr_defbin_b {expr} {...} {iii}{vi} {xintSeq::csv}%
699 \XINT_expr_defbin_b {flexpr}{...} {iii}{vi} {xintSeq::csv}%
700 \XINT_expr_defbin_b {iiexpr}{...} {iii}{vi} {xintiiSeq::csv}%
701 \XINT_expr_defbin_b {expr} {/**} {vii}{vii}{xintDivTrunc}%
702 \XINT_expr_defbin_b {flexpr}{/**} {vii}{vii}{xintDivTrunc}%
703 \XINT_expr_defbin_b {iiexpr}{/**} {vii}{vii}{xintiiDivTrunc}%
704 \XINT_expr_defbin_b {expr} {/:} {vii}{vii}{xintMod}%
705 \XINT_expr_defbin_b {flexpr}{/:} {vii}{vii}{xintMod}%
706 \XINT_expr_defbin_b {iiexpr}{/:} {vii}{vii}{xintiiMod}%
707 \XINT_expr_defbin_b {expr} + {vi}{vi} {xintAdd}%
708 \XINT_expr_defbin_b {flexpr} + {vi}{vi} {XINTinFloatAdd}%
709 \XINT_expr_defbin_b {iiexpr} + {vi}{vi} {xintiiAdd}%
710 \XINT_expr_defbin_b {expr} - {vi}{vi} {xintSub}%
711 \XINT_expr_defbin_b {flexpr} - {vi}{vi} {XINTinFloatSub}%
712 \XINT_expr_defbin_b {iiexpr} - {vi}{vi} {xintiiSub}%
713 \XINT_expr_defbin_b {expr} * {vii}{vii}{xintMul}%
714 \XINT_expr_defbin_b {flexpr} * {vii}{vii}{XINTinFloatMul}%
715 \XINT_expr_defbin_b {iiexpr} * {vii}{vii}{xintiiMul}%
716 \XINT_expr_defbin_b {expr} / {vii}{vii}{xintDiv}%
717 \XINT_expr_defbin_b {flexpr} / {vii}{vii}{XINTinFloatDiv}%
718 \XINT_expr_defbin_b {iiexpr} / {vii}{vii}{xintiiDivRound}% CHANGED IN 1.1!
719 \XINT_expr_defbin_b {expr} ^ {ix}{ix} {xintPow}%
720 \XINT_expr_defbin_b {flexpr} ^ {ix}{ix} {XINTinFloatPowerH}%
721 \XINT_expr_defbin_b {iiexpr} ^ {ix}{ix} {xintiiPow}%
722 \XINT_expr_defbin_b {expr} {...[]} {iii}{vi} {xintSeqA::csv}%
723 \XINT_expr_defbin_b {flexpr}{...[]} {iii}{vi} {XINTinFloatSeqA::csv}%
724 \XINT_expr_defbin_b {iiexpr}{...[]} {iii}{vi} {xintiiSeqA::csv}%
725 \XINT_expr_defbin_b {expr} {...} {iii}{vi} {xintSeqB::csv}%
726 \XINT_expr_defbin_b {flexpr}{...} {iii}{vi} {XINTinFloatSeqB::csv}%
727 \XINT_expr_defbin_b {iiexpr}{...} {iii}{vi} {xintiiSeqB::csv}%

```

10.25.3 The `+`, `-`, `*`, `/`, `^`, `+`, `-`, `*`, `/`, and `^` list operators

`\XINT_expr_binop_inline_b` This handles acting on comma separated values (no need to bother about spaces in this context; expansion in a `\csname...\endcsname`).

```

728 \def\XINT_expr_binop_inline_a
729   {\expandafter\xint_gobble_i\romannumeral`&&\XINT_expr_binop_inline_b}%
730 \def\XINT_expr_binop_inline_b #1#2,{\XINT_expr_binop_inline_c #2,{#1}}%
731 \def\XINT_expr_binop_inline_c #1{%
732   \if ,#1\xint_dothis\XINT_expr_binop_inline_e\fi
733   \if ^#1\xint_dothis\XINT_expr_binop_inline_end\fi
734   \xint_orthat\XINT_expr_binop_inline_d #1}%
735 \def\XINT_expr_binop_inline_d #1,#2{,#2{#1}\XINT_expr_binop_inline_b {#2}}%
736 \def\XINT_expr_binop_inline_e #1,#2{,\XINT_expr_binop_inline_b {#2}}%
737 \def\XINT_expr_binop_inline_end #1,#2{}%
738 \def\XINT_expr_deflistopr_c #1#2#3#4#5#6#7#8%
739 {%
740   \def #1##1% \XINT_expr_op_<op> ou flexpr ou iiexpr
741   {% keep value, get next number and operator, then do until
742     \expandafter #2\expandafter ##1%
743     \romannumeral`&&\expandafter\XINT_expr_getnext}%
744   \def #2##1##2% \XINT_expr_until_<op>_a ou flexpr ou iiexpr
745   {\xint_UDsignfork ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
746     -{#3##1##2}%
747     \krof}%
748   \def #3##1##2##3##4% \XINT_expr_until_<op>_b ou flexpr ou iiexpr
749   {% either execute next operation now, or first do next (possibly unary)
750     \ifnum ##2>#7%
751       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
752         \csname XINT_#8_op_##3\endcsname {##4}}%
753     \else \xint_afterfi {\expandafter ##2\expandafter ##3%
754       \csname .=\expandafter\XINT_expr_binop_inline_a\expandafter
755         {\expandafter\expandafter\expandafter#6\expandafter
756           \xint_exchangetwo_keepbraces\expandafter
757             {\expandafter\XINT_expr_unlock\expandafter ##4\expandafter}\expandafter}%
758         \romannumeral`&&\XINT_expr_unlock ##1,^,\endcsname}%
759     \fi}%
760   \let #7#5%
761}%
762 \def\XINT_expr_deflistopr_b #1#2#3#4%
763 {%
764   \expandafter\XINT_expr_deflistopr_c
765   \csname XINT_#1_op_#2\expandafter\endcsname
766   \csname XINT_#1_until_#2_a\expandafter\endcsname
767   \csname XINT_#1_until_#2_b\expandafter\endcsname
768   \csname XINT_#1_op_-#3\expandafter\endcsname
769   \csname xint_c_#3\expandafter\endcsname
770   \csname #4\expandafter\endcsname
771   \csname XINT_expr_precedence_#2\endcsname {#1}%
772}%

```

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

```

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

```

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

```

815 \XINT_expr_deflistopl_b {expr} {+[]}{vi} {xintAdd}%
816 \XINT_expr_deflistopl_b {expr} {-[]}{vi} {xintSub}%
817 \XINT_expr_deflistopl_b {expr} {*[]}{vii} {xintMul}%
818 \XINT_expr_deflistopl_b {expr} {/[]}{vii} {xintDiv}%
819 \XINT_expr_deflistopl_b {expr} {^[]}{ix} {xintPow}%
820 \XINT_expr_deflistopl_b {iiexpr}{+[]}{vi} {xintiiAdd}%
821 \XINT_expr_deflistopl_b {iiexpr}{-[]}{vi} {xintiiSub}%
822 \XINT_expr_deflistopl_b {iiexpr}{*[]}{vii} {xintiiMul}%

```

```

823 \XINT_expr_deflistopl_b {iiexpr}{/[]{}{vii}{xintiiDivRound}%
824 \XINT_expr_deflistopl_b {iiexpr}{^[]{}{ix} {xintiiPow}%
825 \XINT_expr_deflistopl_b {fexpr}{+[]{}{vi} {XINTinFloatAdd}%
826 \XINT_expr_deflistopl_b {fexpr}{-[]{}{vi} {XINTinFloatSub}%
827 \XINT_expr_deflistopl_b {fexpr}{*[]{}{vii}{XINTinFloatMul}%
828 \XINT_expr_deflistopl_b {fexpr}{/[]{}{vii}{XINTinFloatDiv}%
829 \XINT_expr_deflistopl_b {fexpr}{^[]{}{ix} {XINTinFloatPowerH}%

```

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

```

830 \xintFor #1 in {and,or,xor,mod} \do {%
831   \expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}}%
832 \expandafter\let\csname XINT_expr_precedence_and\expandafter\endcsname
833   \csname XINT_expr_precedence_&\endcsname
834 \expandafter\let\csname XINT_expr_precedence_or\expandafter\endcsname
835   \csname XINT_expr_precedence_|\endcsname
836 \expandafter\let\csname XINT_expr_precedence_mod\expandafter\endcsname
837   \csname XINT_expr_precedence_/\endcsname
838 \xintFor #1 in {expr, fexpr, iiexpr} \do {%
839   \expandafter\let\csname XINT_#1_op_and\expandafter\endcsname
840     \csname XINT_#1_op_&\endcsname
841   \expandafter\let\csname XINT_#1_op_or\expandafter\endcsname
842     \csname XINT_#1_op_|\endcsname
843   \expandafter\let\csname XINT_#1_op_mod\expandafter\endcsname
844     \csname XINT_#1_op_/\endcsname
845 }%

```

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

```

846 \expandafter\let\csname XINT_expr_precedence_==\expandafter\endcsname
847   \csname XINT_expr_precedence_=\endcsname
848 \expandafter\let\csname XINT_expr_precedence_&\string&\expandafter\endcsname
849   \csname XINT_expr_precedence_&\endcsname
850 \expandafter\let\csname XINT_expr_precedence_||\expandafter\endcsname
851   \csname XINT_expr_precedence_|\endcsname
852 \expandafter\let\csname XINT_expr_precedence_**\expandafter\endcsname
853   \csname XINT_expr_precedence_^\endcsname
854 \expandafter\let\csname XINT_expr_precedence_a**\expandafter\endcsname
855   \csname XINT_expr_precedence_a^\endcsname
856 \expandafter\let\csname XINT_expr_precedence_**[\expandafter\endcsname
857   \csname XINT_expr_precedence_^\endcsname
858 \xintFor #1 in {expr, fexpr, iiexpr} \do {%
859   \expandafter\let\csname XINT_#1_op_==\expandafter\endcsname
860     \csname XINT_#1_op_=\endcsname
861   \expandafter\let\csname XINT_#1_op_&\string&\expandafter\endcsname
862     \csname XINT_#1_op_&\endcsname
863   \expandafter\let\csname XINT_#1_op_||\expandafter\endcsname
864     \csname XINT_#1_op_|\endcsname
865   \expandafter\let\csname XINT_#1_op_**\expandafter\endcsname
866     \csname XINT_#1_op_^\endcsname
867   \expandafter\let\csname XINT_#1_op_a**\expandafter\endcsname
868     \csname XINT_#1_op_a^\endcsname
869   \expandafter\let\csname XINT_#1_op_**[\expandafter\endcsname
870     \csname XINT_#1_op_^\endcsname

```

871 }%

10.26 Macros for list selectors: [list][N], [list][:b], [list][a:], [list][a:b]

10.26.1	<code>\xintListSel:f:csv</code>	295
10.26.2	<code>\xintTrim:f:csv</code>	295
10.26.3	<code>\xintKeep:f:csv</code>	295
10.26.4	<code>\xintNthEltPy:f:csv</code>	296
10.26.5	<code>\xintReverse:f:csv</code>	296
10.26.6	<code>\xintLength:f:csv</code>	296

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.

2016/03/16. 1.2g decides that as [list][a:b] obeyed already Python syntax, for example [list][:n] keeps the first n elements, it was annoying that elements were tacitly numbered starting from 1 and not from 0 in the context of the [list][N] syntax. Thus now [list][n] returns the (n-1)-eth element. And one must use `len(list)` for the length.

2016/03/17. I had forgotten the 1.1 code comment above and 1.2g does it at last, in f-expandable versions `\xintTrim:f:csv`, `\xintKeep:f:csv`, `\xintNthEltPy:f:csv` and `\xintLength:f:csv`, which are all f-expandable for easier use with `\xintNewExpr`. Also `\xintReverse:f:csv`. I end up moving them all to `xinttools.sty`.

```

872 \def\xINT_tmpa #1#2#3#4#5#6%
873 {%
874   \def #1##1% \XINT_expr_op_][
875   {%
876     \expandafter #2\expandafter ##1\romannumeral`&&\XINT_expr_getnext
877   }%
878   \def #2##1##2% \XINT_expr_until_][_a
879   {\xint_UDsignfork
880     ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
881     -{#3##1##2}%
882   \krof }%
883   \def #3##1##2##3##4% \XINT_expr_until_][_b
884   {%
885     \ifnum ##2>#5%
886       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
887         \csname XINT_#6_op_##3\endcsname {##4}}%
888     \else
889       \xint_afterfi
890       {\expandafter ##2\expandafter ##3\csname
891         .=\expandafter\xintListSel:f:csv
892           \romannumeral`&&\XINT_expr_unlock ##4;%
893         \XINT_expr_unlock ##1;\endcsname % unlock for \xintNewExpr
894       }%
895     \fi
896   }%
897   \let #5\xint_c_ii

```

```

898 }%
899 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
900 \expandafter\XINT_tmpa
901   \csname XINT_#1_op_][\expandafter\endcsname
902   \csname XINT_#1_until_][_a\expandafter\endcsname
903   \csname XINT_#1_until_][_b\expandafter\endcsname
904   \csname XINT_#1_op_-vi\expandafter\endcsname
905   \csname XINT_expr_precedence_][\endcsname {#1}%
906 }%
907 \def\XINT_tmpa #1#2#3#4#5#6%
908 {%
909   \def #1##1% \XINT_expr_op_:
910   {%
911     \expandafter #2\expandafter ##1\romannumeral`&&\XINT_expr_getnext
912   }%
913   \def #2##1##2% \XINT_expr_until_:_a
914   {\xint_UDsignfork
915     ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
916     -{##1##2}%
917   \krof }%
918   \def #3##1##2##3##4% \XINT_expr_until_:_b
919   {%
920     \ifnum ##2>#5%
921       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
922         \csname XINT_#6_op_##3\endcsname {##4}}%
923     \else
924       \xint_afterfi
925       {\expandafter ##2\expandafter ##3\csname
926         .:=\xintiiifSgn{\XINT_expr_unlock ##1}NPP.%
927         \xintiiifSgn{\XINT_expr_unlock ##4}NPP.%
928         \xintNum{\XINT_expr_unlock ##1};\xintNum{\XINT_expr_unlock ##4}\endcsname
929       }%
930     \fi
931   }%
932   \let #5\xint_c_iii
933 }%
934 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
935 \expandafter\XINT_tmpa
936   \csname XINT_#1_op_:\expandafter\endcsname
937   \csname XINT_#1_until_:_a\expandafter\endcsname
938   \csname XINT_#1_until_:_b\expandafter\endcsname
939   \csname XINT_#1_op_-vi\expandafter\endcsname
940   \csname XINT_expr_precedence_:\endcsname {#1}%
941 }%
942 \catcode`[ 11 \catcode`] 11
943 \let\XINT_expr_precedence_:\ \xint_c_iii
944 \def\XINT_expr_op_:] #1{\expandafter\xint_c_i\expandafter }%
945   \csname .:=\xintiiifSgn{\XINT_expr_unlock #1}npp\XINT_expr_unlock #1\endcsname }%
946 \let\XINT_flexpr_op_:] \XINT_expr_op_:]
947 \let\XINT_iiexpr_op_:] \XINT_expr_op_:]
948 \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).

```

949 \edef\XINT_expr_op_[: #1{\xint_c_ii \expandafter\noexpand
950       \csname XINT_expr_itself_][\endcsname #10\string :}%
951 \let\XINT_flexpr_op_[: \XINT_expr_op_[:
952 \let\XINT_iiexpr_op_[: \XINT_expr_op_[:

```

10.26.1 \xintListSel:f:csv

Some complications here are due to \xintNewExpr matters.

2016/03/17. 1.2g uses Python numbering not only for slices but also item extractors.

It also implements faster extractors, acting directly on comma separated values with custom sub-macros \xintTrim:f:csv, \xintKeep:f:csv, \xintNthEltPy:f:csv and \xintLength:f:csv. We currently use here f-expandable versions for easier compatibility with \xintNewExpr. Also sub-optimal is the way the list argument is grabbed. But still better than the 1.1 version which used \xintCSVtoList everywhere for access to the routines provided with \xinttoolsnameimp.

```

953 \catcode`[ 12 \catcode`] 12
954 \def\xintListSel:f:csv #1{%
955     \if ]\noexpand#1\xint_dothis{\expandafter\XINT_listsel:_s\romannumeral`&&@\fi
956     \if : \noexpand#1\xint_dothis{\XINT_listsel:_:}\fi
957     \xint_orthat {\XINT_listsel:_nth #1}%
958 }%
959 \def\XINT_listsel:_s #1{\if p#1\expandafter\XINT_listsel:_trim\else
960     \expandafter\XINT_listsel:_keep\fi}%
961 \def\XINT_listsel:_: #1.#2.{\csname XINT_listsel:_#1#2\endcsname}%
962 \def\XINT_listsel:_trim #1;#2;{\xintTrim:f:csv {\xintNum{#1}}{#2}}%
963 \def\XINT_listsel:_keep #1;#2;{\xintKeep:f:csv {\xintNum{#1}}{#2}}%
964 \def\XINT_listsel:_nth#1;#2;{\xintNthEltPy:f:csv {\xintNum{#1}}{#2}}%
965 \def\XINT_listsel:_PP #1;#2;#3;%
966     {\xintTrim:f:csv {\xintNum{#1}}{\xintKeep:f:csv {\xintNum{#2}}{#3}}}%
967 \def\XINT_listsel:_NN #1;#2;#3;%
968     {\xintTrim:f:csv {\xintNum{#2}}{\xintKeep:f:csv {\xintNum{#1}}{#3}}}%
969 \def\XINT_listsel:_NP #1;#2;#3;{\expandafter\XINT_listsel:_NP_a
970     \the\numexpr #1+\xintLength:f:csv{#3};#2;#3;%
971 \def\XINT_listsel:_NP_a #1#2;%
972     {\if -#1\expandafter\XINT_listsel:_OP\fi\XINT_listsel:_PP #1#2;%
973 \def\XINT_listsel:_OP\XINT_listsel:_PP #1;{\XINT_listsel:_PP 0;%
974 \def\XINT_listsel:_PN #1;#2;#3;{\expandafter\XINT_listsel:_PN_a
975     \the\numexpr #2+\xintLength:f:csv{#3};#1;#3;%
976 \def\XINT_listsel:_PN_a #1#2;#3;%
977     {\if -#1\expandafter\XINT_listsel:_PO\fi\XINT_listsel:_PP #3;#1#2;%
978 \def\XINT_listsel:_PO\XINT_listsel:_PP #1;#2;{\XINT_listsel:_PP #1;0;%

```

10.26.2 \xintTrim:f:csv

1.2g 2016/03/17. Code in xinttools.sty

10.26.3 \xintKeep:f:csv

1.2g 2016/03/17. Code in xinttools.sty

10.26.4 `\xintNthEltPy:f:csv`

1.2g 2016/03/17. Code in `xinttools.sty`

10.26.5 `\xintReverse:f:csv`

1.2g 2016/03/17. Code in `xinttools.sty`

10.26.6 `\xintLength:f:csv`

1.2g 2016/03/17. Code in `xinttools.sty`

10.27 Macros for a..b list generation

10.27.1	<code>\xintSeq::csv</code>	296
10.27.2	<code>\xintiSeq::csv</code>	297

Ne produit que des listes d'entiers inférieurs à la borne de TeX ! mais sous la forme $N/1[0]$ en ce qui concerne `\xintSeq::csv`.

