

The `xint` source code

JEAN-FRANÇOIS BURNOL

jfbu (at) free (dot) fr

Package version: 1.2q (2018/02/06); documentation date: 2018/02/06.

From source file xint.dtx. Time-stamp: <06-02-2018 at 21:58:51 CET>.

Contents

1	Package <code>xintkernel</code> implementation	2
2	Package <code>xinttools</code> implementation	16
3	Package <code>xintcore</code> implementation	52
4	Package <code>xint</code> implementation	113
5	Package <code>xintbinhex</code> implementation	154
6	Package <code>xintgcd</code> implementation	166
7	Package <code>xintfrac</code> implementation	179
8	Package <code>xintseries</code> implementation	259
9	Package <code>xintcfrc</code> implementation	268
10	Package <code>xintexpr</code> implementation	291

This is 1.2q of 2018/02/06.

- Release 1.2q of 2018/02/06 had some breaking changes:
 - new output for the `\xintBezout` macro (`xintgcd`),
 - the `xintexpr` operators `/:` (aka 'mod') and `//`, and the supporting macros from `xintcore` and `xintfrac`, are now associated with the *floored* division. Formerly it was the *truncated* division. This is breaking change for operands of opposite signs.

Improvements and new features:

- `xinttools` macro `\xintListWithSep` is faster (first update since 1.04-2013/04/25...).
- `divmod()` function added to the `xintexpr` parsers,
- `\xintdefvar`, `\xintdeffloatvar`, `\xintdefiivar` extended to allow multiple simultaneous assignments.
- Release 1.2o of 2017/08/29 deprecated those macros from `xintcore` and `xint` which filtered their arguments via `\xintNum`. Currently these macros execute as formerly but raise an error message. This deprecation is overruled for most if `xintfrac` is loaded as it provides their proper definitions. Some however (like `\xintiAdd`) remain deprecated even if loading `xintfrac`. All these deprecated macros are destined to be removed at some future release.

A few macros got renamed (e.g. `\xintNot` became `\xintNOT`.) Former names emit a deprecation error and will get removed at some future release.
- Release 1.2n of 2017/08/06 removed the `xintbinhex` dependencies upon `xintcore`; the package now depends upon, and loads, only `xintkernel`. The allowed maximal size for the inputs of the base conversion macros got increased. The speed got slightly improved.

1 Package `xintkernel` implementation

- Release 1.2m of 2017/07/31 rewrote entirely the `xintbinhex` module in the style of the techniques from 1.2 `xintcore`. The new macros expand faster but their inputs are now limited to a few thousand characters (the earlier routines, which dated back to 1.08 could handle (slowly) tens of thousands of digits.)
- Release 1.2l of 2017/07/26 refactored the subtraction and also `\xintiiCmp` got a rewrite. It should certainly use `\pdfstrcmp` for dramatic speed-up but I do not want to have to worry about multi-engine usage.

Some utility routines in `xintcore` manipulating blocks of eight digits and still in $O(N^2)$ style got rewritten analogously to the 1.2i version of macros such as `\xintInc`. Also `\xintiNum` was revisited.

The arithmetic macros of `xintcore` and all macros of `xintfrac` using `\XINT_infrac` were made compatible with arguments using non-delimited `\the\numexpr` or `\the\mathcode` etc... But `\xintiiAbs` and `\xintiiOpp` were not modified (to avoid some overhead) as well as routines such as `\xintInc` which are primarily for internal usage.

- Release 1.2i of 2016/12/13 rewrote some legacy macros like `\xintDSR` or `\xintDecSplit` in the style of the techniques of 1.2. But this means also that they got 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 19950 digits from other code parts (such as division.) The macro `\xintXTrunc` (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 entirely rewrote the core arithmetic routines located in `xintcore`. The parser of `xintexpr` got faster and the limitation at 5000 digits per number was removed.
- 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.

Some parts 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.

Warning: pay attention when looking at the code to the catcode configuration as found in `\XINT2_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 . . .	3	.10	<code>\odef</code> , <code>\oodef</code> , <code>\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 . . .	7	.14	<code>\xintLengthUpTo</code>	10
.6	<code>\xint_afterfi</code>	7	.15	<code>\xintreplicate</code>	11
.7	<code>\xint_bye</code> , <code>\xint_Bye</code>	7	.16	<code>\xintgobble</code>	12
.8	<code>\xintdothis</code> , <code>\xintorthat</code>	7	.17	<code>\xintMessage</code> , <code>\ifxintverbose</code>	14
.9	<code>\xint_zapspaces</code>	8	.18	(WIP) Expandable error message	14

1 Package *xintkernel* implementation

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`, `\xintLengthUpTo` and `\xintLastItem`, and improves the efficiency of `\xintLength`.

1.1 Catcodes, ε -TeX and reload detection

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

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

```
1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2   \catcode13=5      % ^^M
3   \endlinechar=13 %
4   \catcode123=1     % {
5   \catcode125=2     % }
6   \catcode35=6      % #
7   \catcode44=12     % ,
8   \catcode45=12     % -
9   \catcode46=12     % .
10  \catcode58=12     % :
11  \catcode95=11     % _
12  \expandafter
13    \ifx\csname PackageInfo\endcsname\relax
14      \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
15    \else
16      \def\y#1#2{\PackageInfo{#1}{#2}}%
17    \fi
18  \let\z\relax
19  \expandafter
20    \ifx\csname numexpr\endcsname\relax
21      \y{xintkernel}{\numexpr not available, aborting input}%
22      \def\z{\endgroup\endinput}%
23    \else
24      \expandafter
25        \ifx\csname XINTsetupcatcodes\endcsname\relax
26          \else
27            \y{xintkernel}{I was already loaded, aborting input}%
28            \def\z{\endgroup\endinput}%
29          \fi
30        \fi
31      \ifx\z\relax\else\expandafter\z\fi%
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
```

1 Package *xintkernel* implementation

```

41      \catcode34=\the\catcode34    % " xintbinhex, and xintexpr
42      \catcode63=\the\catcode63    % ? xintexpr
43      \catcode124=\the\catcode124  % | xintexpr
44      \catcode38=\the\catcode38    % & xintexpr
45      \catcode64=\the\catcode64    % @ xintexpr
46      \catcode33=\the\catcode33    % ! xintexpr
47      \catcode93=\the\catcode93    % ] -, xintfrac, xintseries, xintcfrac
48      \catcode91=\the\catcode91    % [ -, xintfrac, xintseries, xintcfrac
49      \catcode36=\the\catcode36    % $ xintgcd only
50      \catcode94=\the\catcode94    % ^
51      \catcode96=\the\catcode96    % `
52      \catcode47=\the\catcode47    % /
53      \catcode41=\the\catcode41    % )
54      \catcode40=\the\catcode40    % (
55      \catcode42=\the\catcode42    % *
56      \catcode43=\the\catcode43    % +
57      \catcode62=\the\catcode62    % >
58      \catcode60=\the\catcode60    % <
59      \catcode58=\the\catcode58    % :
60      \catcode46=\the\catcode46    % .
61      \catcode45=\the\catcode45    % -
62      \catcode44=\the\catcode44    % ,
63      \catcode35=\the\catcode35    % #
64      \catcode95=\the\catcode95    % _
65      \catcode125=\the\catcode125 % }
66      \catcode123=\the\catcode123 % {
67      \endlinechar=\the\endlinechar
68      \catcode13=\the\catcode13    % ^^M
69      \catcode32=\the\catcode32    %
70      \catcode61=\the\catcode61\relax % =
71  }%
72  \edef\XINT_restorecatcodes_endinput
73  {%
74      \XINT_restorecatcodes\noexpand\endinput %
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    % +

```

1 Package *xintkernel* implementation

```
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 % ! (xintexpr.sty will use catcode 11)
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 [2018/02/06 1.2q Paraphernalia for the xint packages (JFB)]%
```

1.3 Constants

```

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_xii   12
145 \chardef\xint_c_xiv   14
146 \chardef\xint_c_xvi   16
147 \chardef\xint_c_xviii 18
148 \chardef\xint_c_xxii  22
149 \chardef\xint_c_ii^v   32
150 \chardef\xint_c_ii^vi  64
151 \chardef\xint_c_ii^vii 128
152 \mathchardef\xint_c_ii^viii 256
153 \mathchardef\xint_c_ii^xii 4096
154 \mathchardef\xint_c_x^iv 10000

```

1.4 Token management utilities

```

155 \def\xint_tmpa { }%
156 \ifx\xint_tmpa\space\else
157   \immediate\write-1{Package xintkernel Warning: ATTENTION!}%
158   \immediate\write-1{\string\space\xint_tmpa macro does not have its normal
159     meaning.}%
160   \immediate\write-1{\xint_tmpa\xint_tmpa\xint_tmpa\xint_tmpa
161     All kinds of catastrophes will ensue!!!!}%
162 \fi
163 \def\xint_tmppb {}%
164 \ifx\xint_tmppb\empty\else
165   \immediate\write-1{Package xintkernel Warning: ATTENTION!}%
166   \immediate\write-1{\string\empty\xint_tmppb macro does not have its normal
167     meaning.}%
168   \immediate\write-1{\xint_tmpa\xint_tmpa\xint_tmpa\xint_tmpa
169     All kinds of catastrophes will ensue!!!!}%
170 \fi
171 \let\xint_tmpa\relax \let\xint_tmppb\relax
172 \ifdefined\space\else\def\space { }\fi
173 \ifdefined\empty\else\def\empty {} \fi
174 \let\xint_gobble_\empty
175 \long\def\xint_gobble_i   #1{}%
176 \long\def\xint_gobble_ii  #1#2{}%
177 \long\def\xint_gobble_iii  #1#2#3{}%
178 \long\def\xint_gobble_iv   #1#2#3#4{}%
179 \long\def\xint_gobble_v    #1#2#3#4#5{}%
180 \long\def\xint_gobble_vi    #1#2#3#4#5#6{}%
181 \long\def\xint_gobble_vii   #1#2#3#4#5#6#7{}%
182 \long\def\xint_gobble_viii  #1#2#3#4#5#6#7#8{}%
183 \long\def\xint_firstofone  #1{#1}%

```

```

184 \long\def\xint_firstoftwo #1#2{#1}%
185 \long\def\xint_secondoftwo #1#2{#2}%
186 \long\def\xint_gobble_thenstop #1{ }%
187 \long\def\xint_firstofone_thenstop #1{ #1}%
188 \long\def\xint_firstoftwo_thenstop #1#2{ #1}%
189 \long\def\xint_secondoftwo_thenstop #1#2{ #2}%
190 \long\def\xint_exchangetwo_keepbraces #1#2{{#2}{#1}}%

```

1.5 “gob til” macros and UD style fork

```

191 \long\def\xint_gob_til_R #1\R {}%
192 \long\def\xint_gob_til_W #1\W {}%
193 \long\def\xint_gob_til_Z #1\Z {}%
194 \long\def\xint_gob_til_zero #10{}%
195 \long\def\xint_gob_til_one #11{}%
196 \long\def\xint_gob_til_zeros_iii #1000{}%
197 \long\def\xint_gob_til_zeros_iv #10000{}%
198 \long\def\xint_gob_til_eightzeroes #100000000{}%
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_UDzerominusfork #10-#2#3\krof {#2}%
203 \long\def\xint_UDzerofork #10#2#3\krof {#2}%
204 \long\def\xint_UDsignfork #1-#2#3\krof {#2}%
205 \long\def\xint_UDwfork #1\W#2#3\krof {#2}%
206 \long\def\xint_UDXINTWfork #1\XINT_W#2#3\krof {#2}%
207 \long\def\xint_UDzerosfork #100#2#3\krof {#2}%
208 \long\def\xint_UDonezerofork #110#2#3\krof {#2}%
209 \long\def\xint_UDsignsfork #1--#2#3\krof {#2}%
210 \let\xint:\char
211 \long\def\xint_gob_til_xint:#1\xint:{}%
212 \def\xint_bracedstopper{\xint:}%
213 \long\def\xint_gob_til_exclam #1!{}%
214 \long\def\xint_gob_til_sc #1;{}%

```

1.6 `\xint_afterfi`

```

215 \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`.

```

216 \long\def\xint_bye #1\xint_bye {}%
217 \long\def\xint_Bye #1\xint_bye {}%
218 \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.

```

219 \long\def\xint_dothis #1#2\xint_orthat #3{\fi #1}% 1.1
220 \let\xint_orthat \xint_firstofone

```

```

221 \long\def\xintdothis #1#2\xintorthat #3{\fi #1}%
222 \let\xintorthat \xint_firstofone

```

1.9 \xint_zapspaces

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

```
\xint_zapspaces foo<space>\xint_gobble_i
```

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

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

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

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

expands to 12, and not 12\relax.

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

Made long with 1.2i.

```

223 \long\def\xint_zapspaces #1 #2{#1#2\xint_zapspaces }% 1.1
224 \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.

```

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

```

1.11 \xintReverseOrder

\xintReverseOrder: does NOT expand its argument.

Attention: removes brace pairs.

For digit tokens only a faster reverse macro is provided as \xintReverseDigits from 1.2 xint-core.sty.

For comma separated items, 1.2g has (not user documented) \xintCSVReverse in xinttools.sty.

```

234 \def\xintReverseOrder {\romannumeral0\xintreverseorder }%
235 \long\def\xintreverseorder #1%
236 {%
237     \XINT_rord_main {}#1%
238     \xint:

```



```

239      \xint_bye\xint_bye\xint_bye\xint_bye
240      \xint_bye\xint_bye\xint_bye\xint_bye
241      \xint:
242 }%
243 \long\def\xINT_rord_main #1#2#3#4#5#6#7#8#9%
244 {%
245     \xint_bye #9\xINT_rord_cleanup\xint_bye
246     \XINT_rord_main {#9#8#7#6#5#4#3#2#1}%
247 }%
248 \def\xINT_rord_cleanup #1{%
249 \long\def\xINT_rord_cleanup\xint_bye\xINT_rord_main ##1##2\xint:
250 {%
251     \expandafter#1\xint_gob_til_xint: ##1%
252 }}\XINT_rord_cleanup { }%

```

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. Was again slightly changed for 1.2j (cosmetic).

```

253 \def\xintLength {\romannumeral0\xintlength }%
254 \def\xintlength #1{\long\def\xintlength ##1%
255 {%
256     \expandafter#1\the\numexpr\xINT_length_loop
257     ##1\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
258     \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
259     \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
260     \relax
261 }}\xintlength{ }%
262 \long\def\xINT_length_loop #1#2#3#4#5#6#7#8#9%
263 {%
264     \xint_gob_til_xint: #9\xINT_length_finish_a\xint:
265     \xint_c_ix+\XINT_length_loop
266 }%
267 \def\xINT_length_finish_a\xint:\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. Does NOT expand its argument.

```

272 \def\xintLastItem {\romannumeral0\xintlastitem }%
273 \long\def\xintlastitem #1%
274 {%
275     \XINT_last_loop {}.#1%
276     {\xint:\XINT_last_loop_enda}{\xint:\XINT_last_loop_endb}%

```

1 Package *xintkernel* implementation

```

277   {\xint:\XINT_last_loop_endc}{\xint:\XINT_last_loop_endd}%
278   {\xint:\XINT_last_loop_ende}{\xint:\XINT_last_loop_endf}%
279   {\xint:\XINT_last_loop_endg}{\xint:\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: #9%
284   {#8}{#7}{#6}{#5}{#4}{#3}{#2}{#1}\xint:
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#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}{List} produces -0 if $\text{length}(\text{List}) > N$, else it returns $N - \text{length}(\text{List})$. Hence subtracting it from N always computes $\min(N, \text{length}(\text{List}))$.

Does not expand its second argument (it is used by *\xintKeep* and *\xintTrim* with already expanded argument). Not a user macro. 1.2j rewrote the ending and changed the loop interface. The argument N **must be non-negative**.

```

295 \def\xintLengthUpTo {\romannumeral0\xintlengthupto}%
296 \long\def\xintlengthupto #1#2%
297 {%
298   \expandafter\XINT_lengthupto_loop
299   \the\numexpr#1.#2\xint:\xint:\xint:\xint:\xint:\xint:\xint:
300   \xint_c_vii\xint_c_vi\xint_c_v\xint_c_iv
301   \xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye.%
302 }%
303 \def\XINT_lengthupto_loop_a #1%
304 {%
305   \xint_UDsignfork
306   #1\XINT_lengthupto_gt
307   -\XINT_lengthupto_loop
308   \kroft #1%
309 }%
310 \long\def\XINT_lengthupto_gt #1\xint_bye.{-0}%
311 \long\def\XINT_lengthupto_loop #1.#2#3#4#5#6#7#8#9%
312 {%
313   \xint_gob_til_xint: #9\XINT_lengthupto_finish_a\xint:%
314   \expandafter\XINT_lengthupto_loop_a\the\numexpr #1-\xint_c_viii.%
315 }%
316 \def\XINT_lengthupto_finish_a\xint:\expandafter\XINT_lengthupto_loop_a
317   \the\numexpr #1-\xint_c_viii.#2#3#4#5#6#7#8#9%
318 {%
319   \expandafter\XINT_lengthupto_finish_b\the\numexpr #1-#9\xint_bye

```

```

320 }%
321 \def\XINT_lengthupto_finish_b #1#2.%
322 {%
323     \xint_UDsignfork
324     #1{-0}%
325     -{ #1#2}%
326     \krof
327 }%

```

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). The code in Joseph's post does |#1| replications when input #1 is negative and then activates an error triggering macro; here we simply do nothing when #1 is negative.

When #1 is already explicit digits (even #1=0, but non-negative) one can call the macro directly as \romannumeral\XINT_rep #1\endcsname {foo} to skip the \numexpr.

Expansion must be triggered by a \romannumeral.

```

328 \def\xintreplicate#1%
329     {\expandafter\XINT_replicate\the\numexpr#1\endcsname}%
330 \def\XINT_replicate #1{\xint_UDsignfork
331     #1\XINT_rep_neg
332     -\XINT_rep
333     \krof #1}%
334 \long\def\XINT_rep_neg #1\endcsname #2{\xint_c_}%
335 \def\XINT_rep #1{\csname XINT_rep_f#1\XINT_rep_a}%
336 \def\XINT_rep_a #1{\csname XINT_rep_#1\XINT_rep_a}%
337 \def\XINT_rep_\XINT_rep_a{\endcsname}%
338 \long\expandafter\def\csname XINT_rep_0\endcsname #1%
339     {\endcsname{#1#1#1#1#1#1#1#1#1#1}}%
340 \long\expandafter\def\csname XINT_rep_1\endcsname #1%
341     {\endcsname{#1#1#1#1#1#1#1#1#1#1#1}}%
342 \long\expandafter\def\csname XINT_rep_2\endcsname #1%
343     {\endcsname{#1#1#1#1#1#1#1#1#1#1#1#1}}%
344 \long\expandafter\def\csname XINT_rep_3\endcsname #1%
345     {\endcsname{#1#1#1#1#1#1#1#1#1#1#1#1#1}}%
346 \long\expandafter\def\csname XINT_rep_4\endcsname #1%
347     {\endcsname{#1#1#1#1#1#1#1#1#1#1#1#1#1#1}}%
348 \long\expandafter\def\csname XINT_rep_5\endcsname #1%
349     {\endcsname{#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1}}%
350 \long\expandafter\def\csname XINT_rep_6\endcsname #1%
351     {\endcsname{#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1}}%
352 \long\expandafter\def\csname XINT_rep_7\endcsname #1%
353     {\endcsname{#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1}}%
354 \long\expandafter\def\csname XINT_rep_8\endcsname #1%
355     {\endcsname{#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1}}%
356 \long\expandafter\def\csname XINT_rep_9\endcsname #1%
357     {\endcsname{#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1#1}}%
358 \long\expandafter\def\csname XINT_rep_f0\endcsname #1%
359     {\xint_c_}%

```

1 Package *xintkernel* implementation

```

360 \long\expandafter\def\csname XINT_rep_f1\endcsname #1%
361   {\xint_c_ #1}%
362 \long\expandafter\def\csname XINT_rep_f2\endcsname #1%
363   {\xint_c_ #1#1}%
364 \long\expandafter\def\csname XINT_rep_f3\endcsname #1%
365   {\xint_c_ #1#1#1}%
366 \long\expandafter\def\csname XINT_rep_f4\endcsname #1%
367   {\xint_c_ #1#1#1#1}%
368 \long\expandafter\def\csname XINT_rep_f5\endcsname #1%
369   {\xint_c_ #1#1#1#1#1}%
370 \long\expandafter\def\csname XINT_rep_f6\endcsname #1%
371   {\xint_c_ #1#1#1#1#1#1}%
372 \long\expandafter\def\csname XINT_rep_f7\endcsname #1%
373   {\xint_c_ #1#1#1#1#1#1#1}%
374 \long\expandafter\def\csname XINT_rep_f8\endcsname #1%
375   {\xint_c_ #1#1#1#1#1#1#1#1}%
376 \long\expandafter\def\csname XINT_rep_f9\endcsname #1%
377   {\xint_c_ #1#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.

```

378 \def\xintgobble #1%
379   {\csname xint_c_\expandafter\XINT_gobble_a\the\numexpr#1.0}%
380 \def\XINT_gobble #1.{\csname xint_c_\XINT_gobble_a #1.0}%
381 \def\XINT_gobble_a #1{\xint_gob_til_zero#1\XINT_gobble_d0\XINT_gobble_b#1}%
382 \def\XINT_gobble_b #1.#2%
383   {\expandafter\XINT_gobble_c
384     \the\numexpr (#1+\xint_c_v)/\xint_c_ix-\xint_c_i\expandafter.%
385     \the\numexpr #2+\xint_c_i.#1}%
386 \def\XINT_gobble_c #1.#2.#3%
387   {\csname XINT_g#2\the\numexpr#3-\xint_c_ix*#1\relax\XINT_gobble_a #1.#2}%
388 \def\XINT_gobble_d0\XINT_gobble_b0.#1{\endcsname}%
389 \expandafter\let\csname XINT_g10\endcsname\endcsname
390 \long\expandafter\def\csname XINT_g11\endcsname#1{\endcsname}%
391 \long\expandafter\def\csname XINT_g12\endcsname#1#2{\endcsname}%
392 \long\expandafter\def\csname XINT_g13\endcsname#1#2#3{\endcsname}%
393 \long\expandafter\def\csname XINT_g14\endcsname#1#2#3#4{\endcsname}%
394 \long\expandafter\def\csname XINT_g15\endcsname#1#2#3#4#5{\endcsname}%
395 \long\expandafter\def\csname XINT_g16\endcsname#1#2#3#4#5#6{\endcsname}%
396 \long\expandafter\def\csname XINT_g17\endcsname#1#2#3#4#5#6#7{\endcsname}%
397 \long\expandafter\def\csname XINT_g18\endcsname#1#2#3#4#5#6#7#8{\endcsname}%
398 \expandafter\let\csname XINT_g20\endcsname\endcsname
399 \long\expandafter\def\csname XINT_g21\endcsname #1#2#3#4#5#6#7#8#9%
400   {\endcsname}%
401 \long\expandafter\edef\csname XINT_g22\endcsname #1#2#3#4#5#6#7#8#9%
402   {\expandafter\noexpand\csname XINT_g21\endcsname}%
403 \long\expandafter\edef\csname XINT_g23\endcsname #1#2#3#4#5#6#7#8#9%

```

1 Package *xintkernel* implementation

```
404 {\expandafter\noexpand\csname XINT_g22\endcsname}%
405 \long\expandafter\edef\csname XINT_g24\endcsname #1#2#3#4#5#6#7#8#9%
406 {\expandafter\noexpand\csname XINT_g23\endcsname}%
407 \long\expandafter\edef\csname XINT_g25\endcsname #1#2#3#4#5#6#7#8#9%
408 {\expandafter\noexpand\csname XINT_g24\endcsname}%
409 \long\expandafter\edef\csname XINT_g26\endcsname #1#2#3#4#5#6#7#8#9%
410 {\expandafter\noexpand\csname XINT_g25\endcsname}%
411 \long\expandafter\edef\csname XINT_g27\endcsname #1#2#3#4#5#6#7#8#9%
412 {\expandafter\noexpand\csname XINT_g26\endcsname}%
413 \long\expandafter\edef\csname XINT_g28\endcsname #1#2#3#4#5#6#7#8#9%
414 {\expandafter\noexpand\csname XINT_g27\endcsname}%
415 \expandafter\let\csname XINT_g30\endcsname\endcsname
416 \long\expandafter\edef\csname XINT_g31\endcsname #1#2#3#4#5#6#7#8#9%
417 {\expandafter\noexpand\csname XINT_g28\endcsname}%
418 \long\expandafter\edef\csname XINT_g32\endcsname #1#2#3#4#5#6#7#8#9%
419 {\noexpand\csname XINT_g31\expandafter\noexpand\csname XINT_g28\endcsname}%
420 \long\expandafter\edef\csname XINT_g33\endcsname #1#2#3#4#5#6#7#8#9%
421 {\noexpand\csname XINT_g32\expandafter\noexpand\csname XINT_g28\endcsname}%
422 \long\expandafter\edef\csname XINT_g34\endcsname #1#2#3#4#5#6#7#8#9%
423 {\noexpand\csname XINT_g33\expandafter\noexpand\csname XINT_g28\endcsname}%
424 \long\expandafter\edef\csname XINT_g35\endcsname #1#2#3#4#5#6#7#8#9%
425 {\noexpand\csname XINT_g34\expandafter\noexpand\csname XINT_g28\endcsname}%
426 \long\expandafter\edef\csname XINT_g36\endcsname #1#2#3#4#5#6#7#8#9%
427 {\noexpand\csname XINT_g35\expandafter\noexpand\csname XINT_g28\endcsname}%
428 \long\expandafter\edef\csname XINT_g37\endcsname #1#2#3#4#5#6#7#8#9%
429 {\noexpand\csname XINT_g36\expandafter\noexpand\csname XINT_g28\endcsname}%
430 \long\expandafter\edef\csname XINT_g38\endcsname #1#2#3#4#5#6#7#8#9%
431 {\noexpand\csname XINT_g37\expandafter\noexpand\csname XINT_g28\endcsname}%
432 \expandafter\let\csname XINT_g40\endcsname\endcsname
433 \expandafter\edef\csname XINT_g41\endcsname
434 {\noexpand\csname XINT_g38\expandafter\noexpand\csname XINT_g31\endcsname}%
435 \expandafter\edef\csname XINT_g42\endcsname
436 {\noexpand\csname XINT_g41\expandafter\noexpand\csname XINT_g41\endcsname}%
437 \expandafter\edef\csname XINT_g43\endcsname
438 {\noexpand\csname XINT_g42\expandafter\noexpand\csname XINT_g41\endcsname}%
439 \expandafter\edef\csname XINT_g44\endcsname
440 {\noexpand\csname XINT_g43\expandafter\noexpand\csname XINT_g41\endcsname}%
441 \expandafter\edef\csname XINT_g45\endcsname
442 {\noexpand\csname XINT_g44\expandafter\noexpand\csname XINT_g41\endcsname}%
443 \expandafter\edef\csname XINT_g46\endcsname
444 {\noexpand\csname XINT_g45\expandafter\noexpand\csname XINT_g41\endcsname}%
445 \expandafter\edef\csname XINT_g47\endcsname
446 {\noexpand\csname XINT_g46\expandafter\noexpand\csname XINT_g41\endcsname}%
447 \expandafter\edef\csname XINT_g48\endcsname
448 {\noexpand\csname XINT_g47\expandafter\noexpand\csname XINT_g41\endcsname}%
449 \expandafter\let\csname XINT_g50\endcsname\endcsname
450 \expandafter\edef\csname XINT_g51\endcsname
451 {\noexpand\csname XINT_g48\expandafter\noexpand\csname XINT_g41\endcsname}%
452 \expandafter\edef\csname XINT_g52\endcsname
453 {\noexpand\csname XINT_g51\expandafter\noexpand\csname XINT_g51\endcsname}%
454 \expandafter\edef\csname XINT_g53\endcsname
455 {\noexpand\csname XINT_g52\expandafter\noexpand\csname XINT_g51\endcsname}%
```

1 Package *xintkernel* implementation

```
456 \expandafter\edef\csname XINT_g54\endcsname
457 {\noexpand\csname XINT_g53\expandafter\noexpand\csname XINT_g51\endcsname}%
458 \expandafter\edef\csname XINT_g55\endcsname
459 {\noexpand\csname XINT_g54\expandafter\noexpand\csname XINT_g51\endcsname}%
460 \expandafter\edef\csname XINT_g56\endcsname
461 {\noexpand\csname XINT_g55\expandafter\noexpand\csname XINT_g51\endcsname}%
462 \expandafter\edef\csname XINT_g57\endcsname
463 {\noexpand\csname XINT_g56\expandafter\noexpand\csname XINT_g51\endcsname}%
464 \expandafter\edef\csname XINT_g58\endcsname
465 {\noexpand\csname XINT_g57\expandafter\noexpand\csname XINT_g51\endcsname}%
466 \expandafter\let\csname XINT_g60\endcsname\endcsname
467 \expandafter\edef\csname XINT_g61\endcsname
468 {\noexpand\csname XINT_g58\expandafter\noexpand\csname XINT_g51\endcsname}%
469 \expandafter\edef\csname XINT_g62\endcsname
470 {\noexpand\csname XINT_g61\expandafter\noexpand\csname XINT_g61\endcsname}%
471 \expandafter\edef\csname XINT_g63\endcsname
472 {\noexpand\csname XINT_g62\expandafter\noexpand\csname XINT_g61\endcsname}%
473 \expandafter\edef\csname XINT_g64\endcsname
474 {\noexpand\csname XINT_g63\expandafter\noexpand\csname XINT_g61\endcsname}%
475 \expandafter\edef\csname XINT_g65\endcsname
476 {\noexpand\csname XINT_g64\expandafter\noexpand\csname XINT_g61\endcsname}%
477 \expandafter\edef\csname XINT_g66\endcsname
478 {\noexpand\csname XINT_g65\expandafter\noexpand\csname XINT_g61\endcsname}%
479 \expandafter\edef\csname XINT_g67\endcsname
480 {\noexpand\csname XINT_g66\expandafter\noexpand\csname XINT_g61\endcsname}%
481 \expandafter\edef\csname XINT_g68\endcsname
482 {\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.

```
483 \def\xintMessage #1#2#3{%
484   \immediate\write128{Package #1 #2: (on line \the\inputlineno)}%
485   \immediate\write128{\space\space\space\space#3}%
486 }%
487 \newif\ifxintverbose
```

1.18 (WIP) Expandable error message

Incorporated in 1.2l release, but really belongs to next major release.

This is copied over from `l3kernel` code. I am using `\ ! /` control sequence though, which must be left undefined. `\xintError:` would be 6 letters more already. Utiliser `\FPE: ?` (mais ce n'est pas uniquement du « floating point »)

Always used in context where expansion was launched by a `\romannumeral0` or `\romannumeral`^^@`.

```
488 \def\XINT_expandableerror #1#2{%
489   \def\XINT_expandableerror ##1{%
490     \expandafter\expandafter\expandafter
491       \XINT_expandableerror_continue\xint_firstofone{#2#1##1#1}}%
492   \def\XINT_expandableerror_continue ##1#1##2#1{##1}%
493 }%
```

1 Package `xintkernel` implementation

```
494 \begingroup\lccode`$ 32 \catcode`/ 11 \catcode`! 11 \catcode32 11 %
495 \lowercase{\endgroup\XINT_expandableerror$\ ! /}%
496 \XINT_restorecatcodes_endinput%
```

2 Package `xinttools` implementation

.1	Catcodes, ε - \TeX and reload detection . .	16	<code>breakloopanddo</code> , <code>\xintloopskiptonext</code> ,	
.2	Package identification	17	<code>\xintloopskipandredo</code>	32
.3	<code>\xintgodef</code> , <code>\xintgoodef</code> , <code>\xintgfdef</code> .	17	.21 <code>\XINT_xflet</code>	33
.4	<code>\xintRevWithBraces</code>	17	.22 <code>\xintApplyInline</code>	33
.5	<code>\xintZapFirstSpaces</code>	18	.23 <code>\xintFor</code> , <code>\xintFor*</code> , <code>\xintBreakFor</code> ,	
.6	<code>\xintZapLastSpaces</code>	19	<code>\xintBreakForAndDo</code>	34
.7	<code>\xintZapSpaces</code>	20	.24 <code>\XINT_forever</code> , <code>\xintintegers</code> , <code>\xintdi-</code>	
.8	<code>\xintZapSpacesB</code>	20	<code>mensions</code> , <code>\xintrationals</code>	37
.9	<code>\xintCSVtoList</code> , <code>\xintCSVtoListNon-</code>		.25 <code>\xintForpair</code> , <code>\xintForthree</code> , <code>\xintFor-</code>	
	<code>Stripped</code>	20	<code>four</code>	38
.10	<code>\xintListWithSep</code>	22	.26 <code>\xintAssign</code> , <code>\xintAssignArray</code> , <code>\xint-</code>	
.11	<code>\xintNthElt</code>	23	<code>DigitsOf</code>	40
.12	<code>\xintKeep</code>	24	.27 CSV (non user documented) variants of	
.13	<code>\xintKeepUnbraced</code>	25	Length, Keep, Trim, NthElt, Reverse . .	43
.14	<code>\xintTrim</code>	27	.27.1 <code>\xintLength:f:csv</code>	43
.15	<code>\xintTrimUnbraced</code>	28	.27.2 <code>\xintLengthUpTo:f:csv</code>	44
.16	<code>\xintApply</code>	29	.27.3 <code>\xintKeep:f:csv</code>	45
.17	<code>\xintApplyUnbraced</code>	29	.27.4 <code>\xintTrim:f:csv</code>	47
.18	<code>\xintSeq</code>	30	.27.5 <code>\xintNthEltPy:f:csv</code>	48
.19	<code>\xintloop</code> , <code>\xintbreakloop</code> , <code>\xintbreak-</code>		.27.6 <code>\xintReverse:f:csv</code>	50
	<code>loopanddo</code> , <code>\xintloopskiptonext</code> . . .	32	.27.7 <code>\xintFirstItem:f:csv</code>	50
.20	<code>\xintiloop</code> , <code>\xintiloopindex</code> , <code>\xintouter-</code>		.27.8 <code>\xintLastItem:f:csv</code>	50
	<code>iloopindex</code> , <code>\xintbreakiloop</code> , <code>\xint-</code>		.27.9 Public names for the undocumented csv macros	51

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 [2018/02/06 1.2q 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:*, here and in other locations, is in case #1 expands to nothing, the *\romannumeral-`0* must be stopped

```

52 \def\xintRevWithBraces      {\romannumeral0\xintrevwithbraces }%
53 \def\xintRevWithBracesNoExpand {\romannumeral0\xintrevwithbracesnoexpand }%
54 \long\def\xintrevwithbraces #1%

```

```

55 {%
56   \expandafter\XINT_revwbr_loop\expandafter{\expandafter}%
57   \romannumeral`&&@#1\xint:\xint:\xint:\xint:%
58           \xint:\xint:\xint:\xint:\xint_bye
59 }%
60 \long\def\xintrevwithbracesnoexpand #1%
61 {%
62   \XINT_revwbr_loop {%
63   #1\xint:\xint:\xint:\xint:%
64   \xint:\xint:\xint:\xint:\xint_bye
65 }%
66 \long\def\XINT_revwbr_loop #1#2#3#4#5#6#7#8#9%
67 {%
68   \xint_gob_til_xint: #9\XINT_revwbr_finish_a\xint:%
69   \XINT_revwbr_loop {{#9}{#8}{#7}{#6}{#5}{#4}{#3}{#2}{#1}%
70 }%
71 \long\def\XINT_revwbr_finish_a\xint:\XINT_revwbr_loop #1#2\xint_bye
72 {%
73   \XINT_revwbr_finish_b #2\R\R\R\R\R\R\R\Z #1%
74 }%
75 \def\XINT_revwbr_finish_b #1#2#3#4#5#6#7#8\Z
76 {%
77   \xint_gob_til_R
78       #1\XINT_revwbr_finish_c \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 \def\XINT_revwbr_finish_c#1{%
88 \def\XINT_revwbr_finish_c##1##2\Z{\expandafter#1##1}%
89 }\XINT_revwbr_finish_c{ }%

```

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.

```

90 \def\xintZapFirstSpaces {\romannumeral0\xintzapfirstspaces }%
91 \def\xintzapfirstspaces#1{\long
92 \def\xintzapfirstspaces ##1{\XINT_zapbsp_a #1##1\xint:#1#1\xint:}%
93 }\xintzapfirstspaces{ }%

```

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:. 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:<sp><sp>\xint:

```
94 \def\XINT_zapbsp_a#1{\long\def\XINT_zapbsp_a ##1#1#1{%
95 \XINT_zapbsp_again?##1\xint_bye\XINT_zapbsp_b ##1#1#1}%
96 }\XINT_zapbsp_a{ }%
97 \long\def\XINT_zapbsp_again? #1{\xint_bye #1\XINT_zapbsp_again }%
98 \xint_firstofone{\def\XINT_zapbsp_again\XINT_zapbsp_b} {\XINT_zapbsp_a }%
99 \long\def\XINT_zapbsp_b #1\xint:#2\xint:{#1}%
```

2.6 \xintZapLastSpaces

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

```
100 \def\xintZapLastSpaces {\romannumeral0\xintzaplastspaces }%
101 \def\xintzaplastspaces#1{\long
102 \def\xintzaplastspaces ##1{\XINT_zapbsp_a {} \empty##1#1#1\xint_bye\xint:}%
103 }\xintzaplastspaces{ }%
```

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

```
104 \xint_firstofone {\long\def\XINT_zapbsp_a #1#2 } %<- second space here
105 {\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:. 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 ?;-).

```
106 \long\def\XINT_zapbsp_b #1#2#3\xint:%
107 {\XINT_zapbsp_end? #3\XINT_zapbsp_e {#2#1}\empty #3\xint:}%
```

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.

```
108 \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.

```
109 \long\def\XINT_zapbsp_end\XINT_zapbsp_e #1#2\xint:{ #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.

```
110 \def\XINT_zapbsp_e#1{%
111 \long\def\XINT_zapbsp_e ##1{\XINT_zapbsp_a {##1#1#1}}%
112 }\XINT_zapbsp_e{ }%
```

2.7 `\xintZapSpaces`

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

```

113 \def\xintZapSpaces {\romannumeral0\xintzapspaces }%
114 \def\xintzapspaces#1{%
115 \long\def\xintzapspaces ##1% like \xintZapFirstSpaces.
116     {\XINT_zapsp_a #1##1\xint:#1#1\xint:}%
117 }\xintzapspaces{ }%
118 \def\XINT_zapsp_a#1{%
119 \long\def\XINT_zapsp_a ##1#1#1%
120     {\XINT_zapsp_again?##1\xint_bye\XINT_zapsp_b##1#1#1}%
121 }\XINT_zapsp_a{ }%
122 \long\def\XINT_zapsp_again? #1{\xint_bye #1\XINT_zapsp_again }%
123 \xint_firstofone{\def\XINT_zapsp_again\XINT_zapsp_b} {\XINT_zapsp_a }%
124 \xint_firstofone{\def\XINT_zapsp_b} {\XINT_zapsp_c }%
125 \def\XINT_zapsp_c#1{%
126 \long\def\XINT_zapsp_c ##1\xint:##2\xint:%
127     {\XINT_zapsp_a{ }\empty ##1#1#1\xint_bye\xint:}%
128 }\XINT_zapsp_c{ }%

```

2.8 `\xintZapSpacesB`

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

```

129 \def\xintZapSpacesB {\romannumeral0\xintzapspacesb }%
130 \long\def\xintzapspacesb #1{\XINT_zapspb_one? #1\xint:\xint:%
131     \xint_bye\xintzapspaces {#1}}%
132 \long\def\XINT_zapspb_one? #1#2%
133     {\xint_gob_til_xint: #1\XINT_zapspb_onlyspaces\xint:%
134     \xint_gob_til_xint: #2\XINT_zapspb_bracedorone\xint:%
135     \xint_bye {#1}}%
136 \def\XINT_zapspb_onlyspaces\xint:%
137     \xint_gob_til_xint:\xint:\XINT_zapspb_bracedorone\xint:%
138     \xint_bye #1\xint_bye\xintzapspaces #2{ }%
139 \long\def\XINT_zapspb_bracedorone\xint:%
140     \xint_bye #1\xint:\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.