10.27.1 `\xintSeq::csv`

Commence par remplacer a par $\text{ceil}(a)$ 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.

```

979 \def\xintSeq::csv {\romannumeral0\xintseq::csv}%
980 \def\xintseq::csv #1#2%
981 {%
982   \expandafter\XINT_seq::csv\expandafter
983     {\the\numexpr \xintiCeil{#1}\expandafter}\expandafter
984     {\the\numexpr \xintiFloor{#2}}}%
985}%
986 \def\XINT_seq::csv #1#2%
987 {%
988   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
989   \expandafter\XINT_seq::csv_z
990   \or
991   \expandafter\XINT_seq::csv_p
992   \else
993   \expandafter\XINT_seq::csv_n
994   \fi
995   {#2}{#1}}%
996}%
997 \def\XINT_seq::csv_z #1#2{ #1/1[0]}%
998 \def\XINT_seq::csv_p #1#2%
999 {%
1000   \ifnum #1>#2
1001     \expandafter\expandafter\expandafter\XINT_seq::csv_p
1002     \else

```



```

1003     \expandafter\XINT_seq::csv_e
1004     \fi
1005     \expandafter{\the\numexpr #1-\xint_c_i}{#2},#1/1[0]%
1006 }%
1007 \def\XINT_seq::csv_n #1#2%
1008 {%
1009     \ifnum #1<#2
1010         \expandafter\expandafter\expandafter\XINT_seq::csv_n
1011     \else
1012         \expandafter\XINT_seq::csv_e
1013     \fi
1014     \expandafter{\the\numexpr #1+\xint_c_i}{#2},#1/1[0]%
1015 }%
1016 \def\XINT_seq::csv_e #1,{ }%

```

10.27.2 \xintiiseq::csv

```

1017 \def\xintiiseq::csv {\romannumeral0\xintiiseq::csv }%
1018 \def\xintiiseq::csv #1#2%
1019 {%
1020     \expandafter\XINT_iiseq::csv\expandafter
1021     {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
1022 }%
1023 \def\XINT_iiseq::csv #1#2%
1024 {%
1025     \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
1026     \expandafter\XINT_iiseq::csv_z
1027     \or
1028     \expandafter\XINT_iiseq::csv_p
1029     \else
1030     \expandafter\XINT_iiseq::csv_n
1031     \fi
1032     {#2}{#1}%
1033 }%
1034 \def\XINT_iiseq::csv_z #1#2{ #1}%
1035 \def\XINT_iiseq::csv_p #1#2%
1036 {%
1037     \ifnum #1>#2
1038         \expandafter\expandafter\expandafter\XINT_iiseq::csv_p
1039     \else
1040         \expandafter\XINT_seq::csv_e
1041     \fi
1042     \expandafter{\the\numexpr #1-\xint_c_i}{#2},#1%
1043 }%
1044 \def\XINT_iiseq::csv_n #1#2%
1045 {%
1046     \ifnum #1<#2
1047         \expandafter\expandafter\expandafter\XINT_iiseq::csv_n
1048     \else
1049         \expandafter\XINT_seq::csv_e
1050     \fi
1051     \expandafter{\the\numexpr #1+\xint_c_i}{#2},#1%
1052 }%

```

1053 \def\XINT_seq::csv_e #1,{ }%

10.28 Macros for a..[d]..b list generation

10.28.1	\xintSeqA::csv, \xintiiSeqA::csv, \XINTinFloatSeqA::csv	298
10.28.2	\xintSeqB::csv	298
10.28.3	\xintiiSeqB::csv	299
10.28.4	\XINTinFloatSeqB::csv	299

Contrarily to a..b which is limited to small integers, this works with a, b, and d (big) fractions. It will produce a «nil» list, if a>b and d<0 or a<b and d>0.

10.28.1 \xintSeqA::csv, \xintiiSeqA::csv, \XINTinFloatSeqA::csv

```

1054 \def\xintSeqA::csv #1%
1055   {\expandafter\XINT_seq::csv\expandafter{\romannumeral0\xintra {#1}}}%
1056 \def\XINT_seq::csv #1#2{\expandafter\XINT_seq::csv_a \romannumeral0\xintra {#2};#1;}%
1057 \def\xintiiSeqA::csv #1{\expandafter\XINT_iiseq::csv\expandafter{\romannumeral`&&@#1}}%
1058 \def\XINT_iiseq::csv #1#2{\expandafter\XINT_seq::csv_a\romannumeral`&&@#2;#1;}%
1059 \def\XINTinFloatSeqA::csv #1{\expandafter\XINT_flseq::csv\expandafter
1060   {\romannumeral0\XINTinfloat [\XINTdigits]{#1}}}%
1061 \def\XINT_flseq::csv #1#2%
1062   {\expandafter\XINT_seq::csv_a\romannumeral0\XINTinfloat [\XINTdigits]{#2};#1;}%
1063 \def\XINT_seq::csv_a #1{\xint_UDzerominusfork
1064   #1-{z}%
1065   0#1{n}%
1066   0-{p}%
1067   \krof #1}%

```

10.28.2 \xintSeqB::csv

With one year late documentation, let's just say, the #1 is \XINT_expr_unlock\.=Ua;b; with U=z or n or p, a=step, b=start.

```

1068 \def\xintSeqB::csv #1#2%
1069   {\expandafter\XINT_seqb::csv \expandafter{\romannumeral0\xintra{#2}}{#1}}%
1070 \def\XINT_seqb::csv #1#2{\expandafter\XINT_seqb::csv_a\romannumeral`&&@#2#1!}%
1071 \def\XINT_seqb::csv_a #1#2;#3;#4!{\expandafter\XINT_expr_seq_empty?
1072   \romannumeral0\csname XINT_seqb::csv_#1\endcsname {#3}{#4}{#2}}%
1073 \def\XINT_seqb::csv_p #1#2#3%
1074 {%
1075   \xintifCmp {#1}{#2}{, #1\expandafter\XINT_seqb::csv_p\expandafter}%
1076   {, #1\xint_gobble_iii}{\xint_gobble_iii}%

```

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

```

1077   {\romannumeral0\xintadd {#3}{#1}}{#2}{#3}%
1078 }%
1079 \def\XINT_seqb::csv_n #1#2#3%
1080 {%
1081   \xintifCmp {#1}{#2}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1082   {, #1\expandafter\XINT_seqb::csv_n\expandafter}%
1083   {\romannumeral0\xintadd {#3}{#1}}{#2}{#3}%
1084 }%
1085 \def\XINT_seqb::csv_z #1#2#3{, #1}%

```

10.28.3 `\xintiiSeqB::csv`

```

1086 \def\xintiiSeqB::csv #1#2{\XINT_iiseqb::csv #1#2}%
1087 \def\XINT_iiseqb::csv #1#2#3#4%
1088   {\expandafter\XINT_iiseqb::csv_a
1089     \romannumeral`&&\expandafter \XINT_expr_unlock\expandafter#2%
1090     \romannumeral`&&\XINT_expr_unlock #4!}%
1091 \def\XINT_iiseqb::csv_a #1#2;#3;#4!{\expandafter\XINT_expr_seq_empty?
1092   \romannumeral`&&\csname XINT_iiseqb::csv_#1\endcsname {#3}{#4}{#2}}%
1093 \def\XINT_iiseqb::csv_p #1#2#3%
1094 {%
1095   \xintSgnFork{\XINT_Cmp {#1}{#2}}{, #1\expandafter\XINT_iiseqb::csv_p\expandafter}%
1096   {, #1\xint_gobble_iii}{\xint_gobble_iii}%
1097   {\romannumeral0\xintiiadd {#3}{#1}}{#2}{#3}%
1098 }%
1099 \def\XINT_iiseqb::csv_n #1#2#3%
1100 {%
1101   \xintSgnFork{\XINT_Cmp {#1}{#2}}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1102   {, #1\expandafter\XINT_iiseqb::csv_n\expandafter}%
1103   {\romannumeral0\xintiiadd {#3}{#1}}{#2}{#3}%
1104 }%
1105 \def\XINT_iiseqb::csv_z #1#2#3{, #1}%

```

10.28.4 `\XINTinFloatSeqB::csv`

```

1106 \def\XINTinFloatSeqB::csv #1#2{\expandafter\XINT_flseqb::csv \expandafter
1107   {\romannumeral0\XINTinfloat [\XINTdigits]{#2}}{#1}}%
1108 \def\XINT_flseqb::csv #1#2{\expandafter\XINT_flseqb::csv_a\romannumeral`&&#2#1!}%
1109 \def\XINT_flseqb::csv_a #1#2;#3;#4!{\expandafter\XINT_expr_seq_empty?
1110   \romannumeral`&&\csname XINT_flseqb::csv_#1\endcsname {#3}{#4}{#2}}%
1111 \def\XINT_flseqb::csv_p #1#2#3%
1112 {%
1113   \xintifCmp {#1}{#2}{, #1\expandafter\XINT_flseqb::csv_p\expandafter}%
1114   {, #1\xint_gobble_iii}{\xint_gobble_iii}%
1115   {\romannumeral0\XINTinfloatadd {#3}{#1}}{#2}{#3}%
1116 }%
1117 \def\XINT_flseqb::csv_n #1#2#3%
1118 {%
1119   \xintifCmp {#1}{#2}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1120   {, #1\expandafter\XINT_flseqb::csv_n\expandafter}%
1121   {\romannumeral0\XINTinfloatadd {#3}{#1}}{#2}{#3}%
1122 }%
1123 \def\XINT_flseqb::csv_z #1#2#3{, #1}%

```

10.29 The comma as binary operator

New with 1.09a. Suffices to set its precedence level to two.

```

1124 \def\XINT_tmpa #1#2#3#4#5#6%
1125 {%
1126   \def #1##1% \XINT_expr_op_,
1127   {%
1128     \expandafter #2\expandafter ##1\romannumeral`&&\XINT_expr_getnext
1129   }%
1130   \def #2##1##2% \XINT_expr_until_,_a

```

```

1131 {\xint_UDsignfork
1132   ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
1133   -{#3##1##2}%
1134   \krof }%
1135 \def #3##1##2##3##4% \XINT_expr_until_,_b
1136 {%
1137   \ifnum ##2>\xint_c_ii
1138     \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
1139                   \csname XINT_#6_op_##3\endcsname {##4}}%
1140   \else
1141     \xint_afterfi
1142     {\expandafter ##2\expandafter ##3%
1143     \csname .=\XINT_expr_unlock ##1,\XINT_expr_unlock ##4\endcsname }%
1144   \fi
1145 }%
1146 \let #5\xint_c_ii
1147 }%
1148 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1149 \expandafter\XINT_tmpa
1150   \csname XINT_#1_op_,\expandafter\endcsname
1151   \csname XINT_#1_until_,_a\expandafter\endcsname
1152   \csname XINT_#1_until_,_b\expandafter\endcsname
1153   \csname XINT_#1_op_-vi\expandafter\endcsname
1154   \csname XINT_expr_precedence_,\endcsname {#1}%
1155 }%

```

10.30 The minus as prefix operator of variable precedence level

Inherits the precedence level of the previous infix operator.

```

1156 \def\XINT_tmpa #1#2#3%
1157 {%
1158   \expandafter\XINT_tmppb
1159   \csname XINT_#1_op_-#3\expandafter\endcsname
1160   \csname XINT_#1_until_-#3_a\expandafter\endcsname
1161   \csname XINT_#1_until_-#3_b\expandafter\endcsname
1162   \csname xint_c_#3\endcsname {#1}#2%
1163 }%
1164 \def\XINT_tmppb #1#2#3#4#5#6%
1165 {%
1166   \def #1% \XINT_expr_op_-<level>
1167   {% get next number+operator then switch to _until macro
1168     \expandafter #2\romannumeral`&&@\XINT_expr_getnext
1169   }%
1170   \def #2##1% \XINT_expr_until_-<l>_a
1171   {\xint_UDsignfork
1172     ##1{\expandafter #2\romannumeral`&&@#1}%
1173     -{#3##1}%
1174     \krof }%
1175   \def #3##1##2##3% \XINT_expr_until_-<l>_b
1176   {% _until tests precedence level with next op, executes now or postpones
1177     \ifnum ##1>#4%
1178       \xint_afterfi {\expandafter #2\romannumeral`&&@%

```

```

1179          \csname XINT_#5_op_##2\endcsname {##3}}%
1180      \else
1181          \xint_afterfi {\expandafter ##1\expandafter ##2%
1182              \csname .=#6{\XINT_expr_unlock ##3}\endcsname }%
1183      \fi
1184  }%
1185 }%

```

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.

```

1186 \xintApplyInline{\XINT_tmpa {expr}\xintOpp}{\vi}{vii}{viii}{ix}}%
1187 \xintApplyInline{\XINT_tmpa {fexpr}\xintOpp}{\vi}{vii}{viii}{ix}}%
1188 \xintApplyInline{\XINT_tmpa {iiexpr}\xintiiOpp}{\vi}{vii}{viii}{ix}}%

```

10.31 ? as two-way and ?? as three-way conditionals with braced branches

In 1.1, I overload ? with ??, as : will be used for list extraction, problem with (stuff)?{?(1)}{0} for example, one should put a space (stuff)?{ ?(1)}{0} will work. Small idiosyncrasy. (which has been removed in 1.2h, there is no problem anymore with (test)?{?(1)}{0}, however (test)?{?}{!(x)} is not accepted; but (test)?{?(x)}{!(x)} is or even with {?(x)}{!(x)}.)

syntax: ?{yes}{no} and ??{<0}{=0}{>0}.

The difficulty is to recognize the second ? without removing braces as would be the case with standard parsing of operators. Hence the ? operator is intercepted in \XINT_expr_getop_b.

1.2h corrects a bug in \XINT_expr_op_? which in context like (test)?{foo}{bar} would provoke expansion of \foo, or also with (test)?{bar} would result in an error. The fix also solves the (test)?{?(1)}{0} issue mentioned above.

```

1189 \let\XINT_expr_precedence_? \xint_c_x
1190 \def\XINT_expr_op_? #1#2%
1191   {\XINT_expr_op_?checka #2!\xint_bye\XINT_expr_op_?a #1{#2}}%
1192 \def\XINT_expr_op_?checka #1{\expandafter\XINT_expr_op_?checkb\detokenize{#1}}%
1193 \def\XINT_expr_op_?checkb #1{\if ?#1\expandafter\XINT_expr_op_?checkc
1194   \else\expandafter\xint_bye\fi }%
1195 \def\XINT_expr_op_?checkc #1{\xint_gob_til_! #1\XINT_expr_op_?? !\xint_bye}%
1196 \def\XINT_expr_op_?a #1#2#3%
1197 {%
1198   \xintiiifNotZero{\XINT_expr_unlock #1}{\XINT_expr_getnext #2}{\XINT_expr_getnext #3}%
1199 }%
1200 \let\XINT_flexpr_op_?\XINT_expr_op_?
1201 \let\XINT_iiexpr_op_?\XINT_expr_op_?
1202 \def\XINT_expr_op_?? !\xint_bye\xint_bye\XINT_expr_op_?a #1#2#3#4#5%
1203 {%
1204   \xintiiifSgn {\XINT_expr_unlock #1}%
1205   {\XINT_expr_getnext #3}{\XINT_expr_getnext #4}{\XINT_expr_getnext #5}%
1206 }%

```

10.32 ! as postfix factorial operator

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

```

1207 \let\XINT_expr_precedence_! \xint_c_x

```

```

1208 \def\XINT_expr_op_! #1{\expandafter\XINT_expr_getop
1209             \csname .=\xintiFac{\XINT_expr_unlock #1}\endcsname }%
1210 \def\XINT_flexpr_op_! #1{\expandafter\XINT_expr_getop
1211             \csname .=\XINTinFloatFac{\XINT_expr_unlock #1}\endcsname }%
1212 \def\XINT_iiexpr_op_! #1{\expandafter\XINT_expr_getop
1213             \csname .=\xintiiFac{\XINT_expr_unlock #1}\endcsname }%

```

10.33 The A/B[N] mechanism

Releases earlier than 1.1 required the use of braces around A/B[N] input. The [N] is now implemented directly. *BUT* this uses a delimited macro! thus N is not allowed to be itself an expression (I could add it...). \xintE, \xintiiE, and \XINTinFloatE all put #2 in a \numexpr. But attention to the fact that \numexpr stops at spaces separating digits: \the\numexpr 3 + 7 9\relax gives 109\relax !! Hence we have to be careful.

\numexpr will not handle catcode 11 digits, but adding a \detokenize will suddenly make illicit for N to rely on macro expansion.

```

1214 \catcode`[ 11
1215 \catcode`* 11
1216 \let\XINT_expr_precedence_ \xint_c_vii
1217 \def\XINT_expr_op_[ #1#2]{\expandafter\XINT_expr_getop
1218     \csname .=\xintE{\XINT_expr_unlock #1}%
1219     {\xint_zapspace #2 \xint_gobble_i}\endcsname}%
1220 \def\XINT_iiexpr_op_[ #1#2]{\expandafter\XINT_expr_getop
1221     \csname .=\xintiiE{\XINT_expr_unlock #1}%
1222     {\xint_zapspace #2 \xint_gobble_i}\endcsname}%
1223 \def\XINT_flexpr_op_[ #1#2]{\expandafter\XINT_expr_getop
1224     \csname .=\XINTinFloatE{\XINT_expr_unlock #1}%
1225     {\xint_zapspace #2 \xint_gobble_i}\endcsname}%
1226 \catcode`[ 12
1227 \catcode`* 12

```

10.34 \XINT_expr_op_` for recognizing functions

The "onlitteral" intercepts is for bool, togl, protect, ... but also for add, mul, seq, etc... Genuine functions have expr, iiexpr and flexpr versions (or only one or two of the three).

With 1.2c "onlitteral" is also used to disambiguate variables from functions. However as I use only a \ifcsname test, in order to be able to re-define a variable as function, I move the check for being a function first. Each variable name now has its onlitteral_<name> associated macro which is the new way tacit multiplication in front of a parenthesis is implemented. This used to be decided much earlier at the time of \XINT_expr_func.

The advantage of our choices for 1.2c is that the same name can be used for a variable or a function, the parser will apply the correct interpretation which is decided by the presence or not of an opening parenthesis next.

```

1228 \def\XINT_tpa #1#2#3{%
1229     \def #1##1%
1230     {%
1231         \ifcsname XINT_#3_func_##1\endcsname
1232         \xint_dothis{\expandafter\expandafter
1233             \csname XINT_#3_func_##1\endcsname\romannumeral`&&@#2}\fi
1234         \ifcsname XINT_expr_onlitteral_##1\endcsname
1235         \xint_dothis{\csname XINT_expr_onlitteral_##1\endcsname}\fi

```

```

1236 \xint_orthat{\XINT_expr_unknown_function {##1}%
1237 \expandafter\XINT_expr_func_unknown\romannumeral`&&@#2}%
1238 }%
1239 }%
1240 \def\XINT_expr_unknown_function #1{\xintError:removed \xint_gobble_i {#1}}%
1241 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1242 \expandafter\XINT_tmpa
1243 \csname XINT_#1_op_`\expandafter\endcsname
1244 \csname XINT_#1_oparen\endcsname
1245 {#1}%
1246 }%
1247 \def\XINT_expr_func_unknown #1#2#3%
1248 {\expandafter #1\expandafter #2\csname .=#0\endcsname }%

```

10.35 The bool, togl, protect pseudo “functions”

bool, *togl* and *protect* use delimited macros. They are not true functions, they turn off the parser to gather their “variable”.

```

1249 \def\XINT_expr_onlitteral_bool #1)%
1250 {\expandafter\XINT_expr_getop\csname .=\xintBool{#1}\endcsname }%
1251 \def\XINT_expr_onlitteral_togl #1)%
1252 {\expandafter\XINT_expr_getop\csname .=\xintToggle{#1}\endcsname }%
1253 \def\XINT_expr_onlitteral_protect #1)%
1254 {\expandafter\XINT_expr_getop\csname .=\detokenize{#1}\endcsname }%

```

10.36 The break function

break is a true function, the parsing via expansion of the succeeding material proceeded via *_oparen* macros as with any other function.

```

1255 \def\XINT_expr_func_break #1#2#3%
1256 {\expandafter #1\expandafter #2\csname .=?\romannumeral`&&@\XINT_expr_unlock #3\endcsname }%
1257 \let\XINT_flexpr_func_break \XINT_expr_func_break
1258 \let\XINT_iiexpr_func_break \XINT_expr_func_break

```

10.37 The qint, qfrac, qfloat “functions”

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

```

1259 \def\XINT_expr_onlitteral_qint #1)%
1260 {\expandafter\XINT_expr_getop\csname .=\xintiNum{#1}\endcsname }%
1261 \def\XINT_expr_onlitteral_qfrac #1)%
1262 {\expandafter\XINT_expr_getop\csname .=\xintRaw{#1}\endcsname }%
1263 \def\XINT_expr_onlitteral_qfloat #1)%
1264 {\expandafter\XINT_expr_getop\csname .=\XINTinFloatdigits{#1}\endcsname }%

```

10.38 *\XINT_expr_op__* for recognizing variables

The 1.1 mechanism for *\XINT_expr_var_<varname>* has been modified in 1.2c. The *<varname>* associated macro is now only expanded once, not twice. We arrive here via *\XINT_expr_func*.

```

1265 \def\XINT_expr_op__ #1% op__ with two _'s
1266     {%
1267         \ifcsname XINT_expr_var_#1\endcsname
1268             \expandafter\xint_firstoftwo
1269         \else
1270             \expandafter\xint_secondoftwo
1271         \fi
1272         {\expandafter\expandafter\expandafter
1273          \XINT_expr_getop\csname XINT_expr_var_#1\endcsname}%
1274         {\XINT_expr_unknown_variable {#1}%
1275          \expandafter\XINT_expr_getop\csname .:=0\endcsname}%
1276     }%
1277 \def\XINT_expr_unknown_variable #1{\xintError:removed \xint_gobble_i {#1}}%
1278 \let\XINT_flexpr_op__ \XINT_expr_op__
1279 \let\XINT_iexpr_op__ \XINT_expr_op__

```

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

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

Variable names may contain letters, digits, underscores, and must not start with a digit. Names starting with `@` or `_` 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`, `iexpr` 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))`.

```

1280 \catcode`* 11
1281 \def\XINT_expr_defvar #1#2#3;%
1282     \edef\XINT_expr_tmpa{#2}%
1283     \edef\XINT_expr_tmpa {\xint_zapspaces_o\XINT_expr_tmpa}%
1284     \ifnum\expandafter\xintLength\expandafter{\XINT_expr_tmpa}=\z@
1285         \xintMessage {xintexpr}{Warning}
1286         {Error: impossible to declare variable with empty name.}%
1287     \else
1288         \edef\XINT_expr_tmpb {\romannumeral0#1#3\relax }%

```



```

1289 \expandafter\edef\csname XINT_expr_var_\XINT_expr_tmpa\endcsname
1290 {\expandafter\noexpand\XINT_expr_tmpb}%
1291 \expandafter\edef\csname XINT_expr_onlitteral_\XINT_expr_tmpa\endcsname
1292 {\XINT_expr_precedence_*** *\expandafter\noexpand\XINT_expr_tmpb {}%
1293 \ifxintverbose\xintMessage {xintexpr}{Info}
1294 {Variable "\XINT_expr_tmpa" defined with value
1295 \expandafter\XINT_expr_unlock\XINT_expr_tmpb.}%
1296 \fi
1297 \fi
1298 }%
1299 \catcode`* 12
1300 \catcode`: 12
1301 \def\xintdefvar #1:={\XINT_expr_defvar\xintbareeval {#1}}%
1302 \def\xintdefiivar #1:={\XINT_expr_defvar\xintbareiieval {#1}}%
1303 \def\xintdeffloatvar #1:={\XINT_expr_defvar\xintbarefloateval {#1}}%
1304 \catcode`: 11

```

10.40 \xintunassignvar

1.2e. Currently not possible to genuinely ``undefine'' a variable, all we can do is to let it stand for zero and generate an error. The reason is that I chose to use \ifcsname tests in \XINT_expr_op__ and \XINT_expr_op_`.

```

1305 \def\xintunassignvar #1{%
1306 \edef\XINT_expr_tmpa{#1}%
1307 \edef\XINT_expr_tmpa {\xint_zapspaces_o\XINT_expr_tmpa}%
1308 \ifcsname XINT_expr_var_\XINT_expr_tmpa\endcsname
1309 \ifnum\expandafter\xintLength\expandafter{\XINT_expr_tmpa}=\@ne
1310 \expandafter\XINT_expr_makedummy \XINT_expr_tmpa
1311 \ifxintverbose\xintMessage {xintexpr}{Info}
1312 {Character \XINT_expr_tmpa\space usable as dummy variable (if letter).}%
1313 \fi
1314 \else
1315 \expandafter\edef\csname XINT_expr_var_\XINT_expr_tmpa\endcsname
1316 {\csname .=\0\endcsname\noexpand\XINT_expr_undefined {\XINT_expr_tmpa}}%
1317 \expandafter\edef\csname XINT_expr_onlitteral_\XINT_expr_tmpa\endcsname
1318 {\csname .=\0\endcsname\noexpand\XINT_expr_undefined {\XINT_expr_tmpa}*\}%
1319 \ifxintverbose\xintMessage {xintexpr}{Info}
1320 {Variable \XINT_expr_tmpa\space has been ``unassigned''.}%
1321 \fi
1322 \fi
1323 \else
1324 \xintMessage {xintexpr}{Warning}
1325 {Error: there was no such variable \XINT_expr_tmpa\space to unassign.}%
1326 \fi
1327 }%
1328 \def\XINT_expr_undefined #1{\xintError:replaced_by_zero\xint_gobble_i {#1}}%

```

10.41 seq and the implementation of dummy variables

- 10.41.1 All letters usable as dummy variables 306
- 10.41.2 omit and abort 307
- 10.41.3 The special variables @, @1, @2, @3, @4, @@, @@(1), ..., @@@, @@@(1), ... for recursion 307

10.41.4	<code>\XINT_expr_onlitteral_seq</code>	308
10.41.5	<code>\XINT_expr_onlitteral_seq_a</code>	309
10.41.6	<code>\XINT_isbalanced_a</code> for <code>\XINT_expr_onlitteral_seq_a</code>	309
10.41.7	<code>\XINT_allexpr_func_seqx</code>	310
10.41.8	Evaluation over list, <code>\XINT_expr_seq:_a</code> with break, abort, omit	310
10.41.9	Evaluation over ++ generated lists with <code>\XINT_expr_seq:_A</code>	311

All of seq, add, mul, rseq, etc... (actually all of the extensive changes from xintexpr 1.09n to 1.1) was done around June 15-25th 2014, but the problem is that I did not document the code enough, and I had a hard time understanding in October what I had done in June. Despite the lesson, again being short on time, I do not document enough my current understanding of the innards of the beast...

I added subs, and iter in October (also the [:n], [n:] list extractors), proving I did at least understand a bit (or rather could imitate) my earlier code (but don't ask me to explain `\xintNewExpr` !)

The `\XINT_expr_onlitteral_seq_a` parses: "expression, variable=list)" (when it is called the opening (has been swallowed, and it looks for the ending one.) Both expression and list may themselves contain parentheses and commas, we allow nesting. For example "x^2,x=1..10)", at the end of seq_a we have {variable{expression}}{list}, in this example {x{x^2}}{1..10}, or more complicated "seq(add(y,y=1..x),x=1..10)" will work too. The variable is a single lowercase Latin letter.

The complications with `\xint_c_xviii` in seq_f is for the recurrent thing that we don't know in what type of expressions we are, hence we must move back up, with some loss of efficiency (superfluous check for minus sign, etc...). But the code manages simultaneously expr, flexpr and iexpr.

10.41.1 All letters usable as dummy variables

The nil variable was introduced in 1.1 but isn't used under that name. However macros handling a..`[d]`..`b`, or for seq with dummy variable where omit has omitted everyting may in practice inject a nil value as current number.

1.2c has changed the way variables are disambiguated from functions and for this it has added here the definitions of `\XINT_expr_onlitteral_<name>`.

In 1.1 a letter variable say X was acting as a delimited macro looking for `!X{stuff}` and then would expand the stuff inside a `\csname.=...\endcsname`. I don't think I used the possibilities this opened and the 1.2c version has stuff_already_ encapsulated thus a single token. Only one expansion, not two is then needed in `\XINT_expr_op_`.

I had to accordingly modify seq, add, mul and subs, but fortunately realized that the @, @1, etc... variables for rseq, rrseq and iter already had been defined in the way now also followed by the Latin letters as dummy variables.

The 1.2e `\XINT_expr_makedummy` should not be used with multi-letter argument. The add, mul, seq, etc... can only work with one-letter long dummy variable. And this will almost certainly not change. Public interface is via `\xintunassignvar` which one can use with a letter to re-declare it as dummy variable. Also 1.2e does the tacit multiplication `x(stuff)->x*(stuff)` in its higher precedence form. Things are easy now that variables always fetch a single already locked value `\.=<number>`.

The tacit multiplication in case of the ``nil'` variable doesn't make much sense but we do it anyhow.

```

1329 \catcode`* 11
1330 \def\XINT_expr_makedummy #1%
1331 {%
1332   \expandafter\def\csname XINT_expr_var_#1\endcsname ##1\relax !#1##2%
1333   {##2##1\relax !#1##2}%

```

```

1334 \expandafter\def\csname XINT_expr_onlitteral_#1\endcsname ##1\relax !#1##2%
1335      {\XINT_expr_precedence_*** *##2(##1\relax !#1##2}%
1336 }%
1337 \xintApplyUnbraced \XINT_expr_makedummy {abcdefghijklmnopqrstuvwxyz}%
1338 \xintApplyUnbraced \XINT_expr_makedummy {ABCDEFGHIJKLMNOPQRSTUVWXYZ}%
1339 \edef\XINT_expr_var_nil {\expandafter\noexpand\csname . = \endcsname}%
1340 \edef\XINT_expr_onlitteral_nil
1341      {\XINT_expr_precedence_*** *\expandafter\noexpand\csname . = \endcsname (}%
1342 \catcode`* 12

```

10.41.2 omit and abort

June 24 and 25, 2014.

Added comments 2015/11/13:

Et la documentation ? on n'y comprend plus rien. Trop rusé.

```
\def\XINT_expr_var_omit #1\relax !{1^C!{}}{}}\relax !}
```

```
\def\XINT_expr_var_abort #1\relax !{1^C!{}}{}}\relax !}
```

C'était accompagné de \XINT_expr_precedence_^C=0 et d'un hack au sein même des macros until de plus bas niveau.

Le mécanisme sioux était le suivant: ^C est déclaré comme un opérateur de précedence nulle. Lorsque le parseur trouve un "omit" dans un seq ou autre, il va insérer dans le stream \XINT_expr_getop suivi du texte de remplacement. Donc ici on avait un 1 comme place holder, puis l'opérateur ^C. Celui-ci étant de précedence zéro provoque la finalisation de tous les calculs antérieurs dans le sous-bareeval. Mais j'ai dû hacker le until_end_b (et le until_)_b qui confronté à ^C, va se relancer à zéro, le getnext va trouver le !{}}{}}\relax ! et ensuite il y aura \relax, et le résultat sera \relax pour omit ou \relax pour abort. Les routines des boucles seq, iter, etc... peuvent alors repérer le ! ou ^ et agir en conséquence (un long paragraphe pour ne décrire que partiellement une ou deux lignes de codes...).

Mais ^C a été fait alors que je n'avais pas encore les variables muettes. Je dois trouver autre chose, car seq(2^C, C=1..5) est alors impossible. De toute façon ce ^C était à usage interne uniquement.

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

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

```

1343 \edef\XINT_expr_var_omit #1\relax !{1\string !?!\relax !}%
1344 \edef\XINT_expr_var_abort #1\relax !{1\string !?^\relax !}%
1345 \def\XINT_expr_op_!? #1#2\relax {\expandafter\XINT_expr_founded\csname .=#2\endcsname}%
1346 \let\XINT_iexpr_op_!? \XINT_expr_op_!?
1347 \let\XINT_flexpr_op_!? \XINT_expr_op_!?

```

10.41.3 The special variables @, @1, @2, @3, @4, @@, @@(1), ..., @@@, @@@(1), ... for recursion

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

1.2c adds the needed "onlitteral" now that tacit multiplication between a variable and a (has a new mechanism. 1.2e does this tacit multiplication with higher precedence.

For the record, the ~ has catcode 3 in this code.

```

1348 \catcode`? 3 \catcode`* 11
1349 \def\XINT_expr_var_@ #1~#2{#2#1~#2}%
1350 \expandafter\let\csname XINT_expr_var_@1\endcsname \XINT_expr_var_@
1351 \expandafter\def\csname XINT_expr_var_@2\endcsname #1~#2#3{#3#1~#2#3}%
1352 \expandafter\def\csname XINT_expr_var_@3\endcsname #1~#2#3#4{#4#1~#2#3#4}%
1353 \expandafter\def\csname XINT_expr_var_@4\endcsname #1~#2#3#4#5{#5#1~#2#3#4#5}%
1354 \def\XINT_expr_onlitteral_@ #1~#2{\XINT_expr_precedence_*** *#2(#1~#2}%
1355 \expandafter\let\csname XINT_expr_onlitteral_@1\endcsname \XINT_expr_onlitteral_@
1356 \expandafter\def\csname XINT_expr_onlitteral_@2\endcsname #1~#2#3%
1357     {\XINT_expr_precedence_*** *#3(#1~#2#3}%
1358 \expandafter\def\csname XINT_expr_onlitteral_@3\endcsname #1~#2#3#4%
1359     {\XINT_expr_precedence_*** *#4(#1~#2#3#4}%
1360 \expandafter\def\csname XINT_expr_onlitteral_@4\endcsname #1~#2#3#4#5%
1361     {\XINT_expr_precedence_*** *#5(#1~#2#3#4#5}%
1362 \catcode`* 12
1363 \def\XINT_expr_func_@@ #1#2#3#4~#5?%
1364 {%
1365     \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1366         {\xintNum{\XINT_expr_unlock#3}}{#5}#4~#5?%
1367}%
1368 \def\XINT_expr_func_@@@ #1#2#3#4~#5~#6?%
1369 {%
1370     \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1371     {\xintNum{\XINT_expr_unlock#3}}{#6}#4~#5~#6?%
1372}%
1373 \def\XINT_expr_func_@@@# #1#2#3#4~#5~#6~#7?%
1374 {%
1375     \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1376     {\xintNum{\XINT_expr_unlock#3}}{#7}#4~#5~#6~#7?%
1377}%
1378 \let\XINT_flexpr_func_@@\XINT_expr_func_@@
1379 \let\XINT_flexpr_func_@@@\XINT_expr_func_@@@
1380 \let\XINT_flexpr_func_@@@#\XINT_expr_func_@@@#
1381 \def\XINT_iexpr_func_@@ #1#2#3#4~#5?%
1382 {%
1383     \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1384     {\XINT_expr_unlock#3}{#5}#4~#5?%
1385}%
1386 \def\XINT_iexpr_func_@@@ #1#2#3#4~#5~#6?%
1387 {%
1388     \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1389     {\XINT_expr_unlock#3}{#6}#4~#5~#6?%
1390}%
1391 \def\XINT_iexpr_func_@@@# #1#2#3#4~#5~#6~#7?%
1392 {%
1393     \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1394     {\XINT_expr_unlock#3}{#7}#4~#5~#6~#7?%
1395}%
1396 \catcode`? 11

```

10.41.4 \XINT_expr_onlitteral_seq

```

1397 \def\XINT_expr_onlitteral_seq

```

```

1398 {\expandafter\XINT_expr_onlitteral_seq_f\romannumeral`&&\XINT_expr_onlitteral_seq_a {}}%
1399 \def\XINT_expr_onlitteral_seq_f #1#2{\xint_c_xviii `{seqx}#2)\relax #1}%

```

10.41.5 \XINT_expr_onlitteral_seq_a

```

1400 \def\XINT_expr_onlitteral_seq_a #1#2,%
1401 {%
1402     \ifcase\XINT_isbalanced_a \relax #1#2(\xint_bye)\xint_bye
1403         \expandafter\XINT_expr_onlitteral_seq_c
1404         \or\expandafter\XINT_expr_onlitteral_seq_b
1405         \else\expandafter\xintError:we_are_doomed
1406     \fi {#1#2},%
1407 }%
1408 \def\XINT_expr_onlitteral_seq_b #1,{\XINT_expr_onlitteral_seq_a {#1},}%
1409 \def\XINT_expr_onlitteral_seq_c #1,#2#3% #3 pour absorber le =
1410 {%
1411     \XINT_expr_onlitteral_seq_d {#2{#1}}{}}%
1412 }%
1413 \def\XINT_expr_onlitteral_seq_d #1#2#3)%
1414 {%
1415     \ifcase\XINT_isbalanced_a \relax #2#3(\xint_bye)\xint_bye
1416         \or\expandafter\XINT_expr_onlitteral_seq_e
1417         \else\expandafter\xintError:we_are_doomed
1418     \fi
1419     {#1}{#2#3}%
1420 }%
1421 \def\XINT_expr_onlitteral_seq_e #1#2{\XINT_expr_onlitteral_seq_d {#1}{#2}}}%

```

10.41.6 \XINT_isbalanced_a for \XINT_expr_onlitteral_seq_a

Expands to `\xint_c_mone` in case a closing `)` had no opening `(` matching it, to `\@ne` if opening `(` had no closing `)` matching it, to `\z@` if expression was balanced.

```

1422 % use as \XINT_isbalanced_a \relax #1(\xint_bye)\xint_bye
1423 \def\XINT_isbalanced_a #1({\XINT_isbalanced_b #1)\xint_bye }%
1424 \def\XINT_isbalanced_b #1)#2%
1425     {\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
1426 \def\XINT_isbalanced_error #1)\xint_bye {\xint_c_mone}%
    #2 was \xint_bye, was there a ) in original #1?
1427 \def\XINT_isbalanced_c\xint_bye\XINT_isbalanced_error #1%
1428     {\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.
1429 \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.
1430 \def\XINT_isbalanced_d #1)#2%
1431     {\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.
1432 \def\XINT_isbalanced_no\xint_bye #1\xint_bye\xint_bye {\xint_c_i }%

```

10.41.7 \XINT_allexpr_func_seqx

1.2c uses \xintthebareval, ... which strangely were not available at 1.1 time. This spares some tokens from \XINT_expr_seq:_d and cousins. Also now variables have changed their mode of operation they pick only one token which must be an already encapsulated value.

In \XINT_allexpr_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>}.

```

1433 \def\XINT_expr_func_seqx #1#2{\XINT_allexpr_seqx \xintthebareval}%
1434 \def\XINT_flexpr_func_seqx #1#2{\XINT_allexpr_seqx \xintthebarefloateval}%
1435 \def\XINT_iexpr_func_seqx #1#2{\XINT_allexpr_seqx \xintthebareiieval}%
1436 \def\XINT_allexpr_seqx #1#2#3#4%
1437 {%
1438   \expandafter \XINT_expr_getop
1439   \csname .=\expandafter\XINT_expr_seq:_aa
1440     \romannumeral`&&\XINT_expr_unlock #2!{#1#4\relax !#3}\endcsname
1441 }%
1442 \def\XINT_expr_seq:_aa #1{\if +#1\expandafter\XINT_expr_seq:_A\else
1443   \expandafter\XINT_expr_seq:_a\fi #1}%

```

10.41.8 Evaluation over list, \XINT_expr_seq:_a with break, abort, omit

The #2 here is \...bareeval <expression>\relax !<variable name>. The #1 is a comma separated list of values to assign to the dummy variable. The \XINT_expr_seq_empty? intervenes immediately after handling of firstvalue.