```

141 \def\xintCSVtoList {\romannumeral0\xintcsvtolist }%
142 \long\def\xintcsvtolist #1{\expandafter\xintApply
143     \expandafter\xintzapspacesb
144     \expandafter{\romannumeral0\xintcsvtolistnonstripped{#1}}}%

```

2 Package *xinttools* implementation

```

145 \def\xintCSVtoListNoExpand {\romannumeral0\xintcsvtolistnoexpand }%
146 \long\def\xintcsvtolistnoexpand #1{\expandafter\xintApply
147     \expandafter\xintzapspacesb
148     \expandafter{\romannumeral0\xintcsvtolistnonstrippednoexpand{#1}}}%
149 \def\xintCSVtoListNonStripped {\romannumeral0\xintcsvtolistnonstripped }%
150 \def\xintCSVtoListNonStrippedNoExpand
151     {\romannumeral0\xintcsvtolistnonstrippednoexpand }%
152 \long\def\xintcsvtolistnonstripped #1%
153 {%
154     \expandafter\XINT_csvtol_loop_a\expandafter
155     {\expandafter}\romannumeral`&&@#1%
156     ,\xint_bye,\xint_bye,\xint_bye,\xint_bye
157     ,\xint_bye,\xint_bye,\xint_bye,\xint_bye,\Z
158 }%
159 \long\def\xintcsvtolistnonstrippednoexpand #1%
160 {%
161     \XINT_csvtol_loop_a
162     {#1,\xint_bye,\xint_bye,\xint_bye,\xint_bye
163     ,\xint_bye,\xint_bye,\xint_bye,\xint_bye,\Z
164 }%
165 \long\def\XINT_csvtol_loop_a #1#2,#3,#4,#5,#6,#7,#8,#9,%
166 {%
167     \xint_bye #9\XINT_csvtol_finish_a\xint_bye
168     \XINT_csvtol_loop_b {#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}}%
169 }%
170 \long\def\XINT_csvtol_loop_b #1#2{\XINT_csvtol_loop_a {#1#2}}%
171 \long\def\XINT_csvtol_finish_a\xint_bye\XINT_csvtol_loop_b #1#2#3\Z
172 {%
173     \XINT_csvtol_finish_b #3\R,\R,\R,\R,\R,\R,\R,\Z #2{#1}%
174 }%

```

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`).

```

175 \def\XINT_csvtol_finish_b #1,#2,#3,#4,#5,#6,#7,#8\Z
176 {%
177     \xint_gob_til_R
178     #1\expandafter\XINT_csvtol_finish_dviii\xint_gob_til_Z
179     #2\expandafter\XINT_csvtol_finish_dvii \xint_gob_til_Z
180     #3\expandafter\XINT_csvtol_finish_dvi \xint_gob_til_Z
181     #4\expandafter\XINT_csvtol_finish_dv \xint_gob_til_Z
182     #5\expandafter\XINT_csvtol_finish_div \xint_gob_til_Z
183     #6\expandafter\XINT_csvtol_finish_diii \xint_gob_til_Z
184     #7\expandafter\XINT_csvtol_finish_dii \xint_gob_til_Z
185     \R\XINT_csvtol_finish_di \Z
186 }%
187 \long\def\XINT_csvtol_finish_dviii #1#2#3#4#5#6#7#8#9{ #9}%
188 \long\def\XINT_csvtol_finish_dvii #1#2#3#4#5#6#7#8#9{ #9{#1}}%
189 \long\def\XINT_csvtol_finish_dvi #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}}%
190 \long\def\XINT_csvtol_finish_dv #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}}%
191 \long\def\XINT_csvtol_finish_div #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}{#4}}%
192 \long\def\XINT_csvtol_finish_diii #1#2#3#4#5#6#7#8#9{ #9{#1}{#2}{#3}{#4}{#5}}%

```

```

193 \long\def\XINT_csvtol_finish_dii    #1#2#3#4#5#6#7#8#9%
194                                     { #9{#1}{#2}{#3}{#4}{#5}{#6}}%
195 \long\def\XINT_csvtol_finish_di\Z    #1#2#3#4#5#6#7#8#9%
196                                     { #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. The "list" argument may be empty.

`\xintListWithSepNoExpand` does not f-expand its second argument.

This venerable macro from 1.04 remained unchanged for a long time and was finally refactored at 1.2p for increased speed. Tests done with a list of identical `{\x}` items and a sep of `\z` demonstrated a speed increase of about:

- 3x for 30 items,
- 4.5x for 100 items,
- 7.5x--8x for 1000 items.

```

197 \def\xintListWithSep          {\romannumeral0\xintlistwithsep}%
198 \def\xintListWithSepNoExpand {\romannumeral0\xintlistwithsepnoexpand}%
199 \long\def\xintlistwithsep #1#2%
200     {\expandafter\XINT_lws\expandafter {\romannumeral`&&@#2}{#1}}%
201 \long\def\xintlistwithsepnoexpand #1#2%
202 {%
203     \XINT_lws_loop_a {#1}#2{\xint_bye\XINT_lws_e_vi}%
204     {\xint_bye\XINT_lws_e_v}{\xint_bye\XINT_lws_e_iv}%
205     {\xint_bye\XINT_lws_e_iii}{\xint_bye\XINT_lws_e_ii}%
206     {\xint_bye\XINT_lws_e_i}{\xint_bye\XINT_lws_e}%
207     {\xint_bye\expandafter\space}\xint_bye
208}%
209 \long\def\XINT_lws #1#2%
210 {%
211     \XINT_lws_loop_a {#2}#1{\xint_bye\XINT_lws_e_vi}%
212     {\xint_bye\XINT_lws_e_v}{\xint_bye\XINT_lws_e_iv}%
213     {\xint_bye\XINT_lws_e_iii}{\xint_bye\XINT_lws_e_ii}%
214     {\xint_bye\XINT_lws_e_i}{\xint_bye\XINT_lws_e}%
215     {\xint_bye\expandafter\space}\xint_bye
216}%
217 \long\def\XINT_lws_loop_a #1#2#3#4#5#6#7#8#9%
218 {%
219     \xint_bye #9\xint_bye
220     \XINT_lws_loop_b {#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}%
221}%
222 \long\def\XINT_lws_loop_b #1#2#3#4#5#6#7#8#9%
223 {%
224     \XINT_lws_loop_a {#1}{#2#1#3#1#4#1#5#1#6#1#7#1#8#1#9}%
225}%
226 \long\def\XINT_lws_e_vi\xint_bye\XINT_lws_loop_b #1#2#3#4#5#6#7#8#9\xint_bye
227     { #2#1#3#1#4#1#5#1#6#1#7#1#8}%
228 \long\def\XINT_lws_e_v\xint_bye\XINT_lws_loop_b #1#2#3#4#5#6#7#8\xint_bye
229     { #2#1#3#1#4#1#5#1#6#1#7}%
230 \long\def\XINT_lws_e_iv\xint_bye\XINT_lws_loop_b #1#2#3#4#5#6#7\xint_bye
231     { #2#1#3#1#4#1#5#1#6}%

```

```

232 \long\def\XINT_lws_e_iii\xint_bye\XINT_lws_loop_b #1#2#3#4#5#6\xint_bye
233   { #2#1#3#1#4#1#5}%
234 \long\def\XINT_lws_e_ii\xint_bye\XINT_lws_loop_b #1#2#3#4#5\xint_bye
235   { #2#1#3#1#4}%
236 \long\def\XINT_lws_e_i\xint_bye\XINT_lws_loop_b #1#2#3#4\xint_bye
237   { #2#1#3}%
238 \long\def\XINT_lws_e\xint_bye\XINT_lws_loop_b #1#2#3\xint_bye
239   { #2}%

```

2.11 \xintNthElt

First included in release 1.06. Last refactored in 1.2j.

\xintNthElt {i}{List} returns the *i* th item from List (one pair of braces removed). The list is first f-expanded. The \xintNthEltNoExpand does no expansion of its second argument. Both variants expand *i* inside \numexpr.

With *i* = 0, the number of items is returned using \xintlength but with the List argument f-expanded first.

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

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

```

240 \def\xintNthElt          {\romannumeral0\xintnthelt }%
241 \def\xintNthEltNoExpand {\romannumeral0\xintntheltnoexpand}%
242 \long\def\xintnthelt #1#2{\expandafter\XINT_nthelt_a\the\numexpr #1\expandafter.%
243   \expandafter{\romannumeral`&&@#2}}%
244 \def\xintntheltnoexpand #1{\expandafter\XINT_nthelt_a\the\numexpr #1.}%
245 \def\XINT_nthelt_a #1%
246 {%
247   \xint_UDzerominusfork
248     #1-\XINT_nthelt_zero
249     0#1\XINT_nthelt_neg
250     0-{\XINT_nthelt_pos #1}%
251   \krof
252 }%
253 \def\XINT_nthelt_zero #1.{\xintlength}%
254 \long\def\XINT_nthelt_neg #1.#2%
255 {%
256   \expandafter\XINT_nthelt_neg_a\the\numexpr\xint_c_i+\XINT_length_loop
257   #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
258   \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
259   \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c\xint_bye
260   -#1.#2\xint_bye
261 }%
262 \def\XINT_nthelt_neg_a #1%
263 {%
264   \xint_UDzerominusfork
265     #1-\xint_bye_thenstop
266     0#1\xint_bye_thenstop
267     0-{}%
268   \krof
269   \expandafter\XINT_nthelt_neg_b
270   \romannumeral\expandafter\XINT_gobble\the\numexpr-\xint_c_i+#1%
271 }%
272 \long\def\XINT_nthelt_neg_b #1#2\xint_bye{ #1}%

```

```

273 \long\def\XINT_nthelt_pos #1.#2%
274 {%
275     \expandafter\XINT_nthelt_pos_done
276     \romannumeral0\expandafter\XINT_trim_loop\the\numexpr#1-\xint_c_x.%
277     #2\xint:\xint:\xint:\xint:\xint:%
278     \xint:\xint:\xint:\xint:\xint:%
279     \xint_bye
280 }%
281 \def\XINT_nthelt_pos_done #1{%
282 \long\def\XINT_nthelt_pos_done ##1##2\xint_bye{%
283     \xint_gob_til_xint:##1\expandafter#1\xint_gobble_ii\xint:#1##1}%
284 }\XINT_nthelt_pos_done{ }%

```

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.

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.

```

285 \def\xintKeep          {\romannumeral0\xintkeep }%
286 \def\xintKeepNoExpand {\romannumeral0\xintkeepnoexpand }%
287 \long\def\xintkeep #1#2{\expandafter\XINT_keep_a\the\numexpr #1\expandafter.%
288     \expandafter{\romannumeral`&&@#2}}%
289 \def\xintkeepnoexpand #1{\expandafter\XINT_keep_a\the\numexpr #1.}%
290 \def\XINT_keep_a #1%
291 {%
292     \xint_UDzerominusfork
293     #1-\XINT_keep_keeptime
294     0#1\XINT_keep_neg
295     0-{\XINT_keep_pos #1}%
296     \krof
297 }%
298 \long\def\XINT_keep_keeptime .#1{ }%
299 \long\def\XINT_keep_neg #1.#2%
300 {%
301     \expandafter\XINT_keep_neg_a\the\numexpr
302     #1-\numexpr\XINT_length_loop
303     #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
304     \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
305     \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye.#2%
306 }%
307 \def\XINT_keep_neg_a #1%
308 {%

```



```

309 \xint_UDsignfork
310 #1{\expandafter\space\romannumeral\xINT_gobble}%
311 -\XINT_keep_keepall
312 \krof
313 }%
314 \def\xINT_keep_keepall #1.{ }%
315 \long\def\xINT_keep_pos #1.#2%
316 {%
317 \expandafter\xINT_keep_loop
318 \the\numexpr#1-\XINT_lengthupto_loop
319 #1.#2\xint:\xint:\xint:\xint:\xint:\xint:\xint:
320 \xint_c_vii\xint_c_vi\xint_c_v\xint_c_iv
321 \xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye.%
322 -\xint_c_viii.{ }#2\xint_bye%
323 }%
324 \def\xINT_keep_loop #1#2.%
325 {%
326 \xint_gob_til_minus#1\xINT_keep_loop_end-%
327 \expandafter\xINT_keep_loop
328 \the\numexpr#1#2-\xint_c_viii\expandafter.\XINT_keep_loop_pickeight
329 }%
330 \long\def\xINT_keep_loop_pickeight
331 #1#2#3#4#5#6#7#8#9{{#1{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}}}%
332 \def\xINT_keep_loop_end-\expandafter\xINT_keep_loop
333 \the\numexpr-#1-\xint_c_viii\expandafter.\XINT_keep_loop_pickeight
334 {\csname XINT_keep_end#1\endcsname}%
335 \long\expandafter\def\csname XINT_keep_end1\endcsname
336 #1#2#3#4#5#6#7#8#9\xint_bye { #1{#2}{#3}{#4}{#5}{#6}{#7}{#8}}}%
337 \long\expandafter\def\csname XINT_keep_end2\endcsname
338 #1#2#3#4#5#6#7#8\xint_bye { #1{#2}{#3}{#4}{#5}{#6}{#7}}}%
339 \long\expandafter\def\csname XINT_keep_end3\endcsname
340 #1#2#3#4#5#6#7\xint_bye { #1{#2}{#3}{#4}{#5}{#6}}}%
341 \long\expandafter\def\csname XINT_keep_end4\endcsname
342 #1#2#3#4#5#6\xint_bye { #1{#2}{#3}{#4}{#5}}}%
343 \long\expandafter\def\csname XINT_keep_end5\endcsname
344 #1#2#3#4#5\xint_bye { #1{#2}{#3}{#4}}}%
345 \long\expandafter\def\csname XINT_keep_end6\endcsname
346 #1#2#3#4\xint_bye { #1{#2}{#3}}}%
347 \long\expandafter\def\csname XINT_keep_end7\endcsname
348 #1#2#3\xint_bye { #1{#2}}}%
349 \long\expandafter\def\csname XINT_keep_end8\endcsname
350 #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 $\text{length}(L) > i > 0$.

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*.

```

351 \def\xintKeepUnbraced {\romannumeral0\xintkeepunbraced }%
352 \def\xintKeepUnbracedNoExpand {\romannumeral0\xintkeepunbracednoexpand }%

```

2 Package *xinttools* implementation

```

353 \long\def\xintkeepunbraced #1#2%
354   {\expandafter\XINT_keepunbr_a\the\numexpr #1\expandafter.%
355     \expandafter{\romannumeral`&&@#2}}%
356 \def\xintkeepunbracednoexpand #1%
357   {\expandafter\XINT_keepunbr_a\the\numexpr #1.}%
358 \def\XINT_keepunbr_a #1%
359 {%
360   \xint_UDzerominusfork
361     #1-\XINT_keep_keepnone
362     0#1\XINT_keep_neg
363     0-{\XINT_keepunbr_pos #1}%
364   \krof
365 }%
366 \long\def\XINT_keepunbr_pos #1.#2%
367 {%
368   \expandafter\XINT_keepunbr_loop
369   \the\numexpr#1-\XINT_lengthupto_loop
370   #1.#2\xint:\xint:\xint:\xint:\xint:\xint:\xint:
371     \xint_c_vii\xint_c_vi\xint_c_v\xint_c_iv
372     \xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye.%
373   -\xint_c_viii.{ }#2\xint_bye%
374 }%
375 \def\XINT_keepunbr_loop #1#2.%
376 {%
377   \xint_gob_til_minus#1\XINT_keepunbr_loop_end-%
378   \expandafter\XINT_keepunbr_loop
379   \the\numexpr#1#2-\xint_c_viii\expandafter.\XINT_keepunbr_loop_pickeight
380 }%
381 \long\def\XINT_keepunbr_loop_pickeight
382   #1#2#3#4#5#6#7#8#9{{#1#2#3#4#5#6#7#8#9}}%
383 \def\XINT_keepunbr_loop_end-\expandafter\XINT_keepunbr_loop
384   \the\numexpr-#1-\xint_c_viii\expandafter.\XINT_keepunbr_loop_pickeight
385   {\csname XINT_keepunbr_end#1\endcsname}%
386 \long\expandafter\def\csname XINT_keepunbr_end1\endcsname
387   #1#2#3#4#5#6#7#8#9\xint_bye { #1#2#3#4#5#6#7#8}%
388 \long\expandafter\def\csname XINT_keepunbr_end2\endcsname
389   #1#2#3#4#5#6#7#8\xint_bye { #1#2#3#4#5#6#7}%
390 \long\expandafter\def\csname XINT_keepunbr_end3\endcsname
391   #1#2#3#4#5#6#7\xint_bye { #1#2#3#4#5#6}%
392 \long\expandafter\def\csname XINT_keepunbr_end4\endcsname
393   #1#2#3#4#5#6\xint_bye { #1#2#3#4#5}%
394 \long\expandafter\def\csname XINT_keepunbr_end5\endcsname
395   #1#2#3#4#5\xint_bye { #1#2#3#4}%
396 \long\expandafter\def\csname XINT_keepunbr_end6\endcsname
397   #1#2#3#4\xint_bye { #1#2#3}%
398 \long\expandafter\def\csname XINT_keepunbr_end7\endcsname
399   #1#2#3\xint_bye { #1#2}%
400 \long\expandafter\def\csname XINT_keepunbr_end8\endcsname
401   #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 $|i| \geq N$, the empty list is returned.

`\xintTrimNoExpand` does not expand the *L* argument.

Speed improvements with 1.2i for *i*<0 branch (which hands over to `\xintKeep`). Speed improvements with 1.2j for *i*>0 branch which gobbles items nine by nine despite not knowing in advance if it will go too far.

```

402 \def\xintTrim          {\romannumeral0\xinttrim }%
403 \def\xintTrimNoExpand {\romannumeral0\xinttrimnoexpand }%
404 \long\def\xinttrim #1#2{\expandafter\XINT_trim_a\the\numexpr #1\expandafter.%
405                      \expandafter{\romannumeral`&&@#2}}%
406 \def\xinttrimnoexpand #1{\expandafter\XINT_trim_a\the\numexpr #1.}%
407 \def\XINT_trim_a #1%
408 {%
409   \xint_UDzerominusfork
410     #1-\XINT_trim_trimnone
411     0#1\XINT_trim_neg
412     0-{\XINT_trim_pos #1}%
413   \krof
414 }%
415 \long\def\XINT_trim_trimnone .#1{ #1}%
416 \long\def\XINT_trim_neg #1.#2%
417 {%
418   \expandafter\XINT_trim_neg_a\the\numexpr
419   #1-\numexpr\XINT_length_loop
420   #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
421   \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
422   \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
423   .{ }#2\xint_bye
424 }%
425 \def\XINT_trim_neg_a #1%
426 {%
427   \xint_UDsignfork
428     #1{\expandafter\XINT_keep_loop\the\numexpr-\xint_c_viii+}%
429     -\XINT_trim_trimall
430   \krof
431 }%
432 \def\XINT_trim_trimall#1{%
433 \def\XINT_trim_trimall {\expandafter#1\xint_bye}%
434 }\XINT_trim_trimall{ }%

```

This branch doesn't pre-evaluate the length of the list argument. Redone again for 1.2j, manages to trim nine by nine. Some non optimal looking aspect of the code is for allowing sharing with `\xintNthElt`.

```

435 \long\def\XINT_trim_pos #1.#2%
436 {%

```

```

437 \expandafter\XINT_trim_pos_done\expandafter\space
438 \romannumeral0\expandafter\XINT_trim_loop\the\numexpr#1-\xint_c_ix.%
439 #2\xint:\xint:\xint:\xint:\xint:%
440 \xint:\xint:\xint:\xint:\xint:%
441 \xint_bye
442 }%
443 \def\XINT_trim_loop #1#2.%
444 {%
445 \xint_gob_til_minus#1\XINT_trim_finish-%
446 \expandafter\XINT_trim_loop\the\numexpr#1#2\XINT_trim_loop_trimnine
447 }%
448 \long\def\XINT_trim_loop_trimnine #1#2#3#4#5#6#7#8#9%
449 {%
450 \xint_gob_til_xint: #9\XINT_trim_toofew\xint:-\xint_c_ix.%
451 }%
452 \def\XINT_trim_toofew\xint:{*\xint_c_}%
453 \def\XINT_trim_finish#1{%
454 \def\XINT_trim_finish-%
455 \expandafter\XINT_trim_loop\the\numexpr-##1\XINT_trim_loop_trimnine
456 {%
457 \expandafter\expandafter\expandafter#1%
458 \csname xint_gobble_\romannumeral\numexpr\xint_c_ix-##1\endcsname
459 }}\XINT_trim_finish{ }%
460 \long\def\XINT_trim_pos_done #1\xint:#2\xint_bye {#1}%

```

2.15 \xintTrimUnbraced

1.2a. Modified in 1.2i like \xintTrim

```

461 \def\xintTrimUnbraced {\romannumeral0\xinttrimunbraced }%
462 \def\xintTrimUnbracedNoExpand {\romannumeral0\xinttrimunbracednoexpand }%
463 \long\def\xinttrimunbraced #1#2%
464 {\expandafter\XINT_trimunbr_a\the\numexpr #1\expandafter.%
465 \expandafter{\romannumeral`&&@#2}}%
466 \def\xinttrimunbracednoexpand #1%
467 {\expandafter\XINT_trimunbr_a\the\numexpr #1.}%
468 \def\XINT_trimunbr_a #1%
469 {%
470 \xint_UDzerominusfork
471 #1-\XINT_trim_trimnone
472 0#1\XINT_trimunbr_neg
473 0-{\XINT_trim_pos #1}%
474 \krof
475 }%
476 \long\def\XINT_trimunbr_neg #1.#2%
477 {%
478 \expandafter\XINT_trimunbr_neg_a\the\numexpr
479 #1-\numexpr\XINT_length_loop
480 #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
481 \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
482 \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
483 .{ }#2\xint_bye
484 }%

```

```

485 \def\XINT_trimunbr_neg_a #1%
486 {%
487   \xint_UDsignfork
488     #1{\expandafter\XINT_keeppunbr_loop\the\numexpr-\xint_c_viii+}%
489     -\XINT_trim_trimall
490   \krof
491 }%

```

2.16 \xintApply

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

```

492 \def\xintApply          {\romannumeral0\xintapply }%
493 \def\xintApplyNoExpand {\romannumeral0\xintapplynoexpand }%
494 \long\def\xintapply #1#2%
495 {%
496   \expandafter\XINT_apply\expandafter {\romannumeral`&&@#2}%
497   {#1}%
498 }%
499 \long\def\XINT_apply #1#2{\XINT_apply_loop_a {}{#2}#1\xint_bye }%
500 \long\def\xintapplynoexpand #1#2{\XINT_apply_loop_a {}{#1}#2\xint_bye }%
501 \long\def\XINT_apply_loop_a #1#2#3%
502 {%
503   \xint_bye #3\XINT_apply_end\xint_bye
504   \expandafter
505   \XINT_apply_loop_b
506   \expandafter {\romannumeral`&&@#2{#3}}{#1}{#2}%
507 }%
508 \long\def\XINT_apply_loop_b #1#2{\XINT_apply_loop_a {#2{#1}}}%
509 \long\def\XINT_apply_end\xint_bye\expandafter\XINT_apply_loop_b
510   \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.

```

511 \def\xintApplyUnbraced {\romannumeral0\xintapplyunbraced }%
512 \def\xintApplyUnbracedNoExpand {\romannumeral0\xintapplyunbracednoexpand }%
513 \long\def\xintapplyunbraced #1#2%
514 {%
515   \expandafter\XINT_applyunbr\expandafter {\romannumeral`&&@#2}%
516   {#1}%
517 }%
518 \long\def\XINT_applyunbr #1#2{\XINT_applyunbr_loop_a {}{#2}#1\xint_bye }%
519 \long\def\xintapplyunbracednoexpand #1#2%
520   {\XINT_applyunbr_loop_a {}{#1}#2\xint_bye }%
521 \long\def\XINT_applyunbr_loop_a #1#2#3%
522 {%

```

```

523 \xint_bye #3\XINT_applyunbr_end\xint_bye
524 \expandafter\XINT_applyunbr_loop_b
525 \expandafter {\romannumeral`&&@#2{#3}}{#1}{#2}%
526 }%
527 \long\def\XINT_applyunbr_loop_b #1#2{\XINT_applyunbr_loop_a {#2#1}}%
528 \long\def\XINT_applyunbr_end\xint_bye\expandafter\XINT_applyunbr_loop_b
529 \expandafter #1#2#3{ #2}%

```

2.18 \xintSeq

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

```

530 \def\xintSeq {\romannumeral0\xintseq }%
531 \def\xintseq #1{\XINT_seq_chkopt #1\xint_bye }%
532 \def\XINT_seq_chkopt #1%
533 {%
534 \ifx [#1\expandafter\XINT_seq_opt
535 \else\expandafter\XINT_seq_noopt
536 \fi #1%
537 }%
538 \def\XINT_seq_noopt #1\xint_bye #2%
539 {%
540 \expandafter\XINT_seq\expandafter
541 {\the\numexpr#1\expandafter}\expandafter{\the\numexpr #2}%
542 }%
543 \def\XINT_seq #1#2%
544 {%
545 \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
546 \expandafter\xint_firstoftwo_thenstop
547 \or
548 \expandafter\XINT_seq_p
549 \else
550 \expandafter\XINT_seq_n
551 \fi
552 {#2}{#1}%
553 }%
554 \def\XINT_seq_p #1#2%
555 {%
556 \ifnum #1>#2
557 \expandafter\expandafter\expandafter\XINT_seq_p
558 \else
559 \expandafter\XINT_seq_e
560 \fi
561 \expandafter{\the\numexpr #1-\xint_c_i}{#2}{#1}%
562 }%
563 \def\XINT_seq_n #1#2%
564 {%
565 \ifnum #1<#2
566 \expandafter\expandafter\expandafter\XINT_seq_n
567 \else
568 \expandafter\XINT_seq_e
569 \fi

```

2 Package *xinttools* implementation

```

570     \expandafter{\the\numexpr #1+\xint_c_i}{#2}{#1}%
571 }%
572 \def\XINT_seq_e #1#2#3{ }%
573 \def\XINT_seq_opt [\xint_bye #1]#2#3%
574 {%
575     \expandafter\XINT_seqo\expandafter
576     {\the\numexpr #2\expandafter}\expandafter
577     {\the\numexpr #3\expandafter}\expandafter
578     {\the\numexpr #1}%
579 }%
580 \def\XINT_seqo #1#2%
581 {%
582     \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
583     \expandafter\XINT_seqo_a
584     \or
585     \expandafter\XINT_seqo_pa
586     \else
587     \expandafter\XINT_seqo_na
588     \fi
589     {#1}{#2}%
590 }%
591 \def\XINT_seqo_a #1#2#3{ {#1}}%
592 \def\XINT_seqo_o #1#2#3#4{ #4}%
593 \def\XINT_seqo_pa #1#2#3%
594 {%
595     \ifcase\ifnum #3=\xint_c_ 0\else\ifnum #3>\xint_c_ 1\else -1\fi\fi\space
596     \expandafter\XINT_seqo_o
597     \or
598     \expandafter\XINT_seqo_pb
599     \else
600     \xint_afterfi{\expandafter\space\xint_gobble_iv}%
601     \fi
602     {#1}{#2}{#3}{#1}%
603 }%
604 \def\XINT_seqo_pb #1#2#3%
605 {%
606     \expandafter\XINT_seqo_pc\expandafter{\the\numexpr #1+#3}{#2}{#3}%
607 }%
608 \def\XINT_seqo_pc #1#2%
609 {%
610     \ifnum #1>#2
611         \expandafter\XINT_seqo_o
612     \else
613         \expandafter\XINT_seqo_pd
614     \fi
615     {#1}{#2}%
616 }%
617 \def\XINT_seqo_pd #1#2#3#4{\XINT_seqo_pb {#1}{#2}{#3}{#4{#1}}}%
618 \def\XINT_seqo_na #1#2#3%
619 {%
620     \ifcase\ifnum #3=\xint_c_ 0\else\ifnum #3>\xint_c_ 1\else -1\fi\fi\space
621     \expandafter\XINT_seqo_o

```

```

622 \or
623 \xint_afterfi{\expandafter\space\xint_gobble_iv}%
624 \else
625 \expandafter\XINT_sequo_nb
626 \fi
627 {#1}{#2}{#3}{#1}}%
628 }%
629 \def\XINT_sequo_nb #1#2#3%
630 {%
631 \expandafter\XINT_sequo_nc\expandafter{\the\numexpr #1+#3}{#2}{#3}%
632 }%
633 \def\XINT_sequo_nc #1#2%
634 {%
635 \ifnum #1<#2
636 \expandafter\XINT_sequo_o
637 \else
638 \expandafter\XINT_sequo_nd
639 \fi
640 {#1}{#2}%
641 }%
642 \def\XINT_sequo_nd #1#2#3#4{\XINT_sequo_nb {#1}{#2}{#3}{#4{#1}}}%

```

2.19 \xintloop, \xintbreakloop, \xintbreakloopaddo, \xintloopskiptonext

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

```

643 \long\def\xintloop #1#2\repeat {#1#2\xintloop_again\fi\xint_gobble_i {#1#2}}%
644 \long\def\xintloop_again\fi\xint_gobble_i #1{\fi
645 #1\xintloop_again\fi\xint_gobble_i {#1}}%
646 \long\def\xintbreakloop #1\xintloop_again\fi\xint_gobble_i #2{}%
647 \long\def\xintbreakloopaddo #1#2\xintloop_again\fi\xint_gobble_i #3{#1}%
648 \long\def\xintloopskiptonext #1\xintloop_again\fi\xint_gobble_i #2{%
649 #2\xintloop_again\fi\xint_gobble_i {#2}}%

```

2.20 \xintiloop, \xintiloopindex, \xintouteriloopindex, \xintbreakiloop, \xintbreakiloopaddo, \xintiloopskiptonext, \xintiloopskipandredo

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

```

650 \def\xintiloop [#1+#2]{%
651 \expandafter\xintiloop_a\the\numexpr #1\expandafter.\the\numexpr #2.%}
652 \long\def\xintiloop_a #1.#2.#3#4\repeat{%
653 #3#4\xintiloop_again\fi\xint_gobble_iii {#1}{#2}{#3#4}}%
654 \def\xintiloop_again\fi\xint_gobble_iii #1#2{%
655 \fi\expandafter\xintiloop_again_b\the\numexpr#1+#2.#2.%}
656 \long\def\xintiloop_again_b #1.#2.#3{%
657 #3\xintiloop_again\fi\xint_gobble_iii {#1}{#2}{#3}}%
658 \long\def\xintbreakiloop #1\xintiloop_again\fi\xint_gobble_iii #2#3#4{}%
659 \long\def\xintbreakiloopaddo
660 #1.#2\xintiloop_again\fi\xint_gobble_iii #3#4#5{#1}%
661 \long\def\xintiloopindex #1\xintiloop_again\fi\xint_gobble_iii #2%
662 {#2#1\xintiloop_again\fi\xint_gobble_iii {#2}}%
663 \long\def\xintouteriloopindex #1\xintiloop_again

```



```

664             #2\xintiloop_again\fi\xint_gobble_iii #3%
665     {#3#1\xintiloop_again #2\xintiloop_again\fi\xint_gobble_iii {#3}}%
666 \long\def\xintiloopskiptonext #1\xintiloop_again\fi\xint_gobble_iii #2#3{%
667     \expandafter\xintiloop_again_b \the\numexpr#2+#3.#3.%}
668 \long\def\xintiloopskipandredo #1\xintiloop_again\fi\xint_gobble_iii #2#3#4{%
669     #4\xintiloop_again\fi\xint_gobble_iii {#2}{#3}{#4}}%

```

2.21 \XINT_xflet

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

```

670 \def\XINT_xflet #1%
671 {%
672     \def\XINT_xflet_macro {#1}\XINT_xflet_zapsp
673 }%
674 \def\XINT_xflet_zapsp
675 {%
676     \expandafter\futurelet\expandafter\XINT_token
677     \expandafter\XINT_xflet_sp?\romannumeral`&&@%
678 }%
679 \def\XINT_xflet_sp?
680 {%
681     \ifx\XINT_token\XINT_sptoken
682         \expandafter\XINT_xflet_zapsp
683     \else\expandafter\XINT_xflet_zapspB
684     \fi
685 }%
686 \def\XINT_xflet_zapspB
687 {%
688     \expandafter\futurelet\expandafter\XINT_tokenB
689     \expandafter\XINT_xflet_spB?\romannumeral`&&@%
690 }%
691 \def\XINT_xflet_spB?
692 {%
693     \ifx\XINT_tokenB\XINT_sptoken
694         \expandafter\XINT_xflet_zapspB
695     \else\expandafter\XINT_xflet_eq?
696     \fi
697 }%
698 \def\XINT_xflet_eq?
699 {%
700     \ifx\XINT_token\XINT_tokenB
701         \expandafter\XINT_xflet_macro
702     \else\expandafter\XINT_xflet_zapsp
703     \fi
704 }%

```

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.

```

705 \catcode`Z 3
706 \long\def\xintApplyInline #1#2%
707 {%
708   \long\expandafter\def\expandafter\XINT_inline_macro
709   \expandafter ##\expandafter 1\expandafter {#1{##1}}%
710   \XINT_xflet\XINT_inline_b #2Z% this Z has catcode 3
711 }%
712 \def\XINT_inline_b
713 {%
714   \ifx\XINT_token Z\expandafter\xint_gobble_i
715   \else\expandafter\XINT_inline_d\fi
716 }%
717 \long\def\XINT_inline_d #1%
718 {%
719   \long\def\XINT_item{#1}\XINT_xflet\XINT_inline_e
720 }%
721 \def\XINT_inline_e
722 {%
723   \ifx\XINT_token Z\expandafter\XINT_inline_w
724   \else\expandafter\XINT_inline_f\fi
725 }%
726 \def\XINT_inline_f
727 {%
728   \expandafter\XINT_inline_g\expandafter{\XINT_inline_macro {##1}}%
729 }%
730 \long\def\XINT_inline_g #1%
731 {%
732   \expandafter\XINT_inline_macro\XINT_item
733   \long\def\XINT_inline_macro ##1{#1}\XINT_inline_d
734 }%
735 \def\XINT_inline_w #1%
736 {%
737   \expandafter\XINT_inline_macro\XINT_item
738 }%

```

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 to-

2 Package *xinttools* implementation

ken 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.

```

739 \def\XINT_tmpa #1#2{\ifnum #2<#1 \xint_afterfi {{{#####2}}}\fi}%
740 \def\XINT_tmpb #1#2{\ifnum #1<#2 \xint_afterfi {{{#####2}}}\fi}%
741 \def\XINT_tmpc #1%
742 {%
743   \expandafter\edef \csname XINT_for_left#1\endcsname
744     {\xintApplyUnbraced {\XINT_tmpa #1}{123456789}}%
745   \expandafter\edef \csname XINT_for_right#1\endcsname
746     {\xintApplyUnbraced {\XINT_tmpb #1}{123456789}}%
747 }%
748 \xintApplyInline \XINT_tmpc {123456789}%
749 \long\def\xintBreakFor      #1Z{%
750 \long\def\xintBreakForAndDo #1#2Z{#1}%
751 \def\xintFor {\let\xintifForFirst\xint_firstoftwo
752             \let\xintifForLast\xint_secondoftwo
753             \futurelet\XINT_token\XINT_for_ifstar }%
754 \def\XINT_for_ifstar {\ifx\XINT_token*\expandafter\XINT_forx
755                     \else\expandafter\XINT_for \fi }%
756 \catcode\U 3 % with numexpr
757 \catcode\V 3 % with xintfrac.sty (xint.sty not enough)
758 \catcode\D 3 % with dimexpr
759 \def\XINT_flet_zapsp
760 {%
761   \futurelet\XINT_token\XINT_flet_sp?
762 }%
763 \def\XINT_flet_sp?
764 {%
765   \ifx\XINT_token\XINT_sptoken
766     \xint_afterfi{\expandafter\XINT_flet_zapsp\romannumeral0}%
767   \else\expandafter\XINT_flet_macro
768   \fi
769 }%
770 \long\def\XINT_for #1#2in#3#4#5%
771 {%
772   \expandafter\XINT_toks\expandafter
773     {\expandafter\XINT_for_d\the\numexpr #2\relax {#5}}%
774   \def\XINT_flet_macro {\expandafter\XINT_for_forever?\space}%
775   \expandafter\XINT_flet_zapsp #3Z%
776 }%
777 \def\XINT_for_forever? #1Z%
778 {%
779   \ifx\XINT_token U\XINT_to_forever\fi
780   \ifx\XINT_token V\XINT_to_forever\fi
781   \ifx\XINT_token D\XINT_to_forever\fi
782   \expandafter\the\expandafter\XINT_toks\romannumeral0\xintcsvtolist {#1}Z%
783 }%
784 \def\XINT_to_forever\fi #1\xintcsvtolist #2{\fi \XINT_forever #2}%
785 \long\def\XINT_forx *#1#2in#3#4#5%

```

2 Package *xinttools* implementation

```

786 {%
787   \expandafter\XINT_toks\expandafter
788     {\expandafter\XINT_forx_d\the\numexpr #2\relax {#5}}%
789   \XINT_xflet\XINT_forx_forever? #3Z%
790 }%
791 \def\XINT_forx_forever?
792 {%
793   \ifx\XINT_token U\XINT_to_forxever\fi
794   \ifx\XINT_token V\XINT_to_forxever\fi
795   \ifx\XINT_token D\XINT_to_forxever\fi
796   \XINT_forx_empty?
797 }%
798 \def\XINT_to_forxever\fi #1\XINT_forx_empty? {\fi \XINT_forever }%
799 \catcode`U 11
800 \catcode`D 11
801 \catcode`V 11
802 \def\XINT_forx_empty?
803 {%
804   \ifx\XINT_token Z\expandafter\xintBreakFor\fi
805   \the\XINT_toks
806 }%
807 \long\def\XINT_for_d #1#2#3%
808 {%
809   \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
810   \XINT_toks {{#3}}%
811   \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
812     \the\XINT_toks \csname XINT_for_right#1\endcsname }%
813   \XINT_toks {\XINT_x\let\xintifForFirst\xint_secondoftwo
814     \let\xintifForLast\xint_secondoftwo\XINT_for_d #1{#2}}%
815   \futurelet\XINT_token\XINT_for_last?
816 }%
817 \long\def\XINT_forx_d #1#2#3%
818 {%
819   \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
820   \XINT_toks {{#3}}%
821   \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
822     \the\XINT_toks \csname XINT_for_right#1\endcsname }%
823   \XINT_toks {\XINT_x\let\xintifForFirst\xint_secondoftwo
824     \let\xintifForLast\xint_secondoftwo\XINT_forx_d #1{#2}}%
825   \XINT_xflet\XINT_for_last?
826 }%
827 \def\XINT_for_last?
828 {%
829   \ifx\XINT_token Z\expandafter\XINT_for_last?yes\fi
830   \the\XINT_toks
831 }%
832 \def\XINT_for_last?yes
833 {%
834   \let\xintifForLast\xint_firstoftwo
835   \xintBreakForAndDo{\XINT_x\xint_gobble_i Z}%
836 }%

```

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`).

```

837 \catcode`U 3
838 \catcode`D 3
839 \catcode`V 3
840 \let\xintegers      U%
841 \let\xintintegers   U%
842 \let\xintdimensions D%
843 \let\xintrationals  V%
844 \def\XINT_forever #1%
845 {%
846   \expandafter\XINT_forever_a
847   \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi a\expandafter\endcsname
848   \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi i\expandafter\endcsname
849   \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi \endcsname
850 }%
851 \catcode`U 11
852 \catcode`D 11
853 \catcode`V 11
854 \def\XINT_?expr_Ua #1#2%
855   {\expandafter{\expandafter\numexpr\the\numexpr #1\expandafter\relax
856     \expandafter\relax\expandafter}%
857   \expandafter{\the\numexpr #2}}%
858 \def\XINT_?expr_Da #1#2%
859   {\expandafter{\expandafter\dimexpr\the\dimexpr #1\expandafter\relax
860     \expandafter s\expandafter p\expandafter\relax\expandafter}%
861   \expandafter{\the\dimexpr #2}}%
862 \catcode`Z 11
863 \def\XINT_?expr_Va #1#2%
864 {%
865   \expandafter\XINT_?expr_Vb\expandafter
866   {\romannumeral`&&\xintrowwithzeros{\xintZapSpacesB{#2}}}%
867   {\romannumeral`&&\xintrowwithzeros{\xintZapSpacesB{#1}}}%
868 }%
869 \catcode`Z 3
870 \def\XINT_?expr_Vb #1#2{\expandafter\XINT_?expr_Vc #2.#1}%
871 \def\XINT_?expr_Vc #1/#2.#3/#4.%
872 {%
873   \xintifEq {#2}{#4}%
874     {\XINT_?expr_Vf {#3}{#1}{#2}}%
875     {\expandafter\XINT_?expr_Vd\expandafter
876       {\romannumeral0\xintiimul {#2}{#4}}%
877       {\romannumeral0\xintiimul {#1}{#4}}%
878       {\romannumeral0\xintiimul {#2}{#3}}%
879     }%
880 }%

```

```

881 \def\XINT_?expr_Vd #1#2#3{\expandafter\XINT_?expr_Ve\expandafter {#2}{#3}{#1}}%
882 \def\XINT_?expr_Ve #1#2{\expandafter\XINT_?expr_Vf\expandafter {#2}{#1}}%
883 \def\XINT_?expr_Vf #1#2#3{{#2/#3}{0}{#1}{#2}{#3}}%
884 \def\XINT_?expr_Ui {\numexpr 1\relax}{1}}%
885 \def\XINT_?expr_Di {\dimexpr 0pt\relax}{65536}}%
886 \def\XINT_?expr_Vi {{1/1}{0111}}%
887 \def\XINT_?expr_U #1#2%
888   {\expandafter{\expandafter\numexpr\the\numexpr #1+#2\relax\relax}{#2}}%
889 \def\XINT_?expr_D #1#2%
890   {\expandafter{\expandafter\dimexpr\the\numexpr #1+#2\relax sp\relax}{#2}}%
891 \def\XINT_?expr_V #1#2{\XINT_?expr_Vx #2}%
892 \def\XINT_?expr_Vx #1#2%
893 {%
894   \expandafter\XINT_?expr_Vy\expandafter
895     {\romannumeral0\xinttiadd {#1}{#2}}{#2}%
896 }%
897 \def\XINT_?expr_Vy #1#2#3#4%
898 {%
899   \expandafter{\romannumeral0\xinttiadd {#3}{#1}/#4}{{#1}{#2}{#3}{#4}}%
900 }%
901 \def\XINT_forever_a #1#2#3#4%
902 {%
903   \ifx #4[\expandafter\XINT_forever_opt_a
904     \else\expandafter\XINT_forever_b
905   \fi #1#2#3#4%
906 }%
907 \def\XINT_forever_b #1#2#3Z{\expandafter\XINT_forever_c\the\XINT_toks #2#3}%
908 \long\def\XINT_forever_c #1#2#3#4#5%
909   {\expandafter\XINT_forever_d\expandafter #2#4#5{#3}Z}%
910 \def\XINT_forever_opt_a #1#2#3[#4+#5]#6Z%
911 {%
912   \expandafter\expandafter\expandafter
913     \XINT_forever_opt_c\expandafter\the\expandafter\XINT_toks
914     \romannumeral`&&@#1{#4}{#5}#3%
915 }%
916 \long\def\XINT_forever_opt_c #1#2#3#4#5#6{\XINT_forever_d #2{#4}{#5}#6{#3}Z}%
917 \long\def\XINT_forever_d #1#2#3#4#5%
918 {%
919   \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#5}%
920   \XINT_toks {{#2}}%
921   \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
922     \the\XINT_toks \csname XINT_for_right#1\endcsname }%
923   \XINT_x
924   \let\xintifForFirst\xint_secondoftwo
925   \let\xintifForLast\xint_secondoftwo
926   \expandafter\XINT_forever_d\expandafter #1\romannumeral`&&@#4{#2}{#3}#4{#5}%
927 }%

```

2.25 *\xintForpair*, *\xintForthree*, *\xintForfour*

1.09c.

[2013/11/02] 1.09f *\xintForpair* delegate to *\xintCSVtoList* and its *\xintZapSpacesB* the han-

2 Package *xinttools* implementation

dling 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.