1.2c has rewritten to a large extent this and other similar loops because the dummy variables now fetch a single encapsulated token (apart from a good means to lose a few hours needlessly -- as I have had to rewrite and review most everything, this change could make the thing more efficient if the same variable is used many times in an expression, but we are talking micro-seconds here anyhow.)

```

1444 \def\XINT_expr_seq:_a #1!#2{\expandafter\XINT_expr_seq_empty?
1445   \romannumeral0\XINT_expr_seq:_b {#2}#1,^,%}
1446 \def\XINT_expr_seq:_b #1#2#3,{%
1447   \if ,#2\xint_dothis\XINT_expr_seq:_noop\fi
1448   \if ^#2\xint_dothis\XINT_expr_seq:_end\fi
1449   \xint_orthat{\expandafter\XINT_expr_seq:_c}\csname.=#2#3\endcsname {#1}%
1450 }%
1451 \def\XINT_expr_seq:_noop\csname.=,#1\endcsname #2{\XINT_expr_seq:_b {#2}#1,%}
1452 \def\XINT_expr_seq:_end \csname.=^\endcsname #1{%}
1453 \def\XINT_expr_seq:_c #1#2{\expandafter\XINT_expr_seq:_d\romannumeral`&&@#2#1{#2}}%
1454 \def\XINT_expr_seq:_d #1{\if #1^\xint_dothis\XINT_expr_seq:_abort\fi
1455   \if #1?\xint_dothis\XINT_expr_seq:_break\fi
1456   \if #1!\xint_dothis\XINT_expr_seq:_omit\fi
1457   \xint_orthat{\XINT_expr_seq:_goon #1}}%
1458 \def\XINT_expr_seq:_abort #1!#2#3#4#5^,{,%}
1459 \def\XINT_expr_seq:_break #1!#2#3#4#5^,{,#1}%
1460 \def\XINT_expr_seq:_omit #1!#2#3#4{\XINT_expr_seq:_b {#4}}%
1461 \def\XINT_expr_seq:_goon #1!#2#3#4{,#1\XINT_expr_seq:_b {#4}}%

```

If all is omitted or list is empty, _empty? will fetch within the #1 a \endcsname token and construct "nil" via <space>\endcsname, if not #1 will be a comma and the gobble will swallow the space token and the extra \endcsname.

```

1462 \def\XINT_expr_seq_empty? #1{%
1463 \def\XINT_expr_seq_empty? ##1{\if ,##1\expandafter\xint_gobble_i\fi #1\endcsname }}%
1464 \XINT_expr_seq_empty? { }%

```

10.41.9 Evaluation over ++ generated lists with \XINT_expr_seq:_A

This is for index lists generated by n++. The starting point will have been replaced by its ceil (added: in fact with version 1.1. the ceil was not yet evaluated, but `_var_<letter>` did an expansion of what they fetch). We use `\numexpr` rather than `\xintInc`, hence the indexing is limited to small integers.

The 1.2c version of n++ produces a #1 here which is already a single `\.=<value>` token.

```

1465 \def\XINT_expr_seq:_A +#1!%
1466   {\expandafter\XINT_expr_seq_empty?\romannumeral0\XINT_expr_seq:_D #1}%
1467 \def\XINT_expr_seq:_D #1#2{\expandafter\XINT_expr_seq:_E\romannumeral`&&@#2#1{#2}}%
1468 \def\XINT_expr_seq:_E #1{\if #1^\xint_dothis\XINT_expr_seq:_Abort\fi
1469   \if #1?\xint_dothis\XINT_expr_seq:_Break\fi
1470   \if #1!\xint_dothis\XINT_expr_seq:_Omit\fi
1471   \xint_orthat{\XINT_expr_seq:_Goon #1}}%
1472 \def\XINT_expr_seq:_Abort #1!#2#3#4{}%
1473 \def\XINT_expr_seq:_Break #1!#2#3#4{,#1}%
1474 \def\XINT_expr_seq:_Omit #1!#2#3%
1475   {\expandafter\XINT_expr_seq:_D
1476     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
1477 \def\XINT_expr_seq:_Goon #1!#2#3%
1478   {,#1\expandafter\XINT_expr_seq:_D
1479     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%

```

10.42 add, mul

1.2c uses more directly the `\xintiiAdd` etc... macros and has `opxadd/opxmul` rather than a single `opx`. This is less conceptual as I use explicitly the associated macro names for +, * but this makes other things more efficient, and the code more readable.

```

1480 \def\XINT_expr_onlitteral_add
1481   {\expandafter\XINT_expr_onlitteral_add_f\romannumeral`&&\XINT_expr_onlitteral_seq_a {}}%
1482 \def\XINT_expr_onlitteral_add_f #1#2{\xint_c_xviii `{opxadd}#2)\relax #1}%
1483 \def\XINT_expr_onlitteral_mul
1484   {\expandafter\XINT_expr_onlitteral_mul_f\romannumeral`&&\XINT_expr_onlitteral_seq_a {}}%
1485 \def\XINT_expr_onlitteral_mul_f #1#2{\xint_c_xviii `{opxmul}#2)\relax #1}%

```

10.42.1 \XINT_expr_func_opxadd, \XINT_flexpr_func_opxadd, \XINT_iexpr_func_opxadd and same for mul

modified 1.2c.

```

1486 \def\XINT_expr_func_opxadd #1#2{\XINT_allexpr_opx \xintbareeval {\xintAdd 0}}%
1487 \def\XINT_flexpr_func_opxadd #1#2{\XINT_allexpr_opx \xintbarefloateval {\XINTinFloatAdd 0}}%
1488 \def\XINT_iexpr_func_opxadd #1#2{\XINT_allexpr_opx \xintbareiieval {\xintiiAdd 0}}%
1489 \def\XINT_expr_func_opxmul #1#2{\XINT_allexpr_opx \xintbareeval {\xintMul 1}}%
1490 \def\XINT_flexpr_func_opxmul #1#2{\XINT_allexpr_opx \xintbarefloateval {\XINTinFloatMul 1}}%
1491 \def\XINT_iexpr_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

```
1492 \def\XINT_allexpr_opx #1#2#3#4#5%
1493 {%
1494   \expandafter\XINT_expr_getop
1495   \csname.=\romannumeral`&&\expandafter\XINT_expr_op:_a
1496   \romannumeral`&&\XINT_expr_unlock #3!{#1#5\relax !#4}{#2}\endcsname
1497 }%
1498 \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.

```
1499 \def\XINT_expr_op:_b #1#2#3#4#5,{%
1500   \if ,#4\xint_dothis\XINT_expr_op:_noop\fi
1501   \if ^#4\xint_dothis\XINT_expr_op:_end\fi
1502   \xint_orthat{\expandafter\XINT_expr_op:_c}\csname.=#4#5\endcsname {#3}#1{#2}%
1503 }%
1504 \def\XINT_expr_op:_c #1#2#3#4{\expandafter\XINT_expr_op:_d\romannumeral0#2#1#3{#4}{#2}}%
1505 \def\XINT_expr_op:_d #1!#2#3#4#5%
1506   {\expandafter\XINT_expr_op:_b\expandafter #4\expandafter
1507     {\romannumeral`&&#4{\XINT_expr_unlock#1}{#5}}}%

```

The replacement text had `expr_seq:_b` rather than `expr_op:_b` due to a left-over from copy-paste. This made add and mul fail with an empty range for the variable (or "nil" in the list of values). Fixed in 1.2h.

```
1508 \def\XINT_expr_op:_noop\csname.=,#1\endcsname #2#3#4{\XINT_expr_op:_b #3{#4}{#2}#1,}%
1509 \def\XINT_expr_op:_end \csname.=^{\endcsname #1#2#3{#3}%

```

10.43 subs

Got simpler with 1.2c as now the dummy variable fetches an already encapsulated value, which is anyhow the form in which we get it.

```
1510 \def\XINT_expr_onlitteral_subs
1511   {\expandafter\XINT_expr_onlitteral_subs_f\romannumeral`&&\XINT_expr_onlitteral_seq_a {}}%
1512 \def\XINT_expr_onlitteral_subs_f #1#2{\xint_c_xviii `{subx}#2)\relax #1}%
1513 \def\XINT_expr_func_subx #1#2{\XINT_allexpr_subx \xintbareeval }%
1514 \def\XINT_flexpr_func_subx #1#2{\XINT_allexpr_subx \xintbarefloateval}%
1515 \def\XINT_iiepr_func_subx #1#2{\XINT_allexpr_subx \xintbareiieval }%
1516 \def\XINT_allexpr_subx #1#2#3#4% #2 is the value to assign to the dummy variable
1517 {% #3 is the dummy variable, #4 is the expression to evaluate
1518   \expandafter\expandafter\expandafter\XINT_expr_getop
1519   \expandafter\XINT_expr_subx:_end\romannumeral0#1#4\relax !#3#2%
1520 }%
1521 \def\XINT_expr_subx:_end #1!#2#3{#1}%

```

10.44 rseq

10.44.1	\XINT_expr_rseqx	...	313
10.44.2	\XINT_expr_rseqy	...	313
10.44.3	\XINT_expr_rseq:_a etc...	...	313

10.44.4 `\XINT_expr_rseq:_A` etc... 314

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...

```

1522 \def\XINT_expr_func_rseq {\XINT_allexpr_rseq \xintbareeval \xintthebareeval }%
1523 \def\XINT_flexpr_func_rseq {\XINT_allexpr_rseq \xintbarefloateval \xintthebarefloateval }%
1524 \def\XINT_iexpr_func_rseq {\XINT_allexpr_rseq \xintbareiieval \xintthebareiieval }%
1525 \def\XINT_allexpr_rseq #1#2#3%
1526 {%
1527   \expandafter\XINT_expr_rseqx\expandafter #1\expandafter#2\expandafter
1528   #3\romannumeral`&&\XINT_expr_onlitteral_seq_a }%
1529 }%
```

10.44.1 `\XINT_expr_rseqx`

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

```

1530 \def\XINT_expr_rseqx #1#2#3#4#5%
1531 {%
1532   \expandafter\XINT_expr_rseq\romannumeral0#1(#5)\relax #3#4#2%
1533 }%
```

10.44.2 `\XINT_expr_rseqy`

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

```

1534 \def\XINT_expr_rseqy #1#2#3#4#5%
1535 {%
1536   \expandafter \XINT_expr_getop
1537   \csname .=\XINT_expr_unlock #2%
1538   \expandafter\XINT_expr_rseq:_aa
1539   \romannumeral`&&\XINT_expr_unlock #1!{#5#4\relax !#3}#2\endcsname
1540 }%
1541 \def\XINT_expr_rseq:_aa #1{\if +#1\expandafter\XINT_expr_rseq:_A\else
1542   \expandafter\XINT_expr_rseq:_a\fi #1}%

```

10.44.3 `\XINT_expr_rseq:_a` etc...

```

1543 \def\XINT_expr_rseq:_a #1!#2#3{\XINT_expr_rseq:_b {#3}{#2}#1,^,%}
1544 \def\XINT_expr_rseq:_b #1#2#3#4,{%
1545   \if ,#3\xint_dothis\XINT_expr_rseq:_noop\fi
1546   \if ^#3\xint_dothis\XINT_expr_rseq:_end\fi
1547   \xint_orthat{\expandafter\XINT_expr_rseq:_c}\csname.=#3#4\endcsname
1548   {#1}{#2}%
1549 }%
1550 \def\XINT_expr_rseq:_noop\csname.=#1\endcsname #2#3{\XINT_expr_rseq:_b {#2}{#3}#1,%}
1551 \def\XINT_expr_rseq:_end \csname.^=\endcsname #1#2{%}
1552 \def\XINT_expr_rseq:_c #1#2#3%
```

```

1553   {\expandafter\XINT_expr_rseq:_d\romannumeral`&&@#3#1~#2{#3}}%
1554 \def\XINT_expr_rseq:_d #1{%
1555   \if ^#1\xint_dothis\XINT_expr_rseq:_abort\fi
1556   \if ?#1\xint_dothis\XINT_expr_rseq:_break\fi
1557   \if !#1\xint_dothis\XINT_expr_rseq:_omit\fi
1558   \xint_orthat{\XINT_expr_rseq:_goon #1}}%
1559 \def\XINT_expr_rseq:_goon #1!#2#3~#4#5{,#1\expandafter\XINT_expr_rseq:_b
1560   \romannumeral0\XINT_expr_lockit {#1}{#5}}%
1561 \def\XINT_expr_rseq:_omit #1!#2#3~{\XINT_expr_rseq:_b }%
1562 \def\XINT_expr_rseq:_abort #1!#2#3~#4#5#6^,{ }%
1563 \def\XINT_expr_rseq:_break #1!#2#3~#4#5#6^,{,#1}%

```

10.44.4 \XINT_expr_rseq:_A etc...

n++ for rseq. With 1.2c dummy variables pick a single token.

```

1564 \def\XINT_expr_rseq:_A +#1!#2#3{\XINT_expr_rseq:_D #1#3{#2}}%
1565 \def\XINT_expr_rseq:_D #1#2#3%
1566   {\expandafter\XINT_expr_rseq:_E\romannumeral`&&@#3#1~#2{#3}}%
1567 \def\XINT_expr_rseq:_E #1{\if #1^\xint_dothis\XINT_expr_rseq:_Abort\fi
1568   \if #1?\xint_dothis\XINT_expr_rseq:_Break\fi
1569   \if #1!\xint_dothis\XINT_expr_rseq:_Omit\fi
1570   \xint_orthat{\XINT_expr_rseq:_Goon #1}}%
1571 \def\XINT_expr_rseq:_Goon #1!#2#3~#4#5%
1572   {,#1\expandafter\XINT_expr_rseq:_D
1573   \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1574   \romannumeral0\XINT_expr_lockit{#1}{#5}}%
1575 \def\XINT_expr_rseq:_Omit #1!#2#3~%#4#5%
1576   {\expandafter\XINT_expr_rseq:_D
1577   \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname }%
1578 \def\XINT_expr_rseq:_Abort #1!#2#3~#4#5{ }%
1579 \def\XINT_expr_rseq:_Break #1!#2#3~#4#5{,#1}%

```

10.45 iter

10.45.1	\XINT_expr_iterx	...	315
10.45.2	\XINT_expr_itory	...	315
10.45.3	\XINT_expr_iter:_a etc...	...	315
10.45.4	\XINT_expr_iter:_A etc...	...	315

Prior to 1.2g, the iter keyword was what is now called iterr, analogous with rrseq. Somehow I forgot an iter functioning like rseq with the sole difference of printing only the last iteration. Both rseq and iter work well with list selectors, as @ refers to the whole comma separated sequence of the initial values. I have thus deliberately done the backwards incompatible renaming of iter to iterr, and the new iter.

```

1580 \def\XINT_expr_func_iter {\XINT_allexpr_iter \xintbareeval \xintthebareeval }%
1581 \def\XINT_flexpr_func_iter {\XINT_allexpr_iter \xintbarefloateval \xintthebarefloateval }%
1582 \def\XINT_iexpr_func_iter {\XINT_allexpr_iter \xintbareiieval \xintthebareiieval }%
1583 \def\XINT_allexpr_iter #1#2#3%
1584 {%
1585   \expandafter\XINT_expr_iterx\expandafter #1\expandafter#2\expandafter
1586   #3\romannumeral`&&\XINT_expr_onlitteral_seq_a { }%
1587 }%

```

10.45.1 `\XINT_expr_iterx`

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

```
1588 \def\XINT_expr_iterx #1#2#3#4#5%
1589 {%
1590   \expandafter\XINT_expr_itery\romannumeral0#1(#5)\relax #3#4#2%
1591 }%
```

10.45.2 `\XINT_expr_itery`

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

```
1592 \def\XINT_expr_itery #1#2#3#4#5%
1593 {%
1594   \expandafter \XINT_expr_getop
1595   \csname .=%
1596   \expandafter\XINT_expr_iter:_aa
1597   \romannumeral`&&@\XINT_expr_unlock #1!{#5#4\relax !#3}#2\endcsname
1598 }%
1599 \def\XINT_expr_iter:_aa #1{ \if +#1\expandafter\XINT_expr_iter:_A\else
1600   \expandafter\XINT_expr_iter:_a\fi #1}%

```

10.45.3 `\XINT_expr_iter:_a` etc...

```
1601 \def\XINT_expr_iter:_a #1!#2#3{\XINT_expr_iter:_b {#3}{#2}#1,^,%}
1602 \def\XINT_expr_iter:_b #1#2#3#4,{%
1603   \if ,#3\xint_dothis\XINT_expr_iter:_noop\fi
1604   \if ^#3\xint_dothis\XINT_expr_iter:_end\fi
1605   \xint_orthat{\expandafter\XINT_expr_iter:_c}%
1606   \csname.=#3#4\endcsname {#1}{#2}%
1607 }%
1608 \def\XINT_expr_iter:_noop\csname.=#1\endcsname #2#3{\XINT_expr_iter:_b {#2}{#3}#1,%}
1609 \def\XINT_expr_iter:_end \csname.=`\endcsname #1#2{\XINT_expr:_unlock #1}%
1610 \def\XINT_expr_iter:_c #1#2#3%
1611   {\expandafter\XINT_expr_iter:_d\romannumeral`&&@#3#1~#2{#3}}%
1612 \def\XINT_expr_iter:_d #1{%
1613   \if ^#1\xint_dothis\XINT_expr_iter:_abort\fi
1614   \if ?#1\xint_dothis\XINT_expr_iter:_break\fi
1615   \if !#1\xint_dothis\XINT_expr_iter:_omit\fi
1616   \xint_orthat{\XINT_expr_iter:_goon #1}%
1617 \def\XINT_expr_iter:_goon #1!#2#3~#4#5%
1618   {\expandafter\XINT_expr_iter:_b\romannumeral0\XINT_expr_lockit {#1}{#5}}%
1619 \def\XINT_expr_iter:_omit #1!#2#3~{\XINT_expr_iter:_b }%
1620 \def\XINT_expr_iter:_abort #1!#2#3~#4#5#6^,{\XINT_expr_unlock #4}%
1621 \def\XINT_expr_iter:_break #1!#2#3~#4#5#6^,{#1}%

```

10.45.4 `\XINT_expr_iter:_A` etc...

`n++` for iter. With 1.2c dummy variables pick a single token.

```
1622 \def\XINT_expr_iter:_A +#1!#2#3{\XINT_expr_iter:_D #1#3{#2}}%
1623 \def\XINT_expr_iter:_D #1#2#3%
```

```

1624 {\expandafter\XINT_expr_iter:_E\romannumeral`&&@#3#1~#2{#3}}%
1625 \def\XINT_expr_iter:_E #1{\if #1^\xint_dothis\XINT_expr_iter:_Abort\fi
1626         \if #1?\xint_dothis\XINT_expr_iter:_Break\fi
1627         \if #1!\xint_dothis\XINT_expr_iter:_Omit\fi
1628         \xint_orthat{\XINT_expr_iter:_Goon #1}}%
1629 \def\XINT_expr_iter:_Goon #1!#2#3~#4#5%
1630     {\expandafter\XINT_expr_iter:_D
1631     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1632     \romannumeral0\XINT_expr_lockit{#1}{#5}}%
1633 \def\XINT_expr_iter:_Omit #1!#2#3~%#4#5%
1634     {\expandafter\XINT_expr_iter:_D
1635     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname }%
1636 \def\XINT_expr_iter:_Abort #1!#2#3~#4#5{\XINT_expr:_unlock #4}%
1637 \def\XINT_expr_iter:_Break #1!#2#3~#4#5{#1}%

```

10.46 rrseq

10.46.1	\XINT_expr_rrseqx	316
10.46.2	\XINT_expr_rrseqy	316
10.46.3	\XINT_expr_rrseq:_a etc.	317
10.46.4	\XINT_expr_rrseq:_A etc.	317

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.

```

1638 \def\XINT_expr_func_rrseq {\XINT_allexpr_rrseq \xintbareeval \xintthebareeval }%
1639 \def\XINT_flexpr_func_rrseq {\XINT_allexpr_rrseq \xintbarefloateval \xintthebarefloateval }%
1640 \def\XINT_iexpr_func_rrseq {\XINT_allexpr_rrseq \xintbareiieval \xintthebareiieval }%
1641 \def\XINT_allexpr_rrseq #1#2#3%
1642 {%
1643     \expandafter\XINT_expr_rrseqx\expandafter #1\expandafter#2\expandafter
1644     #3\romannumeral`&&@\XINT_expr_onlitteral_seq_a {}%
1645 }%

```

10.46.1 \XINT_expr_rrseqx

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

```

1646 \def\XINT_expr_rrseqx #1#2#3#4#5%
1647 {%
1648     \expandafter\XINT_expr_rrseqy\romannumeral0#1(#5)\expandafter\relax
1649     \expandafter{\romannumeral0\xintapply \XINT_expr_lockit
1650         {\xintRevWithBraces{\xintCSVtoListNonStripped{\XINT_expr_unlock #3}}}%
1651         #3#4#2%
1652 }%

```

10.46.2 \XINT_expr_rrseqy

`#1`=valeurs pour variable (locked), `#2`=initial values (reversed, one (braced) token each) `#3`=toutes les valeurs initiales (csv,locked), `#4`=variable, `#5`=expr, `#6`=\xintthebareeval ou \xintthebarefloateval ou \xintthebareiieval

```

1653 \def\XINT_expr_rrseqy #1#2#3#4#5#6%

```

```

1654 {%
1655   \expandafter \XINT_expr_getop
1656   \csname .=\XINT_expr_unlock #3%
1657   \expandafter\XINT_expr_rrseq:_aa
1658       \romannumeral`&&@\XINT_expr_unlock #1!{#6#5\relax !#4}{#2}\endcsname
1659 }%
1660 \def\XINT_expr_rrseq:_aa #1{\if +#1\expandafter\XINT_expr_rrseq:_A\else
1661     \expandafter\XINT_expr_rrseq:_a\fi #1}%

```

10.46.3 \XINT_expr_rrseq:_a etc...