```
928 \catcode`j 3
929 \long\def\xintForpair #1#2#3in#4#5#6%
930 {%
931   \let\xintifForFirst\xint_firstoftwo
932   \let\xintifForLast\xint_secondoftwo
933   \XINT_toks {\XINT_forpair_d #2{#6}}%
934   \expandafter\the\expandafter\XINT_toks #4jZ%
935 }%
936 \long\def\XINT_forpair_d #1#2#3(#4)#5%
937 {%
938   \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
939   \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
940   \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
941     \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_i\endcsname}%
942   \ifx #5j\expandafter\XINT_for_last?yes\fi
943   \XINT_x
944   \let\xintifForFirst\xint_secondoftwo
945   \let\xintifForLast\xint_secondoftwo
946   \XINT_forpair_d #1{#2}%
947 }%
948 \long\def\xintForthree #1#2#3in#4#5#6%
949 {%
950   \let\xintifForFirst\xint_firstoftwo
951   \let\xintifForLast\xint_secondoftwo
952   \XINT_toks {\XINT_forthree_d #2{#6}}%
953   \expandafter\the\expandafter\XINT_toks #4jZ%
954 }%
955 \long\def\XINT_forthree_d #1#2#3(#4)#5%
956 {%
957   \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
958   \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
959   \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
960     \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_ii\endcsname}%
961   \ifx #5j\expandafter\XINT_for_last?yes\fi
962   \XINT_x
963   \let\xintifForFirst\xint_secondoftwo
964   \let\xintifForLast\xint_secondoftwo
965   \XINT_forthree_d #1{#2}%
966 }%
967 \long\def\xintForfour #1#2#3in#4#5#6%
968 {%
969   \let\xintifForFirst\xint_firstoftwo
970   \let\xintifForLast\xint_secondoftwo
971   \XINT_toks {\XINT_forfour_d #2{#6}}%
972   \expandafter\the\expandafter\XINT_toks #4jZ%
973 }%
974 \long\def\XINT_forfour_d #1#2#3(#4)#5%
975 {%
```

```

976 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
977 \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
978 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
979 \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_iii\endcsname}%
980 \ifx #5j\expandafter\XINT_for_last?yes\fi
981 \XINT_x
982 \let\xintifForFirst\xint_secondoftwo
983 \let\xintifForLast\xint_secondoftwo
984 \XINT_forfour_d #1{#2}%
985 }%
986 \catcode`Z 11
987 \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.

```

988 \def\xintAssign{\def\XINT_flet_macro {\XINT_assign_fork}\XINT_flet_zapsp }%
989 \def\XINT_assign_fork
990 {%
991 \let\XINT_assign_def\def
992 \ifx\XINT_token[\expandafter\XINT_assign_opt
993 \else\expandafter\XINT_assign_a
994 \fi
995 }%
996 \def\XINT_assign_opt [#1]%
997 {%
998 \ifcsname #1def\endcsname
999 \expandafter\let\expandafter\XINT_assign_def \csname #1def\endcsname
1000 \else
1001 \expandafter\let\expandafter\XINT_assign_def \csname xint#1def\endcsname
1002 \fi
1003 \XINT_assign_a
1004 }%
1005 \long\def\XINT_assign_a #1\to
1006 {%
1007 \def\XINT_flet_macro{\XINT_assign_b}%
1008 \expandafter\XINT_flet_zapsp\romannumeral`&&@#1\xint:\to
1009 }%
1010 \long\def\XINT_assign_b
1011 {%
1012 \ifx\XINT_token\bgroup
1013 \expandafter\XINT_assign_c
1014 \else\expandafter\XINT_assign_f
1015 \fi
1016 }%
1017 \long\def\XINT_assign_f #1\xint:\to #2%
1018 {%
1019 \XINT_assign_def #2{#1}%

```


2 Package *xinttools* implementation

```

1020 }%
1021 \long\def\XINT_assign_c #1%
1022 {%
1023   \def\xint_temp {#1}%
1024   \ifx\xint_temp\xint_bracedstopper
1025     \expandafter\XINT_assign_e
1026   \else
1027     \expandafter\XINT_assign_d
1028   \fi
1029 }%
1030 \long\def\XINT_assign_d #1\to #2%
1031 {%
1032   \expandafter\XINT_assign_def\expandafter #2\expandafter{\xint_temp}%
1033   \XINT_assign_c #1\to
1034 }%
1035 \def\XINT_assign_e #1\to {}%
1036 \def\xintRelaxArray #1%
1037 {%
1038   \edef\XINT_restoreescapechar {\escapechar\the\escapechar\relax}%
1039   \escapechar -1
1040   \expandafter\def\expandafter\xint_arrayname\expandafter {\string #1}%
1041   \XINT_restoreescapechar
1042   \xintilop [\csname\xint_arrayname 0\endcsname+-1]
1043   \global
1044   \expandafter\let\csname\xint_arrayname\xintilopindex\endcsname\relax
1045   \ifnum \xintilopindex > \xint_c_
1046   \repeat
1047   \global\expandafter\let\csname\xint_arrayname 00\endcsname\relax
1048   \global\let #1\relax
1049 }%
1050 \def\xintAssignArray{\def\XINT_flet_macro {\XINT_assignarray_fork}%
1051   \XINT_flet_zapspace }%
1052 \def\XINT_assignarray_fork
1053 {%
1054   \let\XINT_assignarray_def\def
1055   \ifx\XINT_token[\expandafter\XINT_assignarray_opt
1056     \else\expandafter\XINT_assignarray
1057   \fi
1058 }%
1059 \def\XINT_assignarray_opt [#1]%
1060 {%
1061   \ifcsname #1def\endcsname
1062     \expandafter\let\expandafter\XINT_assignarray_def \csname #1def\endcsname
1063   \else
1064     \expandafter\let\expandafter\XINT_assignarray_def
1065       \csname xint#1def\endcsname
1066   \fi
1067   \XINT_assignarray
1068 }%
1069 \long\def\XINT_assignarray #1\to #2%
1070 {%
1071   \edef\XINT_restoreescapechar {\escapechar\the\escapechar\relax }%

```

2 Package *xinttools* implementation

```

1072 \escapechar -1
1073 \expandafter\def\expandafter\xint_arrayname\expandafter {\string #2}%
1074 \XINT_restoreescapechar
1075 \def\xint_itemcount {0}%
1076 \expandafter\XINT_assignarray_loop \romannumeral`&&@#1\xint:
1077 \csname\xint_arrayname 00\expandafter\endcsname
1078 \csname\xint_arrayname 0\expandafter\endcsname
1079 \expandafter {\xint_arrayname}#2%
1080 }%
1081 \long\def\XINT_assignarray_loop #1%
1082 {%
1083 \def\xint_temp {#1}%
1084 \ifx\xint_temp\xint_bracedstopper
1085 \expandafter\def\csname\xint_arrayname 0\expandafter\endcsname
1086 \expandafter{\the\numexpr\xint_itemcount}%
1087 \expandafter\expandafter\expandafter\XINT_assignarray_end
1088 \else
1089 \expandafter\def\expandafter\xint_itemcount\expandafter
1090 {\the\numexpr\xint_itemcount+\xint_c_i}%
1091 \expandafter\XINT_assignarray_def
1092 \csname\xint_arrayname\xint_itemcount\expandafter\endcsname
1093 \expandafter{\xint_temp }%
1094 \expandafter\XINT_assignarray_loop
1095 \fi
1096 }%
1097 \def\XINT_assignarray_end #1#2#3#4%
1098 {%
1099 \def #4##1%
1100 {%
1101 \romannumeral0\expandafter #1\expandafter{\the\numexpr ##1}%
1102 }%
1103 \def #1##1%
1104 {%
1105 \ifnum ##1<\xint_c_
1106 \xint_afterfi{\XINT_expandableerror{Array index negative: 0 > ##1} }%
1107 \else
1108 \xint_afterfi {%
1109 \ifnum ##1>#2
1110 \xint_afterfi
1111 {\XINT_expandableerror{Array index beyond range: ##1 > #2} }%
1112 \else\xint_afterfi
1113 {\expandafter\expandafter\expandafter\space\csname #3##1\endcsname}%
1114 \fi}%
1115 \fi
1116 }%
1117 }%
1118 \let\xintDigitsOf\xintAssignArray

```

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

These routines are for use by `\xintListSel:x:csv` and `\xintListSel:f:csv` from `xintexpr`, and also for the `reversed` and `len` functions. Refactored for 1.2j release, following 1.2i updates to `\xintKeep`, `\xintTrim`, ...

These macros will remain undocumented in the user manual:

-- they exist primarily for internal use by the `xintexpr` parsers, hence don't have to be general purpose; for example, they a priori need to handle only catcode 12 tokens (not true in `\xintNewExpr`, though) hence they are not really worried about controlling brace stripping (nevertheless 1.2j has paid some secondary attention to it, see below.) They are not worried about normalizing leading spaces either, because none will be encountered when the macros are used as auxiliaries to the expression parsers.

-- crucial design elements may change in future:

1. whether the handled lists must have or not have a final comma. Currently, the model is the one of comma separated lists with `**no**` final comma. But this means that there can not be a distinction of principle between a truly empty list and a list which contains one item which turns out to be empty. More importantly it makes the coding more complicated as it is needed to distinguish the empty list from the single-item list, both lacking commas.

For the internal use of `xintexpr`, it would be ok to require all list items to be terminated by a comma, and this would bring quite some simplifications here, but as initially I started with non-terminated lists, I have left it this way in the 1.2j refactoring.

2. the way to represent the empty list. I was tempted for matter of optimization and synchronization with `xintexpr` context to require the empty list to be always represented by a space token and to not let the macros admit a completely empty input. But there were complications so for the time being 1.2j does accept truly empty output (it is not distinguished from an input equal to a space token) and produces empty output for empty list. This means that the status of the «nil» object for the `xintexpr` parsers is not completely clarified (currently it is represented by a space token).

The original Python slicing code in `xintexpr 1.1` used `\xintCSVtoList` and `\xintListWithSep{,}` to convert back and forth to token lists and apply `\xintKeep/\xintTrim`. Release 1.2g switched to devoted f-expandable macros added to `xinttools`. Release 1.2j refactored all these macros as a follow-up to 1.2i improvements to `\xintKeep/\xintTrim`. They were made `\long` on this occasion and auxiliary `\xintLengthUpTo:f:csv` was added.

Leading spaces in items are currently maintained as is by the 1.2j macros, even by `\xintNthEltPy:f:csv`, with the exception of the first item, as the list is f-expanded. Perhaps `\xintNthEltPy:f:csv` should remove a leading space if present in the picked item; anyway, there are no spaces for the lists handled internally by the Python slicer of `xintexpr`, except the «nil» object currently represented by exactly one space.

Kept items (with no leading spaces; but first item special as it will have lost a leading space due to f-expansion) will lose a brace pair under `\xintKeep:f:csv` if the first argument was positive and strictly less than the length of the list. This differs of course from `\xintKeep` (which always braces items it outputs when used with positive first argument) and also from `\xintKeepUnbraced` in the case when the whole list is kept. Actually the case of singleton list is special, and brace removal will happen then.

This behaviour was otherwise for releases earlier than 1.2j and may change again.

Directly usable names are provided, but these macros (and the behaviour as described above) are to be considered *unstable* for the time being.

2.27.1 `\xintLength:f:csv`

1.2g. Redone for 1.2j. Contrarily to `\xintLength` from `xintkernel.sty`, this one expands its argument.

```

1119 \def\xintLength:f:csv {\romannumeral0\xintlength:f:csv}%
1120 \def\xintlength:f:csv #1%
1121 {\long\def\xintlength:f:csv ##1{%
1122   \expandafter#1\the\numexpr\expandafter\XINT_length:f:csv_a
1123   \romannumeral`&&##1\xint:,\xint:,\xint:,\xint:,%
1124   \xint:,\xint:,\xint:,\xint:,\xint:,%
1125   \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
1126   \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
1127   \relax
1128 }}\xintlength:f:csv { }%

```

Must first check if empty list.

```

1129 \long\def\XINT_length:f:csv_a #1%
1130 {%
1131   \xint_gob_til_xint: #1\xint_c_\xint_bye\xint:%
1132   \XINT_length:f:csv_loop #1%
1133 }%
1134 \long\def\XINT_length:f:csv_loop #1,#2,#3,#4,#5,#6,#7,#8,#9,%
1135 {%
1136   \xint_gob_til_xint: #9\XINT_length:f:csv_finish\xint:%
1137   \xint_c_ix+\XINT_length:f:csv_loop
1138 }%
1139 \def\XINT_length:f:csv_finish\xint:\xint_c_ix+\XINT_length:f:csv_loop
1140   #1,#2,#3,#4,#5,#6,#7,#8,#9,{#9\xint_bye}%

```

2.27.2 \xintLengthUpTo:f:csv

1.2j. \xintLengthUpTo:f:csv{N}{comma-list}. No ending comma. Returns -0 if length>N, else returns difference N-length. **N must be non-negative!!**

Attention to the dot after \xint_bye for the loop interface.

```

1141 \def\xintLengthUpTo:f:csv {\romannumeral0\xintlengthupto:f:csv}%
1142 \long\def\xintlengthupto:f:csv #1#2%
1143 {%
1144   \expandafter\XINT_lengthupto:f:csv_a
1145   \the\numexpr#1\expandafter.%
1146   \romannumeral`&&#2\xint:,\xint:,\xint:,\xint:,%
1147   \xint:,\xint:,\xint:,\xint:,%
1148   \xint_c_viii,\xint_c_vii,\xint_c_vi,\xint_c_v,%
1149   \xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye.%
1150 }%

```

Must first recognize if empty list. If this is the case, return N.

```

1151 \long\def\XINT_lengthupto:f:csv_a #1.#2%
1152 {%
1153   \xint_gob_til_xint: #2\XINT_lengthupto:f:csv_empty\xint:%
1154   \XINT_lengthupto:f:csv_loop_b #1.#2%
1155 }%
1156 \def\XINT_lengthupto:f:csv_empty\xint:%
1157   \XINT_lengthupto:f:csv_loop_b #1.#2\xint_bye.{ #1}%
1158 \def\XINT_lengthupto:f:csv_loop_a #1%
1159 {%

```

```

1160 \xint_UDsignfork
1161 #1\XINT_lengthupto:f:csv_gt
1162 -\XINT_lengthupto:f:csv_loop_b
1163 \krof #1%
1164 }%
1165 \long\def\XINT_lengthupto:f:csv_gt #1\xint_bye.{-0}%
1166 \long\def\XINT_lengthupto:f:csv_loop_b #1.#2,#3,#4,#5,#6,#7,#8,#9,%
1167 {%
1168 \xint_gob_til_xint: #9\XINT_lengthupto:f:csv_finish_a\xint:%
1169 \expandafter\XINT_lengthupto:f:csv_loop_a\the\numexpr #1-\xint_c_viii.%
1170 }%
1171 \def\XINT_lengthupto:f:csv_finish_a\xint:
1172 \expandafter\XINT_lengthupto:f:csv_loop_a
1173 \the\numexpr #1-\xint_c_viii.#2,#3,#4,#5,#6,#7,#8,#9,%
1174 {%
1175 \expandafter\XINT_lengthupto:f:csv_finish_b\the\numexpr #1-#9\xint_bye
1176 }%
1177 \def\XINT_lengthupto:f:csv_finish_b #1#2.%
1178 {%
1179 \xint_UDsignfork
1180 #1{-0}%
1181 -{ #1#2}%
1182 \krof
1183 }%

```

2.27.3 \xintKeep:f:csv

1.2g 2016/03/17. Redone for 1.2j with use of \xintLengthUpTo:f:csv. Same code skeleton as \xintKeep but handling comma separated but non terminated lists has complications. The \xintKeep in case of a negative #1 uses \xintgobble, we don't have that for comma delimited items, hence we do a special loop here (this style of loop is surely competitive with xintgobble for a few dozens items and even more). The loop knows before starting that it will not go too far.

```

1184 \def\xintKeep:f:csv {\romannumeral0\xintkeep:f:csv }%
1185 \long\def\xintkeep:f:csv #1#2%
1186 {%
1187 \expandafter\xint_gobble_thenstop
1188 \romannumeral0\expandafter\XINT_keep:f:csv_a
1189 \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
1190 }%
1191 \def\XINT_keep:f:csv_a #1%
1192 {%
1193 \xint_UDzerominusfork
1194 #1-\XINT_keep:f:csv_keepnone
1195 0#1\XINT_keep:f:csv_neg
1196 0-{\XINT_keep:f:csv_pos #1}%
1197 \krof
1198 }%
1199 \long\def\XINT_keep:f:csv_keepnone .#1{,}%
1200 \long\def\XINT_keep:f:csv_neg #1.#2%
1201 {%
1202 \expandafter\XINT_keep:f:csv_neg_done\expandafter,%
1203 \romannumeral0%

```

2 Package *xinttools* implementation

```

1204 \expandafter\XINT_keep:f:csv_neg_a\the\numexpr
1205 #1-\numexpr\XINT_length:f:csv_a
1206 #2\xint:,\xint:,\xint:,\xint:,%
1207 \xint:,\xint:,\xint:,\xint:,\xint:,%
1208 \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
1209 \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
1210 .#2\xint_bye
1211 }%
1212 \def\XINT_keep:f:csv_neg_a #1%
1213 {%
1214 \xint_UDsignfork
1215 #1{\expandafter\XINT_keep:f:csv_trimloop\the\numexpr-\xint_c_ix+}%
1216 -\XINT_keep:f:csv_keepall
1217 \krof
1218 }%
1219 \def\XINT_keep:f:csv_keepall #1.{ }%
1220 \long\def\XINT_keep:f:csv_neg_done #1\xint_bye{#1}%
1221 \def\XINT_keep:f:csv_trimloop #1#2.%
1222 {%
1223 \xint_gob_til_minus#1\XINT_keep:f:csv_trimloop_finish-%
1224 \expandafter\XINT_keep:f:csv_trimloop
1225 \the\numexpr#1#2-\xint_c_ix\expandafter.\XINT_keep:f:csv_trimloop_trimnine
1226 }%
1227 \long\def\XINT_keep:f:csv_trimloop_trimnine #1,#2,#3,#4,#5,#6,#7,#8,#9,{}%
1228 \def\XINT_keep:f:csv_trimloop_finish-%
1229 \expandafter\XINT_keep:f:csv_trimloop
1230 \the\numexpr-#1-\xint_c_ix\expandafter.\XINT_keep:f:csv_trimloop_trimnine
1231 {\csname XINT_trim:f:csv_finish#1\endcsname}%
1232 \long\def\XINT_keep:f:csv_pos #1.#2%
1233 {%
1234 \expandafter\XINT_keep:f:csv_pos_fork
1235 \romannumeral0\XINT_lengthupto:f:csv_a
1236 #1.#2\xint:,\xint:,\xint:,\xint:,%
1237 \xint:,\xint:,\xint:,\xint:,%
1238 \xint_c_viii,\xint_c_vii,\xint_c_vi,\xint_c_v,%
1239 \xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye.%
1240 .#1.{ }#2\xint_bye%
1241 }%
1242 \def\XINT_keep:f:csv_pos_fork #1#2.%
1243 {%
1244 \xint_UDsignfork
1245 #1{\expandafter\XINT_keep:f:csv_loop\the\numexpr-\xint_c_viii+}%
1246 -\XINT_keep:f:csv_pos_keepall
1247 \krof
1248 }%
1249 \long\def\XINT_keep:f:csv_pos_keepall #1.#2#3\xint_bye{,#3}%
1250 \def\XINT_keep:f:csv_loop #1#2.%
1251 {%
1252 \xint_gob_til_minus#1\XINT_keep:f:csv_loop_end-%
1253 \expandafter\XINT_keep:f:csv_loop
1254 \the\numexpr#1#2-\xint_c_viii\expandafter.\XINT_keep:f:csv_loop_pickeight
1255 }%

```

```

1256 \long\def\XINT_keep:f:csv_loop_pickeight
1257     #1#2,#3,#4,#5,#6,#7,#8,#9,{\#1,#2,#3,#4,#5,#6,#7,#8,#9}}%
1258 \def\XINT_keep:f:csv_loop_end-\expandafter\XINT_keep:f:csv_loop
1259     \the\numexpr-#1-\xint_c_viii\expandafter.\XINT_keep:f:csv_loop_pickeight
1260     {\csname XINT_keep:f:csv_end#1\endcsname}%
1261 \long\expandafter\def\csname XINT_keep:f:csv_end1\endcsname
1262     #1#2,#3,#4,#5,#6,#7,#8,#9\xint_bye {\#1,#2,#3,#4,#5,#6,#7,#8}%
1263 \long\expandafter\def\csname XINT_keep:f:csv_end2\endcsname
1264     #1#2,#3,#4,#5,#6,#7,#8\xint_bye {\#1,#2,#3,#4,#5,#6,#7}%
1265 \long\expandafter\def\csname XINT_keep:f:csv_end3\endcsname
1266     #1#2,#3,#4,#5,#6,#7\xint_bye {\#1,#2,#3,#4,#5,#6}%
1267 \long\expandafter\def\csname XINT_keep:f:csv_end4\endcsname
1268     #1#2,#3,#4,#5,#6\xint_bye {\#1,#2,#3,#4,#5}%
1269 \long\expandafter\def\csname XINT_keep:f:csv_end5\endcsname
1270     #1#2,#3,#4,#5\xint_bye {\#1,#2,#3,#4}%
1271 \long\expandafter\def\csname XINT_keep:f:csv_end6\endcsname
1272     #1#2,#3,#4\xint_bye {\#1,#2,#3}%
1273 \long\expandafter\def\csname XINT_keep:f:csv_end7\endcsname
1274     #1#2,#3\xint_bye {\#1,#2}%
1275 \long\expandafter\def\csname XINT_keep:f:csv_end8\endcsname
1276     #1#2\xint_bye {\#1}%

```

2.27.4 *\xintTrim:f:csv*

1.2g 2016/03/17. Redone for 1.2j 2016/12/20 on the basis of new *\xintTrim*.

```

1277 \def\xintTrim:f:csv {\romannumeral0\xinttrim:f:csv}%
1278 \long\def\xinttrim:f:csv #1#2%
1279 {%
1280     \expandafter\xint_gobble_thenstop
1281     \romannumeral0\expandafter\XINT_trim:f:csv_a
1282     \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
1283}%
1284 \def\XINT_trim:f:csv_a #1%
1285 {%
1286     \xint_UDzerominusfork
1287     #1-\XINT_trim:f:csv_trimnone
1288     0#1\XINT_trim:f:csv_neg
1289     0-{\XINT_trim:f:csv_pos #1}%
1290     \krof
1291}%
1292 \long\def\XINT_trim:f:csv_trimnone .#1{,#1}%
1293 \long\def\XINT_trim:f:csv_neg #1.#2%
1294 {%
1295     \expandafter\XINT_trim:f:csv_neg_a\the\numexpr
1296     #1-\numexpr\XINT_length:f:csv_a
1297     #2\xint:,\xint:,\xint:,\xint:,%
1298     \xint:,\xint:,\xint:,\xint:,\xint:,%
1299     \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
1300     \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
1301     .{#2\xint_bye
1302}%
1303 \def\XINT_trim:f:csv_neg_a #1%

```

```

1304 {%
1305     \xint_UDsignfork
1306     #1{\expandafter\XINT_keep:f:csv_loop\the\numexpr-\xint_c_viii+}%
1307     -\XINT_trim:f:csv_trimall
1308     \krof
1309 }%
1310 \def\XINT_trim:f:csv_trimall {\expandafter,\xint_bye}%
1311 \long\def\XINT_trim:f:csv_pos #1.#2%
1312 {%
1313     \expandafter\XINT_trim:f:csv_pos_done\expandafter,%
1314     \romannumeral0%
1315     \expandafter\XINT_trim:f:csv_loop\the\numexpr#1-\xint_c_ix.%
1316     #2\xint:,\xint:,\xint:,\xint:,\xint:,%
1317     \xint:,\xint:,\xint:,\xint:,\xint:\xint_bye
1318 }%
1319 \def\XINT_trim:f:csv_loop #1#2.%
1320 {%
1321     \xint_gob_til_minus#1\XINT_trim:f:csv_finish-%
1322     \expandafter\XINT_trim:f:csv_loop\the\numexpr#1#2\XINT_trim:f:csv_loop_trimnine
1323 }%
1324 \long\def\XINT_trim:f:csv_loop_trimnine #1,#2,#3,#4,#5,#6,#7,#8,#9,%
1325 {%
1326     \xint_gob_til_xint: #9\XINT_trim:f:csv_toofew\xint:-\xint_c_ix.%
1327 }%
1328 \def\XINT_trim:f:csv_toofew\xint:{*\xint_c_}%
1329 \def\XINT_trim:f:csv_finish-%
1330     \expandafter\XINT_trim:f:csv_loop\the\numexpr-#1\XINT_trim:f:csv_loop_trimnine
1331 {%
1332     \csname XINT_trim:f:csv_finish#1\endcsname
1333 }%
1334 \long\expandafter\def\csname XINT_trim:f:csv_finish1\endcsname
1335     #1,#2,#3,#4,#5,#6,#7,#8,{ }%
1336 \long\expandafter\def\csname XINT_trim:f:csv_finish2\endcsname
1337     #1,#2,#3,#4,#5,#6,#7,{ }%
1338 \long\expandafter\def\csname XINT_trim:f:csv_finish3\endcsname
1339     #1,#2,#3,#4,#5,#6,{ }%
1340 \long\expandafter\def\csname XINT_trim:f:csv_finish4\endcsname
1341     #1,#2,#3,#4,#5,{ }%
1342 \long\expandafter\def\csname XINT_trim:f:csv_finish5\endcsname
1343     #1,#2,#3,#4,{ }%
1344 \long\expandafter\def\csname XINT_trim:f:csv_finish6\endcsname
1345     #1,#2,#3,{ }%
1346 \long\expandafter\def\csname XINT_trim:f:csv_finish7\endcsname
1347     #1,#2,{ }%
1348 \long\expandafter\def\csname XINT_trim:f:csv_finish8\endcsname
1349     #1,{ }%
1350 \expandafter\let\csname XINT_trim:f:csv_finish9\endcsname\space
1351 \long\def\XINT_trim:f:csv_pos_done #1\xint:#2\xint_bye{#1}%

```

2.27.5 *\xintNthEltPy:f:csv*

Counts like Python starting at zero. Last refactored with 1.2j. Attention, makes currently no effort at removing leading spaces in the picked item.

2 Package *xinttools* implementation

```

1352 \def\xintNthEltPy:f:csv {\romannumeral0\xintntheltpy:f:csv}%
1353 \long\def\xintntheltpy:f:csv #1#2%
1354 {%
1355   \expandafter\XINT_nthelt:f:csv_a
1356   \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
1357}%
1358 \def\XINT_nthelt:f:csv_a #1%
1359 {%
1360   \xint_UDsignfork
1361     #1\XINT_nthelt:f:csv_neg
1362     -\XINT_nthelt:f:csv_pos
1363   \krof #1%
1364}%
1365 \long\def\XINT_nthelt:f:csv_neg -#1.#2%
1366 {%
1367   \expandafter\XINT_nthelt:f:csv_neg_fork
1368   \the\numexpr\XINT_length:f:csv_a
1369   #2\xint:,\xint:,\xint:,\xint:,%
1370   \xint:,\xint:,\xint:,\xint:,\xint:,%
1371   \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
1372   \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
1373   -#1.#2,\xint_bye
1374}%
1375 \def\XINT_nthelt:f:csv_neg_fork #1%
1376 {%
1377   \if#1-\expandafter\xint_bye_thenstop\fi
1378   \expandafter\XINT_nthelt:f:csv_neg_done
1379   \romannumeral0%
1380   \expandafter\XINT_keep:f:csv_trimloop\the\numexpr-\xint_c_ix+#1%
1381}%
1382 \long\def\XINT_nthelt:f:csv_neg_done#1,#2\xint_bye{ #1}%
1383 \long\def\XINT_nthelt:f:csv_pos #1.#2%
1384 {%
1385   \expandafter\XINT_nthelt:f:csv_pos_done
1386   \romannumeral0%
1387   \expandafter\XINT_trim:f:csv_loop\the\numexpr#1-\xint_c_ix.%
1388   #2\xint:,\xint:,\xint:,\xint:,\xint:,%
1389   \xint:,\xint:,\xint:,\xint:,\xint:,\xint_bye
1390}%
1391 \def\XINT_nthelt:f:csv_pos_done #1{%
1392 \long\def\XINT_nthelt:f:csv_pos_done ##1,##2\xint_bye{%
1393   \xint_gob_til_xint:##1\XINT_nthelt:f:csv_pos_cleanup\xint:#1##1}%
1394 }\XINT_nthelt:f:csv_pos_done{ }%

```

This strange thing is in case the picked item was the last one, hence there was an ending `\xint:` (we could not put a comma earlier for matters of not confusing empty list with a singleton list), and we do this here to activate brace-stripping of item as all other items may be brace-stripped if picked. This is done for coherence. Of course, in the context of the `xintexpr.sty` parsers, there are no braces in list items...

```

1395 \xint_firstofone{\long\def\XINT_nthelt:f:csv_pos_cleanup\xint:} %
1396   #1\xint:{ #1}%

```

2.27.6 \xintReverse:f:csv

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

```

1397 \def\xintReverse:f:csv {\romannumeral0\xintreverse:f:csv}%
1398 \long\def\xintreverse:f:csv #1%
1399 {%
1400   \expandafter\XINT_reverse:f:csv_loop
1401   \expandafter{\expandafter}\romannumeral`&&@#1,%
1402   \xint:,%
1403   \xint_bye,\xint_bye,\xint_bye,\xint_bye,%
1404   \xint_bye,\xint_bye,\xint_bye,\xint_bye,%
1405   \xint:
1406}%
1407 \long\def\XINT_reverse:f:csv_loop #1#2,#3,#4,#5,#6,#7,#8,#9,%
1408 {%
1409   \xint_bye #9\XINT_reverse:f:csv_cleanup\xint_bye
1410   \XINT_reverse:f:csv_loop {,#9,#8,#7,#6,#5,#4,#3,#2#1}%
1411}%
1412 \long\def\XINT_reverse:f:csv_cleanup\xint_bye\XINT_reverse:f:csv_loop #1#2\xint:
1413 {%
1414   \XINT_reverse:f:csv_finish #1%
1415}%
1416 \long\def\XINT_reverse:f:csv_finish #1\xint:,{ }%
```

2.27.7 \xintFirstItem:f:csv

Added with 1.2k for use by `first()` in `\xintexpr-essions`, and some amount of compatibility with `\xintNewExpr`.

```

1417 \def\xintFirstItem:f:csv {\romannumeral0\xintfirstitem:f:csv}%
1418 \long\def\xintfirstitem:f:csv #1%
1419 {%
1420   \expandafter\XINT_first:f:csv_a\romannumeral`&&@#1,\xint_bye
1421}%
1422 \long\def\XINT_first:f:csv_a #1,#2\xint_bye{ #1}%
```

2.27.8 \xintLastItem:f:csv

Added with 1.2k, based on and sharing code with `xintkernel`'s `\xintLastItem` from 1.2i. Output empty if input empty. `f`-expands its argument (hence first item, if not protected.) For use by `last()` in `\xintexpr-essions` with to some extent `\xintNewExpr` compatibility.

```

1423 \def\xintLastItem:f:csv {\romannumeral0\xintllastitem:f:csv}%
1424 \long\def\xintllastitem:f:csv #1%
1425 {%
1426   \expandafter\XINT_last:f:csv_loop\expandafter{\expandafter}\expandafter.%
1427   \romannumeral`&&@#1,%
1428   \xint:\XINT_last_loop_enda,\xint:\XINT_last_loop_endb,%
1429   \xint:\XINT_last_loop_endc,\xint:\XINT_last_loop_endd,%
1430   \xint:\XINT_last_loop_ende,\xint:\XINT_last_loop_endf,%
1431   \xint:\XINT_last_loop_endg,\xint:\XINT_last_loop_endh,\xint_bye
```

```

1432 }%
1433 \long\def\XINT_last:f:csv_loop #1.#2,#3,#4,#5,#6,#7,#8,#9,%
1434 {%
1435     \xint_gob_til_xint: #9%
1436     {#8}{#7}{#6}{#5}{#4}{#3}{#2}{#1}\xint:
1437     \XINT_last:f:csv_loop {#9}.%
1438 }%

```

2.27.9 Public names for the undocumented csv macros

Completely unstable macros: currently they expand the list argument and want no final comma. But for matters of *xintexpr.sty* I could as well decide to require a final comma, and then I could simplify implementation but of course this would break the macros if used with current functionalities.

```

1439 \let\xintCSVLength    \xintLength:f:csv
1440 \let\xintCSVKeep      \xintKeep:f:csv
1441 \let\xintCSVTrim      \xintTrim:f:csv
1442 \let\xintCSVNthEltPy  \xintNthEltPy:f:csv
1443 \let\xintCSVReverse   \xintReverse:f:csv
1444 \let\xintCSVFirstItem\xintFirstItem:f:csv
1445 \let\xintCSVLastItem  \xintLastItem:f:csv
1446 \let\XINT_tmpa\relax \let\XINT_tmpb\relax \let\XINT_tmpc\relax
1447 \XINT_restorecatcodes_endinput%

```

3 Package *xintcore* implementation

.1	Catcodes, ε -T _E X and reload detection	52	.26	<code>\XINT_sepbyviii_Z</code>	66
.2	Package identification	53	.27	<code>\XINT_sepbyviii_andcount</code>	66
.3	(WIP!) Error conditions and exceptions	53	.28	<code>\XINT_rsepbyviii</code>	66
.4	Counts for holding needed constants	55	.29	<code>\XINT_sepandrev</code>	67
Routines handling integers as lists of token digits 56			.30	<code>\XINT_sepandrev_andcount</code>	67
.5	<code>\XINT_cuz_small</code>	56	.31	<code>\XINT_rev_nounsep</code>	68
.6	<code>\xintNum</code>	56	.32	<code>\XINT_unrevbyviii</code>	68
.7	<code>\xintiiSgn</code>	57	Core arithmetic 69		
.8	<code>\xintiiOpp</code>	58	.33	<code>\xintiiAdd</code>	69
.9	<code>\xintiiAbs</code>	58	.34	<code>\xintiiCmp</code>	72
.10	<code>\xintFDg</code>	58	.35	<code>\xintiiSub</code>	74
.11	<code>\xintLDg</code>	59	.36	<code>\xintiiMul</code>	80
.12	<code>\xintDouble</code>	59	.37	<code>\xintiiDivision</code>	84
.13	<code>\xintHalf</code>	60	Derived arithmetic 100		
.14	<code>\xintInc</code>	60	.38	<code>\xintiiQuo, \xintiiRem</code>	100
.15	<code>\xintDec</code>	61	.39	<code>\xintiiDivRound</code>	100
.16	<code>\xintDSL</code>	61	.40	<code>\xintiiDivTrunc</code>	101
.17	<code>\xintDSR</code>	62	.41	<code>\xintiiModTrunc</code>	102
.18	<code>\xintDSRr</code>	62	.42	<code>\xintiiDivMod</code>	102
Blocks of eight digits 63			.43	<code>\xintiiDivFloor</code>	103
.19	<code>\XINT_cuz</code>	63	.44	<code>\xintiiMod</code>	103
.20	<code>\XINT_cuz_byviii</code>	63	.45	<code>\xintiiSqr</code>	103
.21	<code>\XINT_unsep_loop</code>	63	.46	<code>\xintiiPow</code>	105
.22	<code>\XINT_unsep_cuzsmall</code>	64	.47	<code>\xintiiFac</code>	108
.23	<code>\XINT_div_unsepQ</code>	64	.48	<code>\XINT_signaldeprecated</code>	111
.24	<code>\XINT_div_unsepR</code>	65	At End of L ^A T _E X Document deprecation message 111		
.25	<code>\XINT_zeroes_forviii</code>	65	.49	<code>\XINT_useiimessage</code>	112

Got split off from *xint* with release 1.1.

The core arithmetic routines have been entirely rewritten for release 1.2. The 1.2i and 1.2l brought again some improvements.

The commenting continues (2018/02/06) 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.

3.1 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 % :
```

3 Package *xintcore* implementation

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

3.2 Package identification

```
44 \XINT_providespackage
45 \ProvidesPackage{xintcore}%
46 [2018/02/06 1.2q Expandable arithmetic on big integers (JFB)]%
```

3.3 (WIP!) Error conditions and exceptions

As per the Mike Cowlishaw/IBM's General Decimal Arithmetic Specification

<http://speleotrove.com/decimal/decarith.html>

and the Python3 implementation in its Decimal module.

Clamped, ConversionSyntax, DivisionByZero, DivisionImpossible, DivisionUndefined, Inexact, InsufficientStorage, InvalidContext, InvalidOperation, Overflow, Inexact, Rounded, Subnormal, Underflow.

X3.274 rajoute LostDigits

Python rajoute FloatOperation (et n'inclut pas InsufficientStorage)

quote de decarith.pdf: The Clamped, Inexact, Rounded, and Subnormal conditions can coincide with each other or with other conditions. In these cases then any trap enabled for another condition takes precedence over (is handled before) all of these, any Subnormal trap takes precedence over Inexact, any Inexact trap takes precedence over Rounded, and any Rounded trap takes precedence over Clamped.

3 Package *xintcore* implementation

WORK IN PROGRESS ! (1.21, 2017/07/26)

I follow the Python terminology: a trapped signal means it raises an exception which for us means an expandable error message with some possible user interaction. In this WIP state, the interaction is commented out. A non-trapped signal or condition would activate a (presumably silent) handler.

Here, no signal-raising condition is "ignored" and all are "trapped" which means that error handlers are never activated, thus left in garbage state in the code.

Various conditions can raise the same signal.

Only signals, not conditions, raise Flags.

If a signal is ignored it does not raise a Flag, but it activates the signal handler (by default now no signal is ignored.)

If a signal is not ignored it raises a Flag and then if it is not trapped it activates the handler of the `_condition_`.

If trapped (which is default now) an «exception» is raised, which means an expandable error message (I copied over the LaTeX3 code for expandable error messages, basically) interrupts the TeX run. In future, user input could be solicited, but currently this is commented out.

For now macros to reset flags are done but without public interface nor documentation.

Only four conditions are currently possibly encountered:

- InvalidOperation
- DivisionByZero
- DivisionUndefined (which signals InvalidOperation)
- Underflow

I did it quickly, anyhow this will become more palpable when some of the Decimal Specification is actually implemented. The plan is to first do the X3.274 norm, then more complete implementation will follow... perhaps...