Attention que ? a catcode 3 ici et dans iter.

```

1662 \catcode`? 3
1663 \def\XINT_expr_rrseq:_a #1!#2#3{\XINT_expr_rrseq:_b {#3}{#2}#1,^,%}
1664 \def\XINT_expr_rrseq:_b #1#2#3#4,{%
1665   \if ,#3\xint_dothis\XINT_expr_rrseq:_noop\fi
1666   \if ^#3\xint_dothis\XINT_expr_rrseq:_end\fi
1667   \xint_orthat{\expandafter\XINT_expr_rrseq:_c}\csname .=#3#4\endcsname
1668   {#1}{#2}%
1669 }%
1670 \def\XINT_expr_rrseq:_noop\csname .=#1\endcsname #2#3{\XINT_expr_rrseq:_b {#2}{#3}#1,%}
1671 \def\XINT_expr_rrseq:_end \csname .=#1\endcsname #1#2{%}
1672 \def\XINT_expr_rrseq:_c #1#2#3%
1673   {\expandafter\XINT_expr_rrseq:_d\romannumeral`&&@#3#1~#2?{#3}}%
1674 \def\XINT_expr_rrseq:_d #1{%
1675   \if ^#1\xint_dothis\XINT_expr_rrseq:_abort\fi
1676   \if ?#1\xint_dothis\XINT_expr_rrseq:_break\fi
1677   \if !#1\xint_dothis\XINT_expr_rrseq:_omit\fi
1678   \xint_orthat{\XINT_expr_rrseq:_goon #1}%
1679 }%
1680 \def\XINT_expr_rrseq:_goon #1!#2#3~#4?#5{,#1\expandafter\XINT_expr_rrseq:_b\expandafter
1681   {\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}{#4}}{#5}}%
1682 \def\XINT_expr_rrseq:_omit #1!#2#3~{\XINT_expr_rrseq:_b }%
1683 \def\XINT_expr_rrseq:_abort #1!#2#3~#4?#5#6^,{}%
1684 \def\XINT_expr_rrseq:_break #1!#2#3~#4?#5#6^,{,#1}%

```

10.46.4 \XINT_expr_rrseq:_A etc...

n++ for rrseq. With 1.2C, the #1 in \XINT_expr_rrseq:_A is a single token.

```

1685 \def\XINT_expr_rrseq:_A +#1!#2#3{\XINT_expr_rrseq:_D #1{#3}{#2}}%
1686 \def\XINT_expr_rrseq:_D #1#2#3%
1687   {\expandafter\XINT_expr_rrseq:_E\romannumeral`&&@#3#1~#2?{#3}}%
1688 \def\XINT_expr_rrseq:_Goon #1!#2#3~#4?#5%
1689   {,#1\expandafter\XINT_expr_rrseq:_D
1690     \csname .=#1\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1691     \expandafter{\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}{#4}}{#5}}%
1692 \def\XINT_expr_rrseq:_Omit #1!#2#3~%#4?#5%
1693   {\expandafter\XINT_expr_rrseq:_D
1694     \csname .=#1\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
1695 \def\XINT_expr_rrseq:_Abort #1!#2#3~#4?#5{%}
1696 \def\XINT_expr_rrseq:_Break #1!#2#3~#4?#5{,#1}%

```

```

1697 \def\XINT_expr_rrseq:_E #1{\if #1^\xint_dothis\XINT_expr_rrseq:_Abort\fi
1698           \if #1?\xint_dothis\XINT_expr_rrseq:_Break\fi
1699           \if #1!\xint_dothis\XINT_expr_rrseq:_Omit\fi
1700           \xint_orthat{\XINT_expr_rrseq:_Goon #1}}%

```

10.47 iterr

10.47.1	<code>\XINT_expr_iterrx</code>	318
10.47.2	<code>\XINT_expr_iterry</code>	318
10.47.3	<code>\XINT_expr_iterr:_a</code> etc...	318
10.47.4	<code>\XINT_expr_iterr:_A</code> etc...	319

```

1701 \def\XINT_expr_func_iterr {\XINT_allexpr_iterr \xintbareeval \xintthebareeval }%
1702 \def\XINT_flexpr_func_iterr {\XINT_allexpr_iterr \xintbarefloateval \xintthebarefloateval }%
1703 \def\XINT_iexpr_func_iterr {\XINT_allexpr_iterr \xintbareiieval \xintthebareiieval }%
1704 \def\XINT_allexpr_iterr #1#2#3%
1705 {%
1706   \expandafter\XINT_expr_iterrx\expandafter #1\expandafter #2\expandafter
1707   #3\romannumeral`&&\XINT_expr_onlitteral_seq_a {}%
1708 }%

```

10.47.1 `\XINT_expr_iterrx`

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

```

1709 \def\XINT_expr_iterrx #1#2#3#4#5%
1710 {%
1711   \expandafter\XINT_expr_iterry\romannumeral0#1(#5)\expandafter\relax
1712   \expandafter{\romannumeral0\xintapply \XINT_expr_lockit
1713     {\xintRevWithBraces{\xintCSVtoListNonStripped{\XINT_expr_unlock #3}}}}%
1714   #3#4#2%
1715 }%

```

10.47.2 `\XINT_expr_iterry`

#1=valeurs pour variable (locked), #2=initial values (reversed, one (braced) token each) #3=toutes les valeurs initiales (csv,locked), #4=variable, #5=expr, #6=\xintbareeval ou \xintbarefloateval ou \xintbareiieval

```

1716 \def\XINT_expr_iterry #1#2#3#4#5#6%
1717 {%
1718   \expandafter \XINT_expr_getop
1719   \csname .=%
1720   \expandafter\XINT_expr_iterr:_aa
1721   \romannumeral`&&\XINT_expr_unlock #1!{#6#5\relax !#4}{#2}\endcsname
1722 }%
1723 \def\XINT_expr_iterr:_aa #1{\if +#1\expandafter\XINT_expr_iterr:_A\else
1724   \expandafter\XINT_expr_iterr:_a\fi #1}%

```

10.47.3 `\XINT_expr_iterr:_a` etc...

```

1725 \def\XINT_expr_iterr:_a #1!#2#3{\XINT_expr_iterr:_b {#3}{#2}#1,^,%
1726 \def\XINT_expr_iterr:_b #1#2#3#4,{%

```

```

1727 \if ,#3\xint_dothis\XINT_expr_iterr:_noop\fi
1728 \if ^#3\xint_dothis\XINT_expr_iterr:_end\fi
1729 \xint_orthat{\expandafter\XINT_expr_iterr:_c}%
1730 \csname.=#3#4\endcsname {#1}{#2}%
1731 }%
1732 \def\XINT_expr_iterr:_noop\csname.=,#1\endcsname #2#3{\XINT_expr_iterr:_b {#2}{#3}#1,}%
1733 \def\XINT_expr_iterr:_end \csname.=^ \endcsname #1#2%
1734 {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1735   {\XINT_expr:_unlock}{\xintReverseOrder{#1\space}}}%
1736 \def\XINT_expr_iterr:_c #1#2#3%
1737 {\expandafter\XINT_expr_iterr:_d\romannumeral`&&@#3#1~#2?{#3}}%
1738 \def\XINT_expr_iterr:_d #1{%
1739   \if ^#1\xint_dothis\XINT_expr_iterr:_abort\fi
1740   \if ?#1\xint_dothis\XINT_expr_iterr:_break\fi
1741   \if !#1\xint_dothis\XINT_expr_iterr:_omit\fi
1742   \xint_orthat{\XINT_expr_iterr:_goon #1}%
1743 }%
1744 \def\XINT_expr_iterr:_goon #1!#2#3~#4?#5{\expandafter\XINT_expr_iterr:_b\expandafter
1745   {\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
1746 \def\XINT_expr_iterr:_omit #1!#2#3~{\XINT_expr_iterr:_b }%
1747 \def\XINT_expr_iterr:_abort #1!#2#3~#4?#5#6^,%
1748 {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1749   {\XINT_expr:_unlock}{\xintReverseOrder{#4\space}}}%
1750 \def\XINT_expr_iterr:_break #1!#2#3~#4?#5#6^,%
1751 {\expandafter\xint_gobble_iv\romannumeral0\xintapplyunbraced
1752   {\XINT_expr:_unlock}{\xintReverseOrder{#4\space}},#1}%
1753 \def\XINT_expr:_unlock #1{\XINT_expr_unlock #1}%

```

10.47.4 \XINT_expr_iterr:_A etc...

n++ for *iterr.* ? is of catcode 3 here.

```

1754 \def\XINT_expr_iterr:_A +#1!#2#3{\XINT_expr_iterr:_D #1{#3}{#2}}%
1755 \def\XINT_expr_iterr:_D #1#2#3%
1756 {\expandafter\XINT_expr_iterr:_E\romannumeral`&&@#3#1~#2?{#3}}%
1757 \def\XINT_expr_iterr:_Goon #1!#2#3~#4?#5%
1758 {\expandafter\XINT_expr_iterr:_D
1759   \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1760   \expandafter{\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
1761 \def\XINT_expr_iterr:_Omit #1!#2#3~#4?#5%
1762 {\expandafter\XINT_expr_iterr:_D
1763   \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
1764 \def\XINT_expr_iterr:_Abort #1!#2#3~#4?#5%
1765 {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1766   {\XINT_expr:_unlock}{\xintReverseOrder{#4\space}}}%
1767 \def\XINT_expr_iterr:_Break #1!#2#3~#4?#5%
1768 {\expandafter\xint_gobble_iv\romannumeral0\xintapplyunbraced
1769   {\XINT_expr:_unlock}{\xintReverseOrder{#4\space}},#1}%
1770 \def\XINT_expr_iterr:_E #1{\if #1^ \xint_dothis\XINT_expr_iterr:_Abort\fi
1771   \if #1? \xint_dothis\XINT_expr_iterr:_Break\fi
1772   \if #1! \xint_dothis\XINT_expr_iterr:_Omit\fi
1773   \xint_orthat{\XINT_expr_iterr:_Goon #1}}%
1774 \catcode`? 11

```

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

10.48.1	<code>\xintANDof:csv</code>	320
10.48.2	<code>\xintORof:csv</code>	320
10.48.3	<code>\xintXORof:csv</code>	320
10.48.4	Generic csv routine	321
10.48.5	<code>\xintMaxof:csv, \xintiiMaxof:csv</code>	321
10.48.6	<code>\xintMinof:csv, \xintiiMinof:csv</code>	321
10.48.7	<code>\xintSum:csv, \xintiiSum:csv</code>	321
10.48.8	<code>\xintPrd:csv, \xintiiPrd:csv</code>	322
10.48.9	<code>\xintGCDof:csv, \xintLCMof:csv</code>	322
10.48.10	<code>\xintiiGCDof:csv, \xintiiLCMof:csv</code>	322
10.48.11	<code>\XINTinFloatdigits, \XINTinFloatSqrtdigits, \XINTinFloatFacdigits</code>	322
10.48.12	<code>\XINTinFloatMaxof:csv, \XINTinFloatMinof:csv</code>	322
10.48.13	<code>\XINTinFloatSum:csv, \XINTinFloatPrd:csv</code>	322

These macros are used inside `\csname...\endcsname`. These things are not initiated by a `\romannumeral` in general, but in some cases they are, especially when involved in an `\xintNewExpr`. They will then be protected against expansion and expand only later in contexts governed by an initial `\romannumeral-`0`. There each new item may need to be expanded, which would not be the case in the use for the `_func_` things.

1.2g adds (to be continued)

10.48.1 `\xintANDof:csv`

1.09a. For use by `\xintexpr` inside `\csname`. 1.1, je remplace `ifTrueAelseB` par `iiNotZero` pour des raisons d'optimisations.

```

1775 \def\xintANDof:csv #1{\expandafter\XINT_andof:_a\romannumeral`&&@#1,,^}%
1776 \def\XINT_andof:_a #1{\if ,#1\expandafter\XINT_andof:_e
1777     \else\expandafter\XINT_andof:_c\fi #1}%
1778 \def\XINT_andof:_c #1,{\xintiifNotZero {#1}{\XINT_andof:_a}{\XINT_andof:_no}}%
1779 \def\XINT_andof:_no #1^{0}%
1780 \def\XINT_andof:_e #1^{1}% works with empty list

```

10.48.2 `\xintORof:csv`

1.09a. For use by `\xintexpr`.

```

1781 \def\xintORof:csv #1{\expandafter\XINT_orof:_a\romannumeral`&&@#1,,^}%
1782 \def\XINT_orof:_a #1{\if ,#1\expandafter\XINT_orof:_e
1783     \else\expandafter\XINT_orof:_c\fi #1}%
1784 \def\XINT_orof:_c #1,{\xintiifNotZero{#1}{\XINT_orof:_yes}{\XINT_orof:_a}}%
1785 \def\XINT_orof:_yes #1^{1}%
1786 \def\XINT_orof:_e #1^{0}% works with empty list

```

10.48.3 `\xintXORof:csv`

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

```

1787 \def\xintXORof:csv #1{\expandafter\XINT_xorof:_a\expandafter 0\romannumeral`&&@#1,,^}%
1788 \def\XINT_xorof:_a #1#2,{\XINT_xorof:_b #2,#1}%

```



```

1789 \def\XINT_xorof:_b #1{\if ,#1\expandafter\XINT_xorof:_e
1790       \else\expandafter\XINT_xorof:_c\fi #1}%
1791 \def\XINT_xorof:_c #1,#2%
1792       {\xintiifNotZero {#1}{\if #20\xint_afterfi{\XINT_xorof:_a 1}%
1793       \else\xint_afterfi{\XINT_xorof:_a 0}\fi}%
1794       {\XINT_xorof:_a #2}%
1795       }%
1796 \def\XINT_xorof:_e ,#1#2^{#1}% allows empty list (then returns 0)

```

10.48.4 Generic csv routine

1.1. generic routine. up to the loss of some efficiency, especially for Sum:csv and Prod:csv, where \XINTinFloat will be done twice for each argument.

```

1797 \def\XINT_oncsv:_empty #1,^,#2{#2}%
1798 \def\XINT_oncsv:_end ^,#1#2#3#4{#1}%
1799 \def\XINT_oncsv:_a #1#2#3%
1800     {\if ,#3\expandafter\XINT_oncsv:_empty\else\expandafter\XINT_oncsv:_b\fi #1#2#3}%
1801 \def\XINT_oncsv:_b #1#2#3,%
1802     {\expandafter\XINT_oncsv:_c \expandafter{\romannumeral`&&@#2{#3}}#1#2}%
1803 \def\XINT_oncsv:_c #1#2#3#4,{\expandafter\XINT_oncsv:_d \romannumeral`&&@#4,{#1}#2#3}%
1804 \def\XINT_oncsv:_d #1%
1805     {\if ^#1\expandafter\XINT_oncsv:_end\else\expandafter\XINT_oncsv:_e\fi #1}%
1806 \def\XINT_oncsv:_e #1,#2#3#4%
1807     {\expandafter\XINT_oncsv:_c\expandafter {\romannumeral`&&@#3{#4{#1}}{#2}}#3#4}%

```

10.48.5 \xintMaxof:csv, \xintiiMaxof:csv

1.09i. Rewritten for 1.1. Compatible avec liste vide donnant valeur par défaut. Pas compatible avec items manquants. ah je m'aperçois au dernier moment que je n'ai pas en effet de \xintiiMax. Je devrais le rajouter. En tout cas ici c'est uniquement pour xintiiexpr, dans il faut bien sûr ne pas faire de xintNum, donc il faut un iimax.

```

1808 \def\xintMaxof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmax
1809       \expandafter\xint_firstofone\romannumeral`&&@#1,^{,0/1[0]}}}%
1810 \def\xintiiMaxof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiimax
1811       \expandafter\xint_firstofone\romannumeral`&&@#1,^{,0}%

```

10.48.6 \xintMinof:csv, \xintiiMinof:csv

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

```

1812 \def\xintMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmin
1813       \expandafter\xint_firstofone\romannumeral`&&@#1,^{,0/1[0]}}}%
1814 \def\xintiiMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiimin
1815       \expandafter\xint_firstofone\romannumeral`&&@#1,^{,0}%

```

10.48.7 \xintSum:csv, \xintiiSum:csv

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

```

1816 \def\xintSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintadd
1817       \expandafter\xint_firstofone\romannumeral`&&@#1,^{,0/1[0]}}}%
1818 \def\xintiiSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiadd
1819       \expandafter\xint_firstofone\romannumeral`&&@#1,^{,0}%

```

10.48.8 `\xintPrd:csv`, `\xintiiPrd:csv`

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

```
1820 \def\xintPrd:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintmul
1821         \expandafter\xint_firstofone\romannumeral`&&@#1,^,{1/1[0]}}}%
1822 \def\xintiiPrd:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintiimul
1823         \expandafter\xint_firstofone\romannumeral`&&@#1,^,1}%

```

10.48.9 `\xintGCDof:csv`, `\xintLCMof:csv`

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

```
1824 \def\xintGCDof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintgcd
1825         \expandafter\xint_firstofone\romannumeral`&&@#1,^,1}%
1826 \def\xintLCMof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintlcm
1827         \expandafter\xint_firstofone\romannumeral`&&@#1,^,0}%

```

10.48.10 `\xintiiGCDof:csv`, `\xintiiLCMof:csv`

1.1a pour `\xintiiexpr`. Ces histoires de ii sont pénibles à la fin.

```
1828 \def\xintiiGCDof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintiigcd
1829         \expandafter\xint_firstofone\romannumeral`&&@#1,^,1}%
1830 \def\xintiiLCMof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintiilcm
1831         \expandafter\xint_firstofone\romannumeral`&&@#1,^,0}%

```

10.48.11 `\XINTinFloatdigits`, `\XINTinFloatSqrtdigits`, `\XINTinFloatFacdigits`

for `\xintNewExpr` matters, mainly.

```
1832 \def\XINTinFloatdigits {\XINTinFloat [\XINTdigits]]}%
1833 \def\XINTinFloatSqrtdigits {\XINTinFloatSqrt[\XINTdigits]]}%
1834 \def\XINTinFloatFacdigits {\XINTinFloatFac [\XINTdigits]]}%

```

10.48.12 `\XINTinFloatMaxof:csv`, `\XINTinFloatMinof:csv`

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

```
1835 \def\XINTinFloatMaxof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintmax
1836         \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^,{0[0]}}}%
1837 \def\XINTinFloatMinof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintmin
1838         \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^,{0[0]}}}%

```

10.48.13 `\XINTinFloatSum:csv`, `\XINTinFloatPrd:csv`

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

```
1839 \def\XINTinFloatSum:csv #1{\expandafter\XINT_onscv:_a\expandafter\XINTinfloatadd
1840         \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^,{0[0]}}}%
1841 \def\XINTinFloatPrd:csv #1{\expandafter\XINT_onscv:_a\expandafter\XINTinfloatmul
1842         \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^,{1[0]}}}%

```

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

```

1843 \def\XINT_expr_twoargs #1,#2,{\{#1\}{#2}}%
1844 \def\XINT_expr_argadopt #1,#2,#3.#4#5%
1845 {%
1846   \if\relax#3\relax\expandafter\xint_firstoftwo\else
1847     \expandafter\xint_secondoftwo\fi
1848   {\#4}{\#5[\xintNum {\#2}]}{\#1}%
1849 }%
1850 \def\XINT_expr_oneortwo #1#2#3,#4,#5.%
1851 {%
1852   \if\relax#5\relax\expandafter\xint_firstoftwo\else
1853     \expandafter\xint_secondoftwo\fi
1854   {\#1{0}}{\#2{\xintNum {\#4}}}{\#3}%
1855 }%
1856 \def\XINT_iiexpr_oneortwo #1#2,#3,#4.%
1857 {%
1858   \if\relax#4\relax\expandafter\xint_firstoftwo\else
1859     \expandafter\xint_secondoftwo\fi
1860   {\#1{0}}{\#1{\#3}}{\#2}%
1861 }%
1862 \def\XINT_expr_func_num #1#2#3%
1863   {\expandafter #1\expandafter #2\csname.=\xintNum {\XINT_expr_unlock #3}\endcsname }%
1864 \let\XINT_flexpr_func_num\XINT_expr_func_num
1865 \let\XINT_iiexpr_func_num\XINT_expr_func_num
1866 % [0] added Oct 25. For interaction with SPRaw::csv
1867 \def\XINT_expr_func_reduce #1#2#3%
1868   {\expandafter #1\expandafter #2\csname.=\xintIrr {\XINT_expr_unlock #3}[0]\endcsname }%
1869 \let\XINT_flexpr_func_reduce\XINT_expr_func_reduce
1870 % no \XINT_iiexpr_func_reduce
1871 \def\XINT_expr_func_abs #1#2#3%
1872   {\expandafter #1\expandafter #2\csname.=\xintAbs {\XINT_expr_unlock #3}\endcsname }%
1873 \let\XINT_flexpr_func_abs\XINT_expr_func_abs
1874 \def\XINT_iiexpr_func_abs #1#2#3%
1875   {\expandafter #1\expandafter #2\csname.=\xintiiAbs {\XINT_expr_unlock #3}\endcsname }%
1876 \def\XINT_expr_func_sgn #1#2#3%
1877   {\expandafter #1\expandafter #2\csname.=\xintSgn {\XINT_expr_unlock #3}\endcsname }%
1878 \let\XINT_flexpr_func_sgn\XINT_expr_func_sgn
1879 \def\XINT_iiexpr_func_sgn #1#2#3%
1880   {\expandafter #1\expandafter #2\csname.=\xintiiSgn {\XINT_expr_unlock #3}\endcsname }%
1881 \def\XINT_expr_func_frac #1#2#3%
1882   {\expandafter #1\expandafter #2\csname.=\xintTFrac {\XINT_expr_unlock #3}\endcsname }%
1883 \def\XINT_flexpr_func_frac #1#2#3{\expandafter #1\expandafter #2\csname
1884   .=\XINTinFloatFracdigits {\XINT_expr_unlock #3}\endcsname }%

no \XINT_iiexpr_func_frac

1885 \def\XINT_expr_func_floor #1#2#3%
1886   {\expandafter #1\expandafter #2\csname .=\xintFloor {\XINT_expr_unlock #3}\endcsname }%
1887 \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.

```

1888 \def\xINT_iiexpr_func_floor #1#2#3%
1889 {%
1890   \expandafter #1\expandafter #2\csname.=\xintiFloor {\XINT_expr_unlock #3}\endcsname }%
1891 \def\xINT_expr_func_ceil #1#2#3%
1892   {\expandafter #1\expandafter #2\csname .=\xintCeil {\XINT_expr_unlock #3}\endcsname }%
1893 \let\xINT_flexpr_func_ceil\xINT_expr_func_ceil
1894 \def\xINT_iiexpr_func_ceil #1#2#3%
1895 {%
1896   \expandafter #1\expandafter #2\csname.=\xintiCeil {\XINT_expr_unlock #3}\endcsname }%
1897 \def\xINT_expr_func_sqr #1#2#3%
1898   {\expandafter #1\expandafter #2\csname.=\xintSqr {\XINT_expr_unlock #3}\endcsname }%
1899 \def\xINT_flexpr_func_sqr #1#2#3%
1900 {%
1901   \expandafter #1\expandafter #2\csname
1902     .=\XINTinFloatMul{\XINT_expr_unlock #3}{\XINT_expr_unlock #3}\endcsname
1903 }%
1904 \def\xINT_expr_func_factorial #1#2#3%
1905 {%
1906   \expandafter #1\expandafter #2\csname .=%
1907   \expandafter\xINT_expr_argandopt
1908   \romannumeral`&&\XINT_expr_unlock#3,,.\xintiFac\xINTinFloatFac
1909   \endcsname
1910 }%
1911 \def\xINT_flexpr_func_factorial #1#2#3%
1912 {%
1913   \expandafter #1\expandafter #2\csname .=%
1914   \expandafter\xINT_expr_argandopt
1915   \romannumeral`&&\XINT_expr_unlock#3,,.\XINTinFloatFacdigits\xINTinFloatFac
1916   \endcsname
1917 }%
1918 \def\xINT_iiexpr_func_factorial #1#2#3%
1919 {%
1920   \expandafter #1\expandafter #2\csname.=\xintiiFac{\XINT_expr_unlock #3}\endcsname
1921 }%
1922 \def\xINT_iiexpr_func_sqr #1#2#3%
1923   {\expandafter #1\expandafter #2\csname.=\xintiiSqr {\XINT_expr_unlock #3}\endcsname }%
1924 \def\xINT_expr_func_sqrt #1#2#3%
1925 {%
1926   \expandafter #1\expandafter #2\csname .=%
1927   \expandafter\xINT_expr_argandopt
1928   \romannumeral`&&\XINT_expr_unlock#3,,.\XINTinFloatSqrtdigits\xINTinFloatSqrt
1929   \endcsname
1930 }%
1931 \let\xINT_flexpr_func_sqrt\xINT_expr_func_sqrt
1932 \def\xINT_iiexpr_func_sqrt #1#2#3%
1933   {\expandafter #1\expandafter #2\csname.=\xintiiSqrt {\XINT_expr_unlock #3}\endcsname }%
1934 \def\xINT_iiexpr_func_sqrtr #1#2#3%
1935   {\expandafter #1\expandafter #2\csname.=\xintiiSqrtr {\XINT_expr_unlock #3}\endcsname }%
1936 \def\xINT_expr_func_round #1#2#3%
1937 {%

```

```

1938 \expandafter #1\expandafter #2\csname .=%
1939 \expandafter\XINT_expr_oneortwo
1940 \expandafter\xintiRound\expandafter\xintRound
1941 \romannumeral`&&\XINT_expr_unlock #3,,.\endcsname
1942 }%
1943 \let\XINT_flexpr_func_round\XINT_expr_func_round
1944 \def\XINT_iiexpr_func_round #1#2#3%
1945 {%
1946 \expandafter #1\expandafter #2\csname .=%
1947 \expandafter\XINT_iiexpr_oneortwo\expandafter\xintiRound
1948 \romannumeral`&&\XINT_expr_unlock #3,,.\endcsname
1949 }%
1950 \def\XINT_expr_func_trunc #1#2#3%
1951 {%
1952 \expandafter #1\expandafter #2\csname .=%
1953 \expandafter\XINT_expr_oneortwo
1954 \expandafter\xintiTrunc\expandafter\xintTrunc
1955 \romannumeral`&&\XINT_expr_unlock #3,,.\endcsname
1956 }%
1957 \let\XINT_flexpr_func_trunc\XINT_expr_func_trunc
1958 \def\XINT_iiexpr_func_trunc #1#2#3%
1959 {%
1960 \expandafter #1\expandafter #2\csname .=%
1961 \expandafter\XINT_iiexpr_oneortwo\expandafter\xintiTrunc
1962 \romannumeral`&&\XINT_expr_unlock #3,,.\endcsname
1963 }%
1964 \def\XINT_expr_func_float #1#2#3%
1965 {%
1966 \expandafter #1\expandafter #2\csname .=%
1967 \expandafter\XINT_expr_argadopt
1968 \romannumeral`&&\XINT_expr_unlock #3,,.\XINTinFloatdigits\XINTinFloat
1969 \endcsname
1970 }%
1971 \let\XINT_flexpr_func_float\XINT_expr_func_float
1972 % \XINT_iiexpr_func_float not defined
1973 \def\XINT_expr_func_mod #1#2#3%
1974 {%
1975 \expandafter #1\expandafter #2\csname .=%
1976 \expandafter\expandafter\expandafter\xintMod
1977 \expandafter\XINT_expr_twoargs
1978 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
1979 }%
1980 \def\XINT_flexpr_func_mod #1#2#3%
1981 {%
1982 \expandafter #1\expandafter #2\csname .=%
1983 \expandafter\XINTinFloatMod
1984 \romannumeral`&&\expandafter\XINT_expr_twoargs
1985 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
1986 }%
1987 \def\XINT_iiexpr_func_mod #1#2#3%
1988 {%
1989 \expandafter #1\expandafter #2\csname .=%

```

```

1990 \expandafter\expandafter\expandafter\xintiMod
1991 \expandafter\XINT_expr_twoargs
1992 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
1993 }%
1994 \def\XINT_expr_func_binomial #1#2#3%
1995 {%
1996 \expandafter #1\expandafter #2\csname .=%
1997 \expandafter\expandafter\expandafter\xintiBinomial
1998 \expandafter\XINT_expr_twoargs
1999 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2000 }%
2001 \def\XINT_flexpr_func_binomial #1#2#3%
2002 {%
2003 \expandafter #1\expandafter #2\csname .=%
2004 \expandafter\expandafter\expandafter\XINTinFloatBinomial
2005 \expandafter\XINT_expr_twoargs
2006 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2007 }%
2008 \def\XINT_iiexpr_func_binomial #1#2#3%
2009 {%
2010 \expandafter #1\expandafter #2\csname .=%
2011 \expandafter\expandafter\expandafter\xintiBinomial
2012 \expandafter\XINT_expr_twoargs
2013 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2014 }%
2015 \def\XINT_expr_func_pfactorial #1#2#3%
2016 {%
2017 \expandafter #1\expandafter #2\csname .=%
2018 \expandafter\expandafter\expandafter\xintiPFactorial
2019 \expandafter\XINT_expr_twoargs
2020 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2021 }%
2022 \def\XINT_flexpr_func_pfactorial #1#2#3%
2023 {%
2024 \expandafter #1\expandafter #2\csname .=%
2025 \expandafter\expandafter\expandafter\XINTinFloatPFactorial
2026 \expandafter\XINT_expr_twoargs
2027 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2028 }%
2029 \def\XINT_iiexpr_func_pfactorial #1#2#3%
2030 {%
2031 \expandafter #1\expandafter #2\csname .=%
2032 \expandafter\expandafter\expandafter\xintiPFactorial
2033 \expandafter\XINT_expr_twoargs
2034 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2035 }%
2036 \def\XINT_expr_func_quo #1#2#3%
2037 {%
2038 \expandafter #1\expandafter #2\csname .=%
2039 \expandafter\expandafter\expandafter\xintiQuo
2040 \expandafter\XINT_expr_twoargs
2041 \romannumeral`&&\XINT_expr_unlock #3,\endcsname

```

```

2042 }%
2043 \let\XINT_flexpr_func_quo\XINT_expr_func_quo
2044 \def\XINT_iiexpr_func_quo #1#2#3%
2045 {%
2046   \expandafter #1\expandafter #2\csname .=%
2047   \expandafter\expandafter\expandafter\xintiiQuo
2048   \expandafter\XINT_expr_twoargs
2049   \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2050 }%
2051 \def\XINT_expr_func_rem #1#2#3%
2052 {%
2053   \expandafter #1\expandafter #2\csname .=%
2054   \expandafter\expandafter\expandafter\xintiRem
2055   \expandafter\XINT_expr_twoargs
2056   \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2057 }%
2058 \let\XINT_flexpr_func_rem\XINT_expr_func_rem
2059 \def\XINT_iiexpr_func_rem #1#2#3%
2060 {%
2061   \expandafter #1\expandafter #2\csname .=%
2062   \expandafter\expandafter\expandafter\xintiRem
2063   \expandafter\XINT_expr_twoargs
2064   \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2065 }%
2066 \def\XINT_expr_func_gcd #1#2#3%
2067   {\expandafter #1\expandafter #2\csname
2068     .=\xintGCDoF:csv{\XINT_expr_unlock #3}\endcsname }%
2069 \let\XINT_flexpr_func_gcd\XINT_expr_func_gcd
2070 \def\XINT_iiexpr_func_gcd #1#2#3%
2071   {\expandafter #1\expandafter #2\csname
2072     .=\xintiiGCDoF:csv{\XINT_expr_unlock #3}\endcsname }%
2073 \def\XINT_expr_func_lcm #1#2#3%
2074   {\expandafter #1\expandafter #2\csname
2075     .=\xintLCMoF:csv{\XINT_expr_unlock #3}\endcsname }%
2076 \let\XINT_flexpr_func_lcm\XINT_expr_func_lcm
2077 \def\XINT_iiexpr_func_lcm #1#2#3%
2078   {\expandafter #1\expandafter #2\csname
2079     .=\xintiiLCMoF:csv{\XINT_expr_unlock #3}\endcsname }%
2080 \def\XINT_expr_func_max #1#2#3%
2081   {\expandafter #1\expandafter #2\csname
2082     .=\xintMaxoF:csv{\XINT_expr_unlock #3}\endcsname }%
2083 \def\XINT_iiexpr_func_max #1#2#3%
2084   {\expandafter #1\expandafter #2\csname
2085     .=\xintiiMaxoF:csv{\XINT_expr_unlock #3}\endcsname }%
2086 \def\XINT_flexpr_func_max #1#2#3%
2087   {\expandafter #1\expandafter #2\csname
2088     .=\XINTinFloatMaxoF:csv{\XINT_expr_unlock #3}\endcsname }%
2089 \def\XINT_expr_func_min #1#2#3%
2090   {\expandafter #1\expandafter #2\csname
2091     .=\xintMinof:csv{\XINT_expr_unlock #3}\endcsname }%
2092 \def\XINT_iiexpr_func_min #1#2#3%
2093   {\expandafter #1\expandafter #2\csname

```



```

2094                                     .=\xintiiMinof:csv{\XINT_expr_unlock #3}\endcsname }%
2095 \def\XINT_flexpr_func_min #1#2#3%
2096     {\expandafter #1\expandafter #2\csname
2097         .=\XINTinFloatMinof:csv{\XINT_expr_unlock #3}\endcsname }%
2098 \expandafter\def\csname XINT_expr_func_+\endcsname #1#2#3%
2099     {\expandafter #1\expandafter #2\csname
2100         .=\xintSum:csv{\XINT_expr_unlock #3}\endcsname }%
2101 \expandafter\def\csname XINT_flexpr_func_+\endcsname #1#2#3%
2102     {\expandafter #1\expandafter #2\csname
2103         .=\XINTinFloatSum:csv{\XINT_expr_unlock #3}\endcsname }%
2104 \expandafter\def\csname XINT_iiexpr_func_+\endcsname #1#2#3%
2105     {\expandafter #1\expandafter #2\csname
2106         .=\xintiiSum:csv{\XINT_expr_unlock #3}\endcsname }%
2107 \expandafter\def\csname XINT_expr_func_*\endcsname #1#2#3%
2108     {\expandafter #1\expandafter #2\csname
2109         .=\xintPrd:csv{\XINT_expr_unlock #3}\endcsname }%
2110 \expandafter\def\csname XINT_flexpr_func_*\endcsname #1#2#3%
2111     {\expandafter #1\expandafter #2\csname
2112         .=\XINTinFloatPrd:csv{\XINT_expr_unlock #3}\endcsname }%
2113 \expandafter\def\csname XINT_iiexpr_func_*\endcsname #1#2#3%
2114     {\expandafter #1\expandafter #2\csname
2115         .=\xintiiPrd:csv{\XINT_expr_unlock #3}\endcsname }%
2116 \def\XINT_expr_func_? #1#2#3%
2117     {\expandafter #1\expandafter #2\csname
2118         .=\xintiiIsNotZero {\XINT_expr_unlock #3}\endcsname }%
2119 \let\XINT_flexpr_func_? \XINT_expr_func_?
2120 \let\XINT_iiexpr_func_? \XINT_expr_func_?
2121 \def\XINT_expr_func_! #1#2#3%
2122     {\expandafter #1\expandafter #2\csname .=\xintiiIsZero {\XINT_expr_unlock #3}\endcsname }%
2123 \let\XINT_flexpr_func_! \XINT_expr_func_!
2124 \let\XINT_iiexpr_func_! \XINT_expr_func_!
2125 \def\XINT_expr_func_not #1#2#3%
2126     {\expandafter #1\expandafter #2\csname .=\xintiiIsZero {\XINT_expr_unlock #3}\endcsname }%
2127 \let\XINT_flexpr_func_not \XINT_expr_func_not
2128 \let\XINT_iiexpr_func_not \XINT_expr_func_not
2129 \def\XINT_expr_func_all #1#2#3%
2130     {\expandafter #1\expandafter #2\csname
2131         .=\xintANDof:csv{\XINT_expr_unlock #3}\endcsname }%
2132 \let\XINT_flexpr_func_all\XINT_expr_func_all
2133 \let\XINT_iiexpr_func_all\XINT_expr_func_all
2134 \def\XINT_expr_func_any #1#2#3%
2135     {\expandafter #1\expandafter #2\csname
2136         .=\xintORof:csv{\XINT_expr_unlock #3}\endcsname }%
2137 \let\XINT_flexpr_func_any\XINT_expr_func_any
2138 \let\XINT_iiexpr_func_any\XINT_expr_func_any
2139 \def\XINT_expr_func_xor #1#2#3%
2140     {\expandafter #1\expandafter #2\csname
2141         .=\xintXORof:csv{\XINT_expr_unlock #3}\endcsname }%
2142 \let\XINT_flexpr_func_xor\XINT_expr_func_xor
2143 \let\XINT_iiexpr_func_xor\XINT_expr_func_xor
2144 \def\xintifNotZero: #1,#2,#3,{\xintiiifNotZero{#1}{#2}{#3}}%
2145 \def\XINT_expr_func_if #1#2#3%

```