```
47 \csname XINT_Clamped_istrapped\endcsname
48 \csname XINT_ConversionSyntax_istrapped\endcsname
49 \csname XINT_DivisionByZero_istrapped\endcsname
50 \csname XINT_DivisionImpossible_istrapped\endcsname
51 \csname XINT_DivisionUndefined_istrapped\endcsname
52 \csname XINT_InvalidOperation_istrapped\endcsname
53 \csname XINT_Overflow_istrapped\endcsname
54 \csname XINT_Underflow_istrapped\endcsname
55 \catcode`- 11
56 \def\XINT_ConversionSyntax-signal {{InvalidOperation}}%
57 \let\XINT_DivisionImpossible-signal\XINT_ConversionSyntax-signal
58 \let\XINT_DivisionUndefined-signal \XINT_ConversionSyntax-signal
59 \let\XINT_InvalidContext-signal \XINT_ConversionSyntax-signal
60 \catcode`- 12
61 \def\XINT_signalcondition #1{\expandafter\XINT_signalcondition_a
62   \romannumeral0\ifcsname XINT_#1-signal\endcsname
63     \xint_dothis{\csname XINT_#1-signal\endcsname}%
64     \fi\xint_orthat{{#1}}{{#1}}%
65 \def\XINT_signalcondition_a #1#2#3#4#5{% copied over from Python Decimal module
66 % #1=signal, #2=condition, #3=explanation for user,
67 % #4=context for error handlers, #5=used
68   \ifcsname XINT_#1_ignoredflag\endcsname
69     \xint_dothis{\csname XINT_#1.handler\endcsname {#4}}%
70   \fi
71   \expandafter\xint_gobble_i\csname XINT_#1Flag_ON\endcsname
72   \unless\ifcsname XINT_#1_istrapped\endcsname
73     \xint_dothis{\csname XINT_#2.handler\endcsname {#4}}%
```

```

74 \fi
75 \xint_orthat{%
76 % the flag raised is named after the signal #1, but we show condition #2
77 \XINT_expandableerror{#2 (hit <RET> thrice)}%
78 \XINT_expandableerror{#3}%
79 \XINT_expandableerror{next: #5}%
80 % not for X3.274
81 %\XINT_expandableerror{<RET>, or I\xintUse{...}<RET>, or I\xintCTRLC<RET>}%
82 \xint_firstofone_thenstop{#5}%
83 }%
84 }%
85 %% \let\xintUse\xint_firstofthree_thenstop % defined in xint.sty
86 \def\XINT_ifFlagRaised #1{%
87 \ifcsname XINT_#1Flag_ON\endcsname
88 \expandafter\xint_firstoftwo
89 \else
90 \expandafter\xint_secondoftwo
91 \fi}%
92 \def\XINT_resetFlag #1%
93 {\expandafter\let\csname XINT_#1Flag_ON\endcsname\XINT_undefined}%
94 \def\XINT_resetFlags {% WIP
95 \XINT_resetFlag{InvalidOperation}% also from DivisionUndefined
96 \XINT_resetFlag{DivisionByZero}%
97 \XINT_resetFlag{Underflow}% (\xintiiPow with negative exponent)
98 \XINT_resetFlag{Overflow}% not encountered so far in xint code 1.21
99 % .. others ..
100 }%
101 \def\XINT_RaiseFlag #1{\expandafter\xint_gobble_i\csname XINT_#1Flag_ON\endcsname}%
102 %% NOT IMPLEMENTED! WORK IN PROGRESS! (ALL SIGNALS TRAPPED, NO HANDLERS USED)
103 \catcode`. 11
104 \let\XINT_Clamped.handler\xint_firstofone % WIP
105 \def\XINT_InvalidOperation.handler#1{NaN}% WIP
106 \def\XINT_ConversionSyntax.handler#1{NaN}% WIP
107 \def\XINT_DivisionByZero.handler#1{SignedInfinity(#1)}% WIP
108 \def\XINT_DivisionImpossible.handler#1{NaN}% WIP
109 \def\XINT_DivisionUndefined.handler#1{NaN}% WIP
110 \let\XINT_Inexact.handler\xint_firstofone % WIP
111 \def\XINT_InvalidContext.handler#1{NaN}% WIP
112 \let\XINT_Rounded.handler\xint_firstofone % WIP
113 \let\XINT_Subnormal.handler\xint_firstofone% WIP
114 \def\XINT_Overflow.handler#1{NaN}% WIP
115 \def\XINT_Underflow.handler#1{NaN}% WIP
116 \catcode`. 12

```

3.4 Counts for holding needed constants

```

117 \ifdefined\m@ne\let\xint_c_mone\m@ne
118 \else\csname newcount\endcsname\xint_c_mone \xint_c_mone -1 \fi
119 \newcount\xint_c_x^viii \xint_c_x^viii 1000000000
120 \ifdefined\xint_c_x^ix\else
121 \csname newcount\endcsname\xint_c_x^ix \xint_c_x^ix 10000000000
122 \fi
123 \newcount\xint_c_x^viii_mone \xint_c_x^viii_mone 99999999

```

```

124 \newcount\xint_c_xii_e_viii          \xint_c_xii_e_viii 12000000000
125 \newcount\xint_c_xi_e_viii_mone      \xint_c_xi_e_viii_mone 10999999999

```

Routines handling integers as lists of token digits

Routines handling big integers which are lists of digit tokens with no special additional structure.

Some routines do not accept non properly terminated inputs like "`\the\numexpr1`", or "`\the\mathcode`\-`", others do.

These routines or their sub-routines are mainly for internal usage.

3.5 \XINT_cuz_small

\XINT_cuz_small removes leading zeroes from the first eight digits. Expands following \romannumeral0. At least one digit is produced.

```

126 \def\XINT_cuz_small#1{%
127 \def\XINT_cuz_small ##1##2##3##4##5##6##7##8%
128 {%
129     \expandafter#1\the\numexpr ##1##2##3##4##5##6##7##8\relax
130 }}\XINT_cuz_small} %

```

3.6 \xintNum

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

Very old routine got completely rewritten at 1.21.

New code uses `\numexpr` governed expansion and fixes some issues of former version particularly regarding inputs of the `\numexpr...\relax` type without `\the` or `\number` prefix, and/or possibly no terminating `\relax`.

`\xintiNum{\numexpr 1}\foo` in earlier versions caused premature expansion of `\foo`.

`\xintiNum{\the\numexpr 1}` was ok, but a bit luckily so.

Also, up to 1.2k inclusive, the macro fetched tokens eight by eight, and not nine by nine as is done now. I have no idea why.

```

131 \def\xintiNum {\romannumeral0\xintinum }%
132 \def\xintinum #1%
133 {%
134     \expandafter\XINT_num_cleanup\the\numexpr\expandafter\XINT_num_loop
135     \romannumeral`&&0#1\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z
136 }%
137 \def\xintNum {\romannumeral0\xintnum }%
138 \let\xintnum\xintinum
139 \def\XINT_num #1%
140 {%
141     \expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
142     #1\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z
143 }%
144 \def\XINT_num_loop #1#2#3#4#5#6#7#8#9%
145 {%
146     \xint_gob_til_xint: #9\XINT_num_end\xint:
147     #1#2#3#4#5#6#7#8#9%
148     \ifnum \numexpr #1#2#3#4#5#6#7#8#9+\xint_c_ = \xint_c_

```


means that so far only signs encountered, (if syntax is legal) then possibly zeroes or a terminated or not terminated `\numexpr` evaluating to zero In that latter case a correct zero will be produced in the end.

```

149     \expandafter\XINT_num_loop
150     \else

non terminated \numexpr (with nine tokens total) are safe as after \fi, there is then \xint:

151     \expandafter\relax
152     \fi
153 }%
154 \def\XINT_num_end\xint:#1\xint:{#1+\xint_c_\xint:}% empty input ok
155 \def\XINT_num_cleanup #1\xint:#2\Z { #1}%

```

3.7 `\xintiiSgn`

1.2l made `\xintiiSgn` robust against non terminated input.

1.2o deprecates here `\xintSgn` (it requires `xintfrac.sty`).

```

156 \def\xintiiSgn {\romannumeral0\xintiisgn }%
157 \def\xintiisgn #1%
158 {%
159     \expandafter\XINT_sgn \romannumeral`&&@#1\xint:
160 }%
161 \def\xintSgn {\romannumeral0\XINT_signaldeprecated{xintcore}{xintSgn}\xintsgn }%
162 \def\xintsgn #1%
163 {%
164     \expandafter\XINT_sgn \romannumeral0\xintnum{#1}\xint:
165 }%
166 \def\XINT_sgn #1#2\xint:
167 {%
168     \xint_UDzerominusfork
169     #1-{ 0}%
170     0#1{-1}%
171     0-{ 1}%
172     \krof
173 }%
174 \def\XINT_Sgn #1#2\xint:
175 {%
176     \xint_UDzerominusfork
177     #1-{0}%
178     0#1{-1}%
179     0-{1}%
180     \krof
181 }%
182 \def\XINT_cntSgn #1#2\xint:
183 {%
184     \xint_UDzerominusfork
185     #1-\xint_c_
186     0#1\xint_c_mone
187     0-\xint_c_i
188     \krof
189 }%

```

3.8 `\xintiiOpp`

Attention, `\xintiiOpp` non robust against non terminated inputs. Reason is I don't want to have to grab a delimiter at the end, as everything happens "upfront".

```

190 \def\xintiiOpp {\romannumeral0\xintiiopp }%
191 \def\xintiiopp #1%
192 {%
193   \expandafter\XINT_opp \romannumeral`&&@#1%
194 }%
195 \def\xintiOpp {\romannumeral0\XINT_signaldeprecated{xintcore}{xintiOpp}\xintiopp }%
196 \def\xintiopp #1%
197 {%
198   \expandafter\XINT_opp \romannumeral0\xintnum{#1}%
199 }%
200 \def\XINT_Opp #1{\romannumeral0\XINT_opp #1}%
201 \def\XINT_opp #1%
202 {%
203   \xint_UDzerominusfork
204   #1-{ 0}%      zero
205   0#1{ }%      negative
206   0-{ -#1}%    positive
207   \krof
208 }%

```

3.9 `\xintiiAbs`

Attention `\xintiiAbs` non robust against non terminated input.

```

209 \def\xintiiAbs {\romannumeral0\xintiiabs }%
210 \def\xintiiabs #1%
211 {%
212   \expandafter\XINT_abs \romannumeral`&&@#1%
213 }%
214 \def\xintiAbs {\romannumeral0\XINT_signaldeprecated{xintcore}{xintiAbs}\xintiiabs }%
215 \def\xintiabs #1%
216 {%
217   \expandafter\XINT_abs \romannumeral0\xintnum{#1}%
218 }%
219 \def\XINT_abs #1%
220 {%
221   \xint_UDsignfork
222   #1{ }%
223   -{ #1}%
224   \krof
225 }%

```

3.10 `\xintFDg`

FIRST DIGIT.

1.2l: `\xintiiFDg` made robust against non terminated input.

1.2o deprecates `\xintiiFDg`, gives to `\xintFDg` former meaning of `\xintiiFDg`.

```

226 \def\xintFDg {\romannumeral0\xintfdg }%
227 \def\xintfdg #1{\expandafter\XINT_fdg \romannumeral`&&@#1\xint:\Z}%
228 \def\xintiifdg {\romannumeral0\XINT_signaldeprecated{xintcore}{xintiifdg}\xintiifdg }%
229 \let\xintiifdg\xintfdg
230 \def\XINT_FDg #1%
231     {\romannumeral0\expandafter\XINT_fdg\romannumeral`&&@\xintnum{#1}\xint:\Z }%
232 \def\XINT_fdg #1#2#3\Z
233 {%
234     \xint_UDzerominusfork
235     #1-{ 0}%    zero
236     0#1{ #2}%  negative
237     0-{ #1}%   positive
238     \krof
239 }%

```

3.11 \xintLDq

LAST DIGIT.

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

1.2o deprecates `\xintiiLDg`, gives to `\xintLDg` former meaning of `\xintiiLDg`.

Attention \xintLDg non robust against non terminated input.

[illegible]

3.12 \xintDouble

Attention \xintDouble non robust against non terminated input.

```

262 \def\xintDouble {\romannumeral0\xintdouble}%
263 \def\xintdouble #1{\expandafter\XINT_dbl_fork\romannumeral'&&#1%
264 \xint bve2345678\xint bve*\xint c ii\relax}%

```

3 Package *xintcore* implementation

```
265 \def\XINT_dbl_fork #1%
266 {%
267   \xint_UDsignfork
268   #1\XINT_dbl_neg
269   -\XINT_dbl
270   \krof #1%
271 }%
272 \def\XINT_dbl_neg-{\expandafter-\romannumeral0\XINT_dbl}%
273 \def\XINT_dbl #1{%
274 \def\XINT_dbl ##1##2##3##4##5##6##7##8%
275   {\expandafter#1\the\numexpr##1##2##3##4##5##6##7##8\XINT_dbl_a}%
276 }\XINT_dbl{ }%
277 \def\XINT_dbl_a #1#2#3#4#5#6#7#8%
278   {\expandafter\XINT_dbl_e\the\numexpr 1#1#2#3#4#5#6#7#8\XINT_dbl_a}%
279 \def\XINT_dbl_e#1{* \xint_c_ii\if#13+\xint_c_i\fi\relax}%
```

3.13 \xintHalf

Attention \xintHalf non robust against non terminated input.

```
280 \def\xintHalf {\romannumeral0\xinthalf}%
281 \def\xinthalf #1{\expandafter\XINT_half_fork\romannumeral`&&@#1%
282   \xint_bye\xint_Bye345678\xint_bye
283   *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax}%
284 \def\XINT_half_fork #1%
285 {%
286   \xint_UDsignfork
287   #1\XINT_half_neg
288   -\XINT_half
289   \krof #1%
290 }%
291 \def\XINT_half_neg-{\xintiiopp\XINT_half}%
292 \def\XINT_half #1{%
293 \def\XINT_half ##1##2##3##4##5##6##7##8%
294   {\expandafter#1\the\numexpr(##1##2##3##4##5##6##7##8\XINT_half_a}%
295 }\XINT_half{ }%
296 \def\XINT_half_a#1{\xint_Bye#1\xint_bye\XINT_half_b#1}%
297 \def\XINT_half_b #1#2#3#4#5#6#7#8%
298   {\expandafter\XINT_half_e\the\numexpr(1#1#2#3#4#5#6#7#8\XINT_half_a}%
299 \def\XINT_half_e#1{* \xint_c_v+#1-\xint_c_v)\relax}%
```

3.14 \xintInc

1.2i much delayed complete rewrite in 1.2 style.

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.

Attention \xintInc non robust against non terminated input.

```
300 \def\xintInc {\romannumeral0\xintinc}%
301 \def\xintinc #1{\expandafter\XINT_inc_fork\romannumeral`&&@#1%
302   \xint_bye23456789\xint_bye+\xint_c_i\relax}%
303 \def\XINT_inc_fork #1%
304 {%
```

```

305 \xint_UDsignfork
306 #1\xINT_inc_neg
307 -\XINT_inc
308 \krof #1%
309 }%
310 \def\xINT_inc_neg-#1\xint_bye#2\relax
311 {\xintiiopt\xINT_dec #1\xINT_dec_bye234567890\xint_bye}%
312 \def\xINT_inc #1{%
313 \def\xINT_inc ##1##2##3##4##5##6##7##8##9%
314 {\expandafter#1\the\numexpr##1##2##3##4##5##6##7##8##9\xINT_inc_a}%
315 }\XINT_inc{ }%
316 \def\xINT_inc_a #1#2#3#4#5#6#7#8#9%
317 {\expandafter\xINT_inc_e\the\numexpr 1#1#2#3#4#5#6#7#8#9\xINT_inc_a}%
318 \def\xINT_inc_e#1{\if#12+\xint_c_i\fi\relax}%

```

3.15 \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.

Attention \xintDec non robust against non terminated input.

```

319 \def\xintDec {\romannumeral0\xintdec}%
320 \def\xintdec #1{\expandafter\xINT_dec_fork\romannumeral`&&@#1%
321 \XINT_dec_bye234567890\xint_bye}%
322 \def\xINT_dec_fork #1%
323 {%
324 \xint_UDsignfork
325 #1\xINT_dec_neg
326 -\XINT_dec
327 \krof #1%
328 }%
329 \def\xINT_dec_neg-#1\xINT_dec_bye#2\xint_bye
330 {\expandafter-%
331 \romannumeral0\xINT_inc #1\xint_bye23456789\xint_bye+\xint_c_i\relax}%
332 \def\xINT_dec #1{%
333 \def\xINT_dec ##1##2##3##4##5##6##7##8##9%
334 {\expandafter#1\the\numexpr##1##2##3##4##5##6##7##8##9\xINT_dec_a}%
335 }\XINT_dec{ }%
336 \def\xINT_dec_a #1#2#3#4#5#6#7#8#9%
337 {\expandafter\xINT_dec_e\the\numexpr 1#1#2#3#4#5#6#7#8#9\xINT_dec_a}%
338 \def\xINT_dec_bye #1\xINT_dec_a#2#3\xint_bye
339 {\if#20-\xint_c_ii\relax+\else-\fi\xint_c_i\relax}%
340 \def\xINT_dec_e#1{\unless\if#11\xint_dothis{-\xint_c_i#1}\fi\xint_orthat\relax}%

```

3.16 \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.

Attention \xintDSL non robust against non terminated input.

```

341 \def\xintDSL {\romannumeral0\xintdsl }%
342 \def\xintdsl #1{\expandafter\xINT_dsl\romannumeral`&&@#10}%
343 \def\xINT_dsl#1{%

```

```

344 \def\XINT_dsl ##1{\xint_gob_til_zero ##1\xint_dsl_zero 0#1##1}%
345 }\XINT_dsl{ }%
346 \def\xint_dsl_zero 0 0{ }%

```

3.17 \xintDSR

Decimal shift right, truncates towards zero. Rewritten for 1.2i. Limited to 22483 digits on input.
 Attention \xintDSR non robust against non terminated input.

```

347 \def\xintDSR{\romannumeral0\xintdsr}%
348 \def\xintdsr #1{\expandafter\XINT_dsr_fork\romannumeral`&&@#1%
349   \xint_bye\xint_Bye3456789\xint_bye+\xint_c_v)/\xint_c_x-\xint_c_i\relax}%
350 \def\XINT_dsr_fork #1%
351 {%
352   \xint_UDsignfork
353   #1\XINT_dsr_neg
354   -\XINT_dsr
355   \krof #1%
356 }%
357 \def\XINT_dsr_neg-{\xintiopp\XINT_dsr}%
358 \def\XINT_dsr #1{%
359 \def\XINT_dsr ##1##2##3##4##5##6##7##8##9%
360   {\expandafter#1\the\numexpr(##1##2##3##4##5##6##7##8##9\XINT_dsr_a}%
361 }\XINT_dsr{ }%
362 \def\XINT_dsr_a#1{\xint_Bye#1\xint_bye\XINT_dsr_b#1}%
363 \def\XINT_dsr_b #1#2#3#4#5#6#7#8#9%
364   {\expandafter\XINT_dsr_e\the\numexpr(1#1#2#3#4#5#6#7#8#9\XINT_dsr_a}%
365 \def\XINT_dsr_e #1{)\relax}%

```

3.18 \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!

Attention \xintDSRr non robust against non terminated input.

```

366 \def\xintDSRr{\romannumeral0\xintdsrr}%
367 \def\xintdsrr #1{\expandafter\XINT_dsrr_fork\romannumeral`&&@#1%
368   \xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax}%
369 \def\XINT_dsrr_fork #1%
370 {%
371   \xint_UDsignfork
372   #1\XINT_dsrr_neg
373   -\XINT_dsrr
374   \krof #1%
375 }%
376 \def\XINT_dsrr_neg-{\xintiopp\XINT_dsrr}%
377 \def\XINT_dsrr #1{%
378 \def\XINT_dsrr ##1##2##3##4##5##6##7##8##9%
379   {\expandafter#1\the\numexpr##1##2##3##4##5##6##7##8##9\XINT_dsrr_a}%
380 }\XINT_dsrr{ }%
381 \def\XINT_dsrr_a#1{\xint_Bye#1\xint_bye\XINT_dsrr_b#1}%
382 \def\XINT_dsrr_b #1#2#3#4#5#6#7#8#9%

```

```
383 {\expandafter\XINT_dsrr_e\the\numexpr1#1#2#3#4#5#6#7#8#9\XINT_dsrr_a}%
384 \let\XINT_dsrr_e\XINT_inc_e
```

Blocks of eight digits

The lingua of release 1.2.

3.19 \XINT_cuz

This (launched by \romannumeral0) iterately removes all leading zeroes from a sequence of 8N digits ended by \R.

Rewritten for 1.2l, now uses \numexpr governed expansion and \ifnum test rather than delimited gobbling macros.

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 back then.

```
385 \def\XINT_cuz #1{%
386 \def\XINT_cuz {\expandafter#1\the\numexpr\XINT_cuz_loop}%
387 }\XINT_cuz{ }%
388 \def\XINT_cuz_loop #1#2#3#4#5#6#7#8#9%
389 {%
390     #1#2#3#4#5#6#7#8%
391     \xint_gob_til_R #9\XINT_cuz_hitend\R
392     \ifnum #1#2#3#4#5#6#7#8>\xint_c_
393         \expandafter\XINT_cuz_cleantoend
394     \else\expandafter\XINT_cuz_loop
395     \fi #9%
396 }%
397 \def\XINT_cuz_hitend\R #1\R{\relax}%
398 \def\XINT_cuz_cleantoend #1\R{\relax #1}%
```

3.20 \XINT_cuz_byviii

This removes eight by eight leading zeroes from a sequence of 8N digits ended by \R. Thus, we still have 8N digits on output. Expansion started by \romannumeral0

```
399 \def\XINT_cuz_byviii #1#2#3#4#5#6#7#8#9%
400 {%
401     \xint_gob_til_R #9\XINT_cuz_byviii_e \R
402     \xint_gob_til_eightzeroes #1#2#3#4#5#6#7#8\XINT_cuz_byviii_z 00000000%
403     \XINT_cuz_byviii_done #1#2#3#4#5#6#7#8#9%
404 }%
405 \def\XINT_cuz_byviii_z 00000000\XINT_cuz_byviii_done 00000000{\XINT_cuz_byviii}%
406 \def\XINT_cuz_byviii_done #1\R { #1}%
407 \def\XINT_cuz_byviii_e\R #1\XINT_cuz_byviii_done #2\R{ #2}%
```

3.21 \XINT_unsep_loop

This is used as

```
\the\numexpr0\XINT_unsep_loop (blocks of 1<8digits>!)
\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax
```

3 Package *xintcore* implementation

It removes the 1's and !'s, and outputs the 8N digits with a 0 token as as prefix which will have to be cleaned out by caller.

Actually it does not matter whether the blocks contain really 8 digits, all that matters is that they have 1 as first digit (and at most 9 digits after that to obey the TeX-\numexpr bound).

Done at 1.2l for usage by other macros. The similar code in earlier releases was strangely in $O(N^2)$ style, apparently to avoid some memory constraints. But these memory constraints related to \numexpr chaining seems to be in many places in xint code base. The 1.2l version is written in the 1.2i style of \xintInc etc... and is compatible with some 1! block without digits among the treated blocks, they will disappear.

```
408 \def\xINT_unsep_loop #1!#2!#3!#4!#5!#6!#7!#8!#9!%
409 {%
410   \expandafter\xINT_unsep_clean
411   \the\numexpr #1\expandafter\xINT_unsep_clean
412   \the\numexpr #2\expandafter\xINT_unsep_clean
413   \the\numexpr #3\expandafter\xINT_unsep_clean
414   \the\numexpr #4\expandafter\xINT_unsep_clean
415   \the\numexpr #5\expandafter\xINT_unsep_clean
416   \the\numexpr #6\expandafter\xINT_unsep_clean
417   \the\numexpr #7\expandafter\xINT_unsep_clean
418   \the\numexpr #8\expandafter\xINT_unsep_clean
419   \the\numexpr #9\xINT_unsep_loop
420 }%
421 \def\xINT_unsep_clean 1{\relax}%
```

3.22 \XINT_unsep_cuzsmall

This is used as

```
\romannumeral0\xINT_unsep_cuzsmall (blocks of 1<8d>!)
\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax
```

It removes the 1's and !'s, and removes the leading zeroes *of the first block*.

Redone for 1.2l: the 1.2 variant was strangely in $O(N^2)$ style.

```
422 \def\xINT_unsep_cuzsmall
423 {%
424   \expandafter\xINT_unsep_cuzsmall_x\the\numexpr0\xINT_unsep_loop
425 }%
426 \def\xINT_unsep_cuzsmall_x #1{%
427 \def\xINT_unsep_cuzsmall_x 0##1##2##3##4##5##6##7##8%
428 {%
429   \expandafter#1\the\numexpr ##1##2##3##4##5##6##7##8\relax
430 }}\xINT_unsep_cuzsmall_x{ }%
```

3.23 \XINT_div_unsepQ

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. Attention that there might be a single block of 8 zeroes. Expansion launched by \romannumeral0.

Rewritten for 1.2l in 1.2i style.

```
431 \def\xINT_div_unsepQ_delim {\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax\Z}%
```



```

432 \def\XINT_div_unsepQ
433 {%
434   \expandafter\XINT_div_unsepQ_x\the\numexpr0\XINT_unsep_loop
435 }%
436 \def\XINT_div_unsepQ_x #1{%
437 \def\XINT_div_unsepQ_x 0##1##2##3##4##5##6##7##8##9%
438 {%
439   \xint_gob_til_Z ##9\XINT_div_unsepQ_one\Z
440   \xint_gob_til_eightzeroes ##1##2##3##4##5##6##7##8\XINT_div_unsepQ_y 00000000%
441   \expandafter#1\the\numexpr ##1##2##3##4##5##6##7##8\relax ##9%
442 }}\XINT_div_unsepQ_x{ }%
443 \def\XINT_div_unsepQ_y #1{%
444 \def\XINT_div_unsepQ_y ##1\relax ##2##3##4##5##6##7##8##9%
445 {%
446   \expandafter#1\the\numexpr ##2##3##4##5##6##7##8##9\relax
447 }}\XINT_div_unsepQ_y{ }%
448 \def\XINT_div_unsepQ_one#1\expandafter{\expandafter}%

```

3.24 \XINT_div_unsepR

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.

Also rewritten for 1.2l, the 1.2 version was $O(N^2)$ style.

Terminator \xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax\R

We have a need for something like \R because it is not guaranteed the thing is not actually zero.

```

449 \def\XINT_div_unsepR
450 {%
451   \expandafter\XINT_div_unsepR_x\the\numexpr0\XINT_unsep_loop
452 }%
453 \def\XINT_div_unsepR_x#1{%
454 \def\XINT_div_unsepR_x 0{\expandafter#1\the\numexpr\XINT_cuz_loop}%
455 }\XINT_div_unsepR_x{ }%

```

3.25 \XINT_zeroes_forviii

\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.

```

456 \def\XINT_zeroes_forviii #1#2#3#4#5#6#7#8%
457 {%
458   \xint_gob_til_R #8\XINT_zeroes_forviii_end\R\XINT_zeroes_forviii
459 }%
460 \def\XINT_zeroes_forviii_end#1{%
461 \def\XINT_zeroes_forviii_end\R\XINT_zeroes_forviii ##1##2##3##4##5##6##7##8##9\W
462 {%
463   \expandafter#1\xint_gob_til_one ##2##3##4##5##6##7##8%
464 }}\XINT_zeroes_forviii_end{ }%

```

3.26 `\XINT_sepbyviii_Z`

This is used as

```
\the\numexpr\XINT_sepbyviii_Z <8Ndigits>\XINT_sepbyviii_Z_end 2345678\relax
```

It produces `1<8d>!\...1<8d>!\;`

Prior to 1.21 it used `\Z` as terminator not the semi-colon (hence the name). The switch to `;` was done at a time I thought perhaps I would use an internal format maintaining such 8 digits blocks, and this has to be compatible with the `\csname...\endcsname` encapsulation in `\xintexpr` parsers.

```
465 \def\XINT_sepbyviii_Z #1#2#3#4#5#6#7#8%
466 {%
467     1#1#2#3#4#5#6#7#8\expandafter!\the\numexpr\XINT_sepbyviii_Z
468 }%
469 \def\XINT_sepbyviii_Z_end #1\relax {;!}%
```

3.27 `\XINT_sepbyviii_andcount`

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\xint:<count of blocks>\xint:
```

Used by `\XINT_div_prepare_g` for `\XINT_div_prepare_h`, and also by `\xintiiCmp`.

```
470 \def\XINT_sepbyviii_andcount
471 {%
472     \expandafter\XINT_sepbyviii_andcount_a\the\numexpr\XINT_sepbyviii
473 }%
474 \def\XINT_sepbyviii #1#2#3#4#5#6#7#8%
475 {%
476     1#1#2#3#4#5#6#7#8\expandafter!\the\numexpr\XINT_sepbyviii
477 }%
478 \def\XINT_sepbyviii_end #1\relax {\relax\XINT_sepbyviii_andcount_end!}%
479 \def\XINT_sepbyviii_andcount_a {\XINT_sepbyviii_andcount_b \xint_c_\xint:}%
480 \def\XINT_sepbyviii_andcount_b #1\xint:#2!#3!#4!#5!#6!#7!#8!#9!%
481 {%
482     #2\expandafter!\the\numexpr#3\expandafter!\the\numexpr#4\expandafter
483     !\the\numexpr#5\expandafter!\the\numexpr#6\expandafter!\the\numexpr
484     #7\expandafter!\the\numexpr#8\expandafter!\the\numexpr#9\expandafter!\the\numexpr
485     \expandafter\XINT_sepbyviii_andcount_b\the\numexpr #1+\xint_c_viii\xint:%
486 }%
487 \def\XINT_sepbyviii_andcount_end #1\XINT_sepbyviii_andcount_b\the\numexpr
488     #2+\xint_c_viii\xint:#3#4\W {\expandafter\xint:\the\numexpr #2+#3\xint:}%
```

3.28 `\XINT_rsepbyviii`

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>\xint:1<8digits>!...

where the original digits are organized by eight, and the order inside successive pairs of blocks separated by \xint: has been reversed. Output ends either in 1<8d>!1<8d>\xint:1U\xint: (even) or 1<8d>!1<8d>\xint:1V!1<8d>\xint: (odd)

The U an V should be \numexpr1 stoppers (or will expand and be ended by !). This macro is currently (1.2..1.21) exclusively used in combination with \XINT_sepandrev_andcount or \XINT_sepandrev.

```

489 \def\xINT_rsepbyviii #1#2#3#4#5#6#7#8%
490 {%
491   \XINT_rsepbyviii_b {#1#2#3#4#5#6#7#8}%
492 }%
493 \def\xINT_rsepbyviii_b #1#2#3#4#5#6#7#8#9%
494 {%
495   #2#3#4#5#6#7#8#9\expandafter!\the\numexpr
496   1#1\expandafter\xint:\the\numexpr 1\xINT_rsepbyviii
497 }%
498 \def\xINT_rsepbyviii_end_B #1\relax #2#3{#2\xint:}%
499 \def\xINT_rsepbyviii_end_A #1#2\expandafter #3\relax #4#5{#5!1#2\xint:}%

```

3.29 \XINT_sepandrev

This is used typically as

```

\romannumeral0\xINT_sepandrev <8Ndigits>%
    \XINT_rsepbyviii_end_A 2345678%
    \XINT_rsepbyviii_end_B 2345678\relax UV%
    \R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\W

```

and will produce

1<8digits>!1<8digits>!1<8digits>!...

where the blocks have been globally reversed. The UV here are only place holders (must be \numexpr1 stoppers) to share same syntax as \XINT_sepandrev_andcount, they are gobbled (#2 in \XINT_sepandrev_done).

```

500 \def\xINT_sepandrev
501 {%
502   \expandafter\xINT_sepandrev_a\the\numexpr 1\xINT_rsepbyviii
503 }%
504 \def\xINT_sepandrev_a {\XINT_sepandrev_b {}}%
505 \def\xINT_sepandrev_b #1#2\xint:#3\xint:#4\xint:#5\xint:#6\xint:#7\xint:#8\xint:#9\xint:%
506 {%
507   \xint_gob_til_R #9\xINT_sepandrev_end\R
508   \XINT_sepandrev_b {#9!#8!#7!#6!#5!#4!#3!#2!#1}%
509 }%
510 \def\xINT_sepandrev_end\R\xINT_sepandrev_b #1#2\W {\XINT_sepandrev_done #1}%
511 \def\xINT_sepandrev_done #1#2!{ }%

```

3.30 \XINT_sepandrev_andcount

This is used typically as

```

\romannumeral0\xINT_sepandrev_andcount <8Ndigits>%
    \XINT_rsepbyviii_end_A 2345678%
    \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
    \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
    \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c\W

```

and will produce

<length>.1<8digits>!1<8digits>!1<8digits>!...

where the blocks have been globally reversed and <length> is the number of blocks.

```

512 \def\XINT_sepandrev_andcount
513 {%
514   \expandafter\XINT_sepandrev_andcount_a\the\numexpr 1\XINT_rsepyviii
515 }%
516 \def\XINT_sepandrev_andcount_a {\XINT_sepandrev_andcount_b 0!{}}%
517 \def\XINT_sepandrev_andcount_b #1!#2#3\xint:#4\xint:#5\xint:#6\xint:#7\xint:#8\xint:#9\xint:%
518 {%
519   \xint_gob_til_R #9\XINT_sepandrev_andcount_end\R
520   \expandafter\XINT_sepandrev_andcount_b \the\numexpr #1+\xint_c_i!%
521   {#9!#8!#7!#6!#5!#4!#3!#2}%
522 }%
523 \def\XINT_sepandrev_andcount_end\R
524   \expandafter\XINT_sepandrev_andcount_b\the\numexpr #1+\xint_c_i!#2#3#4\W
525 {\expandafter\XINT_sepandrev_andcount_done\the\numexpr #3+\xint_c_xiv*#1!#2}%
526 \def\XINT_sepandrev_andcount_done#1{%
527 \def\XINT_sepandrev_andcount_done##1!##2!##3!{\expandafter#1\the\numexpr##1-##3\xint:}%
528 }\XINT_sepandrev_andcount_done{ }%

```

3.31 \XINT_rev_nounsep

This is used as

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

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

```

529 \def\XINT_rev_nounsep #1#2!#3!#4!#5!#6!#7!#8!#9!%
530 {%
531   \xint_gob_til_R #9\XINT_rev_nounsep_end\R
532   \XINT_rev_nounsep {#9!#8!#7!#6!#5!#4!#3!#2!#1}%
533 }%
534 \def\XINT_rev_nounsep_end\R\XINT_rev_nounsep #1#2\W {\XINT_rev_nounsep_done #1}%
535 \def\XINT_rev_nounsep_done #11{ 1}%

```

3.32 \XINT_unrevbyviii

Used as \romannumeral0\XINT_unrevbyviii 1<8d>!...1<8d>! terminated by

1;!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). This mechanism is used by 1.2 subtraction (still true for 1.21).

```

536 \def\XINT_unrevbyviii #11#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
537 {%
538   \xint_gob_til_R #9\XINT_unrevbyviii_a\R
539   \XINT_unrevbyviii {#9#8#7#6#5#4#3#2#1}%
540 }%
541 \def\XINT_unrevbyviii_a#1{%
542 \def\XINT_unrevbyviii_a\R\XINT_unrevbyviii ##1##2\W
543   {\expandafter#1\romannumeral`&&@\xint_gob_til_sc ##1}%

```

```
544 }\XINT_unrevbyviii_a{ }%
```

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

```
545 \def\XINT_smallunrevbyviii 1#1!1#2!1#3!1#4!1#5!1#6!1#7!1#8!#9\W%
546 {%
547   \expandafter\XINT_cuz_small\xint_gob_til_sc #8#7#6#5#4#3#2#1%
548 }%
```

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.

3.33 \xintiiAdd

1.21: \xintiiAdd made robust against non terminated input.

```
549 \def\xintiAdd    {\romannumeral0\XINT_signaldeprecated{xintcore}{xintiAdd}\xintiadd }%
550 \def\xintiadd   #1{\expandafter\XINT_iadd\romannumeral0\xintnum{#1}\xint:}%
551 \def\xintiiAdd   {\romannumeral0\xintiiadd }%
552 \def\xintiiadd   #1{\expandafter\XINT_iiadd\romannumeral`&&@#1\xint:}%
553 \def\XINT_iiadd  #1#2\xint:#3%
554 {%
555   \expandafter\XINT_add_nfork\expandafter#1\romannumeral`&&@#3\xint:#2\xint:
556 }%
557 \def\XINT_iadd  #1#2\xint:#3%
558 {%
559   \expandafter\XINT_add_nfork\expandafter
560   #1\romannumeral0\xintnum{#3}\xint:#2\xint:
561 }%
562 \def\XINT_add_fork #1#2\xint:#3\xint:{\XINT_add_nfork #1#3\xint:#2\xint:}%
563 \def\XINT_add_nfork #1#2%
564 {%
565   \xint_UDzerofork
566   #1\XINT_add_firstiszero
567   #2\XINT_add_secondiszero
568   0{}%
569   \krof
570   \xint_UDsignsfork
571   #1#2\XINT_add_minusminus
572   #1-\XINT_add_minusplus
573   #2-\XINT_add_plusminus
574   --\XINT_add_plusplus
```

3 Package *xintcore* implementation

```

575 \krof #1#2%
576 }%
577 \def\XINT_add_firstiszero #1\krof 0#2#3\xint:#4\xint:{ #2#3}%
578 \def\XINT_add_secondiszero #1\krof #20#3\xint:#4\xint:{ #2#4}%
579 \def\XINT_add_minusminus #1#2%
580 {\expandafter\romannumeral0\XINT_add_pp_a {}{}}%
581 \def\XINT_add_minusplus #1#2{\XINT_sub_mm_a {}#2}%
582 \def\XINT_add_plusminus #1#2%
583 {\expandafter\XINT_opp\romannumeral0\XINT_sub_mm_a #1{}}%
584 \def\XINT_add_pp_a #1#2#3\xint:
585 {%
586 \expandafter\XINT_add_pp_b
587 \romannumeral0\expandafter\XINT_sepandrev_andcount
588 \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
589 #2#3\XINT_rsepbyviii_end_A 2345678%
590 \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
591 \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
592 \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
593 \X #1%
594 }%
595 \let\XINT_add_plusplus \XINT_add_pp_a
596 \def\XINT_add_pp_b #1\xint:#2\X #3\xint:
597 {%
598 \expandafter\XINT_add_checklengths
599 \the\numexpr #1\expandafter\xint:%
600 \romannumeral0\expandafter\XINT_sepandrev_andcount
601 \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
602 #3\XINT_rsepbyviii_end_A 2345678%
603 \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
604 \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
605 \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
606 1;!1;!1;!1;!1\W #21;!1;!1;!1;!1\W
607 1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\W
608 }%

```

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

```

609 \def\XINT_add_checklengths #1\xint:#2\xint:%
610 {%
611 \ifnum #2>#1
612 \expandafter\XINT_add_exchange
613 \else
614 \expandafter\XINT_add_A
615 \fi
616 #1\xint:#2\xint:%
617 }%
618 \def\XINT_add_exchange #1\xint:#2\xint:#3\W #4\W
619 {%
620 \XINT_add_A #2\xint:#1\xint:#4\W #3\W
621 }%
622 \def\XINT_add_A #1\xint:#2\xint:%
623 {%

```

3 Package *xintcore* implementation

```

624 \ifnum #1>\xint_c_vi
625 \expandafter\XINT_add_aa
626 \else \expandafter\XINT_add_aa_small
627 \fi
628 }%
629 \def\XINT_add_aa {\expandafter\XINT_add_out\the\numexpr\XINT_add_a \xint_c_ii}%
630 \def\XINT_add_out{\expandafter\XINT_cuz_small\romannumeral0\XINT_unrevbyviii {}}%
631 \def\XINT_add_aa_small
632 {\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;!, replacing the \Z! used in 1.2.

Call: \the\numexpr\XINT_add_a 2#11;!1;!1;!1;!1\W #21;!1;!1;!1;!1\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;!. In recursive algorithm this 1;! terminator can thus conveniently be reused as part of input terminator (up to the length problem).

```

633 \def\XINT_add_a #1!#2!#3!#4!#5\W
634 #6!#7!#8!#9!%
635 {%
636 \XINT_add_b
637 #1!#6!#2!#7!#3!#8!#4!#9!%
638 #5\W
639 }%
640 \def\XINT_add_b #11#2#3!#4!%
641 {%
642 \xint_gob_til_sc #2\XINT_add_bi ;%
643 \expandafter\XINT_add_c\the\numexpr#1+1#2#3+#4-\xint_c_ii\xint:%
644 }%
645 \def\XINT_add_bi;\expandafter\XINT_add_c
646 \the\numexpr#1+#2+#3-\xint_c_ii\xint:#4!#5!#6!#7!#8!#9!\W
647 {%
648 \XINT_add_k #1#3!#5!#7!#9!%
649 }%
650 \def\XINT_add_c #1#2\xint:%
651 {%
652 1#2\expandafter!\the\numexpr\XINT_add_d #1%
653 }%
654 \def\XINT_add_d #11#2#3!#4!%
655 {%
656 \xint_gob_til_sc #2\XINT_add_di ;%
657 \expandafter\XINT_add_e\the\numexpr#1+1#2#3+#4-\xint_c_ii\xint:%
658 }%
659 \def\XINT_add_di;\expandafter\XINT_add_e
660 \the\numexpr#1+#2+#3-\xint_c_ii\xint:#4!#5!#6!#7!#8\W
661 {%
662 \XINT_add_k #1#3!#5!#7!%
663 }%
664 \def\XINT_add_e #1#2\xint:%
665 {%
666 1#2\expandafter!\the\numexpr\XINT_add_f #1%

```

```

667 }%
668 \def\XINT_add_f #1#2#3!#4!%
669 {%
670   \xint_gob_til_sc #2\XINT_add_fi ;%
671   \expandafter\XINT_add_g\the\numexpr#1+1#2#3+#4-\xint_c_ii\xint:%
672 }%
673 \def\XINT_add_fi;\expandafter\XINT_add_g
674   \the\numexpr#1+#2+#3-\xint_c_ii\xint:#4!#5!#6\W
675 {%
676   \XINT_add_k #1#3!#5!%
677 }%
678 \def\XINT_add_g #1#2\xint:%
679 {%
680   1#2\expandafter!\the\numexpr\XINT_add_h #1%
681 }%
682 \def\XINT_add_h #1#2#3!#4!%
683 {%
684   \xint_gob_til_sc #2\XINT_add_hi ;%
685   \expandafter\XINT_add_i\the\numexpr#1+1#2#3+#4-\xint_c_ii\xint:%
686 }%
687 \def\XINT_add_hi;%
688   \expandafter\XINT_add_i\the\numexpr#1+#2+#3-\xint_c_ii\xint:#4\W
689 {%
690   \XINT_add_k #1#3!%
691 }%
692 \def\XINT_add_i #1#2\xint:%
693 {%
694   1#2\expandafter!\the\numexpr\XINT_add_a #1%
695 }%
696 \def\XINT_add_k #1{\if #12\expandafter\XINT_add_ke\else\expandafter\XINT_add_l \fi}%
697 \def\XINT_add_ke #11;#2\W {\XINT_add_kf #11;!}%
698 \def\XINT_add_kf 1{1\relax }%
699 \def\XINT_add_l 1#1#2{\xint_gob_til_sc #1\XINT_add_lf ;\XINT_add_m 1#1#2}%
700 \def\XINT_add_lf #1\W {1\relax 00000001!1;!}%
701 \def\XINT_add_m #1!{\expandafter\XINT_add_n\the\numexpr\xint_c_i+#1\xint:%}%
702 \def\XINT_add_n #1#2\xint:{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>.)

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

```

3.34 \xintiiCmp

Moved from *xint.sty* to *xintcore.sty* and rewritten for 1.21.

1.21's *\xintiiCmp* is robust against non terminated input.

1.2o deprecates *\xintCmp*, with *xintfrac* loaded it will get overwritten anyhow.

```

704 \def\xintCmp    {\romannumeral0\XINT_signaldeprecated{xintcore}{xintCmp}\xintcmp }%
705 \def\xintcmp    #1{\expandafter\XINT_icmp\romannumeral0\xintnum{#1}\xint:%}%
706 \def\xintiiCmp  {\romannumeral0\xintiicmp }%
707 \def\xintiicmp  #1{\expandafter\XINT_iicmp\romannumeral`&&@#1\xint:%}%
708 \def\XINT_iicmp #1#2\xint:#3%
709 {%

```


3 Package *xintcore* implementation

```

710 \expandafter\XINT_cmp_nfork\expandafter #1\romannumeral`&&@#3\xint:#2\xint:
711 }%
712 \def\XINT_icmp #1#2\xint:#3%
713 {%
714 \expandafter\XINT_cmp_nfork\expandafter #1\romannumeral0\xintnum{#3}\xint:#2\xint:
715 }%
716 \def\XINT_cmp_nfork #1#2%
717 {%
718 \xint_UDzerofork
719 #1\XINT_cmp_firstiszero
720 #2\XINT_cmp_secondiszero
721 0{}%
722 \krof
723 \xint_UDsignsfork
724 #1#2\XINT_cmp_minusminus
725 #1-\XINT_cmp_minusplus
726 #2-\XINT_cmp_plusminus
727 --\XINT_cmp_plusplus
728 \krof #1#2%
729 }%
730 \def\XINT_cmp_firstiszero #1\krof 0#2#3\xint:#4\xint:
731 {%
732 \xint_UDzerominusfork
733 #2-{ 0}%
734 0#2{ 1}%
735 0-{ -1}%
736 \krof
737 }%
738 \def\XINT_cmp_secondiszero #1\krof #20#3\xint:#4\xint:
739 {%
740 \xint_UDzerominusfork
741 #2-{ 0}%
742 0#2{ -1}%
743 0-{ 1}%
744 \krof
745 }%
746 \def\XINT_cmp_plusminus #1\xint:#2\xint:{ 1}%
747 \def\XINT_cmp_minusplus #1\xint:#2\xint:{ -1}%
748 \def\XINT_cmp_minusminus
749 --{\expandafter\XINT_opp\romannumeral0\XINT_cmp_plusplus {}{}}%
750 \def\XINT_cmp_plusplus #1#2#3\xint:
751 {%
752 \expandafter\XINT_cmp_pp
753 \the\numexpr\expandafter\XINT_sepbyviii_andcount
754 \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\{10}0000001\W
755 #2#3\XINT_sepbyviii_end 2345678\relax
756 \xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!%
757 \xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_\W
758 #1%
759 }%
760 \def\XINT_cmp_pp #1\xint:#2\xint:#3\xint:
761 {%

```

```

762 \expandafter\XINT_cmp_checklengths
763 \the\numexpr #2\expandafter\xint:%
764 \the\numexpr\expandafter\XINT_sepyviii_andcount
765 \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\{10}0000001\W
766 #3\XINT_sepyviii_end 2345678\relax
767 \xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!%
768 \xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_\W
769 #1;!1;!1;!1;!1;\W
770 }%
771 \def\XINT_cmp_checklengths #1\xint:#2\xint:#3\xint:
772 {%
773 \ifnum #1=#3
774 \expandafter\xint_firstoftwo
775 \else
776 \expandafter\xint_secondoftwo
777 \fi
778 \XINT_cmp_a {\XINT_cmp_distinctlengths {#1}{#3}}#2;!1;!1;!1;\W
779 }%
780 \def\XINT_cmp_distinctlengths #1#2#3\W #4\W
781 {%
782 \ifnum #1>#2
783 \expandafter\xint_firstoftwo
784 \else
785 \expandafter\xint_secondoftwo
786 \fi
787 { -1}{ 1}%
788 }%
789 \def\XINT_cmp_a 1#1!1#2!1#3!1#4!#5\W 1#6!1#7!1#8!1#9!%
790 {%
791 \xint_gob_til_sc #1\XINT_cmp_equal ;%
792 \ifnum #1>#6 \XINT_cmp_gt\fi
793 \ifnum #1<#6 \XINT_cmp_lt\fi
794 \xint_gob_til_sc #2\XINT_cmp_equal ;%
795 \ifnum #2>#7 \XINT_cmp_gt\fi
796 \ifnum #2<#7 \XINT_cmp_lt\fi
797 \xint_gob_til_sc #3\XINT_cmp_equal ;%
798 \ifnum #3>#8 \XINT_cmp_gt\fi
799 \ifnum #3<#8 \XINT_cmp_lt\fi
800 \xint_gob_til_sc #4\XINT_cmp_equal ;%
801 \ifnum #4>#9 \XINT_cmp_gt\fi
802 \ifnum #4<#9 \XINT_cmp_lt\fi
803 \XINT_cmp_a #5\W
804 }%
805 \def\XINT_cmp_lt#1{\def\XINT_cmp_lt\fi ##1\W ##2\W {\fi#1-1}}\XINT_cmp_lt{ }%
806 \def\XINT_cmp_gt#1{\def\XINT_cmp_gt\fi ##1\W ##2\W {\fi#11}}\XINT_cmp_gt{ }%
807 \def\XINT_cmp_equal #1\W #2\W { 0}%

```

3.35 \xintiiSub

Entirely rewritten for 1.2.

Refactored at 1.2l. I was initially aiming at clinching some internal format of the type `1<8digits>!...1<8digits>!` for chaining the arithmetic operations (as a preliminary step to de-

3 Package *xintcore* implementation

cided upon some internal format for `xintfrac` macros), thus I wanted to uniformize delimiters in particular and have some core macros inputting and outputting such formats. But the way division is implemented makes it currently very hard to obtain a satisfactory solution. For subtraction I got there almost, but there was added overhead and, as the core sub-routine still assumed the shorter number will be positioned first, one would need to record the length also in the basic internal format, or add the overhead to not make assumption on which one is shorter. I thus but back-tracked my steps but in passing I improved the efficiency (probably) in the worst case branch.

Sadly this 1.2l refactoring left an extra ! in macro `\XINT_sub_1_Ida`. This bug shows only in rare circumstances which escaped out test suite :(Fixed at 1.2q.

The other reason for backtracking was in relation with the decimal numbers. Having a core format in base 10^8 but ultimately the radix is actually 10 leads to complications. I could use radix 10^8 for `\xintiexpr` only, but then I need to make it compatible with sub-`\xintiexpr` in `\xintexpr`, etc... there are many issues of this type.

I considered also an approach like in the 1.2l `\xintiiCmp`, but decided to stick with the method here for now.

```
808 \def\xintiSub    {\romannumeral0\xintiisub }%
809 \def\xintiisub  #1{\expandafter\XINT_iisub\romannumeral`&&@#1\xint:}%
810 \def\XINT_iisub  #1#2\xint:#3%
811 {%
812   \expandafter\XINT_sub_nfork\expandafter
813   #1\romannumeral`&&@#3\xint:#2\xint:
814 }%
815 \def\xintiSub    {\romannumeral0\XINT_signaldeprecated{xintcore}{xintiSub}\xintisub }%
816 \def\xintisub    #1{\expandafter\XINT_isub\romannumeral0\xintnum{#1}\xint:}%
817 \def\XINT_isub   #1#2\xint:#3%
818 {%
819   \expandafter\XINT_sub_nfork\expandafter
820   #1\romannumeral0\xintnum{#3}\xint:#2\xint:
821 }%
822 \def\XINT_sub_nfork #1#2%
823 {%
824   \xint_UDzerofork
825   #1\XINT_sub_firstiszero
826   #2\XINT_sub_secondiszero
827   0{}%
828   \krof
829   \xint_UDsignsfork
830   #1#2\XINT_sub_minusminus
831   #1-\XINT_sub_minusplus
832   #2-\XINT_sub_plusminus
833   --\XINT_sub_plusplus
834   \krof #1#2%
835 }%
836 \def\XINT_sub_firstiszero #1\krof 0#2#3\xint:#4\xint:{\XINT_opp #2#3}%
837 \def\XINT_sub_secondiszero #1\krof #20#3\xint:#4\xint:{ #2#4}%
838 \def\XINT_sub_plusminus   #1#2{\XINT_add_pp_a #1{}}%
839 \def\XINT_sub_plusplus    #1#2%
840   {\expandafter\XINT_opp\romannumeral0\XINT_sub_mm_a #1#2}%
841 \def\XINT_sub_minusplus    #1#2%
842   {\expandafter-\romannumeral0\XINT_add_pp_a {}#2}%
843 \def\XINT_sub_minusminus  #1#2{\XINT_sub_mm_a {}{}}%
```

3 Package *xintcore* implementation

```

844 \def\XINT_sub_mm_a #1#2#3\xint:
845 {%
846   \expandafter\XINT_sub_mm_b
847     \romannumeral0\expandafter\XINT_sepandrev_andcount
848     \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\{10}0000001\W
849     #2#3\XINT_rsepbyviii_end_A 2345678%
850     \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
851     \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
852     \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
853   \X #1%
854 }%
855 \def\XINT_sub_mm_b #1\xint:#2\X #3\xint:
856 {%
857   \expandafter\XINT_sub_checklengths
858   \the\numexpr #1\expandafter\xint:%
859   \romannumeral0\expandafter\XINT_sepandrev_andcount
860   \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\{10}0000001\W
861   #3\XINT_rsepbyviii_end_A 2345678%
862   \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
863   \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
864   \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
865   1;!1;!1;!1;!1;\W
866   #21;!1;!1;!1;!1;\W
867   1;!1\R!1\R!1\R!1\R!%
868   1\R!1\R!1\R!1\R!\W
869 }%
870 \def\XINT_sub_checklengths #1\xint:#2\xint:%
871 {%
872   \ifnum #2>#1
873     \expandafter\XINT_sub_exchange
874   \else
875     \expandafter\XINT_sub_aa
876   \fi
877 }%
878 \def\XINT_sub_exchange #1\W #2\W
879 {%
880   \expandafter\XINT_opp\romannumeral0\XINT_sub_aa #2\W #1\W
881 }%
882 \def\XINT_sub_aa
883 {%
884   \expandafter\XINT_sub_out\the\numexpr\XINT_sub_a\xint_c_i
885 }%

```

The post-processing (clean-up of zeros, or rescue of situation with A-B where actually B turns out bigger than A) will be done by a macro which depends on circumstances and will be initially last token before the reversion done by `\XINT_unrevbyviii`.

```

886 \def\XINT_sub_out {\XINT_unrevbyviii{}}%

```

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

Call: `\the\numexpr`

```

\XINT_sub_a 1#11;!1;!1;!1;!1;\W
#21;!1;!1;!1;!1;\W

```

3 Package *xintcore* implementation

where #1 and #2 are blocks of $1 < 8d > !$, #1 (=B) *must* be at most as long as #2 (=A), (in radix 10^8) and the routine wants to compute $\#2 - \#1 = A - B$

1.2l uses $1; !$ delimiters to match those of addition (and multiplication). But in the end I reverted the code branch which made it possible to chain such operations keeping internal format in 8 digits blocks throughout.

\numexpr governed expansion stops with various possibilities:

- Type Ia: #1 shorter than #2, no final carry
- Type Ib: #1 shorter than #2, a final carry but next block of #2 > 1
- Type Ica: #1 shorter than #2, a final carry, next block of #2 is final and $= 1$
- Type Icb: as Ica except that 00000001 block from #2 was not final
- Type Id: #1 shorter than #2, a final carry, next block of #2 $= 0$
- Type IIa: #1 same length as #2, turns out it was $\leq \#2$.
- Type IIb: #1 same length as #2, but turned out $> \#2$.

Various type of post actions are then needed:

- Ia: clean up of zeros in most significant block of 8 digits
- Ib: as Ia
- Ic: there may be significant blocks of 8 zeros to clean up from result. Only case Ica may have arbitrarily many of them, case Icb has only one such block.
- Id: blocks of 99999999 may propagate and there might a be final zero block created which has to be cleaned up.
- IIa: arbitrarily many zeros might have to be removed.
- IIb: We wanted $\#2 - \#1 = -(\#1 - \#2)$, but we got $10^{\{8N\}} + \#2 - \#1 = 10^{\{8N\}} - (\#1 - \#2)$. We need to do the correction then we are as in IIa situation, except that final result can not be zero.

The 1.2l method for this correction is (presumably, testing takes lots of time, which I do not have) more efficient than in 1.2 release.

```
887 \def\XINT_sub_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
888 {%
889   \XINT_sub_b
890   #1!#6!#2!#7!#3!#8!#4!#9!%
891   #5\W
892 }%
```

As 1.2l code uses $1 < 8digits > !$ blocks one has to be careful with the carry digit 1 or 0: A #1#2#3 pattern would result into an empty #1 if the carry digit which is upfront is 1, rather than setting #1=1.

```
893 \def\XINT_sub_b #1#2#3#4!#5!%
894 {%
895   \xint_gob_til_sc #3\XINT_sub_bi ;%
896   \expandafter\XINT_sub_c\the\numexpr#1+1#5-#3#4-\xint_c_i\xint:%
897 }%
898 \def\XINT_sub_c 1#1#2\xint:%
899 {%
900   1#2\expandafter!\the\numexpr\XINT_sub_d #1%
901 }%
902 \def\XINT_sub_d #1#2#3#4!#5!%
903 {%
904   \xint_gob_til_sc #3\XINT_sub_di ;%
905   \expandafter\XINT_sub_e\the\numexpr#1+1#5-#3#4-\xint_c_i\xint:%
906 }%
907 \def\XINT_sub_e 1#1#2\xint:%
908 {%
```

3 Package *xintcore* implementation

```

909      1#2\expandafter!\the\numexpr\XINT_sub_f #1%
910 }%
911 \def\XINT_sub_f #1#2#3#4!#5!%
912 {%
913     \xint_gob_til_sc #3\XINT_sub_fi ;%
914     \expandafter\XINT_sub_g\the\numexpr#1+1#5-#3#4-\xint_c_i\xint:
915 }%
916 \def\XINT_sub_g 1#1#2\xint:%
917 {%
918     1#2\expandafter!\the\numexpr\XINT_sub_h #1%
919 }%
920 \def\XINT_sub_h #1#2#3#4!#5!%
921 {%
922     \xint_gob_til_sc #3\XINT_sub_hi ;%
923     \expandafter\XINT_sub_i\the\numexpr#1+1#5-#3#4-\xint_c_i\xint:
924 }%
925 \def\XINT_sub_i 1#1#2\xint:%
926 {%
927     1#2\expandafter!\the\numexpr\XINT_sub_a #1%
928 }%
929 \def\XINT_sub_bi;%
930     \expandafter\XINT_sub_c\the\numexpr#1+1#2-#3\xint:
931     #4!#5!#6!#7!#8!#9!\W
932 {%
933     \XINT_sub_k #1#2!#5!#7!#9!%
934 }%
935 \def\XINT_sub_di;%
936     \expandafter\XINT_sub_e\the\numexpr#1+1#2-#3\xint:
937     #4!#5!#6!#7!#8\W
938 {%
939     \XINT_sub_k #1#2!#5!#7!%
940 }%
941 \def\XINT_sub_fi;%
942     \expandafter\XINT_sub_g\the\numexpr#1+1#2-#3\xint:
943     #4!#5!#6\W
944 {%
945     \XINT_sub_k #1#2!#5!%
946 }%
947 \def\XINT_sub_hi;%
948     \expandafter\XINT_sub_i\the\numexpr#1+1#2-#3\xint:
949     #4\W
950 {%
951     \XINT_sub_k #1#2!%
952 }%

```

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 possibly is just $1 < 00000001 > !$, we will have those eight zeros to clean up.

If A and B have the same length (in base 10^8) then arbitrarily many zeros might have to be cleaned up, and if $A < B$, the whole result will have to be complemented first.

3 Package *xintcore* implementation

```

953 \def\XINT_sub_k #1#2#3%
954 {%
955   \xint_gob_til_sc #3\XINT_sub_p;\XINT_sub_l #1#2#3%
956 }%
957 \def\XINT_sub_l #1%
958   {\xint_UDzerofork #1\XINT_sub_l_carry 0\XINT_sub_l_Ia\krof}%
959 \def\XINT_sub_l_Ia 1#1;!#2\W{1\relax#1;!1\XINT_sub_fix_none!}%

960 \def\XINT_sub_l_carry 1#1!{\ifcase #1
961   \expandafter \XINT_sub_l_Id
962   \or \expandafter \XINT_sub_l_Ic
963   \else\expandafter \XINT_sub_l_Ib\fi 1#1!}%
964 \def\XINT_sub_l_Ib #1;!#2\W {-\xint_c_i+#1;!1\XINT_sub_fix_none!}%
965 \def\XINT_sub_l_Ic 1#1!1#2#3!#4;!#5\W
966 {%
967   \xint_gob_til_sc #2\XINT_sub_l_Ica;%
968   1\relax 00000000!1#2#3!#4;!1\XINT_sub_fix_none!%
969 }%

```

We need to add some extra delimiters at the end for post-action by `\XINT_num`, so we first grab the material up to `\W`

```

970 \def\XINT_sub_l_Ica#1\W
971 {%
972   1;!1\XINT_sub_fix_cuz!%
973   1;!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\W
974   \xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z
975 }%
976 \def\XINT_sub_l_Id 1#1!%
977   {199999999\expandafter!\the\numexpr \XINT_sub_l_Id_a}%
978 \def\XINT_sub_l_Id_a 1#1!{\ifcase #1
979   \expandafter \XINT_sub_l_Id
980   \or \expandafter \XINT_sub_l_Id_b
981   \else\expandafter \XINT_sub_l_Ib\fi 1#1!}%
982 \def\XINT_sub_l_Id_b 1#1!1#2#3!#4;!#5\W
983 {%
984   \xint_gob_til_sc #2\XINT_sub_l_Ida;%
985   1\relax 00000000!1#2#3!#4;!1\XINT_sub_fix_none!%
986 }%
987 \def\XINT_sub_l_Ida#1\XINT_sub_fix_none{1;!1\XINT_sub_fix_none}%

```

This is the case where both operands have same 10^8 -base length.

We were handling $A < B$ but perhaps $B > A$. The situation with $A = B$ is also annoying because we then have to clean up all zeros but don't know where to stop (if $A > B$ the first non-zero 8 digits block would tell use when).

Here again we need to grab `#3\W` to position the actually used terminating delimiters.

```

988 \def\XINT_sub_p;\XINT_sub_l #1#2\W #3\W
989 {%
990   \xint_UDzerofork
991     #1{1;!1\XINT_sub_fix_neg!%
992     1;!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\W

```

3 Package *xintcore* implementation

```
993      \xint_bye2345678\xint_bye1099999988\relax}% A - B, B > A
994      0{1;!1\XINT_sub_fix_cuz!%
995      1;!1\R!1\R!1\R!1\R!1\R!1\R!1\W}%
996      \krof
997      \xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z
998 }%
```

Routines for post-processing after reversal, and removal of separators. It is a matter of cleaning up zeros, and possibly in the bad case to take a complement before that.

```
999 \def\XINT_sub_fix_none;{\XINT_cuz_small}%
1000 \def\XINT_sub_fix_cuz ;{\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop}%
```

Case with A and B same number of digits in base 10^8 and $B > A$.

1.2l subtle chaining on the model of the 1.2i rewrite of `\xintInc` and similar routines. After taking complement, leading zeroes need to be cleaned up as in $B \leq A$ branch.

```
1001 \def\XINT_sub_fix_neg;%
1002 {%
1003     \expandafter-\romannumeral0\expandafter
1004     \XINT_sub_comp_finish\the\numexpr\XINT_sub_comp_loop
1005 }%
1006 \def\XINT_sub_comp_finish 0{\XINT_sub_fix_cuz;}%
1007 \def\XINT_sub_comp_loop #1#2#3#4#5#6#7#8%
1008 {%
1009     \expandafter\XINT_sub_comp_clean
1010     \the\numexpr \xint_c_xi_e_viii_mone-#1#2#3#4#5#6#7#8\XINT_sub_comp_loop
1011 }%
```

#1 = 0 signifie une retenue, #1 = 1 pas de retenue, ce qui ne peut arriver que tant qu'il n'y a que des zéros du côté non significatif. Lorsqu'on est revenu au début on a forcément une retenue.

```
1012 \def\XINT_sub_comp_clean 1#1{+ #1\relax}%
```

3.36 `\xintiiMul`

Completely rewritten for 1.2.

1.2l: `\xintiiMul` made robust against non terminated input.

```
1013 \def\xintiMul {\romannumeral0\XINT_signaldeprecated{xintcore}{xintiMul}\xintimul }%
1014 \def\xintimul #1%
1015 {%
1016     \expandafter\XINT_imul\romannumeral0\xintnum{#1}\xint:
1017 }%
1018 \def\XINT_imul #1#2\xint:#3%
1019 {%
1020     \expandafter\XINT_mul_nfork\expandafter #1\romannumeral0\xintnum{#3}\xint:#2\xint:
1021 }%
1022 \def\xintiiMul {\romannumeral0\xintiimul }%
1023 \def\xintiimul #1%
1024 {%
1025     \expandafter\XINT_iimul\romannumeral`&&@#1\xint:
1026 }%
1027 \def\XINT_iimul #1#2\xint:#3%
```


3 Package *xintcore* implementation

```

1028 {%
1029   \expandafter\XINT_mul_nfork\expandafter #1\romannumeral`&&@#3\xint:#2\xint:
1030 }%

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

1031 \def\XINT_mul_fork #1#2\xint:#3\xint:{\XINT_mul_nfork #1#3\xint:#2\xint:}%
1032 \def\XINT_mul_nfork #1#2%
1033 {%
1034   \xint_UDzerofork
1035   #1\XINT_mul_zero
1036   #2\XINT_mul_zero
1037   0{}%
1038   \krof
1039   \xint_UDsignsfork
1040   #1#2\XINT_mul_minusminus
1041   #1-\XINT_mul_minusplus
1042   #2-\XINT_mul_plusminus
1043   --\XINT_mul_plusplus
1044   \krof #1#2%
1045 }%
1046 \def\XINT_mul_zero #1\krof #2#3\xint:#4\xint:{ 0}%
1047 \def\XINT_mul_minusminus #1#2{\XINT_mul_plusplus {}}}%
1048 \def\XINT_mul_minusplus #1#2%
1049   {\expandafter-\romannumeral0\XINT_mul_plusplus {}}#2}%
1050 \def\XINT_mul_plusminus #1#2%
1051   {\expandafter-\romannumeral0\XINT_mul_plusplus #1{}}}%
1052 \def\XINT_mul_plusplus #1#2#3\xint:
1053 {%
1054   \expandafter\XINT_mul_pre_b
1055   \romannumeral0\expandafter\XINT_sepandrev_andcount
1056   \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
1057   #2#3\XINT_rsepbyviii_end_A 2345678%
1058   \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
1059   \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
1060   \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
1061   \W #1%
1062 }%
1063 \def\XINT_mul_pre_b #1\xint:#2\W #3\xint:
1064 {%
1065   \expandafter\XINT_mul_checklengths
1066   \the\numexpr #1\expandafter\xint:%
1067   \romannumeral0\expandafter\XINT_sepandrev_andcount
1068   \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
1069   #3\XINT_rsepbyviii_end_A 2345678%
1070   \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
1071   \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
1072   \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
1073   1;\!W #21;!\%
1074   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1075 }%

```

Cooking recipe, 2015/10/05.

3 Package *xintcore* implementation

```

1076 \def\XINT_mul_checklengths #1\xint:#2\xint:%
1077 {%
1078   \ifnum #2=\xint_c_i\expandafter\XINT_mul_smallbyfirst\fi
1079   \ifnum #1=\xint_c_i\expandafter\XINT_mul_smallbysecond\fi
1080   \ifnum #2<#1
1081     \ifnum \numexpr (#2-\xint_c_i)*(#1-#2)<383
1082       \XINT_mul_exchange
1083     \fi
1084   \else
1085     \ifnum \numexpr (#1-\xint_c_i)*(#2-#1)>383
1086       \XINT_mul_exchange
1087     \fi
1088   \fi
1089   \XINT_mul_start
1090 }%
1091 \def\XINT_mul_smallbyfirst #1\XINT_mul_start 1#2!1;!\W
1092 {%
1093   \ifnum#2=\xint_c_i\expandafter\XINT_mul_oneisone\fi
1094   \ifnum#2<\xint_c_xxii\expandafter\XINT_mul_verysmall\fi
1095   \expandafter\XINT_mul_out\the\numexpr\XINT_smallmul 1#2!%
1096 }%
1097 \def\XINT_mul_smallbysecond #1\XINT_mul_start #2\W 1#3!1;!\%
1098 {%
1099   \ifnum#3=\xint_c_i\expandafter\XINT_mul_oneisone\fi
1100   \ifnum#3<\xint_c_xxii\expandafter\XINT_mul_verysmall\fi
1101   \expandafter\XINT_mul_out\the\numexpr\XINT_smallmul 1#3!#2%
1102 }%
1103 \def\XINT_mul_oneisone #1!\XINT_mul_out }%
1104 \def\XINT_mul_verysmall\expandafter\XINT_mul_out
1105   \the\numexpr\XINT_smallmul 1#1!%
1106   {\expandafter\XINT_mul_out\the\numexpr\XINT_verysmallmul 0\xint:#1!}%
1107 \def\XINT_mul_exchange #1\XINT_mul_start #2\W #3!;!\%
1108   {\fi\fi\XINT_mul_start #3!;!\W #2}%

1109 \def\XINT_mul_start
1110   {\expandafter\XINT_mul_out\the\numexpr\XINT_mul_loop 100000000!1;!\W}%
1111 \def\XINT_mul_out
1112   {\expandafter\XINT_cuz_small\romannumeral0\XINT_unrevbyviii {}}%

```

Call:

\the\numexpr \XINT_mul_loop 100000000!1;!\W #1;!\W #2;!

where #1 and #2 are (globally reversed) blocks 1<8d>!. Its is generally more efficient if #1 is the shorter one, but a better recipe is implemented in \XINT_mul_checklengths. 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;!. The last 8-digits block is significant one. It can not be 100000000! except if 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.

```

1113 \def\XINT_mul_loop #1\W #2\W 1#3!%

```

3 Package *xintcore* implementation

```

1114 {%
1115   \xint_gob_til_sc #3\XINT_mul_e ;%
1116   \expandafter\XINT_mul_a\the\numexpr \XINT_smallmul 1#3!#2\W
1117   #1\W #2\W
1118 }%

```

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

```

1119 \def\XINT_mul_a #1\W #2\W
1120 {%
1121   \expandafter\XINT_mul_b\the\numexpr
1122   \XINT_add_a \xint_c_ii #21;!1;!1;!1\W #11;!1;!1;!1\W\W
1123 }%
1124 \def\XINT_mul_b 1#1!\{1#1\expandafter!\the\numexpr\XINT_mul_loop }%
1125 \def\XINT_mul_e;#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;!

Multiplication by zero will produce blocks of zeros.

```

1126 \def\XINT_minimulwc_a 1#1\xint:#2\xint:#3!#4#5#6#7#8\xint:%
1127 {%
1128   \expandafter\XINT_minimulwc_b
1129   \the\numexpr \xint_c_x^ix+#1+#3*#8\xint:
1130               #3*#4#5#6#7+#2*#8\xint:
1131               #2*#4#5#6#7\xint:%
1132 }%
1133 \def\XINT_minimulwc_b 1#1#2#3#4#5#6\xint:#7\xint:%
1134 {%
1135   \expandafter\XINT_minimulwc_c
1136   \the\numexpr \xint_c_x^ix+#1#2#3#4#5+#7\xint:#6\xint:%
1137 }%
1138 \def\XINT_minimulwc_c 1#1#2#3#4#5#6\xint:#7\xint:#8\xint:%
1139 {%
1140   1#6#7\expandafter!%
1141   \the\numexpr\expandafter\XINT_smallmul_a
1142   \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8\xint:%
1143 }%
1144 \def\XINT_smallmul 1#1#2#3#4#5!\{\XINT_smallmul_a 100000000\xint:#1#2#3#4\xint:#5!}%
1145 \def\XINT_smallmul_a #1\xint:#2\xint:#3!#4!%
1146 {%
1147   \xint_gob_til_sc #4\XINT_smallmul_e;%
1148   \XINT_minimulwc_a #1\xint:#2\xint:#3!#4\xint:#2\xint:#3!%
1149 }%
1150 \def\XINT_smallmul_e;\XINT_minimulwc_a 1#1\xint:#2;#3!%
1151   {\xint_gob_til_eightzeroes #1\XINT_smallmul_f 000000001\relax #1!1;!}%
1152 \def\XINT_smallmul_f 000000001\relax 00000000!1{1\relax}%

```

```

1153 \def\XINT_verysmallmul #1\xint:#2!1#3!%
1154 {%
1155     \xint_gob_til_sc #3\XINT_verysmallmul_e;%
1156     \expandafter\XINT_verysmallmul_a
1157     \the\numexpr #2*#3+#1\xint:#2!%
1158 }%
1159 \def\XINT_verysmallmul_e;\expandafter\XINT_verysmallmul_a\the\numexpr
1160     #1+#2#3\xint:#4!%
1161 {\xint_gob_til_zero #2\XINT_verysmallmul_f 0\xint_c_x^viii+#2#3!1;!}%
1162 \def\XINT_verysmallmul_f #1!1{1\relax}%
1163 \def\XINT_verysmallmul_a #1#2\xint:%
1164 {%
1165     \unless\ifnum #1#2<\xint_c_x^ix
1166     \expandafter\XINT_verysmallmul_bi\else
1167     \expandafter\XINT_verysmallmul_bj\fi
1168     \the\numexpr \xint_c_x^ix+#1#2\xint:%
1169 }%
1170 \def\XINT_verysmallmul_bj{\expandafter\XINT_verysmallmul_cj }%
1171 \def\XINT_verysmallmul_cj 1#1#2\xint:%
1172     {1#2\expandafter!\the\numexpr\XINT_verysmallmul #1\xint:}%
1173 \def\XINT_verysmallmul_bi\the\numexpr\xint_c_x^ix+#1#2#3\xint:%
1174     {1#3\expandafter!\the\numexpr\XINT_verysmallmul #1#2\xint:}%

```

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.

```

1175 \def\XINT_minimul_a #1\xint:#2!#3#4#5#6#7!%
1176 {%
1177     \expandafter\XINT_minimul_b
1178     \the\numexpr \xint_c_x^viii+#2*#7\xint:#2*#3#4#5#6+#1*#7\xint:#1*#3#4#5#6\xint:%
1179 }%
1180 \def\XINT_minimul_b 1#1#2#3#4#5\xint:#6\xint:%
1181 {%
1182     \expandafter\XINT_minimul_c
1183     \the\numexpr \xint_c_x^ix+#1#2#3#4+#6\xint:#5\xint:%
1184 }%
1185 \def\XINT_minimul_c 1#1#2#3#4#5#6\xint:#7\xint:#8\xint:%
1186 {%
1187     1#6#7\expandafter!\the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8!%
1188 }%

```

3.37 \xintiiDivision

Completely rewritten for 1.2.

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

3 Package *xintcore* implementation

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.

1.21: `\xintiDivision` et al. made robust against non terminated input.

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

```
1189 \def\xintiDivision {\romannumeral0\XINT_signaldeprecated{xintcore}{xintiDivision}\xintidivision }%
1190 \def\xintidivision #1{\expandafter\XINT_idivision\romannumeral0\xintnum{#1}\xint:}%
1191 \def\XINT_idivision #1#2\xint:#3{\expandafter\XINT_iidivision_a\expandafter #1%
1192 \romannumeral0\xintnum{#3}\xint:#2\xint:}%
1193 \def\xintiiDivision {\romannumeral0\xintiidivision }%
1194 \def\xintiidivision #1{\expandafter\XINT_iidivision \romannumeral`&&@#1\xint:}%
1195 \def\XINT_iidivision #1#2\xint:#3{\expandafter\XINT_iidivision_a\expandafter #1%
1196 \romannumeral`&&@#3\xint:#2\xint:}%

```

On regarde les signes de A et de B.

```
1197 \def\XINT_iidivision_a #1#2% #1 de A, #2 de B.
1198 {%
1199 \if0#2\xint_dothis{\XINT_iidivision_divbyzero #1#2}\fi
1200 \if0#1\xint_dothis\XINT_iidivision_aiszero\fi
1201 \if-#2\xint_dothis{\expandafter\XINT_iidivision_bneg
1202 \romannumeral0\XINT_iidivision_bpos #1}\fi
1203 \xint_orthat{\XINT_iidivision_bpos #1#2}%
1204 }%
1205 \def\XINT_iidivision_divbyzero#1#2#3\xint:#4\xint:
1206 {\if0#1\xint_dothis{\XINT_signalcondition{DivisionUndefined}}\fi
1207 \xint_orthat{\XINT_signalcondition{DivisionByZero}}%
1208 {Division of #1#4 by #2#3}{\{0\}{0}}}%
1209 \def\XINT_iidivision_aiszero #1\xint:#2\xint:{\{0\}{0}}}%
1210 \def\XINT_iidivision_bneg #1% q->-q, r unchanged
1211 {\expandafter{\romannumeral0\XINT_opp #1}}%
1212 \def\XINT_iidivision_bpos #1%
1213 {%
1214 \xint_UDsignfork
1215 #1\XINT_iidivision_aneg
1216 -{\XINT_iidivision_apos #1}%
1217 \krof
1218 }%

```

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.

```
1219 \def\XINT_iidivision_apos #1#2\xint:#3\xint:{\XINT_div_prepare {#2}{#1#3}}%
1220 \def\XINT_iidivision_aneg #1\xint:#2\xint:
1221 {\expandafter
1222 \XINT_iidivision_aneg_b\romannumeral0\XINT_div_prepare {#1}{#2}{#1}}%

```

3 Package *xintcore* implementation

```

1223 \def\XINT_iidivision_aneg_b #1#2{\if0\XINT_Sgn #2\xint:
1224         \expandafter\XINT_iidivision_aneg_rzero
1225     \else
1226         \expandafter\XINT_iidivision_aneg_rpos
1227     \fi {#1}{#2}}%
1228 \def\XINT_iidivision_aneg_rzero #1#2#3{{-#1}{0}}% necessarily q was >0
1229 \def\XINT_iidivision_aneg_rpos #1%
1230 {%
1231     \expandafter\XINT_iidivision_aneg_end\expandafter
1232     {\expandafter-\romannumeral0\xintinc {#1}}% q-> -(1+q)
1233 }%
1234 \def\XINT_iidivision_aneg_end #1#2#3%
1235 {%
1236     \expandafter\xint_exchangetwo_keepbraces
1237     \expandafter{\romannumeral0\XINT_sub_mm_a {}}{#3\xint:#2\xint:}{#1}% r-> b-r
1238 }%

```

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.

```

1239 \def\XINT_div_prepare #1%
1240 {%
1241     \XINT_div_prepare_a #1\R\R\R\R\R\R\R {10}0000001\W !{#1}%
1242 }%
1243 \def\XINT_div_prepare_a #1#2#3#4#5#6#7#8#9%
1244 {%
1245     \xint_gob_til_R #9\XINT_div_prepare_small\R
1246     \XINT_div_prepare_b #9%
1247 }%

```

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).

```

1248 \def\XINT_div_prepare_small\R #1!#2%
1249 {%
1250     \ifcase #2
1251     \or\expandafter\XINT_div_BisOne
1252     \or\expandafter\XINT_div_BisTwo
1253     \else\expandafter\XINT_div_small_a
1254     \fi {#2}%
1255 }%
1256 \def\XINT_div_BisOne #1#2{{#2}{0}}%
1257 \def\XINT_div_BisTwo #1#2%
1258 {%
1259     \expandafter\expandafter\expandafter\XINT_div_BisTwo_a
1260     \ifodd\xintLDg{#2} \expandafter1\else \expandafter0\fi {#2}%
1261 }%
1262 \def\XINT_div_BisTwo_a #1#2%
1263 {%
1264     \expandafter{\romannumeral0\XINT_half
1265     #2\xint_bye\xint_Bye345678\xint_bye
1266     *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax}{#1}%
1267 }%

```

3 Package *xintcore* implementation

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.

```
1268 \def\XINT_div_small_a #1#2%
1269 {%
1270   \expandafter\XINT_div_small_b
1271   \the\numexpr #1/\xint_c_ii\expandafter
1272   \xint:\the\numexpr \xint_c_x^viii+#1\expandafter!%
1273   \romannumeral0%
1274   \XINT_div_small_ba #2\R\R\R\R\R\R\R\R{10}0000001\W
1275   #2\XINT_sepbyviii_Z_end 2345678\relax
1276 }%
```

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

```
1277 \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 $A < 10^8$.

```
1278 \def\XINT_div_small_ba #1#2#3#4#5#6#7#8#9%
1279 {%
1280   \xint_gob_til_R #9\XINT_div_smallsmall\R
1281   \expandafter\XINT_div_dosmallldiv
1282   \the\numexpr\expandafter\XINT_sepbyviii_Z
1283   \romannumeral0\XINT_zeroes_forviii
1284   #1#2#3#4#5#6#7#8#9%
1285 }%
```

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}.

```
1286 \def\XINT_div_smallsmall\R
1287   \expandafter\XINT_div_dosmallldiv
1288   \the\numexpr\expandafter\XINT_sepbyviii_Z
1289   \romannumeral0\XINT_zeroes_forviii #1\R #2\relax
1290   {{\XINT_div_dosmallsmall}}{#1}}%
1291 \def\XINT_div_dosmallsmall #1\xint:1#2!#3%
1292 {%
1293   \expandafter\XINT_div_smallsmallend
1294   \the\numexpr (#3+#1)/#2-\xint_c_i\xint:#2\xint:#3\xint:%
1295 }%
1296 \def\XINT_div_smallsmallend #1\xint:#2\xint:#3\xint:{\expandafter
1297   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #3-#1*#2}}%
```

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

```
1298 \def\XINT_div_dosmallldiv
1299   {{\expandafter\XINT_sdiv_out\the\numexpr\XINT_smalldivx_a}}%
```

3 Package *xintcore* implementation

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

```

1300 \def\XINT_div_prepare_b
1301   {\expandafter\XINT_div_prepare_c\romannumeral0\XINT_zeroes_forviii }%
1302 \def\XINT_div_prepare_c #1!%
1303 {%
1304   \XINT_div_prepare_d #1.00000000!{#1}%
1305 }%
1306 \def\XINT_div_prepare_d #1#2#3#4#5#6#7#8#9%
1307 {%
1308   \expandafter\XINT_div_prepare_e\xint_gob_til_dot #1#2#3#4#5#6#7#8#9!%
1309 }%
1310 \def\XINT_div_prepare_e #1!#2!#3#4%
1311 {%
1312   \XINT_div_prepare_f #4#3\X {#1}{#3}%
1313 }%

```

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.