```

2146   {\expandafter #1\expandafter #2\csname
2147     .=\expandafter\xintifNotZero:\romannumeral`&&\XINT_expr_unlock #3,\endcsname }%
2148 \let\XINT_flexpr_func_if\XINT_expr_func_if
2149 \let\XINT_iiexpr_func_if\XINT_expr_func_if
2150 \def\xintifSgn: #1,#2,#3,#4,{\xintiifSgn{#1}{#2}{#3}{#4}}%
2151 \def\XINT_expr_func_ifsgn #1#2#3%
2152 {%
2153   \expandafter #1\expandafter #2\csname
2154     .=\expandafter\xintifSgn:\romannumeral`&&\XINT_expr_unlock #3,\endcsname
2155 }%
2156 \let\XINT_flexpr_func_ifsgn\XINT_expr_func_ifsgn
2157 \let\XINT_iiexpr_func_ifsgn\XINT_expr_func_ifsgn
2158 \def\XINT_expr_func_len #1#2#3%
2159   {\expandafter#1\expandafter#2%
2160     \csname.=\xintLength:f:csv {\XINT_expr_unlock#3}\endcsname }%
2161 \let\XINT_flexpr_func_len \XINT_expr_func_len
2162 \let\XINT_iiexpr_func_len \XINT_expr_func_len
2163 \def\XINT_expr_func_first #1#2#3%
2164   {\expandafter #1\expandafter #2\csname.=\expandafter\XINT_expr_func_firsta
2165     \romannumeral`&&\XINT_expr_unlock #3,^\endcsname }%
2166 \def\XINT_expr_func_firsta #1,#2^{#1}%
2167 \let\XINT_flexpr_func_first\XINT_expr_func_first
2168 \let\XINT_iiexpr_func_first\XINT_expr_func_first
2169 \def\XINT_expr_func_last #1#2#3% will not work in \xintNewExpr if macro param involved
2170   {\expandafter #1\expandafter #2\csname.=\expandafter\XINT_expr_func_lasta
2171     \romannumeral`&&\XINT_expr_unlock #3,^\endcsname }%
2172 \def\XINT_expr_func_lasta #1,#2%
2173   {\if ^#2 #1\expandafter\xint_gobble_ii\fi \XINT_expr_func_lasta #2}%
2174 \let\XINT_flexpr_func_last\XINT_expr_func_last
2175 \let\XINT_iiexpr_func_last\XINT_expr_func_last
2176 \def\XINT_expr_func_odd #1#2#3%
2177   {\expandafter #1\expandafter #2\csname.=\xintOdd{\XINT_expr_unlock #3}\endcsname}%
2178 \let\XINT_flexpr_func_odd\XINT_expr_func_odd
2179 \def\XINT_iiexpr_func_odd #1#2#3%
2180   {\expandafter #1\expandafter #2\csname.=\xintiiOdd{\XINT_expr_unlock #3}\endcsname}%
2181 \def\XINT_expr_func_even #1#2#3%
2182   {\expandafter #1\expandafter #2\csname.=\xintEven{\XINT_expr_unlock #3}\endcsname}%
2183 \let\XINT_flexpr_func_even\XINT_expr_func_even
2184 \def\XINT_iiexpr_func_even #1#2#3%
2185   {\expandafter #1\expandafter #2\csname.=\xintiiEven{\XINT_expr_unlock #3}\endcsname}%
2186 \def\XINT_expr_func_nuple #1#2#3%
2187   {\expandafter #1\expandafter #2\csname .=\XINT_expr_unlock #3\endcsname }%
2188 \let\XINT_flexpr_func_nuple\XINT_expr_func_nuple
2189 \let\XINT_iiexpr_func_nuple\XINT_expr_func_nuple

```

1.2c I hesitated but left the function "reversed" from 1.1 with this name, not "reverse". But the inner not public macro got renamed into `\xintReverse::csv`. 1.2g opts for the name `\xintReverse:f:csv`, and rewrites it for direct handling of csv lists. 2016/03/17.

```

2190 \def\XINT_expr_func_reversed #1#2#3%
2191   {\expandafter #1\expandafter #2\csname .=%
2192     \xintReverse:f:csv {\XINT_expr_unlock #3}\endcsname }%
2193 \let\XINT_flexpr_func_reversed\XINT_expr_func_reversed

```

```
2194 \let\XINT_iiexpr_func_reversed\XINT_expr_func_reversed
```

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

10.50.1	<code>\xintSeqB:f:csv</code>	330
10.50.2	<code>\xintiiSeqB:f:csv</code>	330
10.50.3	<code>\XINTinFloatSeqB:f:csv</code>	331

10.50.1 `\xintSeqB:f:csv`

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

```
2195 \def\xintSeqB:f:csv #1#2%
2196   {\expandafter\XINT_seqb:f:csv \expandafter{\romannumeral0\xintra{#2}}{#1}}%
2197 \def\XINT_seqb:f:csv #1#2{\expandafter\XINT_seqb:f:csv_a\romannumeral`&&@#2#1!}%
2198 \def\XINT_seqb:f:csv_a #1#2;#3;#4!{%
2199   \expandafter\xint_gobble_i\romannumeral`&&@%
2200   \xintifCmp {#3}{#4}\XINT_seqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_seqb:f:csv_bg
2201   #1{#3}{#4}{#2}}%
2202 \def\XINT_seqb:f:csv_be #1#2#3#4#5{,#2}%
2203 \def\XINT_seqb:f:csv_bl #1{\if #1p\expandafter\XINT_seqb:f:csv_pa\else
2204   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2205 \def\XINT_seqb:f:csv_pa #1#2#3#4{\expandafter\XINT_seqb:f:csv_p\expandafter
2206   {\romannumeral0\xintadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2207 \def\XINT_seqb:f:csv_p #1#2%
2208   {%
2209   \xintifCmp {#1}{#2}\XINT_seqb:f:csv_pa\XINT_seqb:f:csv_pb\XINT_seqb:f:csv_pc
2210   {#1}{#2}%
2211   }%
2212 \def\XINT_seqb:f:csv_pb #1#2#3#4{#3,#1}%
2213 \def\XINT_seqb:f:csv_pc #1#2#3#4{#3}%
2214 \def\XINT_seqb:f:csv_bg #1{\if #1n\expandafter\XINT_seqb:f:csv_na\else
2215   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2216 \def\XINT_seqb:f:csv_na #1#2#3#4{\expandafter\XINT_seqb:f:csv_n\expandafter
2217   {\romannumeral0\xintadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2218 \def\XINT_seqb:f:csv_n #1#2%
2219   {%
2220   \xintifCmp {#1}{#2}\XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_seqb:f:csv_na
2221   {#1}{#2}%
2222   }%
2223 \def\XINT_seqb:f:csv_nb #1#2#3#4{#3,#1}%
2224 \def\XINT_seqb:f:csv_nc #1#2#3#4{#3}%

```

10.50.2 `\xintiiSeqB:f:csv`

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

2015/11/11. I correct a typo dating back to release 1.1 (2014/10/29): the macro name had a "b" rather than "B", hence was not functional (causing `\xintNewIIExpr` to fail on inputs such as `#1..[1]..#2`).

```

2225 \def\xintiiSeqB:f:csv #1#2%
2226   {\expandafter\XINT_iiseqb:f:csv \expandafter{\romannumeral`&&@#2}{#1}}%
2227 \def\XINT_iiseqb:f:csv #1#2{\expandafter\XINT_iiseqb:f:csv_a\romannumeral`&&@#2#1!}%
2228 \def\XINT_iiseqb:f:csv_a #1#2;#3;#4!{%
2229   \expandafter\xint_gobble_i\romannumeral`&&@%
2230   \xintSgnFork{\XINT_Cmp {#3}{#4}}%
2231     \XINT_iiseqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_iiseqb:f:csv_bg
2232     #1{#3}{#4}{}{#2}}%
2233 \def\XINT_iiseqb:f:csv_bl #1{\if #1p\expandafter\XINT_iiseqb:f:csv_pa\else
2234   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2235 \def\XINT_iiseqb:f:csv_pa #1#2#3#4{\expandafter\XINT_iiseqb:f:csv_p\expandafter
2236   {\romannumeral0\xintiiadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2237 \def\XINT_iiseqb:f:csv_p #1#2%
2238 {%
2239   \xintSgnFork{\XINT_Cmp {#1}{#2}}%
2240   \XINT_iiseqb:f:csv_pa\XINT_iiseqb:f:csv_pb\XINT_iiseqb:f:csv_pc {#1}{#2}%
2241 }%
2242 \def\XINT_iiseqb:f:csv_pb #1#2#3#4{#3,#1}%
2243 \def\XINT_iiseqb:f:csv_pc #1#2#3#4{#3}%
2244 \def\XINT_iiseqb:f:csv_bg #1{\if #1n\expandafter\XINT_iiseqb:f:csv_na\else
2245   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2246 \def\XINT_iiseqb:f:csv_na #1#2#3#4{\expandafter\XINT_iiseqb:f:csv_n\expandafter
2247   {\romannumeral0\xintiiadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2248 \def\XINT_iiseqb:f:csv_n #1#2%
2249 {%
2250   \xintSgnFork{\XINT_Cmp {#1}{#2}}%
2251   \XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_iiseqb:f:csv_na {#1}{#2}%
2252 }%

```

10.50.3 \XINTinFloatSeqB:f:csv

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

```

2253 \def\XINTinFloatSeqB:f:csv #1#2{\expandafter\XINT_flseqb:f:csv \expandafter
2254   {\romannumeral0\XINTinfloat [\XINTdigits]{#2}}{#1}}%
2255 \def\XINT_flseqb:f:csv #1#2{\expandafter\XINT_flseqb:f:csv_a\romannumeral`&&@#2#1!}%
2256 \def\XINT_flseqb:f:csv_a #1#2;#3;#4!{%
2257   \expandafter\xint_gobble_i\romannumeral`&&@%
2258   \xintifCmp {#3}{#4}\XINT_flseqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_flseqb:f:csv_bg
2259   #1{#3}{#4}{}{#2}}%
2260 \def\XINT_flseqb:f:csv_bl #1{\if #1p\expandafter\XINT_flseqb:f:csv_pa\else
2261   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2262 \def\XINT_flseqb:f:csv_pa #1#2#3#4{\expandafter\XINT_flseqb:f:csv_p\expandafter
2263   {\romannumeral0\XINTinfloatadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2264 \def\XINT_flseqb:f:csv_p #1#2%
2265 {%
2266   \xintifCmp {#1}{#2}%
2267   \XINT_flseqb:f:csv_pa\XINT_flseqb:f:csv_pb\XINT_flseqb:f:csv_pc {#1}{#2}%
2268 }%
2269 \def\XINT_flseqb:f:csv_pb #1#2#3#4{#3,#1}%
2270 \def\XINT_flseqb:f:csv_pc #1#2#3#4{#3}%
2271 \def\XINT_flseqb:f:csv_bg #1{\if #1n\expandafter\XINT_flseqb:f:csv_na\else

```

```

2272 \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2273 \def\XINT_flseqb:f:csv_na #1#2#3#4{\expandafter\XINT_flseqb:f:csv_n\expandafter
2274 \romannumeral0\XINTinfloatadd{#4}{#1}{#2}{#3,#1}{#4}}%
2275 \def\XINT_flseqb:f:csv_n #1#2%
2276 {%
2277 \xintifCmp {#1}{#2}%
2278 \XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_flseqb:f:csv_na {#1}{#2}%
2279 }%

```

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

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

2015/11/13: function candidates names first completely expanded, then detokenized and cleaned of spaces.

2015/11/21: no more `\detokenize` on the function names. Also I use `#1(#2)#3:=#4` rather than `#1(#2):=#3`. Ah, rather `#1(#2)#3=#4`, then I don't have to worry about active `:`.

2016/02/22: 1.2f la macro associée à la fonction ne débute plus par un `\romannumeral`, de toute façon est pour emploi dans `\csname..\endcsname`.

2016/03/08: adding a pair of braces thus allowing comma separated expressions; until then the user had to do `\xintdeffunc foo(x,...):=(..., .., ..)\relax`.

```

2280 \catcode`: 12
2281 \def\XINT_tmpa #1#2#3#4%
2282 {%
2283 \def #1##1(##2)##3==#4;%
2284 \edef\XINT_expr_tmpa {##1}%
2285 \edef\XINT_expr_tmpa {\xint_zapspace_o \XINT_expr_tmpa}%
2286 \def\XINT_expr_tmppb {0}%
2287 \def\XINT_expr_tmppc {(##4)}%
2288 \xintFor ####1 in {##2} \do
2289 {\edef\XINT_expr_tmppb {\the\numexpr\XINT_expr_tmppb+\xint_c_i}%
2290 \edef\XINT_expr_tmppc {subs(\unexpanded\expandafter{\XINT_expr_tmppc},%
2291 ####1=#####\XINT_expr_tmppb)}}%
2292 }%
2293 \expandafter#3\csname XINT_#2_userfunc_\XINT_expr_tmpa\endcsname
2294 [\XINT_expr_tmppb]{\XINT_expr_tmppc}%
2295 \expandafter\XINT_expr_defuserfunc
2296 \csname XINT_#2_func_\XINT_expr_tmpa\expandafter\endcsname
2297 \csname XINT_#2_userfunc_\XINT_expr_tmpa\endcsname
2298 \ifxintverbose\xintMessage {xintexpr}{Info}
2299 {Function \XINT_expr_tmpa\space for \string\xint #4 parser
2300 associated to \string\XINT_#2_userfunc_\XINT_expr_tmpa\space
2301 with meaning \expandafter\meaning
2302 \csname XINT_#2_userfunc_\XINT_expr_tmpa\endcsname}%
2303 \fi
2304 }%
2305 }%
2306 \catcode`: 11
2307 \XINT_tmpa\xintdeffunc {expr} \XINT_NewFunc {expr}%

```

```

2308 \XINT_tmpa\xintdefiifunc {iiexpr}\XINT_NewIIFunc {iiexpr}%
2309 \XINT_tmpa\xintdeffloatfunc{flexpr}\XINT_NewFloatFunc{floatexpr}%
2310 \def\XINT_expr_defuserfunc #1#2{%
2311     \def #1##1##2##3{\expandafter ##1\expandafter ##2%
2312         \csname .=\expandafter #2\romannumeral`&&\XINT_expr_unlock ##3,\endcsname
2313     }%
2314 }%

```

10.52 \xintNewFunction

1.2h (2016/11/20). Syntax is `\xintNewFunction{<name>}[nb of arguments]{expression with #1, #2,... as in \xintNewExpr}`. This defines a function for all three parsers but the expression parsing is delayed until function execution. Hence the expression admits all constructs, contrarily to `\xintNewExpr` or `\xintdeffunc`.

```

2315 \def\XINT_expr_wrapit #1{\expandafter\XINT_expr_wrap\csname.=#1\endcsname}%
2316 \def\xintNewFunction #1#2[#3]#4%
2317 {%
2318     \edef\XINT_expr_tmpa {#1}%
2319     \edef\XINT_expr_tmpa {\xint_zapspace_o \XINT_expr_tmpa}%
2320     \def\XINT_expr_tmppb ##1##2##3##4##5##6##7##8##9{#4}%
2321     \begingroup
2322         \ifcase #3\relax
2323             \toks0{,}%
2324         \or \toks0{##1,}%
2325         \or \toks0{##1,##2,}%
2326         \or \toks0{##1,##2,##3,}%
2327         \or \toks0{##1,##2,##3,##4,}%
2328         \or \toks0{##1,##2,##3,##4,##5,}%
2329         \or \toks0{##1,##2,##3,##4,##5,##6,}%
2330         \or \toks0{##1,##2,##3,##4,##5,##6,##7,}%
2331         \or \toks0{##1,##2,##3,##4,##5,##6,##7,##8,}%
2332         \else \toks0{##1,##2,##3,##4,##5,##6,##7,##8,##9,}%
2333         \fi
2334         \expandafter
2335     \endgroup
2336     \expandafter
2337     \def\csname XINT_expr_macrofunc_\XINT_expr_tmpa\expandafter\endcsname
2338     \the\toks0\expandafter{\XINT_expr_tmppb
2339         {\XINT_expr_wrapit{##1}}{\XINT_expr_wrapit{##2}}{\XINT_expr_wrapit{##3}}%
2340         {\XINT_expr_wrapit{##4}}{\XINT_expr_wrapit{##5}}{\XINT_expr_wrapit{##6}}%
2341         {\XINT_expr_wrapit{##7}}{\XINT_expr_wrapit{##8}}{\XINT_expr_wrapit{##9}}}%
2342     \expandafter\XINT_expr_newfunction
2343     \csname XINT_expr_func_\XINT_expr_tmpa\expandafter\endcsname
2344     \csname XINT_expr_macrofunc_\XINT_expr_tmpa\endcsname\xintbareeval
2345     \expandafter\XINT_expr_newfunction
2346     \csname XINT_iiexpr_func_\XINT_expr_tmpa\expandafter\endcsname
2347     \csname XINT_expr_macrofunc_\XINT_expr_tmpa\endcsname\xintbareieval
2348     \expandafter\XINT_expr_newfunction
2349     \csname XINT_flexpr_func_\XINT_expr_tmpa\expandafter\endcsname
2350     \csname XINT_expr_macrofunc_\XINT_expr_tmpa\endcsname\xintbarefloateval
2351     \ifxintverbose
2352         \xintMessage {xintexpr}{Info}

```

```

2353      {Function \XINT_expr_tmpa\space for the expression parsers is
2354      associated to \string\XINT_expr_macrofunc_\XINT_expr_tmpa\space
2355      with meaning \expandafter\meaning
2356      \csname XINT_expr_macrofunc_\XINT_expr_tmpa\endcsname}%
2357 \fi
2358 }%
2359 \def\XINT_expr_newfunction #1#2#3%
2360 {%
2361 \def#1##1##2##3{\expandafter ##1\expandafter ##2\romannumeral`&&@%
2362      #3\expandafter#2\romannumeral`&&@\XINT_expr_unlock##3,\relax}%
2363 }%

```

10.53 \xintNewExpr, \xintNewIExpr, \xintNewFloatExpr, \xintNewIIExpr

10.53.1	\xintApply::csv	334
10.53.2	\xintApply:::csv	335
10.53.3	\XINT_expr_RApply::csv, \XINT_expr_LApply::csv, \XINT_expr_RLApply:::csv	335
10.53.4	Mysterious stuff	335
10.53.5	\xintNewExpr, ..., at last.	338

There was an `\xintNewExpr` already in 1.07 from May 2013, which was modified in September 2013 to work with the `#` macro parameter character, and then refactored into a more powerful version in June 2014 for 1.1 release of 2014/10/28. List handling causes special challenges, addressed by `\xintApply::csv`, `\xintApply:::csv`, ... next.

Comments finally added 2015/12/11 (with later edits):

The whole point is to expand completely macros when they have only numerical arguments and to inhibit this expansion if not. This is done in a recursive way: the catcode 12 `~` is used to register a macro name whose expansion must be inhibited. Any argument itself starting with such a `~` will force use of `~` for the macro which receives it.

In this context the catcode 12 `$` is used to signal a "virtual list" argument. It triggers insertion of `\xintApply::csv` or `\xintApply:::csv` for delayed handling later. This succeeds into handling inputs such as `[#1..[#2]..#3][#4:#5]...`

A final `\scantokens` converts the `"~"` prefixed names into real control sequences.

For this whole mechanism we need to have everything expressed using exclusively f-expandable macros. We avoid `\csname... \endcsname` like construct, but if absolutely needed perhaps we will do it ultimately.

For the iterating loops `seq`, `iter`, etc..., and dummy variables, we have no macros to our disposal to handle the case where the list of indices is not explicit. Moreover `omit`, `abort`, `break` can not work with non numerical data. Thus the whole mechanism is currently not applicable to them. It does work when the macro parameters (or variables for `\xintdeffunc`) do not intervene in the list of values to iterate over. But we can not delay expansion of dummy variables.

10.53.1 \xintApply::csv

```

2364 \def\xintApply::csv #1#2%
2365   {\expandafter\XINT_applyon::_a\expandafter {\romannumeral`&&@#2}{#1}}%
2366 \def\XINT_applyon::_a #1#2{\XINT_applyon::_b {#2}}{#1,,}%
2367 \def\XINT_applyon::_b #1#2#3,{\expandafter\XINT_applyon::_c \romannumeral`&&@#3,{#1}{#2}}%
2368 \def\XINT_applyon::_c #1{\if #1,\expandafter\XINT_applyon::_end
2369   \else\expandafter\XINT_applyon::_d\fi #1}%
2370 \def\XINT_applyon::_d #1,#2{\expandafter\XINT_applyon::_e\romannumeral`&&@#2{#1},{#2}}%
2371 \def\XINT_applyon::_e #1,#2#3{\XINT_applyon::_b {#2}{#3, #1}}%

```

```
2372 \def\XINT_applyon::_end #1,#2#3{\xint_secondoftwo #3}%
```

10.53.2 \xintApply:::csv

```
2373 \def\xintApply:::csv #1#2#3%
2374   {\expandafter\XINT_applyon::_a\expandafter{\romannumeral`&&@#2}{#1}{#3}}%
2375 \def\XINT_applyon::_a #1#2#3{\XINT_applyon::_b {#2}{#3}{#1},}%
2376 \def\XINT_applyon::_b #1#2#3#4,%
2377   {\expandafter\XINT_applyon::_c \romannumeral`&&@#4,{#1}{#2}{#3}}%
2378 \def\XINT_applyon::_c #1{\if #1,\expandafter\XINT_applyon::_end
2379   \else\expandafter\XINT_applyon::_d\fi #1}%
2380 \def\XINT_applyon::_d #1,#2#3%
2381   {\expandafter\XINT_applyon::_e\expandafter
2382    {\romannumeral`&&\xintApply:::csv {#2}{#1}}{#3}}{#2}{#3}}%
2383 \def\XINT_applyon::_e #1,#2#3#4{\XINT_applyon::_b {#2}{#3}{#4}, #1}%
2384 \def\XINT_applyon::_end #1,#2#3#4{\xint_secondoftwo #4}%
```

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

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

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

```
2385 \catcode`~ 12
2386 \def\XINT_expr_RApply:::csv #1#2#3#4%
2387   {\xintApply:::csv{~expandafter#1~xint_exchangetwo_keepbraces{#4}}{#3}}%
2388 \def\XINT_expr_LApply:::csv #1#2#3#4{\xintApply:::csv{#1}{#3}}{#4}}%
2389 \def\XINT_expr_RLApply:::csv #1#2{\xintApply:::csv{#1}}%
```

10.53.4 Mysterious stuff

`~` and `$` of catcode 12 in what follows.