```

1314 \def\XINT_div_prepare_f #1#2#3#4#5#6#7#8#9\X
1315 {%
1316   \expandafter\XINT_div_prepare_g
1317   \the\numexpr #1#2#3#4#5#6#7#8+\xint_c_i\expandafter
1318   \xint:\the\numexpr (#1#2#3#4#5#6#7#8+\xint_c_i)/\xint_c_ii\expandafter
1319   \xint:\the\numexpr #1#2#3#4#5#6#7#8\expandafter
1320   \xint:\romannumeral0\XINT_sepandrev_andcount
1321   #1#2#3#4#5#6#7#8#9\XINT_rsepbyviii_end_A 2345678%
1322   \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
1323   \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
1324   \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_W
1325   \X
1326 }%
1327 \def\XINT_div_prepare_g #1\xint:#2\xint:#3\xint:#4\xint:#5\X #6#7#8%
1328 {%
1329   \expandafter\XINT_div_prepare_h
1330   \the\numexpr\expandafter\XINT_sepbyviii_andcount
1331   \romannumeral0\XINT_zeroes_forviii #8#7\R\R\R\R\R\R\R\{10}0000001\W
1332   #8#7\XINT_sepbyviii_end 2345678\relax
1333   \xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!%
1334   \xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_W
1335   {#1}{#2}{#3}{#4}{#5}{#6}%
1336 }%
1337 \def\XINT_div_prepare_h #11\xint:#2\xint:#3#4#5#6#7#8%
1338 {%
1339   \XINT_div_start_a {#2}{#6}{#1}{#3}{#4}{#5}{#7}{#8}%
1340 }%

```

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.

3 Package *xintcore* implementation

Attention que ce n'est pas ici qu'on boucle mais en `\XINT_div_I_a`.

```

1341 \def\XINT_div_start_a #1#2%
1342 {%
1343   \ifnum #1 < #2
1344     \expandafter\XINT_div_zeroQ
1345   \else
1346     \expandafter\XINT_div_start_b
1347   \fi
1348   {#1}{#2}%
1349 }%
1350 \def\XINT_div_zeroQ #1#2#3#4#5#6#7%
1351 {%
1352   \expandafter\XINT_div_zeroQ_end
1353   \romannumeral0\XINT_unsep_cuzsmall
1354   #3\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax\xint:
1355 }%
1356 \def\XINT_div_zeroQ_end #1\xint:#2%
1357   {\expandafter{\expandafter0\expandafter}\XINT_div_cleanR #1#2\xint:}%

```

$L, K, A, x', y, x, B, \langle C \rangle \rightarrow K.A.x\{LK\{x'y\}x\}B\langle C \rangle$

```

1358 \def\XINT_div_start_b #1#2#3#4#5#6%
1359 {%
1360   \expandafter\XINT_div_finish\the\numexpr
1361   \XINT_div_start_c {#2}\xint:#3\xint:{#6}{#{1}{#2}{#{4}{#5}}{#6}}%
1362 }%
1363 \def\XINT_div_finish
1364 {%
1365   \expandafter\XINT_div_finish_a \romannumeral`&&\XINT_div_unsepQ
1366 }%
1367 \def\XINT_div_finish_a #1\Z #2\xint:{\XINT_div_finish_b #2\xint:{#1}}%

```

Ici ce sont routines de fin. Le reste déjà nettoyé. $R.Q\langle C \rangle$.

```

1368 \def\XINT_div_finish_b #1%
1369 {%
1370   \if0#1%
1371     \expandafter\XINT_div_finish_bRzero
1372   \else
1373     \expandafter\XINT_div_finish_bRpos
1374   \fi
1375   #1%
1376 }%
1377 \def\XINT_div_finish_bRzero 0\xint:#1#2{#{1}{0}}%
1378 \def\XINT_div_finish_bRpos #1\xint:#2#3%
1379 {%
1380   \expandafter\xint_exchangetwo_keepbraces\XINT_div_cleanR #1#3\xint:{#2}%
1381 }%
1382 \def\XINT_div_cleanR #1000000000\xint:{#{1}}%

```

$K\alpha.A.x\{LK\{x'y\}x\}, B, \langle C \rangle$, 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 α .

3 Package *xintcore* implementation

```

1383 \def\XINT_div_start_c #1%
1384 {%
1385   \ifnum #1>\xint_c_vi
1386     \expandafter\XINT_div_start_ca
1387   \else
1388     \expandafter\XINT_div_start_cb
1389   \fi {#1}%
1390 }%
1391 \def\XINT_div_start_ca #1#2\xint:#3!#4!#5!#6!#7!#8!#9!%
1392 {%
1393   \expandafter\XINT_div_start_c\expandafter
1394   {\the\numexpr #1-\xint_c_vii}\xint:#2#3!#4!#5!#6!#7!#8!#9!\xint:%
1395 }%
1396 \def\XINT_div_start_cb #1%
1397   {\csname XINT_div_start_c_\romannumeral\xint:#1\endcsname}%
1398 \def\XINT_div_start_c_i #1\xint:#2!%
1399   {\XINT_div_start_c_ #1#2!\xint:}%
1400 \def\XINT_div_start_c_ii #1\xint:#2!#3!%
1401   {\XINT_div_start_c_ #1#2!#3!\xint:}%
1402 \def\XINT_div_start_c_iii #1\xint:#2!#3!#4!%
1403   {\XINT_div_start_c_ #1#2!#3!#4!\xint:}%
1404 \def\XINT_div_start_c_iv #1\xint:#2!#3!#4!#5!%
1405   {\XINT_div_start_c_ #1#2!#3!#4!#5!\xint:}%
1406 \def\XINT_div_start_c_v #1\xint:#2!#3!#4!#5!#6!%
1407   {\XINT_div_start_c_ #1#2!#3!#4!#5!#6!\xint:}%
1408 \def\XINT_div_start_c_vi #1\xint:#2!#3!#4!#5!#6!#7!%
1409   {\XINT_div_start_c_ #1#2!#3!#4!#5!#6!#7!\xint:}%

```

#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».

```

1410 \def\XINT_div_start_c_ #1!#2\xint:#3\xint:#4#5#6%
1411 {%
1412   \XINT_div_I_a {#1}{#4}{1#1!#2}{#6}{00000000}#5{#3}{#6}%
1413 }%

```

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

```

1414 \def\XINT_div_I_a #1#2%
1415 {%
1416   \expandafter\XINT_div_I_b\the\numexpr #1/#2\xint:{#1}{#2}%
1417 }%
1418 \def\XINT_div_I_b #1%
1419 {%
1420   \xint_gob_til_zero #1\XINT_div_I_czero 0\XINT_div_I_c #1%
1421 }%

```

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».

```

1422 \def\XINT_div_I_czero 0\XINT_div_I_c 0\xint:#1#2#3#4#5{1#5\XINT_div_I_g {#3}}%
1423 \def\XINT_div_I_c #1\xint:#2#3%
1424 {%

```

3 Package *xintcore* implementation

```
1425 \expandafter\XINT_div_I_da\the\numexpr #2-#1*#3\xint:#1\xint:{#2}{#3}%
1426 }%
```

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

```
1427 \def\XINT_div_I_da #1\xint:%
1428 {%
1429 \ifnum #1>\xint_c_ix
1430 \expandafter\XINT_div_I_dP
1431 \else
1432 \ifnum #1<\xint_c_
1433 \expandafter\expandafter\expandafter\XINT_div_I_dN
1434 \else
1435 \expandafter\expandafter\expandafter\XINT_div_I_db
1436 \fi
1437 \fi
1438 }%
```

attention très mauvaises notations avec _b et _db.

```
1439 \def\XINT_div_I_dN #1\xint:%
1440 {%
1441 \expandafter\XINT_div_I_b\the\numexpr #1-\xint_c_i\xint:%
1442 }%
1443 \def\XINT_div_I_db #1\xint:#2#3#4#5%
1444 {%
1445 \expandafter\XINT_div_I_dc\expandafter #1%
1446 \romannumeral0\expandafter\XINT_div_sub\expandafter
1447 {\romannumeral0\XINT_rev_nounsep }#4\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1448 {\the\numexpr\XINT_div_verysmallmul #1!#51;!}%
1449 \Z {#4}{#5}%
1450 }%
```

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

```
1451 \def\XINT_div_I_dc #1#2%
1452 {%
1453 \if-#2\expandafter\XINT_div_I_dd\else\expandafter\XINT_div_I_de\fi
1454 #1#2%
1455 }%
1456 \def\XINT_div_I_dd #1-\Z
1457 {%
1458 \if #11\expandafter\XINT_div_I_dz\fi
1459 \expandafter\XINT_div_I_dP\the\numexpr #1-\xint_c_i\xint: XX%
1460 }%
1461 \def\XINT_div_I_dz #1XX#2#3#4%
1462 {%
1463 1#4\XINT_div_I_g {#2}%
1464 }%
1465 \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»

3 Package *xintcore* implementation

```

1466 \def\XINT_div_I_dP #1\xint:#2#3#4#5#6%
1467 {%
1468     1#6+#1\expandafter\XINT_div_I_g\expandafter
1469     {\romannumeral0\expandafter\XINT_div_sub\expandafter
1470     {\romannumeral0\XINT_rev_nounsep {}}#4\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1471     {\the\numexpr\XINT_div_verysmallmul #1!#51;!}%
1472 }%
1473 }%

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»

1474 \def\XINT_div_I_g #1#2#3#4#5#6#7%
1475 {%
1476     \expandafter !\the\numexpr
1477     \ifnum#2=#3
1478         \expandafter\XINT_div_exittofinish
1479     \else
1480         \expandafter\XINT_div_I_h
1481     \fi
1482     {#4}#1\xint:#6\xint:{{#4}{#5}{#3}{#2}}{#7}%
1483 }%

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

1484 \def\XINT_div_exittofinish #1#2\xint:#3\xint:#4#5%
1485 {%
1486     1\expandafter\expandafter\expandafter!\expandafter\XINT_div_unsepQ_delim
1487     \romannumeral0\XINT_div_unsepR #2#3%
1488     \xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax\R\xint:
1489 }%

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

1490 \def\XINT_div_I_h #1\xint:#2!#3\xint:#4#5%
1491 {%
1492     \XINT_div_II_b #1#2!\xint:{{#5}{#4}{#3}}{#5}%
1493 }%

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

1494 \def\XINT_div_II_b #1#2!#3!%
1495 {%
1496     \xint_gob_til_eightzeroes #2\XINT_div_II_skipc 00000000%
1497     \XINT_div_II_c #1{1#2}{#3}%
1498 }%

```

3 Package *xintcore* implementation

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

```
1499 \def\XINT_div_II_skipc 00000000\XINT_div_II_c #1#2#3#4#5\xint:#6#7%
1500 {%
1501     \XINT_div_II_k #7{#4!#5}{#6}{00000000}%
1502 }%
```

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

```
1503 \def\XINT_div_II_c #1#2#3#4%
1504 {%
1505     \expandafter\XINT_div_II_d\the\numexpr\XINT_div_xmini
1506     #1\xint:#2!#3!#4!{#1}{#2}{#3!#4!}%
1507 }%
1508 \def\XINT_div_xmini #1%
1509 {%
1510     \xint_gob_til_one #1\XINT_div_xmini_a 1\XINT_div_mini #1%
1511 }%
1512 \def\XINT_div_xmini_a 1\XINT_div_mini 1#1%
1513 {%
1514     \xint_gob_til_zero #1\XINT_div_xmini_b 0\XINT_div_mini 1#1%
1515 }%
1516 \def\XINT_div_xmini_b 0\XINT_div_mini 10#1#2#3#4#5#6#7%
1517 {%
1518     \xint_gob_til_zero #7\XINT_div_xmini_c 0\XINT_div_mini 10#1#2#3#4#5#6#7%
1519 }%
```

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

```
1520 \def\XINT_div_xmini_c 0\XINT_div_mini 100000000\xint:50000000!#1!#2!{#1!}%
```

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

```
1521 \def\XINT_div_II_d 1#1#2#3#4#5!#6#7#8\xint:#9%
1522 {%
1523     \expandafter\XINT_div_II_e
1524     \romannumeral0\expandafter\XINT_div_sub\expandafter
1525     {\romannumeral0\XINT_rev_nounsep {#8\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1526     {\the\numexpr\XINT_div_smallmul_a 100000000\xint:#1#2#3#4\xint:#5!#91;!}%
1527     \xint:{#6}{#7}{#9}{#1#2#3#4#5}%
1528 }%
```

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

```
1529 \def\XINT_div_II_e 1#1!%
1530 {%
1531     \xint_gob_til_eightzeroes #1\XINT_div_II_skipf 00000000%
1532     \XINT_div_II_f 1#1!%
1533 }%
```

3 Package *xintcore* implementation

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»

```
1534 \def\XINT_div_II_skipf 00000000\XINT_div_II_f 100000000!#1\xint:#2#3#4#5#6%
```

```
1535 {%
```

```
1536 \XINT_div_II_k #6{#1}{#4}{#5}%
```

```
1537 }%
```

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.

```
1538 \def\XINT_div_II_f #1!#2!#3\xint:%
```

```
1539 {%
```

```
1540 \XINT_div_II_fa {#1!#2!}{#1!#2!#3}%
```

```
1541 }%
```

```
1542 \def\XINT_div_II_fa #1#2#3#4%
```

```
1543 {%
```

```
1544 \expandafter\XINT_div_II_g \the\numexpr\XINT_div_xmini #3\xint:#4!#1{#2}%
```

```
1545 }%
```

#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»

```
1546 \def\XINT_div_II_g 1#1#2#3#4#5!#6#7#8%
```

```
1547 {%
```

```
1548 \expandafter \XINT_div_II_h
```

```
1549 \the\numexpr 1#1#2#3#4#5+#8\expandafter\expandafter\expandafter
```

```
1550 \xint:\expandafter\expandafter\expandafter
```

```
1551 {\expandafter\xint_gob_til_exclam
```

```
1552 \romannumeral0\expandafter\XINT_div_sub\expandafter
```

```
1553 {\romannumeral0\XINT_rev_nounsep }#6\R!\R!\R!\R!\R!\R!\R!\R!\W}%
```

```
1554 {\the\numexpr\XINT_div_smallmul_a 100000000\xint:#1#2#3#4\xint:#5!#71;!}%
```

```
1555 {#7}%
```

```
1556 }%
```

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»

```
1557 \def\XINT_div_II_h 1#1\xint:#2#3#4%
```

```
1558 {%
```

```
1559 \XINT_div_II_k #4{#2}{#3}{#1}%
```

```
1560 }%
```

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

```
1561 \def\XINT_div_II_k #1#2#3#4#5%
```

```
1562 {%
```

```
1563 \expandafter\XINT_div_II_l \the\numexpr #4-\xint_c_i\xint:{#3}#1{#2}#5\xint:%
```

```
1564 }%
```

```
1565 \def\XINT_div_II_l #1\xint:#2#3#4#5!#6!%
```

```
1566 {%
```

```
1567 \XINT_div_II_m {{#1}{#2}{{#3}{#4}}{#5}}{#5}{#6}1#6!%
```

```
1568 }%
```

3 Package *xintcore* implementation

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

```
1569 \def\XINT_div_II_m #1#2#3#4\xint:#5#6%
1570 {%
1571     \XINT_div_I_a {#3}{#2}{#4}{#5}{#6}#1%
1572 }%
```

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

```
1573 \def\XINT_div_minimulwc_a 1#1\xint:#2\xint:#3!#4#5#6#7#8\xint:%
1574 {%
1575     \expandafter\XINT_div_minimulwc_b
1576     \the\numexpr \xint_c_x^ix+#1+#3*#8\xint:#3*#4#5#6#7+#2*#8\xint:#2*#4#5#6#7\xint:%
1577 }%
1578 \def\XINT_div_minimulwc_b 1#1#2#3#4#5#6\xint:#7\xint:%
1579 {%
1580     \expandafter\XINT_div_minimulwc_c
1581     \the\numexpr \xint_c_x^ix+#1#2#3#4#5+#7\xint:#6\xint:%
1582 }%
1583 \def\XINT_div_minimulwc_c 1#1#2#3#4#5#6\xint:#7\xint:#8\xint:%
1584 {%
1585     1#6#7\expandafter!%
1586     \the\numexpr\expandafter\XINT_div_smallmul_a
1587     \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8\xint:%
1588 }%
1589 \def\XINT_div_smallmul_a #1\xint:#2\xint:#3!1#4!%
1590 {%
1591     \xint_gob_til_sc #4\XINT_div_smallmul_e;%
1592     \XINT_div_minimulwc_a #1\xint:#2\xint:#3!#4\xint:#2\xint:#3!%
1593 }%
1594 \def\XINT_div_smallmul_e;\XINT_div_minimulwc_a 1#1\xint:#2;#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;! is added. If multiplicand is just 1 let's not forget to add the zero carry 100000000! at the end.

```
1595 \def\XINT_div_verysmallmul #1%
1596     {\xint_gob_til_one #1\XINT_div_verysmallisone 1\XINT_div_verysmallmul_a 0\xint:#1}%
1597 \def\XINT_div_verysmallisone 1\XINT_div_verysmallmul_a 0\xint:1!1#11;!%
1598     {1\relax #1100000000!}%
1599 \def\XINT_div_verysmallmul_a #1\xint:#2!1#3!%
1600 {%
1601     \xint_gob_til_sc #3\XINT_div_verysmallmul_e;%
1602     \expandafter\XINT_div_verysmallmul_b
1603     \the\numexpr \xint_c_x^ix+#2*#3+#1\xint:#2!%
1604 }%
1605 \def\XINT_div_verysmallmul_b 1#1#2\xint:%
1606     {1#2\expandafter!\the\numexpr\XINT_div_verysmallmul_a #1\xint:}%
1607 \def\XINT_div_verysmallmul_e;#1;+#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.

3 Package *xintcore* implementation

```

1608 \def\XINT_div_sub #1#2%
1609 {%
1610     \expandafter\XINT_div_sub_clean
1611     \the\numexpr\expandafter\XINT_div_sub_a\expandafter
1612     1#2;!!;!!;!!\W #1;!!;!!;!!\W
1613 }%
1614 \def\XINT_div_sub_clean #1-#2#3\W
1615 {%
1616     \if1#2\expandafter\XINT_rev_nounsep\else\expandafter\XINT_div_sub_neg\fi
1617     {}#1\R!\R!\R!\R!\R!\R!\R!\R!\W
1618 }%
1619 \def\XINT_div_sub_neg #1\W { -}%
1620 \def\XINT_div_sub_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
1621 {%
1622     \XINT_div_sub_b #1!#6!#2!#7!#3!#8!#4!#9!#5\W
1623 }%
1624 \def\XINT_div_sub_b #1#2#3!#4!%
1625 {%
1626     \xint_gob_til_sc #4\XINT_div_sub_bi ;%
1627     \expandafter\XINT_div_sub_c\the\numexpr#1-#3+1#4-\xint_c_i\xint:%
1628 }%
1629 \def\XINT_div_sub_c 1#1#2\xint:%
1630 {%
1631     1#2\expandafter!\the\numexpr\XINT_div_sub_d #1%
1632 }%
1633 \def\XINT_div_sub_d #1#2#3!#4!%
1634 {%
1635     \xint_gob_til_sc #4\XINT_div_sub_di ;%
1636     \expandafter\XINT_div_sub_e\the\numexpr#1-#3+1#4-\xint_c_i\xint:%
1637 }%
1638 \def\XINT_div_sub_e 1#1#2\xint:%
1639 {%
1640     1#2\expandafter!\the\numexpr\XINT_div_sub_f #1%
1641 }%
1642 \def\XINT_div_sub_f #1#2#3!#4!%
1643 {%
1644     \xint_gob_til_sc #4\XINT_div_sub_fi ;%
1645     \expandafter\XINT_div_sub_g\the\numexpr#1-#3+1#4-\xint_c_i\xint:%
1646 }%
1647 \def\XINT_div_sub_g 1#1#2\xint:%
1648 {%
1649     1#2\expandafter!\the\numexpr\XINT_div_sub_h #1%
1650 }%
1651 \def\XINT_div_sub_h #1#2#3!#4!%
1652 {%
1653     \xint_gob_til_sc #4\XINT_div_sub_hi ;%
1654     \expandafter\XINT_div_sub_i\the\numexpr#1-#3+1#4-\xint_c_i\xint:%
1655 }%
1656 \def\XINT_div_sub_i 1#1#2\xint:%
1657 {%
1658     1#2\expandafter!\the\numexpr\XINT_div_sub_a #1%
1659 }%

```


3 Package *xintcore* implementation

```

1660 \def\XINT_div_sub_bi;%
1661   \expandafter\XINT_div_sub_c\the\numexpr#1-#2+#3\xint:#4!#5!#6!#7!#8!#9!;! \W
1662 {%
1663   \XINT_div_sub_l #1#2!#5!#7!#9!%
1664 }%
1665 \def\XINT_div_sub_di;%
1666   \expandafter\XINT_div_sub_e\the\numexpr#1-#2+#3\xint:#4!#5!#6!#7!#8\W
1667 {%
1668   \XINT_div_sub_l #1#2!#5!#7!%
1669 }%
1670 \def\XINT_div_sub_fi;%
1671   \expandafter\XINT_div_sub_g\the\numexpr#1-#2+#3\xint:#4!#5!#6\W
1672 {%
1673   \XINT_div_sub_l #1#2!#5!%
1674 }%
1675 \def\XINT_div_sub_hi;%
1676   \expandafter\XINT_div_sub_i\the\numexpr#1-#2+#3\xint:#4\W
1677 {%
1678   \XINT_div_sub_l #1#2!%
1679 }%
1680 \def\XINT_div_sub_l #1%
1681 {%
1682   \xint_UDzerofork
1683   #1{-2\relax}%
1684   0\XINT_div_sub_r
1685   \krof
1686 }%
1687 \def\XINT_div_sub_r #1!%
1688 {%
1689   -\ifnum 0#1=\xint_c_ 1\else2\fi\relax
1690 }%

```

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

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

```

1691 \def\XINT_sdiv_out #1;!#2!%
1692   {\expandafter
1693   {\romannumeral0\XINT_unsep_cuzsmall
1694   #1\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax}%
1695   {#2}}%

```

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

```

1696 \def\XINT_smallldivx_a #1\xint:1#2!1#3!%
1697 {%
1698   \expandafter\XINT_smallldivx_b
1699   \the\numexpr (#3+#1)/#2-\xint_c_i!#1\xint:#2!#3!%
1700 }%
1701 \def\XINT_smallldivx_b #1#2!%

```

3 Package *xintcore* implementation

```
1702 {%
1703   \if0#1\else
1704     \xint_c_x^viii+#1#2\xint_afterfi{\expandafter!\the\numexpr}\fi
1705   \XINT_smallldiv_c #1#2!%
1706}%
1707\def\XINT_smallldiv_c #1!#2\xint:#3!#4!%
1708{%
1709  \expandafter\XINT_smallldiv_d\the\numexpr #4-#1*#3!#2\xint:#3!%
1710}%

```

On va boucler ici: #1 est un reste, #2 est x.B (avec B sans le 1 mais sur huit chiffres). #3#4 est le premier bloc qui reste de A. Si on a terminé avec A, alors #1 est le reste final. Le quotient lui est terminé par un 1! ce 1! disparaîtra dans le nettoyage par \XINT_unsep_cuzsmall.

```
1711\def\XINT_smallldiv_d #1!#2!1#3#4!%
1712{%
1713  \xint_gob_til_sc #3\XINT_smallldiv_end;%
1714  \XINT_smallldiv_e #1!#2!1#3#4!%
1715}%
1716\def\XINT_smallldiv_end;\XINT_smallldiv_e #1!#2!1;!{1!;!#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.

```
1717\def\XINT_smallldiv_e #1!#2\xint:#3!%
1718{%
1719  \expandafter\XINT_smallldiv_f\the\numexpr
1720  \xint_c_xi_e_viii_mone+#1*\xint_c_x^viii/#3!#2\xint:#3!#1!%
1721}%
1722\def\XINT_smallldiv_f 1#1#2#3#4#5#6!#7\xint:#8!%
1723{%
1724  \xint_gob_til_zero #1\XINT_smallldiv_fz 0%
1725  \expandafter\XINT_smallldiv_g
1726  \the\numexpr\XINT_minimul_a #2#3#4#5\xint:#6!#8!#2#3#4#5#6!#7\xint:#8!%
1727}%
1728\def\XINT_smallldiv_fz 0%
1729  \expandafter\XINT_smallldiv_g\the\numexpr\XINT_minimul_a
1730  9999\xint:9999!#1!99999999!#2!0!1#3!%
1731{%
1732  \XINT_smallldiv_i \xint:#3!\xint_c_!#2!%
1733}%
1734\def\XINT_smallldiv_g 1#1!1#2!#3!#4!#5!#6!%
1735{%
1736  \expandafter\XINT_smallldiv_h\the\numexpr 1#6-#1\xint:#2!#5!#3!#4!%
1737}%
1738\def\XINT_smallldiv_h 1#1#2\xint:#3!#4!%
1739{%
1740  \expandafter\XINT_smallldiv_i\the\numexpr #4-#3+#1-\xint_c_i\xint:#2!%
1741}%
1742\def\XINT_smallldiv_i #1\xint:#2!#3!#4\xint:#5!%
1743{%
1744  \expandafter\XINT_smallldiv_j\the\numexpr (#1#2+#4)/#5-\xint_c_i!#3!#1#2!#4\xint:#5!%
1745}%

```

3 Package *xintcore* implementation

```

1746 \def\XINT_smallldiv_j #1!#2!%
1747 {%
1748   \xint_c_x^viii+#1+#2\expandafter!\the\numexpr\XINT_smallldiv_k
1749   #1!%
1750 }%

```

On boucle vers \XINT_smallldiv_d.

```

1751 \def\XINT_smallldiv_k #1!#2!#3\xint:#4!%
1752 {%
1753   \expandafter\XINT_smallldiv_d\the\numexpr #2-#1*#4!#3\xint:#4!%
1754 }%

```

Cette routine fait la division euclidienne d'un nombre de seize chiffres par $\#1 = C =$ diviseur sur huit chiffres $\geq 10^7$, avec $\#2 =$ sa moitié utilisée dans \numexpr pour contrebalancer l'arrondi (ARRRRRRGGGGGHHH) fait par /. 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 $1<8\text{chiffres}>.$

```

1755 \def\XINT_div_mini #1\xint:#2!#3!%
1756 {%
1757   \expandafter\XINT_div_mini_a\the\numexpr
1758   \xint_c_xi_eviii_mone+#3*\xint_c_x^viii/#1!#1\xint:#2!#3!%
1759 }%

```

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.

```

1760 \def\XINT_div_mini_a 1#1#2#3#4#5#6!#7\xint:#8!%
1761 {%
1762   \xint_gob_til_zero #1\XINT_div_mini_w 0%
1763   \expandafter\XINT_div_mini_b
1764   \the\numexpr\XINT_minimul_a #2#3#4#5\xint:#6!#7!#2#3#4#5#6!#7\xint:#8!%
1765 }%
1766 \def\XINT_div_mini_w 0%
1767   \expandafter\XINT_div_mini_b\the\numexpr\XINT_minimul_a
1768   9999\xint:9999!#1!99999999!#2\xint:#3!00000000!#4!%
1769 {%
1770   \xint_c_x^viii_mone+(#4+#3)/#2!%
1771 }%
1772 \def\XINT_div_mini_b 1#1!1#2!#3!#4!#5!#6!%
1773 {%
1774   \expandafter\XINT_div_mini_c
1775   \the\numexpr 1#6-#1\xint:#2!#5!#3!#4!%
1776 }%
1777 \def\XINT_div_mini_c 1#1#2\xint:#3!#4!%
1778 {%
1779   \expandafter\XINT_div_mini_d
1780   \the\numexpr #4-#3+#1-\xint_c_i\xint:#2!%

```

```

1781 }%
1782 \def\XINT_div_mini_d #1\xint:#2!#3!#4\xint:#5!%
1783 {%
1784   \xint_c_x^viii_mone+#3+(#1#2+#5)/#4!%
1785 }%

```

Derived arithmetic

3.38 `\xintiiQuo`, `\xintiiRem`

```

1786 \def\xintiiQuo {\romannumeral0\xintiiquo }%
1787 \def\xintiiRem {\romannumeral0\xintiirem }%
1788 \def\xintiiquo
1789   {\expandafter\xint_firstoftwo_thenstop\romannumeral0\xintiidivision }%
1790 \def\xintiirem
1791   {\expandafter\xint_secondoftwo_thenstop\romannumeral0\xintiidivision }%
1792 \def\xintiQuo {\romannumeral0\XINT_signaldeprecated{xintcore}{xintiQuo}\xintiquo }%
1793 \def\xintiRem {\romannumeral0\XINT_signaldeprecated{xintcore}{xintiRem}\xintirem }%
1794 \def\xintiquo
1795   {\expandafter\xint_firstoftwo_thenstop\romannumeral0\xintidivision }%
1796 \def\xintirem
1797   {\expandafter\xint_secondoftwo_thenstop\romannumeral0\xintidivision }%

```

3.39 `\xintiiDivRound`

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

1.2l: `\xintiiDivRound` made robust against non terminated input.

```

1798 \def\xintiDivRound {\romannumeral0\XINT_signaldeprecated{xintcore}{xintiDivRound}\xintidivround }%
1799 \def\xintidivround #1%
1800   {\expandafter\XINT_idivround\romannumeral0\xintnum{#1}\xint:}%
1801 \def\xintiiDivRound {\romannumeral0\xintiidivround }%
1802 \def\xintiidivround #1{\expandafter\XINT_iidivround\romannumeral`&&@#1\xint:}%
1803 \def\XINT_idivround #1#2\xint:#3%
1804   {\expandafter\XINT_iidivround_a\expandafter #1%
1805     \romannumeral0\xintnum{#3}\xint:#2\xint:}%
1806 \def\XINT_iidivround #1#2\xint:#3%
1807   {\expandafter\XINT_iidivround_a\expandafter #1\romannumeral`&&@#3\xint:#2\xint:}%
1808 \def\XINT_iidivround_a #1#2% #1 de A, #2 de B.
1809 {%
1810   \if0#2\xint_dothis{\XINT_iidivround_divbyzero#1#2}\fi
1811   \if0#1\xint_dothis\XINT_iidivround_aiszero\fi
1812   \if-#2\xint_dothis{\XINT_iidivround_bneg #1}\fi
1813   \xint_orthat{\XINT_iidivround_bpos #1#2}%
1814 }%
1815 \def\XINT_iidivround_divbyzero #1#2#3\xint:#4\xint:
1816   {\XINT_signalcondition{DivisionByZero}{Division of #1#4 by #2#3}{0}}%
1817 \def\XINT_iidivround_aiszero #1\xint:#2\xint:{ 0}%
1818 \def\XINT_iidivround_bpos #1%
1819 {%
1820   \xint_UDsignfork
1821     #1{\xintiiopp\XINT_iidivround_pos }}%
1822     -{\XINT_iidivround_pos #1}%

```

```

1823 \krof
1824 }%
1825 \def\XINT_iidivround_bneg #1%
1826 {%
1827 \xint_UDsignfork
1828 #1{\XINT_iidivround_pos {}}%
1829 -{\xintiioopp\XINT_iidivround_pos #1}%
1830 \krof
1831 }%
1832 \def\XINT_iidivround_pos #1#2\xint:#3\xint:
1833 {%
1834 \expandafter\expandafter\expandafter\XINT_dsrr
1835 \expandafter\xint_firstoftwo
1836 \romannumeral0\XINT_div_prepare {#2}{#1#30}%
1837 \xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax
1838 }%

```

3.40 \xintiDivTrunc

1.21: \xintiDivTrunc made robust against non terminated input.

```

1839 \def\xintiDivTrunc {\romannumeral0\XINT_signaldeprecated{xintcore}{xintiDivTrunc}\xintidivtrunc }%
1840 \def\xintidivtrunc #1{\expandafter\XINT_iidivtrunc\romannumeral0\xintnum{#1}\xint:}%
1841 \def\xintiDivTrunc {\romannumeral0\xintiDivTrunc }%
1842 \def\xintiDivTrunc #1{\expandafter\XINT_iidivtrunc\romannumeral`&&@#1\xint:}%
1843 \def\XINT_iidivtrunc #1#2\xint:#3{\expandafter\XINT_iidivtrunc_a\expandafter #1%
1844 \romannumeral`&&@#3\xint:#2\xint:}%
1845 \def\XINT_iidivtrunc_a #1#2% #1 de A, #2 de B.
1846 {%
1847 \if0#2\xint_dothis{\XINT_iidivtrunc_divbyzero#1#2}\fi
1848 \if0#1\xint_dothis\XINT_iidivtrunc_aiszero\fi
1849 \if-#2\xint_dothis{\XINT_iidivtrunc_bneg #1}\fi
1850 \xint_orthat{\XINT_iidivtrunc_bpos #1#2}%
1851 }%

```

Attention to not move DivRound code beyond that point.

```

1852 \let\XINT_iidivtrunc_divbyzero\XINT_iidivround_divbyzero
1853 \let\XINT_iidivtrunc_aiszero \XINT_iidivround_aiszero
1854 \def\XINT_iidivtrunc_bpos #1%
1855 {%
1856 \xint_UDsignfork
1857 #1{\xintiioopp\XINT_iidivtrunc_pos {}}%
1858 -{\XINT_iidivtrunc_pos #1}%
1859 \krof
1860 }%
1861 \def\XINT_iidivtrunc_bneg #1%
1862 {%
1863 \xint_UDsignfork
1864 #1{\XINT_iidivtrunc_pos {}}%
1865 -{\xintiioopp\XINT_iidivtrunc_pos #1}%
1866 \krof
1867 }%

```

```

1868 \def\XINT_iidivtrunc_pos #1#2\xint:#3\xint:
1869     {\expandafter\xint_firstoftwo_thenstop
1870     \romannumeral0\XINT_div_prepare {#2}{#1#3}}%

```

3.41 \xintiModTrunc

Renamed from \xintiMod to \xintiModTrunc at 1.2p.

```

1871 \def\xintiMod {\romannumeral0\XINT_signaldeprecated{xintcore}{xintiMod}\xintimod }%
1872 \def\xintimod #1{\expandafter\XINT_iimodtrunc\romannumeral0\xintnum{#1}\xint:}%
1873 \def\xintiModTrunc {\romannumeral0\xintiimodtrunc }%
1874 \def\xintiimodtrunc #1{\expandafter\XINT_iimodtrunc\romannumeral`&&@#1\xint:}%
1875 \def\XINT_iimodtrunc #1#2\xint:#3{\expandafter\XINT_iimodtrunc_a\expandafter #1%
1876     \romannumeral`&&@#3\xint:#2\xint:}%
1877 \def\XINT_iimodtrunc_a #1#2% #1 de A, #2 de B.
1878 {%
1879     \if0#2\xint_dothis{\XINT_iimodtrunc_divbyzero#1#2}\fi
1880     \if0#1\xint_dothis\XINT_iimodtrunc_aiszero\fi
1881     \if-#2\xint_dothis{\XINT_iimodtrunc_bneg #1}\fi
1882     \xint_orthat{\XINT_iimodtrunc_bpos #1#2}%
1883 }%

```

Attention to not move DivRound code beyond that point. A bit of abuse here for divbyzero defaulted-to value, which happily works in both.

```

1884 \let\XINT_iimodtrunc_divbyzero\XINT_iidivround_divbyzero
1885 \let\XINT_iimodtrunc_aiszero \XINT_iidivround_aiszero
1886 \def\XINT_iimodtrunc_bpos #1%
1887 {%
1888     \xint_UDsignfork
1889     #1{\xintiiopt\XINT_iimodtrunc_pos {}}%
1890     -{\XINT_iimodtrunc_pos #1}%
1891     \krof
1892 }%
1893 \def\XINT_iimodtrunc_bneg #1%
1894 {%
1895     \xint_UDsignfork
1896     #1{\xintiiopt\XINT_iimodtrunc_pos {}}%
1897     -{\XINT_iimodtrunc_pos #1}%
1898     \krof
1899 }%
1900 \def\XINT_iimodtrunc_pos #1#2\xint:#3\xint:
1901     {\expandafter\xint_secondoftwo_thenstop\romannumeral0\XINT_div_prepare
1902     {#2}{#1#3}}%

```

3.42 \xintiDivMod

1.2p. Associated with floored division like Python's divmod.

```

1903 \def\xintiDivMod {\romannumeral0\xintiDivMod }%
1904 \def\xintiDivMod #1{\expandafter\XINT_iidivmod\romannumeral`&&@#1\xint:}%
1905 \def\XINT_iidivmod #1#2\xint:#3{\expandafter\XINT_iidivmod_a\expandafter #1%
1906     \romannumeral`&&@#3\xint:#2\xint:}%

```

```

1907 \def\XINT_iidivmod_a #1#2% #1 de A, #2 de B.
1908 {%
1909     \if0#2\xint_dothis{\XINT_iidivmod_divbyzero#1#2}\fi
1910     \if0#1\xint_dothis\XINT_iidivmod_aiszero\fi
1911     \if-#2\xint_dothis{\XINT_iidivmod_bneg #1}\fi
1912     \xint_orthat{\XINT_iidivmod_bpos #1#2}%
1913 }%
1914 \def\XINT_iidivmod_divbyzero #1#2\xint:#3\xint:
1915 {%
1916     \XINT_signalcondition{DivisionByZero}{Division by #2 of #1#3}{}%
1917     {{0}{0}}% à revoir...
1918 }%
1919 \def\XINT_iidivmod_aiszero #1#2\xint:#3\xint:{{0}{0}}%
1920 \def\XINT_iidivmod_bneg #1%
1921 {%
1922     \expandafter\XINT_iidivmod_bneg_finish
1923     \romannumeral0\xint_UDsignfork
1924     #1{\XINT_iidivmod_bpos {}}%
1925     -{\XINT_iidivmod_bpos {-#1}}%
1926     \krof
1927 }%
1928 \def\XINT_iidivmod_bneg_finish#1#2%
1929 {%
1930     \expandafter\xint_exchangetwo_keepbraces\expandafter
1931     {\romannumeral0\xintiioff#2}{#1}%
1932 }%
1933 \def\XINT_iidivmod_bpos #1#2\xint:#3\xint:{{\xintiividivision{#1#3}{#2}}}%

```

3.43 \xintiiDivFloor

1.2p. For `bnumexpr` actually, because `\xintiiepr` could use `\xintDivFloor` which also outputs an integer in strict format.

```

1934 \def\xintiiDivFloor {\romannumeral0\xintiividivfloor}%
1935 \def\xintiividivfloor {\expandafter\xint_firstoftwo_thenstop
1936     \romannumeral0\xintiividivmod}%

```

3.44 \xintiiMod

Associated with floored division at 1.2p. Formerly was associated with truncated division.

```

1937 \def\xintiiMod {\romannumeral0\xintiimod}%
1938 \def\xintiimod {\expandafter\xint_secondoftwo_thenstop
1939     \romannumeral0\xintiividivmod}%

```

3.45 \xintiiSqr

1.2l: `\xintiiSqr` made robust against non terminated input.

```

1940 \def\xintiiSqr {\romannumeral0\xintiisqr}%
1941 \def\xintiisqr #1%
1942 {%
1943     \expandafter\XINT_sqr\romannumeral0\xintiiaabs{#1}\xint:

```

3 Package *xintcore* implementation

```

1944 }%
1945 \def\xintisqr {\romannumeral0\XINT_signaldeprecated{xintcore}{xintisqr}\xintisqr }%
1946 \def\xintisqr #1%
1947 {%
1948   \expandafter\XINT_sqr\romannumeral0\xintiabs{#1}\xint:
1949 }%
1950 \def\XINT_sqr #1\xint:
1951 {%
1952   \expandafter\XINT_sqr_a
1953   \romannumeral0\expandafter\XINT_sepandrev_andcount
1954   \romannumeral0\XINT_zeroes_forviii #1\R\R\R\R\R\R\R\{10}0000001\W
1955   #1\XINT_rsepbyviii_end_A 2345678%
1956   \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
1957   \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
1958   \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
1959   \xint:
1960 }%

```

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

```

1961 \def\XINT_sqr_a #1\xint:
1962 {%
1963   \ifnum #1=\xint_c_i \expandafter\XINT_sqr_small
1964   \else\expandafter\XINT_sqr_start\fi
1965 }%
1966 \def\XINT_sqr_small 1#1#2#3#4#5!\xint:
1967 {%
1968   \ifnum #1#2#3#4#5<46341 \expandafter\XINT_sqr_verysmall\fi
1969   \expandafter\XINT_sqr_small_out
1970   \the\numexpr\XINT_minimul_a #1#2#3#4\xint:#5!#1#2#3#4#5!%
1971 }%
1972 \def\XINT_sqr_verysmall#1{%
1973 \def\XINT_sqr_verysmall
1974   \expandafter\XINT_sqr_small_out\the\numexpr\XINT_minimul_a ##1!##2!%
1975   {\expandafter#1\the\numexpr ##2*##2\relax}%
1976 }\XINT_sqr_verysmall{ }%
1977 \def\XINT_sqr_small_out 1#1!1#2!%
1978 {%
1979   \XINT_cuz #2#1\R
1980 }%

```

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

```

1981 \def\XINT_sqr_start #1\xint:
1982 {%
1983   \expandafter\XINT_mul_out
1984   \the\numexpr\XINT_mul_loop
1985   100000000!1;! \W #11;! \W #11;!%
1986   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1987 }%

```


3.46 `\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.

1.2l: `\xintiiPow` made robust against non terminated input.

```

1988 \def\xintiiPow {\romannumeral0\xintiipow }%
1989 \def\xintiipow #1#2%
1990 {%
1991   \expandafter\xint_pow\the\numexpr #2\expandafter
1992   .\romannumeral`&&@#1\xint:
1993 }%
1994 \def\xintiPow  {\romannumeral0\XINT_signaldeprecated{xintcore}{xintiPow}\xintipow }%
1995 \def\xintipow #1#2%
1996 {%
1997   \expandafter\xint_pow\the\numexpr #2\expandafter
1998   .\romannumeral0\xintnum{#1}\xint:
1999 }%
2000 \def\xint_pow #1.#2#3\xint:
2001 {%
2002   \xint_UDzerominusfork
2003   #2-\XINT_pow_AisZero
2004   0#2\XINT_pow_Aneg
2005   0-\XINT_pow_Apos #2}%
2006   \krof {#1}%
2007 }%
2008 \def\XINT_pow_AisZero #1#2\xint:
2009 {%
2010   \ifcase\XINT_cntSgn #1\xint:
2011     \xint_afterfi { 1}%
2012   \or
2013     \xint_afterfi { 0}%
2014   \else
2015     \xint_afterfi
2016     {\XINT_signalcondition{DivisionByZero}{Zero to power #1}{\{0}}}%
2017   \fi
2018 }%
2019 \def\XINT_pow_Aneg #1%
2020 {%
2021   \ifodd #1
2022     \expandafter\XINT_opp\romannumeral0%
2023   \fi
2024   \XINT_pow_Apos {\}{#1}%
2025 }%
2026 \def\XINT_pow_Apos #1#2{\XINT_pow_Apos_a {#2}#1}%
2027 \def\XINT_pow_Apos_a #1#2#3%
2028 {%
2029   \xint_gob_til_xint: #3\XINT_pow_Apos_short\xint:
2030   \XINT_pow_AatleastTwo {#1}#2#3%

```

3 Package *xintcore* implementation

```

2031 }%
2032 \def\XINT_pow_Apos_short\xint:\XINT_pow_AatleastTwo #1#2\xint:
2033 {%
2034   \ifcase #2
2035     \xintError:thiscannothappen
2036   \or \expandafter\XINT_pow_AisOne
2037   \else\expandafter\XINT_pow_AatleastTwo
2038   \fi {#1}#2\xint:
2039 }%
2040 \def\XINT_pow_AisOne #1\xint:{ 1}%
2041 \def\XINT_pow_AatleastTwo #1%
2042 {%
2043   \ifcase\XINT_cntSgn #1\xint:
2044     \expandafter\XINT_pow_BisZero
2045   \or
2046     \expandafter\XINT_pow_I_in
2047   \else
2048     \expandafter\XINT_pow_BisNegative
2049   \fi
2050   {#1}%
2051 }%
2052 \def\XINT_pow_BisNegative #1\xint:{\XINT_signalcondition{Underflow}{Inverse power
2053   can not be represented by an integer}}{0}}%
2054 \def\XINT_pow_BisZero #1\xint:{ 1}%

```

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

```

2055 \def\XINT_pow_I_in #1#2\xint:
2056 {%
2057   \expandafter\XINT_pow_I_loop
2058   \the\numexpr #1\expandafter\xint:%
2059   \romannumeral0\expandafter\XINT_sepandrev
2060   \romannumeral0\XINT_zeroes_forviii #2\R\R\R\R\R\R\R\R{10}0000001\W
2061   #2\XINT_rsepbyviii_end_A 2345678%
2062   \XINT_rsepbyviii_end_B 2345678\relax XX%
2063   \R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\W
2064   1;! \W
2065   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
2066 }%
2067 \def\XINT_pow_I_loop #1\xint:%
2068 {%
2069   \ifnum #1 = \xint_c_i\expandafter\XINT_pow_I_exit\fi
2070   \ifodd #1
2071     \expandafter\XINT_pow_II_in
2072   \else
2073     \expandafter\XINT_pow_I_squareit
2074   \fi #1\xint:%
2075 }%
2076 \def\XINT_pow_I_exit \ifodd #1\fi #2\xint:#3\W {\XINT_mul_out #3}%

```

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

3 Package *xintcore* implementation

```
2077 \def\XINT_pow_I_squareit #1\xint:#2\W%
2078 {%
2079   \expandafter\XINT_pow_I_loop
2080   \the\numexpr #1/\xint_c_ii\expandafter\xint:%
2081   \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W
2082 }%
2083 \def\XINT_pow_mulbutcheckifsmall #1!1#2%
2084 {%
2085   \xint_gob_til_sc #2\XINT_pow_mul_small;%
2086   \XINT_mul_loop 100000000!1;! \W #1!1#2%
2087 }%
2088 \def\XINT_pow_mul_small;\XINT_mul_loop
2089   100000000!1;! \W 1#1!1;! \W
2090 {%
2091   \XINT_smallmul 1#1!%
2092 }%
2093 \def\XINT_pow_II_in #1\xint:#2\W
2094 {%
2095   \expandafter\XINT_pow_II_loop
2096   \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter\xint:%
2097   \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W #2\W
2098 }%
2099 \def\XINT_pow_II_loop #1\xint:%
2100 {%
2101   \ifnum #1 = \xint_c_i\expandafter\XINT_pow_II_exit\fi
2102   \ifodd #1
2103     \expandafter\XINT_pow_II_odda
2104   \else
2105     \expandafter\XINT_pow_II_even
2106   \fi #1\xint:%
2107 }%
2108 \def\XINT_pow_II_exit\ifodd #1\fi #2\xint:#3\W #4\W
2109 {%
2110   \expandafter\XINT_mul_out
2111   \the\numexpr\XINT_pow_mulbutcheckifsmall #4\W #3%
2112 }%
2113 \def\XINT_pow_II_even #1\xint:#2\W
2114 {%
2115   \expandafter\XINT_pow_II_loop
2116   \the\numexpr #1/\xint_c_ii\expandafter\xint:%
2117   \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W
2118 }%
2119 \def\XINT_pow_II_odda #1\xint:#2\W #3\W
2120 {%
2121   \expandafter\XINT_pow_II_oddb
2122   \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter\xint:%
2123   \the\numexpr\XINT_pow_mulbutcheckifsmall #3\W #2\W #2\W
2124 }%
2125 \def\XINT_pow_II_oddb #1\xint:#2\W #3\W
2126 {%
2127   \expandafter\XINT_pow_II_loop
2128   \the\numexpr #1\expandafter\xint:%
```

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

Partially rewritten with release 1.2 to benefit from the inner format of the 1.2 multiplication.
With current default settings of the etex memory and a.t.t.o.w (11/2015) the maximal possible computation is 5971! (which has 19956 digits).

```
\catcode\_ 11
\catcode\_^ 11
\long\def\xint_firstofthree #1#2#3{#1}%
\long\def\xint_secondofthree #1#2#3{#2}%
\long\def\xint_thirdofthree #1#2#3{#3}%
% quickly written factorial using binary split recursive method
\def\tFac {\romannumeral-`0\tfac }%
\def\tfac #1{\expandafter\XINT_mul_out
    \romannumeral-`0\ufac {1}{#1}1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W}%
\def\ufac #1#2{\ifcase\numexpr#2-#1\relax
    \expandafter\xint_firstofthree
  \or
    \expandafter\xint_secondofthree
  \else
    \expandafter\xint_thirdofthree
  \fi
  {\the\numexpr\xint_c_x^viii+#1!1;!}%
  {\the\numexpr\xint_c_x^viii+#1*#2!1;!}%
  {\expandafter\vfac\the\numexpr (#1+#2)/\xint_c_ii.#1.#2.}%
}%
\def\vfac #1.#2.#3.%
{%
  \expandafter
  \wfac\expandafter
    {\romannumeral-`0\expandafter
      \ufac\expandafter{\the\numexpr #1+\xint_c_i}{#3}}%
    {\ufac {#2}{#1}}%
}%
\def\wfac #1#2{\expandafter\zfac\romannumeral-`0#2\W #1}%
\def\zfac {\the\numexpr\XINT_mul_loop 100000000!1;! \W }% core multiplication..
\catcode\_ 8
\catcode\_^ 7
```

Note (2017, 1.21): I found out some code comment of mine that the code here should be more in the style of `\xintiiBinomial`, but I left matters untouched.

1.2o modifies `\xintiFac` to be coherent with `\xintiBinomial`: only with `xintfrac.sty` loaded does it use `\xintNum`. It is documented only as macro of `xintfrac.sty`, not as macro of `xint.sty`.

109

3 Package *xintcore* implementation

```

2181 \expandafter\XINT_fac_medloop_b
2182 \the\numexpr #1+\xint_c_i-\xint_c_iii*((#1-100)/\xint_c_iii).#1.%
2183 }%
2184 \def\XINT_fac_medloop_b #1.#2.%
2185 {%
2186 \expandafter\XINT_fac_smallloop_a
2187 \the\numexpr #1-\xint_c_i.{\XINT_fac_medloop_loop #1.#2.}%
2188 }%
2189 \def\XINT_fac_medloop_loop #1.#2.%
2190 {%
2191 \ifnum #1>#2 \expandafter\XINT_fac_loop_exit\fi
2192 \expandafter\XINT_fac_medloop_loop
2193 \the\numexpr #1+\xint_c_iii\expandafter.%
2194 \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_medloop_mul #1!%
2195 }%
2196 \def\XINT_fac_medloop_mul #1!%
2197 {%
2198 \expandafter\XINT_smallmul
2199 \the\numexpr
2200 \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2201 }%
2202 \def\XINT_fac_smallloop_a #1.%
2203 {%
2204 \csname
2205 XINT_fac_smallloop_\the\numexpr #1-\xint_c_iv*(#1/\xint_c_iv)\relax
2206 \endcsname #1.%
2207 }%
2208 \expandafter\def\csname XINT_fac_smallloop_1\endcsname #1.%
2209 {%
2210 \XINT_fac_smallloop_loop 2.#1.100000001!1!;%
2211 }%
2212 \expandafter\def\csname XINT_fac_smallloop_-2\endcsname #1.%
2213 {%
2214 \XINT_fac_smallloop_loop 3.#1.100000002!1!;%
2215 }%
2216 \expandafter\def\csname XINT_fac_smallloop_-1\endcsname #1.%
2217 {%
2218 \XINT_fac_smallloop_loop 4.#1.100000006!1!;%
2219 }%
2220 \expandafter\def\csname XINT_fac_smallloop_0\endcsname #1.%
2221 {%
2222 \XINT_fac_smallloop_loop 5.#1.1000000024!1!;%
2223 }%
2224 \def\XINT_fac_smallloop_loop #1.#2.%
2225 {%
2226 \ifnum #1>#2 \expandafter\XINT_fac_loop_exit\fi
2227 \expandafter\XINT_fac_smallloop_loop
2228 \the\numexpr #1+\xint_c_iv\expandafter.%
2229 \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_smallloop_mul #1!%
2230 }%
2231 \def\XINT_fac_smallloop_mul #1!%
2232 {%

```

```

2233 \expandafter\XINT_smallmul
2234 \the\numexpr
2235 \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2236 }%
2237 \def\XINT_fac_loop_exit #1!#2;!#3{#3#2;!}%

```

3.48 `\XINT_signaldeprecated`

1.2o

```

2238 \def\XINT_signaldeprecated #1#2%
2239 {%
2240 \XINT_ifFlagRaised{#2}%
2241 }% already encountered (if not hidden in a group...)
2242 {\XINT_RaiseFlag{Deprecated-#1}\XINT_RaiseFlag{#2}%
2243 \expandafter\XINT_expandableerror\expandafter
2244 {\csname#2\endcsname (#1) is deprecated! (RET to proceed)}}%
2245 }%

```

At End of \LaTeX Document deprecation message

1.2o

```

2246 \ifdefined\documentclass\ifdefined\AtEndDocument
2247 \AtEndDocument{%
2248 \XINT_ifFlagRaised{Deprecated-xintcore}%
2249 {\PackageError{xintcore}
2250 {Usage of deprecated macros!}
2251 {These deprecated macros from xintcore.sty have been detected:\MessageBreak
2252 \XINT_useiiimage{xintSgn}%
2253 \XINT_useiiimage{xintCmp}%
2254 \XINT_ifFlagRaised{xintiOpp}{\string\xintiOpp\MessageBreak}{}%
2255 \XINT_ifFlagRaised{xintiAbs}{\string\xintiAbs\MessageBreak}{}%
2256 \XINT_ifFlagRaised{xintiiFDg}{\string\xintiiFDg\space(renamed to \string\xintFDg!)\MessageBreak}{}%
2257 \XINT_ifFlagRaised{xintiiLDg}{\string\xintiiLDg\space(renamed to \string\xintLDg!)\MessageBreak}{}%
2258 \XINT_ifFlagRaised{xintiAdd}{\string\xintiAdd\MessageBreak}{}%
2259 \XINT_ifFlagRaised{xintiSub}{\string\xintiSub\MessageBreak}{}%
2260 \XINT_ifFlagRaised{xintiMul}{\string\xintiMul\MessageBreak}{}%
2261 \XINT_ifFlagRaised{xintiDivision}{\string\xintiDivision\MessageBreak}{}%
2262 \XINT_ifFlagRaised{xintiQuo}{\string\xintiQuo\MessageBreak}{}%
2263 \XINT_ifFlagRaised{xintiRem}{\string\xintiRem\MessageBreak}{}%
2264 \XINT_ifFlagRaised{xintiDivRound}{\string\xintiDivRound\MessageBreak}{}%
2265 \XINT_ifFlagRaised{xintiDivTrunc}{\string\xintiDivTrunc\MessageBreak}{}%
2266 \XINT_ifFlagRaised{xintiMod}{\string\xintiMod\MessageBreak}{}%
2267 \XINT_ifFlagRaised{xintiSqr}{\string\xintiSqr\MessageBreak}{}%
2268 \XINT_ifFlagRaised{xintiPow}{\string\xintiPow\MessageBreak}{}%
2269 \XINT_ifFlagRaised{xintiFac}{\string\xintiFac\MessageBreak}{}%
2270 They will get removed at some future release.}}%
2271 {% no deprecated macro used (at top level...)
2272 }%
2273 }\fi\fi

```

3.49 `\XINT_useiimessage`

1.2o

```
2274 \def\XINT_useiimessage #1% used in LaTeX only
2275 {%
2276   \XINT_ifFlagRaised {#1}%
2277   {\@backslashchar#1
2278     (load xintfrac or use \@backslashchar xintii\xint_gobble_iv#1!)\MessageBreak}%
2279   }%
2280 }%
2281 \XINT_restorecatcodes_endinput%
```


4 Package `xint` implementation

.1	Package identification	114	.28	<code>\xintiiifCmp</code>	126
.2	More token management	114	.29	<code>\xintiiifEq</code>	126
.3	<code>\xintLen</code>	114	.30	<code>\xintiiifGt</code>	127
.4	<code>\xintReverseDigits</code>	115	.31	<code>\xintiiifLt</code>	127
.5	<code>\xintiiE</code>	116	.32	<code>\xintiiifZero</code>	128
.6	<code>\xintDecSplit</code>	116	.33	<code>\xintiiifNotZero</code>	128
.7	<code>\xintDecSplitL</code>	118	.34	<code>\xintiiifOne</code>	129
.8	<code>\xintDecSplitR</code>	118	.35	<code>\xintiiifOdd</code>	129
.9	<code>\xintDSHr</code>	118	.36	<code>\xintifTrueAelseB, \xintifFalseAelseB</code>	129
.10	<code>\xintDSH</code>	119	.37	<code>\xintIsTrue, \xintIsFalse</code>	130
.11	<code>\xintDSx</code>	119	.38	<code>\xintNOT</code>	130
.12	<code>\xintiiEq</code>	121	.39	<code>\xintAND, \xintOR, \xintXOR</code>	130
.13	<code>\xintiiNotEq</code>	121	.40	<code>\xintANDof</code>	130
.14	<code>\xintiiGeq</code>	122	.41	<code>\xintORof</code>	131
.15	<code>\xintiiGt</code>	122	.42	<code>\xintXORof</code>	131
.16	<code>\xintiiLt</code>	123	.43	<code>\xintiiMax</code>	131
.17	<code>\xintiiGtorEq</code>	123	.44	<code>\xintiiMin</code>	132
.18	<code>\xintiiLtorEq</code>	123	.45	<code>\xintiiMaxof</code>	134
.19	<code>\xintiiIsZero</code>	123	.46	<code>\xintiiMinof</code>	134
.20	<code>\xintiiIsNotZero</code>	123	.47	<code>\xintiiSum</code>	135
.21	<code>\xintiiIsOne</code>	123	.48	<code>\xintiiPrd</code>	135
.22	<code>\xintiiOdd</code>	124	.49	<code>\xintiiSquareRoot</code>	135
.23	<code>\xintiiEven</code>	124	.50	<code>\xintiiSqrt, \xintiiSqrtR</code>	143
.24	<code>\xintiiMON</code>	124	.51	<code>\xintiiBinomial</code>	143
.25	<code>\xintiiMMON</code>	125	.52	<code>\xintiiPFactorial</code>	149
.26	<code>\xintSgnFork</code>	125	.53	<code>\xintBool, \xintToggle</code>	152
.27	<code>\xintiiifSgn</code>	126		At End of L ^A T _E X Document deprecation message	153

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 [2018/02/06 1.2q Expandable operations on big integers (JFB)]%

```

4.2 More token management

```

47 \long\def\xint_firstofthree #1#2#3{#1}%
48 \long\def\xint_secondofthree #1#2#3{#2}%
49 \long\def\xint_thirdofthree #1#2#3{#3}%
50 \long\def\xint_firstofthree_thenstop #1#2#3{ #1}% 1.09i
51 \long\def\xint_secondofthree_thenstop #1#2#3{ #2}%
52 \long\def\xint_thirdofthree_thenstop #1#2#3{ #3}%

```

4.3 `\xintLen`

`\xintLen` gets extended to fractions by `xintfrac.sty`: A/B is given length $\text{len}(A)+\text{len}(B)-1$ (somewhat arbitrary). It applies `\xintNum` to its argument. A minus sign is accepted and ignored.

For parallelism with `\xintiNum`/`\xintNum`, 1.2o defines `\xintilen`.

```

53 \def\xintilen {\romannumeral0\xintilen }%
54 \def\xintilen #1{\def\xintilen ##1%
55 {%
56   \expandafter#1\the\numexpr
57   \expandafter\XINT_len_fork\romannumeral0\xintinum{##1}%
58   \xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
59   \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
60   \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye\relax
61 }}\xintilen{ }%
62 \def\xintLen {\romannumeral0\xintlen }%
63 \let\xintlen\xintilen

```

```

64 \def\XINT_len_fork #1%
65 {%
66   \expandafter\XINT_length_loop\xint_UDsignfork#1{-#1\krof
67 }%

```

4.4 \xintReverseDigits

1.2.

This puts digits in reverse order, not suppressing leading zeros after reverse. Despite lacking the "ii" in its name, it does not apply \xintNum to its argument (contrarily to \xintLen, this is not very coherent).

1.2l variant is robust against non terminated \the\numexpr input.

This macro is currently not used elsewhere in xint code.

```

68 \def\xintReverseDigits {\romannumeral0\xintreversedigits }%
69 \def\xintreversedigits #1%
70 {%
71   \expandafter\XINT_revdigits\romannumeral`&&@#1%
72   {\XINT_microrevsep_end\W}\XINT_microrevsep_end
73   \XINT_microrevsep_end\XINT_microrevsep_end
74   \XINT_microrevsep_end\XINT_microrevsep_end
75   \XINT_microrevsep_end\XINT_microrevsep_end\XINT_microrevsep_end\Z
76   1\Z!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
77 }%
78 \def\XINT_revdigits #1%
79 {%
80   \xint_UDsignfork
81   #1{\expandafter-\romannumeral0\XINT_revdigits_a}%
82   -{\XINT_revdigits_a #1}%
83   \krof
84 }%
85 \def\XINT_revdigits_a
86 {%
87   \expandafter\XINT_revdigits_b\expandafter{\expandafter}%
88   \the\numexpr\XINT_microrevsep
89 }%
90 \def\XINT_microrevsep #1#2#3#4#5#6#7#8#9%
91 {%
92   1#9#8#7#6#5#4#3#2#1\expandafter!\the\numexpr\XINT_microrevsep
93 }%
94 \def\XINT_microrevsep_end #1\W #2\expandafter #3\Z{\relax#2!}%
95 \def\XINT_revdigits_b #11#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
96 {%
97   \xint_gob_til_R #9\XINT_revdigits_end\R
98   \XINT_revdigits_b {#9#8#7#6#5#4#3#2#1}%
99 }%
100 \def\XINT_revdigits_end#1{%
101 \def\XINT_revdigits_end\R\XINT_revdigits_b ##1##2\W
102   {\expandafter#1\xint_gob_til_Z ##1}%
103 }\XINT_revdigits_end{ }%
104 \let\xintRev\xintReverseDigits

```

4.5 `\xintiiE`

Originally was used in `\xintiiexpr`. Transferred from `xintfrac` for 1.1. Code rewritten for 1.2i. `\xintiiE{x}{e}` extends `x` with `e` zeroes if `e` is positive and simply outputs `x` if `e` is zero or negative. Attention, le comportement pour `e < 0` ne doit pas être modifié car `\xintMod` et autres macros en dépendent.

```

105 \def\xintiiE {\romannumeral0\xintiiE }%
106 \def\xintiiE #1#2%
107   {\expandafter\XINT_iiE_fork\the\numexpr #2\expandafter.\romannumeral`&&@#1;}%
108 \def\XINT_iiE_fork #1%
109 {%
110   \xint_UDsignfork
111   #1\XINT_iiE_neg
112   -\XINT_iiE_a
113   \krof #1%
114 }%

```

le #2 a le bon pattern terminé par ; #1=0 est OK pour `\XINT_rep`.

```

115 \def\XINT_iiE_a #1.%
116 {\expandafter\XINT_dsx_append\romannumeral\XINT_rep #1\endcsname 0.}%
117 \def\XINT_iiE_neg #1.#2;{ #2}%

```

4.6 `\xintDecSplit`

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).

Attention: `\xintDecSplit` not robust against non terminated second argument.

```

118 \def\xintDecSplit {\romannumeral0\xintdecsplit }%
119 \def\xintdecsplit #1#2%
120 {%
121   \expandafter\XINT_split_finish
122   \romannumeral0\expandafter\XINT_split_xfork
123   \the\numexpr #1\expandafter.\romannumeral`&&@#2%
124   \xint_bye2345678\xint_bye..%
125 }%
126 \def\XINT_split_finish #1.#2.{{#1}{#2}}%

127 \def\XINT_split_xfork #1%
128 {%
129   \xint_UDzerominusfork
130   #1-\XINT_split_zerosplit

```

4 Package `xint` implementation

```

131      \xint_split_fromleft
132      \xint_split_fromright #1}%
133      \kroft
134 }%
135 \def\xint_split_zerosplit .#1\xint_bye#2\xint_bye..{ #1..}%
136 \def\xint_split_fromleft
137      {\expandafter\xint_split_fromleft_a\the\numexpr\xint_c_viii-}%
138 \def\xint_split_fromleft_a #1%
139 {%
140      \xint_UDsignfork
141      #1\xint_split_fromleft_b
142      -{\xint_split_fromleft_end_a #1}%
143      \kroft
144 }%
145 \def\xint_split_fromleft_b #1.#2#3#4#5#6#7#8#9%
146 {%
147      \expandafter\xint_split_fromleft_clean
148      \the\numexpr1#2#3#4#5#6#7#8#9\expandafter
149      \xint_split_fromleft_a\the\numexpr\xint_c_viii-#1.%
150 }%
151 \def\xint_split_fromleft_end_a #1.%
152 {%
153      \expandafter\xint_split_fromleft_clean
154      \the\numexpr1\csname XINT_split_fromleft_end#1\endcsname
155 }%
156 \def\xint_split_fromleft_clean 1{ }%
157 \expandafter\def\csname XINT_split_fromleft_end7\endcsname #1%
158      {#1\xint_split_fromleft_end_b}%
159 \expandafter\def\csname XINT_split_fromleft_end6\endcsname #1#2%
160      {#1#2\xint_split_fromleft_end_b}%
161 \expandafter\def\csname XINT_split_fromleft_end5\endcsname #1#2#3%
162      {#1#2#3\xint_split_fromleft_end_b}%
163 \expandafter\def\csname XINT_split_fromleft_end4\endcsname #1#2#3#4%
164      {#1#2#3#4\xint_split_fromleft_end_b}%
165 \expandafter\def\csname XINT_split_fromleft_end3\endcsname #1#2#3#4#5%
166      {#1#2#3#4#5\xint_split_fromleft_end_b}%
167 \expandafter\def\csname XINT_split_fromleft_end2\endcsname #1#2#3#4#5#6%
168      {#1#2#3#4#5#6\xint_split_fromleft_end_b}%
169 \expandafter\def\csname XINT_split_fromleft_end1\endcsname #1#2#3#4#5#6#7%
170      {#1#2#3#4#5#6#7\xint_split_fromleft_end_b}%
171 \expandafter\def\csname XINT_split_fromleft_end0\endcsname #1#2#3#4#5#6#7#8%
172      {#1#2#3#4#5#6#7#8\xint_split_fromleft_end_b}%

173 \def\xint_split_fromleft_end_b #1\xint_bye#2\xint_bye.{.#1}% puis .
174 \def\xint_split_fromright #1.#2\xint_bye
175 {%
176      \expandafter\xint_split_fromright_a
177      \the\numexpr#1-\numexpr\xint_length_loop
178      #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
179      \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
180      \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye

```

```

181     .#2\xint_bye
182 }%

183 \def\xint_split_fromright_a #1%
184 {%
185     \xint_UDsignfork
186     #1\xint_split_fromleft
187     -\xint_split_fromright_Lempty
188     \krof
189 }%
190 \def\xint_split_fromright_Lempty #1.#2\xint_bye#3..{.#2.}%

```

4.7 \xintDecSplitL

```

191 \def\xintDecSplitL {\romannumeral0\xintdecsplitl }%
192 \def\xintdecsplitl #1#2%
193 {%
194     \expandafter\xint_splitl_finish
195     \romannumeral0\expandafter\xint_split_xfork
196     \the\numexpr #1\expandafter.\romannumeral`&&@#2%
197     \xint_bye2345678\xint_bye..%
198 }%
199 \def\xint_splitl_finish #1.#2.{ #1}%

```

4.8 \xintDecSplitR

```

200 \def\xintDecSplitR {\romannumeral0\xintdecsplitr }%
201 \def\xintdecsplitr #1#2%
202 {%
203     \expandafter\xint_splitr_finish
204     \romannumeral0\expandafter\xint_split_xfork
205     \the\numexpr #1\expandafter.\romannumeral`&&@#2%
206     \xint_bye2345678\xint_bye..%
207 }%
208 \def\xint_splitr_finish #1.#2.{ #2}%

```

4.9 \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.

```

209 \def\xintDSHr {\romannumeral0\xintdshr }%

```

```

210 \def\xintdshr #1#2%
211 {%

```

```

212 \expandafter\XINT_dshr_fork\the\numexpr#1\expandafter.\romannumeral`&&@#2;%
213 }%
214 \def\XINT_dshr_fork #1%
215 {%
216 \xint_UDzerominusfork
217 0#1\XINT_dshr_xzeroorneg
218 #1-\XINT_dshr_xzeroorneg
219 0-\XINT_dshr_xpositive
220 \krof #1%
221 }%
222 \def\XINT_dshr_xzeroorneg #1;{ 0}%
223 \def\XINT_dshr_xpositive
224 {%

225 \expandafter\xint_secondoftwo_thenstop\romannumeral0\XINT_dsx_xisPos
226 }%

```

4.10 \xintDSH

```

227 \def\xintDSH {\romannumeral0\xintdsh }%
228 \def\xintdsh #1#2%
229 {%

```

```

230 \expandafter\XINT_dsh_fork\the\numexpr#1\expandafter.\romannumeral`&&@#2;%
231 }%
232 \def\XINT_dsh_fork #1%
233 {%
234 \xint_UDzerominusfork
235 #1-\XINT_dsh_xiszero

236 0#1\XINT_dsx_xisNeg_checkA
237 0-{ \XINT_dsh_xisPos #1}%
238 \krof
239 }%
240 \def\XINT_dsh_xiszero #1.#2;{ #2}%
241 \def\XINT_dsh_xisPos
242 {%

\xexpandafter\xint_firstoftwo_thenstop\romannumeral0\XINT_dsx_xisPos
243 }%

```

4.11 \xintDSx

```

--> Attention le cas x=0 est traité dans la même catégorie que x > 0 <--
si x < 0, fait A -> A.10^(|x|)
si x >= 0, et A >= 0, fait A -> {quo(A,10^(x))}{rem(A,10^(x))}
si x >= 0, et A < 0, d'abord on calcule {quo(-A,10^(x))}{rem(-A,10^(x))}

```

4 Package *xint* implementation

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.

```
244 \def\xintDSx {\romannumeral0\xintdsx}%
245 \def\xintdsx #1#2%
246 {%

247   \expandafter\XINT_dsx_fork\the\numexpr#1\expandafter.\romannumeral`&&@#2;%
248}%
249 \def\XINT_dsx_fork #1%
250 {%
251   \xint_UDzerominusfork
252   #1-\XINT_dsx_xisZero
253   0#1\XINT_dsx_xisNeg_checkA
254   0-\XINT_dsx_xisPos #1}%
255   \krof
256}%
257 \def\XINT_dsx_xisZero #1.#2;{{#2}{0}}%
258 \def\XINT_dsx_xisNeg_checkA #1.#2%
259 {%
260   \xint_gob_til_zero #2\XINT_dsx_xisNeg_Azero 0%

261   \expandafter\XINT_dsx_append\romannumeral\XINT_rep #1\endcsname 0.#2%
262}%
263 \def\XINT_dsx_xisNeg_Azero #1;{ 0}%

264 \def\XINT_dsx_addzeros #1%
265   {\expandafter\XINT_dsx_append\romannumeral\XINT_rep#1\endcsname0.}%

266 \def\XINT_dsx_addzerosnofuss #1%
267   {\expandafter\XINT_dsx_append\romannumeral\xintreplicate{#1}0.}%
268 \def\XINT_dsx_append #1.#2;{ #2#1}%

269 \def\XINT_dsx_xisPos #1.#2%
270 {%
271   \xint_UDzerominusfork
272   #2-\XINT_dsx_AisZero
273   0#2\XINT_dsx_AisNeg
274   0-\XINT_dsx_AisPos
275   \krof #1.#2%
276}%
277 \def\XINT_dsx_AisZero #1;{{0}{0}}%
```


4 Package `xint` implementation

```
278 \def\XINT_dsx_AisNeg #1.-#2;%
279 {%
280   \expandafter\XINT_dsx_AisNeg_checkiffirstempty
281   \romannumeral0\XINT_split_xfork #1.#2\xint_bye2345678\xint_bye..%
282 }%

283 \def\XINT_dsx_AisNeg_checkiffirstempty #1%
284 {%
285   \xint_gob_til_dot #1\XINT_dsx_AisNeg_finish_zero.%
286   \XINT_dsx_AisNeg_finish_notzero #1%
287 }%
288 \def\XINT_dsx_AisNeg_finish_zero.\XINT_dsx_AisNeg_finish_notzero.#1.%
289 {%
290   \expandafter\XINT_dsx_end
291   \expandafter {\romannumeral0\XINT_num {-#1}}{0}%
292 }%
293 \def\XINT_dsx_AisNeg_finish_notzero #1.#2.%
294 {%
295   \expandafter\XINT_dsx_end
296   \expandafter {\romannumeral0\XINT_num {#2}}{-#1}%
297 }%

298 \def\XINT_dsx_AisPos #1.#2;%
299 {%
300   \expandafter\XINT_dsx_AisPos_finish
301   \romannumeral0\XINT_split_xfork #1.#2\xint_bye2345678\xint_bye..%
302 }%

303 \def\XINT_dsx_AisPos_finish #1.#2.%
304 {%
305   \expandafter\XINT_dsx_end
306   \expandafter {\romannumeral0\XINT_num {#2}}%
307   {\romannumeral0\XINT_num {#1}}%
308 }%
309 \def\XINT_dsx_end #1#2{\expandafter{#2}{#1}}%
```

4.12 `\xintiiEq`

`no \xintiiEq.`

```
310 \def\xintEq   {\romannumeral0\XINT_signaldeprecated{xint}{xintEq}\xinteq }%
311 \def\xinteq   #1#2{\xintifeq{#1}{#2}{1}{0}}%
312 \def\xintiiEq #1#2{\romannumeral0\xintiiifeq{#1}{#2}{1}{0}}%
```

4.13 `\xintiiNotEq`

`Pour xintexpr. Pas de version en lowercase.`

```
313 \def\xintNeq  #1#2{\romannumeral0\XINT_signaldeprecated{xint}{xintNeq}\xintifeq {#1}{#2}{0}{1}}%
314 \def\xintiiNotEq #1#2{\romannumeral0\xintiiifeq {#1}{#2}{0}{1}}%
```

4.14 `\xintiiGeq`

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

1.2l made `\xintiiGeq` robust against non terminated items.

1.2l rewrote `\xintiiCmp`, but forgot to handle `\xintiiGeq` too. Done at 1.2m.

This macro should have been called `\xintGEq` for example.

```

315 \def\xintGeq    {\romannumeral0\XINT_signaldeprecated{xint}{xintGeq}\xintgeq }%
316 \def\xintgeq    #1{\expandafter\XINT_geq\romannumeral0\xintnum{#1}\xint:}%
317 \def\xintiiGeq  {\romannumeral0\xintiigeq }%
318 \def\xintiigeq  #1{\expandafter\XINT_iigeq\romannumeral`&&@#1\xint:}%
319 \def\XINT_iigeq #1#2\xint:#3%
320 {%
321   \expandafter\XINT_geq_fork\expandafter #1\romannumeral`&&@#3\xint:#2\xint:
322 }%

323 \def\XINT_geq #1#2\xint:#3%
324 {%
325   \expandafter\XINT_geq_fork\expandafter #1\romannumeral0\xintnum{#3}\xint:#2\xint:
326 }%
327 \def\XINT_geq_fork #1#2%
328 {%
329   \xint_UDzerofork
330   #1\XINT_geq_firstiszero
331   #2\XINT_geq_secondiszero
332   0{}}%
333   \krof
334   \xint_UDsignsfork
335   #1#2\XINT_geq_minusminus
336   #1-\XINT_geq_minusplus
337   #2-\XINT_geq_plusminus
338   --\XINT_geq_plusplus
339   \krof #1#2%
340 }%
341 \def\XINT_geq_firstiszero #1\krof 0#2#3\xint:#4\xint:
342   {\xint_UDzerofork #2{ 1}0{ 0}\krof }%
343 \def\XINT_geq_secondiszero #1\krof #20#3\xint:#4\xint:{ 1}%
344 \def\XINT_geq_plusminus #1-{\XINT_geq_plusplus #1{}}%
345 \def\XINT_geq_minusplus -#1{\XINT_geq_plusplus { }#1}%
346 \def\XINT_geq_minusminus --{\XINT_geq_plusplus { }{ }}%
347 \def\XINT_geq_plusplus
348   {\expandafter\XINT_geq_finish\romannumeral0\XINT_cmp_plusplus}%
349 \def\XINT_geq_finish #1{\if-#1\expandafter\XINT_geq_no
350   \else\expandafter\XINT_geq_yes\fi}%
351 \def\XINT_geq_no 1{ 0}%
352 \def\XINT_geq_yes { 1}%

```

4.15 `\xintiiGt`

```

353 \def\xintGt {\romannumeral0\XINT_signaldeprecated{xint}{xintGt}\xintgt }%
354 \def\xintgt #1#2{\xintifgt{#1}{#2}{1}{0}}%
355 \def\xintiiGt #1#2{\romannumeral0\xintiiifgt{#1}{#2}{1}{0}}%

```

4.16 `\xintiLt`

```
356 \def\xintLt {\romannumeral0\XINT_signaldeprecated{xint}{xintLt}\xintlt }%
357 \def\xintlt #1#2{\xintiflt{#1}{#2}{1}{0}}%
358 \def\xintiLt #1#2{\romannumeral0\xintiiiflt{#1}{#2}{1}{0}}%
```

4.17 `\xintiiGtorEq`

```
359 \def\xintGtorEq #1#2{\romannumeral0\XINT_signaldeprecated{xint}{xintGtorEq}\xintiflt {#1}{#2}{0}{1}}%
360 \def\xintiiGtorEq #1#2{\romannumeral0\xintiiiflt {#1}{#2}{0}{1}}%
```

4.18 `\xintiiLtorEq`

```
361 \def\xintLtorEq #1#2{\romannumeral0\XINT_signaldeprecated{xint}{xintLtorEq}\xintifgt {#1}{#2}{0}{1}}%
362 \def\xintiiLtorEq #1#2{\romannumeral0\xintiiifgt {#1}{#2}{0}{1}}%
```

4.19 `\xintiiIsZero`

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

```
363 \def\xintIsZero {\romannumeral0\XINT_signaldeprecated{xint}{xintIsZero}\xintiszero }%
364 \def\xintiszero #1{\if0\xintSgn{#1}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%
365 \def\xintiiIsZero {\romannumeral0\xintiiiszero }%
366 \def\xintiiiszero #1{\if0\xintiiSgn{#1}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%
```

4.20 `\xintiiIsNotZero`

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

```
367 \def\xintIsNotZero {\romannumeral0\XINT_signaldeprecated{xint}{xintIsNotZero}\xintisnotzero }%
368 \def\xintisnotzero
369     #1{\if0\xintSgn{#1}\xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi}%
370 \def\xintiiIsNotZero {\romannumeral0\xintiiisnotzero }%
371 \def\xintiiisnotzero
372     #1{\if0\xintiiSgn{#1}\xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi}%
```

4.21 `\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.

```
373 \def\xintiiIsOne {\romannumeral0\xintiiisone }%
374 \def\xintiiisone #1{\expandafter\XINT_isone\romannumeral`&&@#1XY}%
375 \def\xintIsOne {\romannumeral0\XINT_signaldeprecated{xint}{xintIsOne}\xintisone }%
376 \def\xintisone #1{\expandafter\XINT_isone\romannumeral0\xintnum{#1}XY}%
377 \def\XINT_isone #1#2#3Y%
378 {%
379     \unless\if#2X\xint_dothis{ 0}\fi
380     \unless\if#11\xint_dothis{ 0}\fi
381     \xint_orthat{ 1}%
382 }%
383 \def\XINT_isOne #1{\XINT_is_One#1XY}%
384 \def\XINT_is_One #1#2#3Y%
385 {%
386     \unless\if#2X\xint_dothis0\fi
387     \unless\if#11\xint_dothis0\fi
```

```

388 \xint_orthat1%
389 }%

```

4.22 `\xintiiOdd`

`\xintOdd` is needed for the `xintexpr`-essions `even()` and `odd()` functions (and also by `\xintNewExpr`).

```

390 \def\xintiiOdd {\romannumeral0\xintiiodd }%
391 \def\xintiiodd #1%
392 {%
393   \ifodd\xintLDg{#1} %<- intentional space
394     \xint_afterfi{ 1}%
395   \else
396     \xint_afterfi{ 0}%
397   \fi
398 }%
399 \def\xintOdd {\romannumeral0\XINT_signaldeprecated{xint}{xintOdd}\xintodd }%
400 \def\xintodd #1%
401 {%
402   \ifodd\xintLDg{\xintNum{#1}} %<- intentional space
403     \xint_afterfi{ 1}%
404   \else
405     \xint_afterfi{ 0}%
406   \fi
407 }%

```

4.23 `\xintiiEven`

```

408 \def\xintiiEven {\romannumeral0\xintiieven }%
409 \def\xintiieven #1%
410 {%
411   \ifodd\xintLDg{#1} %<- intentional space
412     \xint_afterfi{ 0}%
413   \else
414     \xint_afterfi{ 1}%
415   \fi
416 }%
417 \def\xintEven {\romannumeral0\XINT_signaldeprecated{xint}{xintEven}\xinteven }%
418 \def\xinteven #1%
419 {%
420   \ifodd\xintLDg{\xintNum{#1}} %<- intentional space
421     \xint_afterfi{ 0}%
422   \else
423     \xint_afterfi{ 1}%
424   \fi
425 }%

```

4.24 `\xintiiMON`

MINUS ONE TO THE POWER N

```

426 \def\xintiiMON {\romannumeral0\xintiimon }%
427 \def\xintiimon #1%
428 {%

```

```

429 \ifodd\xintLDg {#1} %<- intentional space
430 \xint_afterfi{ -1}%
431 \else
432 \xint_afterfi{ 1}%
433 \fi
434 }%
435 \def\xintMON {\romannumeral0\XINT_signaldeprecated{xint}{xintMON}\xintmon }%
436 \def\xintmon #1%
437 {%
438 \ifodd\xintLDg{\xintNum{#1}} %<- intentional space
439 \xint_afterfi{ -1}%
440 \else
441 \xint_afterfi{ 1}%
442 \fi
443 }%

```

4.25 \xintiiMMON

MINUS ONE TO THE POWER N-1

```

444 \def\xintiiMMON {\romannumeral0\xintiimmon }%
445 \def\xintiimmon #1%
446 {%
447 \ifodd\xintLDg {#1} %<- intentional space
448 \xint_afterfi{ 1}%
449 \else
450 \xint_afterfi{ -1}%
451 \fi
452 }%
453 \def\xintMMON {\romannumeral0\XINT_signaldeprecated{xint}{xintMMON}\xintmmon }%
454 \def\xintmmon #1%
455 {%
456 \ifodd\xintLDg{\xintNum{#1}} %<- intentional space
457 \xint_afterfi{ 1}%
458 \else
459 \xint_afterfi{ -1}%
460 \fi
461 }%

```

4.26 \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`.

```

462 \def\xintSgnFork {\romannumeral0\xintsgnfork }%
463 \def\xintsgnfork #1%
464 {%
465 \ifcase #1 \expandafter\xint_secondofthree_thenstop
466 \or\expandafter\xint_thirdofthree_thenstop
467 \else\expandafter\xint_firstofthree_thenstop
468 \fi
469 }%

```

4.27 `\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`)

```

470 \def\xintifSgn {\romannumeral0\XINT_signaldeprecated{xint}{xintifSgn}\xintifsgn }%
471 \def\xintifsgn #1%
472 {%
473   \ifcase \xintSgn{#1}
474     \expandafter\xint_secondofthree_thenstop
475     \or\expandafter\xint_thirdofthree_thenstop
476     \else\expandafter\xint_firstofthree_thenstop
477   \fi
478 }%
479 \def\xintiiifSgn {\romannumeral0\xintiiifsgn }%
480 \def\xintiiifsgn #1%
481 {%
482   \ifcase \xintiiSgn{#1}
483     \expandafter\xint_secondofthree_thenstop
484     \or\expandafter\xint_thirdofthree_thenstop
485     \else\expandafter\xint_firstofthree_thenstop
486   \fi
487 }%

```

4.28 `\xintiiifCmp`

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

```

488 \def\xintifCmp {\romannumeral0\XINT_signaldeprecated{xint}{xintifCmp}\xintifcmp }%
489 \def\xintifcmp #1#2%
490 {%
491   \ifcase\xintCmp {#1}{#2}
492     \expandafter\xint_secondofthree_thenstop
493     \or\expandafter\xint_thirdofthree_thenstop
494     \else\expandafter\xint_firstofthree_thenstop
495   \fi
496 }%
497 \def\xintiiifCmp {\romannumeral0\xintiiifcmp }%
498 \def\xintiiifcmp #1#2%
499 {%
500   \ifcase\xintiiCmp {#1}{#2}
501     \expandafter\xint_secondofthree_thenstop
502     \or\expandafter\xint_thirdofthree_thenstop
503     \else\expandafter\xint_firstofthree_thenstop
504   \fi
505 }%

```

4.29 `\xintiiifEq`

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

```

506 \def\xintifEq {\romannumeral0\XINT_signaldeprecated{xint}{xintifEq}\xintifeq }%
507 \def\xintifeq #1#2%
508 {%
509     \if0\xintCmp{#1}{#2}%
510         \expandafter\xint_firstoftwo_thenstop
511     \else\expandafter\xint_secondoftwo_thenstop
512     \fi
513 }%
514 \def\xintiiifEq {\romannumeral0\xintiiifeq }%
515 \def\xintiiifeq #1#2%
516 {%
517     \if0\xintiiCmp{#1}{#2}%
518         \expandafter\xint_firstoftwo_thenstop
519     \else\expandafter\xint_secondoftwo_thenstop
520     \fi
521 }%

```

4.30 \xintiiifGt

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

```

522 \def\xintifGt {\romannumeral0\XINT_signaldeprecated{xint}{xintifGt}\xintifgt }%
523 \def\xintifgt #1#2%
524 {%
525     \if1\xintCmp{#1}{#2}%
526         \expandafter\xint_firstoftwo_thenstop
527     \else\expandafter\xint_secondoftwo_thenstop
528     \fi
529 }%
530 \def\xintiiifGt {\romannumeral0\xintiiifgt }%
531 \def\xintiiifgt #1#2%
532 {%
533     \if1\xintiiCmp{#1}{#2}%
534         \expandafter\xint_firstoftwo_thenstop
535     \else\expandafter\xint_secondoftwo_thenstop
536     \fi
537 }%

```

4.31 \xintiiifLt

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

```

538 \def\xintifLt {\romannumeral0\XINT_signaldeprecated{xint}{xintifLt}\xintiflt }%
539 \def\xintiflt #1#2%
540 {%
541     \ifnum\xintCmp{#1}{#2}<\xint_c_
542         \expandafter\xint_firstoftwo_thenstop
543     \else \expandafter\xint_secondoftwo_thenstop
544     \fi
545 }%
546 \def\xintiiifLt {\romannumeral0\xintiiiflt }%
547 \def\xintiiiflt #1#2%
548 {%

```

```

549 \ifnum\xintiiCmp{#1}{#2}<\xint_c_
550 \expandafter\xint_firstoftwo_thenstop
551 \else \expandafter\xint_secondoftwo_thenstop
552 \fi
553 }%

```

4.32 `\xintiiifZero`

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.