```
2390 \catcode`$ 12 % $
2391 \def\XINT_xptwo_getab_b #1#2!#3%
2392   {\expandafter\XINT_xptwo_getab_c\romannumeral`&&@#3!#1{#1#2}}%
2393 \def\XINT_xptwo_getab_c #1#2!#3#4#5#6{#1#3{#5}{#6}{#1#2}{#4}}%
2394 \def\xint_ddfork #1$#2#3\krof {#2}% $$
2395 \def\XINT_NEfork #1#2{\xint_ddfork
2396   #1#2\XINT_expr_RLApply:::csv
2397   #1$\XINT_expr_RApply:::csv% $
2398   $#2\XINT_expr_LApply:::csv% $
2399   $${\XINT_NEfork_nn #1#2}% $$
2400   \krof }%
2401 \def\XINT_NEfork_nn #1#2#3#4{%
2402   \if #1#\xint_dothis{#3}\fi
2403   \if #1~\xint_dothis{#3}\fi
2404   \if #2#\xint_dothis{#3}\fi
2405   \if #2~\xint_dothis{#3}\fi
2406   \xint_orthat {\csname #4NE\endcsname }%
2407   }%
2408 \def\XINT_NEfork_one #1#2!#3#4#5#6{%
2409   \if ###1\xint_dothis {#3}\fi
2410   \if ~#1\xint_dothis {#3}\fi
2411   \if $#1\xint_dothis {\xintApply:::csv{#3#5}}\fi %$
```



```

2412 \xint_orthat {\csname #4NE\endcsname #6}{#1#2}%
2413 }%
2414 \toks0 {}%
2415 \xintFor #1 in
2416     {DivTrunc,Mod,Round,Trunc,iRound,iTrunc,iQuo,iRem,
2417     iiDivTrunc,iiDivRound,iiMod,iiQuo,iiRem,%
2418     Lt,Gt,Eq,LtorEq,GtorEq,Neq,%
2419     iLt,iiGt,iiEq,iiLtorEq,iiGtorEq,iiNeq,%
2420     Add,Sub,Mul,Div,Pow,E,%
2421     iiAdd,iiSub,iiMul,iiPow,iiE,%
2422     AND,OR,XOR,%
2423     SeqA::csv,iiSeqA::csv}\do
2424 {\toks0
2425 \expandafter{\the\toks0% no space!
2426 \expandafter\let\csname xint#1NE\expandafter\endcsname\csname xint#1\expandafter
2427 \endcsname\expandafter\def\csname xint#1\endcsname #####1####2{%
2428 \expandafter\XINT_NEfork
2429 \romannumeral`&&\expandafter\XINT_xptwo_getab_b
2430 \romannumeral`&&#####2!{#####1}{~xint#1}{xint#1}}%
2431 }%
2432 }%
2433 % cela aurait-il un sens d'ajouter Raw et iNum (à cause de qint, qfrac,
2434 % qfloat?). Pas le temps d'y réfléchir. Je ne fais rien.
2435 \xintFor #1 in {Num,Irr,Abs,iiAbs,Sgn,iiSgn,TFrac,Floor,iFloor,Ceil,iCeil,%
2436 Sqr,iiSqr,iiSqrt,iiSqrtR,iiIsZero,iiIsNotZero,iiifNotZero,iiifSgn,%
2437 Odd,Even,iiOdd,iiEven,Opp,iiOpp,iiifZero,iFac,iBinomial,%
2438 iPFactorial,iiFac,iiBinomial,iiPFactorial,Bool,Toggle}\do
2439 {\toks0 \expandafter{\the\toks0%
2440 \expandafter\let\csname xint#1NE\expandafter\endcsname\csname
2441 xint#1\expandafter
2442 \endcsname\expandafter\def\csname xint#1\endcsname #####1{%
2443 \expandafter\XINT_NEfork_one\romannumeral`&&#####1!{~xint#1}{xint#1}{}}}%
2444 }%
2445 }%
2446 \toks0
2447 \expandafter{\the\toks0
2448 \let\XINTinFloatFacNE\XINTinFloatFac
2449 \def\XINTinFloatFac ##1{%
2450 \expandafter\XINT_NEfork_one
2451 \romannumeral`&&#####1!{~XINTinFloatFac}{XINTinFloatFac}{}}}%
2452 }%
2453 \xintFor #1 in {Add,Sub,Mul,Div,Binomial,PFactorial,PowerH,E,Mod,SeqA::csv}\do
2454 {\toks0
2455 \expandafter{\the\toks0%
2456 \expandafter\let\csname XINTinFloat#1NE\expandafter\endcsname
2457 \csname XINTinFloat#1\expandafter\endcsname
2458 \expandafter\def\csname XINTinFloat#1\endcsname #####1####2{%
2459 \expandafter\XINT_NEfork
2460 \romannumeral`&&\expandafter\XINT_xptwo_getab_b
2461 \romannumeral`&&#####2!{#####1}{~XINTinFloat#1}{XINTinFloat#1}}%
2462 }%
2463 }%

```



```

2464 \xintFor #1 in {XINTinFloatdigits,XINTinFloatFracdigits,XINTinFloatSqrtdigits,XINTinFloatFacdigits}\do
2465 {\toks0
2466   \expandafter{\the\toks0%
2467   \expandafter\let\csname #1NE\expandafter\endcsname\csname #1\expandafter
2468   \endcsname\expandafter\def\csname #1\endcsname #####1{\expandafter
2469     \XINT_NEfork_one\romannumeral`&&@####1!{~#1}{#1}{}}}%
2470   }%
2471 }%
2472 \xintFor #1 in {xintSeq::csv,xintiiSeq::csv,XINTinFloatSeq::csv}\do
2473 {\toks0
2474   \expandafter{\the\toks0% no space
2475   \expandafter\let\csname #1NE\expandafter\endcsname\csname #1\expandafter
2476   \endcsname\expandafter\def\csname #1\endcsname #####1####2{%
2477     \expandafter\XINT_NEfork
2478     \romannumeral`&&\expandafter\XINT_xptwo_getab_b
2479     \romannumeral`&&@####2!{####1}{$noexpand$#1}{#1}}}%
2480   }%
2481 }%
2482 \xintFor #1 in {xintSeqB,xintiiSeqB,XINTinFloatSeqB}\do
2483 {\toks0
2484   \expandafter{\the\toks0% no space
2485   \expandafter\let\csname #1::csvNE\expandafter\endcsname\csname #1::csv\expandafter
2486   \endcsname\expandafter\def\csname #1::csv\endcsname #####1####2{%
2487     \expandafter\XINT_NEfork
2488     \romannumeral`&&\expandafter\XINT_xptwo_getab_b
2489     \romannumeral`&&@####2!{####1}{$noexpand$#1:f:csv}{#1::csv}}}%
2490   }%
2491 }%
2492 \toks0
2493   \expandafter{\the\toks0
2494   \let\XINTinFloatNE\XINTinFloat
2495   \def\XINTinFloat [##1]##2{% not ultimately general, but got tired
2496     \expandafter\XINT_NEfork_one
2497     \romannumeral`&&@##2!{~XINTinFloat[##1]}{XINTinFloat}{}}{[##1]}}%
2498   \let\XINTinFloatSqrtNE\XINTinFloatSqrt
2499   \def\XINTinFloatSqrt [##1]##2{%
2500     \expandafter\XINT_NEfork_one
2501     \romannumeral`&&@##2!{~XINTinFloatSqrt[##1]}{XINTinFloatSqrt}{}}{[##1]}}%
2502   \let\XINTinFloatFacNE\XINTinFloatFac
2503   \def\XINTinFloatFac [##1]##2{%
2504     \expandafter\XINT_NEfork_one
2505     \romannumeral`&&@##2!{~XINTinFloatFac[##1]}{XINTinFloatFac}{}}{[##1]}}%
2506   }%
2507 \xintFor #1 in {ANDof,ORof,XORof,iiMaxof,iiMinof,iiSum,iiPrd,
2508               GCDof,LCMof,Sum,Prd,Maxof,Minof}\do
2509 {\toks0
2510   \expandafter{\the\toks0\expandafter\def\csname xint#1:csv\endcsname {~xint#1:csv}}%
2511 }%
2512 \xintFor #1 in
2513   {XINTinFloatMaxof,XINTinFloatMinof,XINTinFloatSum,XINTinFloatPrd}\do
2514 {\toks0
2515   \expandafter{\the\toks0\expandafter\def\csname #1:csv\endcsname {~#1:csv}}%

```

2516 }%

~xintListSel:f:csv must have space after it, the reason being in \XINT_expr_until_:b which inserts a : as fist token of something which will reappear later following ~xintListSel:f:csv.

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).

```

2517 \toks0 \expandafter{\the\toks0
2518             \def\xintReverse:f:csv {\xintReverse:f:csv }%
2519             \def\xintListSel:f:csv {\xintListSel:f:csv }%
2520 }%
2521 \odef\XINT_expr_redefinemacros {\the\toks0}% Not \edef ! (subtle)
2522 \def\XINT_expr_redefineprints
2523 {%
2524     \def\XINT_flexpr_noopt
2525     {\expandafter\XINT_flexpr_withopt_b\expandafter-\romannumeral0\xintbarefloateval }%
2526     \def\XINT_flexpr_withopt_b ##1##2%
2527     {\expandafter\XINT_flexpr_wrap\csname .;##1.=\XINT_expr_unlock ##2\endcsname }%
2528     \def\XINT_expr_unlock_sp ##1.;##2##3.=##4!%
2529     {\if -##2\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
2530     \XINTdigits{##2##3}{##4}}%
2531     \def\XINT_expr_print ##1{\expandafter\xintSPRaw::csv\expandafter
2532     {\romannumeral`&&\XINT_expr_unlock ##1}}%
2533     \def\XINT_iexpr_print ##1{\expandafter\xintCSV::csv\expandafter
2534     {\romannumeral`&&\XINT_expr_unlock ##1}}%
2535     \def\XINT_boolexpr_print ##1{\expandafter\xintIsTrue::csv\expandafter
2536     {\romannumeral`&&\XINT_expr_unlock ##1}}%
2537     \def\xintCSV::csv {\xintCSV::csv }%
2538     \def\xintSPRaw::csv {\xintSPRaw::csv }%
2539     \def\xintPFloat::csv {\xintPFloat::csv }%
2540     \def\xintIsTrue::csv {\xintIsTrue::csv }%
2541     \def\xintRound::csv {\xintRound::csv }%
2542 }%
2543 \toks0 {}%
```

10.53.5 \xintNewExpr, ..., at last.

1.2c modifications to accomodate \XINT_expr_deffunc_newexpr etc..

1.2f adds token \XINT_newexpr_clean to be able to have a different \XINT_newfunc_clean

```

2544 \def\xintNewExpr {\XINT_NewExpr}\XINT_expr_redefineprints\xint_firstofone
2545             \xinttheexpr\XINT_newexpr_clean}%
2546 \def\xintNewFloatExpr{\XINT_NewExpr}\XINT_expr_redefineprints\xint_firstofone
2547             \xintthefloatexpr\XINT_newexpr_clean}%
2548 \def\xintNewIExpr {\XINT_NewExpr}\XINT_expr_redefineprints\xint_firstofone
2549             \xinttheiexpr\XINT_newexpr_clean}%
2550 \def\xintNewIIExpr {\XINT_NewExpr}\XINT_expr_redefineprints\xint_firstofone
2551             \xinttheiexpr\XINT_newexpr_clean}%
2552 \def\xintNewBoolExpr {\XINT_NewExpr}\XINT_expr_redefineprints\xint_firstofone
2553             \xinttheboolexpr\XINT_newexpr_clean}%
2554 \def\XINT_newexpr_clean #1>{\noexpand\romannumeral`&&}%
```

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

```

2555 \def\xINT_NewFunc
2556 {\xINT_NewExpr,{ }\xint_gobble_i\xintthebareeval \xINT_newfunc_clean }%
2557 \def\xINT_NewFloatFunc
2558 {\xINT_NewExpr,{ }\xint_gobble_i\xintthebarefloateval\xINT_newfunc_clean }%
2559 \def\xINT_NewIIFunc
2560 {\xINT_NewExpr,{ }\xint_gobble_i\xintthebareiieval \xINT_newfunc_clean }%
2561 \def\xINT_newfunc_clean #1>{}%
```

1.2c adds optional logging. For this needed to pass to `_NewExpr_a` the macro name as parameter. And `_NewExpr` itself receives two new parameters to treat both `\xintNewExpr` and `\xintdeffunc`.

Up to and including 1.2c the definition was global. Starting with 1.2d it is done locally.

```

2562 \def\xINT_NewExpr #1#2#3#4#5#6#7[#8]%
2563 {%
2564 \begingroup
2565 \ifcase #8\relax
2566 \toks0 {\endgroup\def#6}%
2567 \or \toks0 {\endgroup\def#6##1#1}%
2568 \or \toks0 {\endgroup\def#6##1#1##2#1}%
2569 \or \toks0 {\endgroup\def#6##1#1##2#1##3#1}%
2570 \or \toks0 {\endgroup\def#6##1#1##2#1##3#1##4#1}%
2571 \or \toks0 {\endgroup\def#6##1#1##2#1##3#1##4#1##5#1}%
2572 \or \toks0 {\endgroup\def#6##1#1##2#1##3#1##4#1##5#1##6#1}%
2573 \or \toks0 {\endgroup\def#6##1#1##2#1##3#1##4#1##5#1##6#1##7#1}%
2574 \or \toks0 {\endgroup\def#6##1#1##2#1##3#1##4#1##5#1##6#1##7#1##8#1}%
2575 \or \toks0 {\endgroup\def#6##1#1##2#1##3#1##4#1##5#1##6#1##7#1##8#1##9#1}%
2576 \fi
2577 \xintexprSafeCatcodes
2578 \xINT_expr_redefinmacros
2579 #2%
2580 \xINT_NewExpr_a #3#4#5#6%
2581 }%
```

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.

```

2582 \catcode`~ 13 \catcode`@ 14 \catcode`\% 6 \catcode`# 12 \catcode`$ 11 @ $
2583 \def\xINT_NewExpr_a %1%2%3%4%5@
2584 {@
2585 \def\xINT_tmpa %1%2%3%4%5%6%7%8%9{%5}@
2586 \def~{$noexpand$}@
2587 \catcode`: 11 \catcode`_ 11
2588 \catcode`# 12 \catcode`~ 13 \escapechar 126
2589 \endlinechar -1 \everyeof {\noexpand }@
2590 \edef\xINT_tmpb
2591 {\scantokens\expandafter
2592 {\romannumeral`&&&\expandafter%2\xINT_tmpa {#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}\relax}@
2593 }@
```

```

2594 \escapechar 92 \catcode`# 6 \catcode`$ 0 @ $
2595 \edef\XINT_tmpa %%1%%2%%3%%4%%5%%6%%7%%8%%9@
2596 {\scantokens\expandafter{\expandafter%3\meaning\XINT_tmpb}}@
2597 \the\toks0\expandafter{\XINT_tmpa{%%1}{%%2}{%%3}{%%4}{%%5}{%%6}{%%7}{%%8}{%%9}}@
2598 %1{\ifxintverbose
2599 \xintMessage{xintexpr}{Info}@
2600 \string%4\space now with meaning \meaning%4}@
2601 \fi}@
2602 }@
2603 \catcode`% 14
2604 \let\xintexprRestoreCatcodes\empty
2605 \def\xintexprSafeCatcodes
2606 {%
2607 \edef\xintexprRestoreCatcodes {%
2608 \catcode59=\the\catcode59 % ;
2609 \catcode34=\the\catcode34 % "
2610 \catcode63=\the\catcode63 % ?
2611 \catcode124=\the\catcode124 % |
2612 \catcode38=\the\catcode38 % &
2613 \catcode33=\the\catcode33 % !
2614 \catcode93=\the\catcode93 % ]
2615 \catcode91=\the\catcode91 % [
2616 \catcode94=\the\catcode94 % ^
2617 \catcode95=\the\catcode95 % _
2618 \catcode47=\the\catcode47 % /
2619 \catcode41=\the\catcode41 % )
2620 \catcode40=\the\catcode40 % (
2621 \catcode42=\the\catcode42 % *
2622 \catcode43=\the\catcode43 % +
2623 \catcode62=\the\catcode62 % >
2624 \catcode60=\the\catcode60 % <
2625 \catcode58=\the\catcode58 % :
2626 \catcode46=\the\catcode46 % .
2627 \catcode45=\the\catcode45 % -
2628 \catcode44=\the\catcode44 % ,
2629 \catcode61=\the\catcode61 % =
2630 \catcode32=\the\catcode32\relax % space
2631 }%
2632 \catcode59=12 % ;
2633 \catcode34=12 % "
2634 \catcode63=12 % ?
2635 \catcode124=12 % |
2636 \catcode38=4 % &
2637 \catcode33=12 % !
2638 \catcode93=12 % ]
2639 \catcode91=12 % [
2640 \catcode94=7 % ^
2641 \catcode95=8 % _
2642 \catcode47=12 % /
2643 \catcode41=12 % )
2644 \catcode40=12 % (
2645 \catcode42=12 % *

```

Package `xintexpr` implementation

```
2646      \catcode43=12 % +
2647      \catcode62=12 % >
2648      \catcode60=12 % <
2649      \catcode58=12 % :
2650      \catcode46=12 % .
2651      \catcode45=12 % -
2652      \catcode44=12 % ,
2653      \catcode61=12 % =
2654      \catcode32=10 % space
2655 }%
2656 \let\XINT_tmpa\relax \let\XINT_tmpb\relax \let\XINT_tmpc\relax
2657 \XINT_restorecatcodes_endinput%

xintkernel: 492. Total number of code lines: 13780. (but 3189 lines among them
xinttools:1351. start either with {% or with }%.)
xintcore:2000. Each package starts with circa 50 lines dealing with cat-
xint:1785. codes, package identification and reloading management,
xintbinhex: 629. also for Plain TeX. Version 1.2i of 2016/12/13.
xintgcd: 455.
xintfrac:3004.
xintseries: 386.
xintcfraction:1021.
xintexpr:2657.
```