1.2o deprecates `\xintifZero`.

```

554 \def\xintifZero {\romannumeral0\XINT_signaldeprecated{xint}{xintifZero}\xintifzero }%
555 \def\xintifzero #1%
556 {%
557 \if0\xintSgn{#1}%
558 \expandafter\xint_firstoftwo_thenstop
559 \else
560 \expandafter\xint_secondoftwo_thenstop
561 \fi
562 }%
563 \def\xintiiifZero {\romannumeral0\xintiiifzero }%
564 \def\xintiiifzero #1%
565 {%
566 \if0\xintiiSgn{#1}%
567 \expandafter\xint_firstoftwo_thenstop
568 \else
569 \expandafter\xint_secondoftwo_thenstop
570 \fi
571 }%

```

4.33 `\xintiiifNotZero`

```

572 \def\xintifNotZero {\romannumeral0\XINT_signaldeprecated{xint}{xintifNotZero}\xintifnotzero }%
573 \def\xintifnotzero #1%
574 {%
575 \if0\xintSgn{#1}%
576 \expandafter\xint_secondoftwo_thenstop
577 \else
578 \expandafter\xint_firstoftwo_thenstop
579 \fi
580 }%
581 \def\xintiiifNotZero {\romannumeral0\xintiiifnotzero }%
582 \def\xintiiifnotzero #1%
583 {%
584 \if0\xintiiSgn{#1}%
585 \expandafter\xint_secondoftwo_thenstop
586 \else
587 \expandafter\xint_firstoftwo_thenstop
588 \fi
589 }%

```


4.34 `\xintiiifOne`

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

```

590 \def\xintiiifOne {\romannumeral0\xintiiifone }%
591 \def\xintiiifone #1%
592 {%
593   \if1\xintiiIsOne{#1}%
594     \expandafter\xint_firstoftwo_thenstop
595   \else
596     \expandafter\xint_secondoftwo_thenstop
597   \fi
598 }%
599 \def\xintifOne {\romannumeral0\XINT_signaldeprecated{xint}{xintifOne}\xintifone }%
600 \def\xintifone #1%
601 {%
602   \if1\xintIsOne{#1}%
603     \expandafter\xint_firstoftwo_thenstop
604   \else
605     \expandafter\xint_secondoftwo_thenstop
606   \fi
607 }%

```

4.35 `\xintiiifOdd`

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

```

608 \def\xintiiifOdd {\romannumeral0\xintiiifodd }%
609 \def\xintiiifodd #1%
610 {%
611   \if\xintiiOdd{#1}1%
612     \expandafter\xint_firstoftwo_thenstop
613   \else
614     \expandafter\xint_secondoftwo_thenstop
615   \fi
616 }%
617 \def\xintifOdd {\romannumeral0\XINT_signaldeprecated{xint}{xintifOdd}\xintifodd }%
618 \def\xintifodd #1%
619 {%
620   \if\xintOdd{#1}1%
621     \expandafter\xint_firstoftwo_thenstop
622   \else
623     \expandafter\xint_secondoftwo_thenstop
624   \fi
625 }%

```

4.36 `\xintifTrueAelseB`, `\xintifFalseAelseB`

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

1.2o uses `\xintiiifNotZero`, see comments to `\xintAND` etc... This will work fine with arguments being nested `xintfrac.sty` macros, without the overhead of `\xintNum` or `\xintRaw` parsing.

```

626 \def\xintifTrueAelseB {\romannumeral0\xintiiifnotzero}%
627 \def\xintifFalseAelseB{\romannumeral0\xintiiifzero}%

```

4.37 `\xintIsTrue`, `\xintIsFalse`

1.09c. Suppressed at 1.2o. They seem not to have been documented, fortunately.

```
628 %\let\xintIsTrue \xintIsNotZero
629 %\let\xintIsFalse\xintIsZero
```

4.38 `\xintNOT`

1.09c. But it should have been called `\xintNOT`, not `\xintNot`. Former denomination deprecated at 1.2o. Besides, the macro is now defined as `ii`-type.

```
630 \def\xintNot{\romannumeral0\XINT_signaldeprecated{xint}{xintNot}\xintiiszero}%
631 \def\xintNOT{\romannumeral0\xintiiszero}%
```

4.39 `\xintAND`, `\xintOR`, `\xintXOR`

Added with 1.09a. But they used `\xintSgn`, etc... rather than `\xintiiSgn`. This brings `\xintNum` overhead which is not really desired, and which is not needed for use by `xintexpr.sty`. At 1.2o I modify them to use only `ii` macros. This is enough for sign or zeroness even for `xintfrac` format, as manipulated inside the `\xintexpr`. Big hesitation whether there should be however `\xintiiAND` outputting 1 or 0 versus an `\xintAND` outputting 1[0] versus 0[0] for example.

```
632 \def\xintAND {\romannumeral0\xintand }%
633 \def\xintand #1#2{\if0\xintiiSgn{#1}\expandafter\xint_firstoftwo
634 \else\expandafter\xint_secondoftwo\fi
635 { 0}{\xintiisnotzero{#2}}}%
636 \def\xintOR {\romannumeral0\xintor }%
637 \def\xintor #1#2{\if0\xintiiSgn{#1}\expandafter\xint_firstoftwo
638 \else\expandafter\xint_secondoftwo\fi
639 {\xintiisnotzero{#2}}{ 1}}%
640 \def\xintXOR {\romannumeral0\xintxor }%
641 \def\xintxor #1#2{\if\xintiiIsZero{#1}\xintiiIsZero{#2}%
642 \xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi }%
```

4.40 `\xintANDof`

New with 1.09a. `\xintANDof` works also with an empty list. Empty items however are not accepted.

1.2l made `\xintANDof` robust against non terminated items.

1.2o's `\xintifTrueAelseB` is now an `ii` macro, actually.

This macro as well as `ORof` and `XORof` are actually not used by `xintexpr`, which has its own `csv` handling macros.

```
643 \def\xintANDof {\romannumeral0\xintandof }%
644 \def\xintandof #1{\expandafter\XINT_andof_a\romannumeral`&&@#1\xint:}%
645 \def\XINT_andof_a #1{\expandafter\XINT_andof_b\romannumeral`&&@#1!}%
646 \def\XINT_andof_b #1%
647 {\xint_gob_til_xint: #1\XINT_andof_e\xint:\XINT_andof_c #1}%
648 \def\XINT_andof_c #1!%
649 {\xintifTrueAelseB {#1}{\XINT_andof_a}{\XINT_andof_no}}%
650 \def\XINT_andof_no #1\xint:{ 0}%
651 \def\XINT_andof_e #1!{ 1}%
```

4.41 `\xintORof`

New with 1.09a. Works also with an empty list. Empty items however are not accepted.
1.2l made `\xintORof` robust against non terminated items.

```

652 \def\xintORof      {\romannumeral0\xintorof }%
653 \def\xintorof      #1{\expandafter\XINT_orof_a\romannumeral`&&@#1\xint:}%
654 \def\XINT_orof_a #1{\expandafter\XINT_orof_b\romannumeral`&&@#1!}%
655 \def\XINT_orof_b #1%
656     {\xint_gob_til_xint: #1\XINT_orof_e\xint:\XINT_orof_c #1}%
657 \def\XINT_orof_c #1!%
658     {\xintifTrueAelseB {#1}{\XINT_orof_yes}{\XINT_orof_a}}%
659 \def\XINT_orof_yes #1\xint:{ 1}%
660 \def\XINT_orof_e   #1!{ 0}%

```

4.42 `\xintXORof`

New with 1.09a. Works with an empty list, too. Empty items however are not accepted. `\XINT_xorof_c` more efficient in 1.09i.

1.2l made `\xintXORof` robust against non terminated items.

```

661 \def\xintXORof      {\romannumeral0\xintxorof }%
662 \def\xintxorof      #1{\expandafter\XINT_xorof_a\expandafter
663     0\romannumeral`&&@#1\xint:}%
664 \def\XINT_xorof_a #1#2{\expandafter\XINT_xorof_b\romannumeral`&&@#2!#1}%
665 \def\XINT_xorof_b #1%
666     {\xint_gob_til_xint: #1\XINT_xorof_e\xint:\XINT_xorof_c #1}%
667 \def\XINT_xorof_c #1!#2%
668     {\xintifTrueAelseB {#1}{\if #20\xint_afterfi{\XINT_xorof_a 1}%
669     \else\xint_afterfi{\XINT_xorof_a 0}\fi}%
670     {\XINT_xorof_a #2}%
671     }%
672 \def\XINT_xorof_e #1!#2{ #2}%

```

4.43 `\xintiiMax`

At 1.2m, a long-standing bug was fixed: `\xintiiMax` had the overhead of applying `\xintNum` to its arguments due to use of a sub-macro of `\xintGeq` code to which this overhead was added at some point.
And on this occasion I reduced even more number of times input is grabbed.

```

673 \def\xintiMax {\romannumeral0\XINT_signaldeprecated{xint}{xintiMax}\xintimax }%
674 \def\xintimax #1%
675 {%
676     \expandafter\xint_max\romannumeral0\xintnum{#1}\xint:
677 }%
678 \def\xint_max #1\xint:#2%
679 {%
680     \expandafter\XINT_max_fork\romannumeral0\xintnum{#2}\xint:#1\xint:
681 }%
682 \def\xintiiMax {\romannumeral0\xintiimax }%
683 \def\xintiimax #1%
684 {%
685     \expandafter\xint_iimax \romannumeral`&&@#1\xint:

```

4 Package `xint` implementation

```

686 }%
687 \def\xint_iimax #1\xint:#2%
688 {%
689   \expandafter\XINT_max_fork\romannumeral`&&@#2\xint:#1\xint:
690 }%

```

`#3#4` vient du **premier**, `#1#2` vient du **second**. I have renamed the sub-macros at 1.2m because the terminology was quite counter-intuitive; there was no bug, but still.

```

691 \def\XINT_max_fork #1#2\xint:#3#4\xint:
692 {%
693   \xint_UDsignsfork
694     #1#3\XINT_max_minusminus % A < 0, B < 0
695     #1-\XINT_max_plusminus % B < 0, A >= 0
696     #3-\XINT_max_minusplus % A < 0, B >= 0
697     --{\xint_UDzerosfork
698       #1#3\XINT_max_zerozero % A = B = 0
699       #10\XINT_max_pluszero % B = 0, A > 0
700       #30\XINT_max_zeropplus % A = 0, B > 0
701       00\XINT_max_plusplus % A, B > 0
702     \krof }%
703   \krof
704   #3#1#2\xint:#4\xint:
705   \expandafter\xint_firstoftwo_thenstop
706   \else
707   \expandafter\xint_secondoftwo_thenstop
708   \fi
709   {#3#4}{#1#2}%
710 }%

```

Refactored at 1.2m for avoiding grabbing arguments. Position of inputs shared with `iiCmp` and `iiGeq` code.

```

711 \def\XINT_max_zerozero #1\fi{\xint_firstoftwo_thenstop }%
712 \def\XINT_max_zeropplus #1\fi{\xint_secondoftwo_thenstop }%
713 \def\XINT_max_pluszero #1\fi{\xint_firstoftwo_thenstop }%
714 \def\XINT_max_minusplus #1\fi{\xint_secondoftwo_thenstop }%
715 \def\XINT_max_plusminus #1\fi{\xint_firstoftwo_thenstop }%
716 \def\XINT_max_plusplus
717 {%
718   \if1\romannumeral0\XINT_geq_plusplus
719 }%

```

Premier des testés $|A|=-A$, second est $|B|=-B$. On veut le $\max(A,B)$, c'est donc A si $|A|<|B|$ (ou $|A|=|B|$, mais peu importe alors). Donc on peut faire cela avec `\unless`. Simple.

```

720 \def\XINT_max_minusminus --%
721 {%
722   \unless\if1\romannumeral0\XINT_geq_plusplus{}}%
723 }%

```

4.44 `\xintiiMin`

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

4 Package *xint* implementation

fracnameimp.

At 1.2m, a long-standing bug was fixed: `\xintiiMin` had the overhead of applying `\xintNum` to its arguments due to use of a sub-macro of `\xintGeq` code to which this overhead was added at some point.

And on this occasion I reduced even more number of times input is grabbed.

```

724 \def\xintiMin {\romannumeral0\XINT_signaldeprecated{xint}{\xintiMin}\xintimin }%
725 \def\xintimin #1%
726 {%
727   \expandafter\xint_min\romannumeral0\xintnum{#1}\xint:
728 }%
729 \def\xint_min #1\xint:#2%
730 {%
731   \expandafter\XINT_min_fork\romannumeral0\xintnum{#2}\xint:#1\xint:
732 }%
733 \def\xintiiMin {\romannumeral0\xintiimin }%
734 \def\xintiimin #1%
735 {%
736   \expandafter\xint_iimin \romannumeral`&&@#1\xint:
737 }%
738 \def\xint_iimin #1\xint:#2%
739 {%
740   \expandafter\XINT_min_fork\romannumeral`&&@#2\xint:#1\xint:
741 }%
742 \def\XINT_min_fork #1#2\xint:#3#4\xint:
743 {%
744   \xint_UDsignsfork
745     #1#3\XINT_min_minusminus % A < 0, B < 0
746     #1-\XINT_min_plusminus % B < 0, A >= 0
747     #3-\XINT_min_minusplus % A < 0, B >= 0
748     --{\xint_UDzerosfork
749       #1#3\XINT_min_zerozero % A = B = 0
750       #10\XINT_min_pluszero % B = 0, A > 0
751       #30\XINT_min_zeroplus % A = 0, B > 0
752       00\XINT_min_plusplus % A, B > 0
753     \krof }%
754   \krof
755   #3#1#2\xint:#4\xint:
756   \expandafter\xint_secondoftwo_thenstop
757   \else
758   \expandafter\xint_firstoftwo_thenstop
759   \fi
760   {#3#4}{#1#2}%
761 }%
762 \def\XINT_min_zerozero #1\fi{\xint_firstoftwo_thenstop }%
763 \def\XINT_min_zeroplus #1\fi{\xint_firstoftwo_thenstop }%
764 \def\XINT_min_pluszero #1\fi{\xint_secondoftwo_thenstop }%
765 \def\XINT_min_minusplus #1\fi{\xint_firstoftwo_thenstop }%
766 \def\XINT_min_plusminus #1\fi{\xint_secondoftwo_thenstop }%
767 \def\XINT_min_plusplus
768 {%
769   \if1\romannumeral0\XINT_geq_plusplus
770 }%
771 \def\XINT_min_minusminus --%

```

```

772 {%
773   \unless\if1\romannumeral0\XINT_geq_plusplus{}}}%
774 }%

```

4.45 `\xintiiMaxof`

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

NOT compatible with empty list.

1.2l made `\xintiiMaxof` robust against non terminated items.

```

775 \def\xintiMaxof      {\romannumeral0\XINT_signaldeprecated{xint}{xintiMaxof}\xintimaxof }%
776 \def\xintimaxof      #1{\expandafter\XINT_imaxof_a\romannumeral`&&@#1\xint:}%
777 \def\XINT_imaxof_a
778 #1{\expandafter\XINT_imaxof_b\romannumeral0\xintnum{#1}!}%

```

No `\xintnum` on #2 which might be `\xint:`, of course. But if list not terminated the `\xintNum` will be done via `\xintimax`.

```

779 \def\XINT_imaxof_b #1!#2%
780   {\expandafter\XINT_imaxof_c\romannumeral`&&@#2!{#1}!}%
781 \def\XINT_imaxof_c #1%
782   {\xint_gob_til_xint: #1\XINT_imaxof_e\xint:\XINT_imaxof_d #1}%
783 \def\XINT_imaxof_d #1!%
784   {\expandafter\XINT_imaxof_b\romannumeral0\xintimax {#1}}%
785 \def\XINT_imaxof_e #1!#2!{ #2}%
786 \def\xintiiMaxof      {\romannumeral0\xintiimaxof }%
787 \def\xintiimaxof      #1{\expandafter\XINT_iimaxof_a\romannumeral`&&@#1\xint:}%
788 \def\XINT_iimaxof_a #1{\expandafter\XINT_iimaxof_b\romannumeral`&&@#1!}%
789 \def\XINT_iimaxof_b #1!#2%
790   {\expandafter\XINT_iimaxof_c\romannumeral`&&@#2!{#1}!}%
791 \def\XINT_iimaxof_c #1%
792   {\xint_gob_til_xint: #1\XINT_iimaxof_e\xint:\XINT_iimaxof_d #1}%
793 \def\XINT_iimaxof_d #1!%
794   {\expandafter\XINT_iimaxof_b\romannumeral0\xintiimax {#1}}%
795 \def\XINT_iimaxof_e #1!#2!{ #2}%

```

4.46 `\xintiiMinof`

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

```

796 \def\xintiMinof      {\romannumeral0\XINT_signaldeprecated{xint}{xintiMinof}\xintiminof }%
797 \def\xintiminof      #1{\expandafter\XINT_iminof_a\romannumeral`&&@#1\xint:}%
798 \def\XINT_iminof_a #1{\expandafter\XINT_iminof_b\romannumeral0\xintnum{#1}!}%
799 \def\XINT_iminof_b #1!#2%
800   {\expandafter\XINT_iminof_c\romannumeral`&&@#2!{#1}!}%
801 \def\XINT_iminof_c #1%
802   {\xint_gob_til_xint: #1\XINT_iminof_e\xint:\XINT_iminof_d #1}%
803 \def\XINT_iminof_d #1!%
804   {\expandafter\XINT_iminof_b\romannumeral0\xintimin {#1}}%
805 \def\XINT_iminof_e #1!#2!{ #2}%
806 \def\xintiiMinof      {\romannumeral0\xintiiminof }%
807 \def\xintiiminof      #1{\expandafter\XINT_iiminof_a\romannumeral`&&@#1\xint:}%

```

```

808 \def\XINT_iiminof_a #1{\expandafter\XINT_iiminof_b\romannumeral`&&@#1!}%
809 \def\XINT_iiminof_b #1!#2%
810     {\expandafter\XINT_iiminof_c\romannumeral`&&@#2!{#1}!}%
811 \def\XINT_iiminof_c #1%
812     {\xint_gob_til_xint: #1\XINT_iiminof_e\xint:\XINT_iiminof_d #1}%
813 \def\XINT_iiminof_d #1!%
814     {\expandafter\XINT_iiminof_b\romannumeral0\xintiimin {#1}}%
815 \def\XINT_iiminof_e #1!#2!{ #2}%

```

4.47 \xintiiSum

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

```

816 \def\xintiiSum {\romannumeral0\xintiisum }%
817 \def\xintiisum #1{\expandafter\XINT_sumexpr\romannumeral`&&@#1\xint:}%
818 \def\XINT_sumexpr {\XINT_sum_loop_a 0\Z }%
819 \def\XINT_sum_loop_a #1\Z #2%
820     {\expandafter\XINT_sum_loop_b \romannumeral`&&@#2\xint:#1\xint:\Z}%
821 \def\XINT_sum_loop_b #1%
822     {\xint_gob_til_xint: #1\XINT_sum_finished\xint:\XINT_sum_loop_c #1}%
823 \def\XINT_sum_loop_c
824     {\expandafter\XINT_sum_loop_a\romannumeral0\XINT_add_fork }%
825 \def\XINT_sum_finished\xint:\XINT_sum_loop_c\xint:\xint:#1\xint:\Z{ #1}%

```

4.48 \xintiiPrd

\xintiiPrd {{a}}...{z}}

```

826 \def\xintiiPrd {\romannumeral0\xintiiprd }%
827 \def\xintiiprd #1{\expandafter\XINT_prdexpr\romannumeral`&&@#1\xint:}%
828 \def\XINT_prdexpr {\XINT_prod_loop_a 1\Z }%
829 \def\XINT_prod_loop_a #1\Z #2%
830     {\expandafter\XINT_prod_loop_b\romannumeral`&&@#2\xint:#1\xint:\Z}%
831 \def\XINT_prod_loop_b #1%
832     {\xint_gob_til_xint: #1\XINT_prod_finished\xint:\XINT_prod_loop_c #1}%
833 \def\XINT_prod_loop_c
834     {\expandafter\XINT_prod_loop_a\romannumeral0\XINT_mul_fork }%
835 \def\XINT_prod_finished\xint:\XINT_prod_loop_c\xint:\xint:#1\xint:\Z { #1}%

```

4.49 \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.

4 Package *xint* implementation

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

Incidentally, as `\xintiSqrt` 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.

1.2k belatedly corrects the output to `{1}{1}` and not `11` when input is zero. As braces are used in all other cases they should have been used here too.

Also, 1.2k adds an `\xintiSqrtR` macro, for coherence as `\xintiSqrt` is defined (and mentioned in user manual.)

```

836 \def\xintiiSquareRoot {\romannumeral0\xintiisquareroot }%
837 \def\xintiSquareRoot{\romannumeral0\XINT_signaldeprecated{xint}{xintiSquareRoot}\xintisquareroot}%
838 \def\xintisquareroot #1%
839   {\expandafter\XINT_sqrt_checkin\romannumeral0\xintnum{#1}\xint:}%
840 \def\xintiisquareroot #1{\expandafter\XINT_sqrt_checkin\romannumeral`&&@#1\xint:}%
841 \def\XINT_sqrt_checkin #1%
842 {%
843   \xint_UDzerominusfork
844   #1-\XINT_sqrt_iszero
845   0#1\XINT_sqrt_isneg
846   0-\XINT_sqrt
847   \krof #1%
848 }%
849 \def\XINT_sqrt_iszero #1\xint:{{1}{1}}%
850 \def\XINT_sqrt_isneg #1\xint:{\XINT_signalcondition{InvalidOperation}{square
851   root of negative: #1}{0}{0}}}%
852 \def\XINT_sqrt #1\xint:
853 {%
854   \expandafter\XINT_sqrt_start\romannumeral0\xintlength {#1}.#1.%
855 }%
856 \def\XINT_sqrt_start #1.%
857 {%
858   \ifnum #1<\xint_c_x\xint_dothis\XINT_sqrt_small_a\fi
859   \xint_orthat\XINT_sqrt_big_a #1.%
860 }%
861 \def\XINT_sqrt_small_a #1.{\XINT_sqrt_a #1.\XINT_sqrt_small_d }%
862 \def\XINT_sqrt_big_a #1.{\XINT_sqrt_a #1.\XINT_sqrt_big_d }%
863 \def\XINT_sqrt_a #1.%
864 {%
865   \ifodd #1
866     \expandafter\XINT_sqrt_b0
867   \else
868     \expandafter\XINT_sqrt_bE
869   \fi
870   #1.%
871 }%

872 \def\XINT_sqrt_bE #1.#2#3#4%
873 {%

```


4 Package *xint* implementation

```

874 \XINT_sqrt_c {#3#4}#2{#1}#3#4%
875 }%

876 \def\XINT_sqrt_b0 #1.#2#3%
877 {%
878 \XINT_sqrt_c #3#2{#1}#3%
879 }%

880 \def\XINT_sqrt_c #1#2%
881 {%
882 \expandafter #2%
883 \the\numexpr \ifnum #1>\xint_c_ii
884 \ifnum #1>\xint_c_vi
885 \ifnum #1>12 \ifnum #1>20 \ifnum #1>30
886 \ifnum #1>42 \ifnum #1>56 \ifnum #1>72
887 \ifnum #1>90
888 10\else 9\fi \else 8\fi \else 7\fi \else 6\fi \else 5\fi
889 \else 4\fi \else 3\fi \else 2\fi \else 1\fi .%
890 }%

891 \def\XINT_sqrt_small_d #1.#2%
892 {%
893 \expandafter\XINT_sqrt_small_e
894 \the\numexpr #1\ifcase \numexpr #2/\xint_c_ii-\xint_c_i\relax
895 \or 0\or 00\or 000\or 0000\fi .%
896 }%

897 \def\XINT_sqrt_small_e #1.#2.%
898 {%
899 \expandafter\XINT_sqrt_small_ea\the\numexpr #1*#1-#2.#1.%
900 }%

901 \def\XINT_sqrt_small_ea #1%
902 {%
903 \if0#1\xint_dothis\XINT_sqrt_small_ez\fi
904 \if-#1\xint_dothis\XINT_sqrt_small_eb\fi
905 \xint_orthat\XINT_sqrt_small_f #1%
906 }%
907 \def\XINT_sqrt_small_ez 0.#1.{\expandafter{\the\numexpr#1+\xint_c_i
908 \expandafter}\expandafter{\the\numexpr #1*\xint_c_ii+\xint_c_i}}%

909 \def\XINT_sqrt_small_eb -#1.#2.%
910 {%
911 \expandafter\XINT_sqrt_small_ec \the\numexpr
912 (#1-\xint_c_i+#2)/(\xint_c_ii*#2).#1.#2.%
913 }%

```

4 Package *xint* implementation

```

914 \def\XINT_sqrt_small_ec #1.#2.#3.%
915 {%
916   \expandafter\XINT_sqrt_small_f \the\numexpr
917     -#2+\xint_c_ii*#3*#1+#1*#1\expandafter.\the\numexpr #3+#1.%
918 }%

919 \def\XINT_sqrt_small_f #1.#2.%
920 {%
921   \expandafter\XINT_sqrt_small_g
922   \the\numexpr (#1+#2)/(\xint_c_ii*#2)-\xint_c_i.#1.#2.%
923 }%

924 \def\XINT_sqrt_small_g #1#2.%
925 {%
926   \if 0#1%
927     \expandafter\XINT_sqrt_small_end
928   \else
929     \expandafter\XINT_sqrt_small_h
930   \fi
931   #1#2.%
932 }%

933 \def\XINT_sqrt_small_h #1.#2.#3.%
934 {%
935   \expandafter\XINT_sqrt_small_f
936   \the\numexpr #2-\xint_c_ii*#1*#3+#1*#1\expandafter.%
937   \the\numexpr #3-#1.%
938 }%
939 \def\XINT_sqrt_small_end #1.#2.#3.{{#3}{#2}}%

940 \def\XINT_sqrt_big_d #1.#2%
941 {%
942   \ifodd #2 \xint_dothis{\expandafter\XINT_sqrt_big_e0}\fi
943   \xint_orthat{\expandafter\XINT_sqrt_big_e}%

944   \the\numexpr (#2-\xint_c_i)/\xint_c_ii.#1;%
945 }%

946 \def\XINT_sqrt_big_eE #1;#2#3#4#5#6#7#8#9%
947 {%
948   \XINT_sqrt_big_eE_a #1;{#2#3#4#5#6#7#8#9}%
949 }%

```

4 Package `xint` implementation

```

950 \def\XINT_sqrt_big_eE_a #1.#2;#3%
951 {%
952   \expandafter\XINT_sqrt_bigormed_f
953   \romannumeral0\XINT_sqrt_small_e #2000.#3.#1;%
954 }%

955 \def\XINT_sqrt_big_e0 #1;#2#3#4#5#6#7#8#9%
956 {%
957   \XINT_sqrt_big_e0_a #1;{#2#3#4#5#6#7#8#9}%
958 }%
959 \def\XINT_sqrt_big_e0_a #1.#2;#3#4%
960 {%
961   \expandafter\XINT_sqrt_bigormed_f
962   \romannumeral0\XINT_sqrt_small_e #20000.#3#4.#1;%
963 }%

964 \def\XINT_sqrt_bigormed_f #1#2#3;%
965 {%
966   \ifnum#3<\xint_c_ix
967     \xint_dothis {\csname XINT_sqrt_med_f\romannumeral#3\endcsname}%
968   \fi
969   \xint_orthat\XINT_sqrt_big_f #1.#2.#3;%
970 }%
971 \def\XINT_sqrt_med_fv {\XINT_sqrt_med_fa .}%
972 \def\XINT_sqrt_med_fvi {\XINT_sqrt_med_fa 0.}%
973 \def\XINT_sqrt_med_fvii {\XINT_sqrt_med_fa 00.}%
974 \def\XINT_sqrt_med_fviii{\XINT_sqrt_med_fa 000.}%

975 \def\XINT_sqrt_med_fa #1.#2.#3.#4;%
976 {%
977   \expandafter\XINT_sqrt_med_fb
978   \the\numexpr (#30#1-5#1)/(\xint_c_ii*#2).#1.#2.#3.%
979 }%

980 \def\XINT_sqrt_med_fb #1.#2.#3.#4.#5.%
981 {%
982   \expandafter\XINT_sqrt_small_ea
983   \the\numexpr (#40#2-\xint_c_ii*#3*#1)*10#2+(#1*#1-#5)\expandafter.%
984   \the\numexpr #30#2-#1.%
985 }%

986 \def\XINT_sqrt_big_f #1;#2#3#4#5#6#7#8#9%
987 {%
988   \XINT_sqrt_big_fa #1;{#2#3#4#5#6#7#8#9}%
989 }%

```

4 Package `xint` implementation

```

990 \def\XINT_sqrt_big_fa #1.#2.#3;#4%
991 {%
992   \expandafter\XINT_sqrt_big_ga
993   \the\numexpr #3-\xint_c_viii\expandafter.%
994   \romannumeral0\XINT_sqrt_med_fa 000.#1.#2.;#4.%
995 }%

996 \def\XINT_sqrt_big_ga #1.#2#3%
997 {%
998   \ifnum #1>\xint_c_viii
999     \expandafter\XINT_sqrt_big_gb\else
1000   \expandafter\XINT_sqrt_big_ka
1001   \fi #1.#3.#2.%
1002 }%

1003 \def\XINT_sqrt_big_gb #1.#2.#3.%
1004 {%
1005   \expandafter\XINT_sqrt_big_gc
1006   \the\numexpr (\xint_c_ii*#2-\xint_c_i)*\xint_c_x^viii/(\xint_c_iv*#3).%
1007   #3.#2.#1;%
1008 }%

1009 \def\XINT_sqrt_big_gc #1.#2.#3.%
1010 {%
1011   \expandafter\XINT_sqrt_big_gd
1012   \romannumeral0\xintiiaadd
1013     {\xintiiSub {#3000000000}\xintDouble{\xintiiMul{#2}{#1}}00000000}%
1014     {\xintiiSqr {#1}}.%
1015   \romannumeral0\xintiisub{#2000000000}{#1}.%
1016 }%

1017 \def\XINT_sqrt_big_gd #1.#2.%
1018 {%
1019   \expandafter\XINT_sqrt_big_ge #2.#1.%
1020 }%

1021 \def\XINT_sqrt_big_ge #1;#2#3#4#5#6#7#8#9%
1022   {\XINT_sqrt_big_gf #1.#2#3#4#5#6#7#8#9;}%
1023 \def\XINT_sqrt_big_gf #1;#2#3#4#5#6#7#8#9%
1024   {\XINT_sqrt_big_gg #1#2#3#4#5#6#7#8#9.}%

```

4 Package *xint* implementation

```
1025 \def\XINT_sqrt_big_gg #1.#2.#3.#4.%
1026 {%
1027   \expandafter\XINT_sqrt_big_gloop
1028   \expandafter\xint_c_xvi\expandafter.%
1029   \the\numexpr #3-\xint_c_viii\expandafter.%
1030   \romannumeral0\xintiisub {#2}{\xintiNum{#4}}.#1.%
1031 }%

1032 \def\XINT_sqrt_big_gloop #1.#2.%
1033 {%
1034   \unless\ifnum #1<#2 \xint_dothis\XINT_sqrt_big_ka \fi
1035   \xint_orthat{\XINT_sqrt_big_gi #1.}#2.%
1036 }%

1037 \def\XINT_sqrt_big_gi #1.%
1038 {%
1039   \expandafter\XINT_sqrt_big_gj\romannumeral\xintreplicate{#1}0.#1.%
1040 }%

1041 \def\XINT_sqrt_big_gj #1.#2.#3.#4.#5.%
1042 {%
1043   \expandafter\XINT_sqrt_big_gk
1044   \romannumeral0\xintiidivision {#4#1}%
1045   {\XINT_dbl #5\xint_bye2345678\xint_bye*\xint_c_ii\relax}.%
1046   #1.#5.#2.#3.%
1047 }%

1048 \def\XINT_sqrt_big_gk #1#2.#3.#4.%
1049 {%
1050   \expandafter\XINT_sqrt_big_gl
1051   \romannumeral0\xintiiadd {#2#3}{\xintiiSqr{#1}}.%
1052   \romannumeral0\xintiisub {#4#3}{#1}.%
1053 }%

1054 \def\XINT_sqrt_big_gl #1.#2.%
1055 {%
1056   \expandafter\XINT_sqrt_big_gm #2.#1.%
1057 }%

1058 \def\XINT_sqrt_big_gm #1.#2.#3.#4.#5.%
1059 {%
1060   \expandafter\XINT_sqrt_big_gn
```

4 Package *xint* implementation

```

1061 \romannumeral0\XINT_split_fromleft\xint_c_ii*#3.#5\xint_bye2345678\xint_bye..%
1062 #1.#2.#3.#4.%
1063 }%

1064 \def\XINT_sqrt_big_gn #1.#2.#3.#4.#5.#6.%
1065 {%
1066 \expandafter\XINT_sqrt_big_gloop
1067 \the\numexpr \xint_c_ii*#5\expandafter.%
1068 \the\numexpr #6-#5\expandafter.%
1069 \romannumeral0\xintiisub{#4}{\xintiNum{#1}}.#3.#2.%
1070 }%

1071 \def\XINT_sqrt_big_ka #1.#2.#3.#4.%
1072 {%
1073 \expandafter\XINT_sqrt_big_kb

1074 \romannumeral0\XINT_dsx_addzeros {#1}#3;.%
1075 \romannumeral0\xintiisub
1076 {\XINT_dsx_addzerosnofuss {\xint_c_ii*#1}#2;}%
1077 {\xintiNum{#4}}.%
1078 }%
1079 \def\XINT_sqrt_big_kb #1.#2.%
1080 {%
1081 \expandafter\XINT_sqrt_big_kc #2.#1.%
1082 }%

1083 \def\XINT_sqrt_big_kc #1%
1084 {%
1085 \if0#1\xint_dothis\XINT_sqrt_big_kz\fi
1086 \xint_orthat\XINT_sqrt_big_kloop #1%
1087 }%
1088 \def\XINT_sqrt_big_kz 0.#1.%
1089 {%
1090 \expandafter\XINT_sqrt_big_kend
1091 \romannumeral0%
1092 \xintinc{\XINT_dbl#1\xint_bye2345678\xint_bye*\xint_c_ii\relax}.#1.%
1093 }%
1094 \def\XINT_sqrt_big_kend #1.#2.%
1095 {%
1096 \expandafter{\romannumeral0\xintinc{#2}}{#1}%
1097 }%

1098 \def\XINT_sqrt_big_kloop #1.#2.%

```

4 Package *xint* implementation

```

1099 {%
1100   \expandafter\XINT_sqrt_big_ke
1101   \romannumeral0\xintiidevision{#1}%
1102   {\romannumeral0\XINT_dbl #2\xint_bye2345678\xint_bye*\xint_c_ii\relax}{#2}%
1103 }%

1104 \def\XINT_sqrt_big_ke #1%
1105 {%
1106   \if0\XINT_Sgn #1\xint:
1107     \expandafter \XINT_sqrt_big_end
1108   \else \expandafter \XINT_sqrt_big_kf
1109   \fi {#1}%
1110 }%

1111 \def\XINT_sqrt_big_kf #1#2#3%
1112 {%
1113   \expandafter\XINT_sqrt_big_kg
1114   \romannumeral0\xintiisub {#3}{#1}.%
1115   \romannumeral0\xintiiaadd {#2}{\xintiiSqr {#1}}.%
1116 }%
1117 \def\XINT_sqrt_big_kg #1.#2.%
1118 {%
1119   \expandafter\XINT_sqrt_big_kloop #2.#1.%
1120 }%

1121 \def\XINT_sqrt_big_end #1#2#3{{#3}{#2}}%

```

4.50 *\xintiiSqr*, *\xintiiSqrR*

```

1122 \def\xintiiSqr {\romannumeral0\xintiisqr }%
1123 \def\xintiisqr {\expandafter\XINT_sqrt_post\romannumeral0\xintiisquareroot }%
1124 \def\XINT_sqrt_post #1#2{\XINT_dec #1\XINT_dec_bye234567890\xint_bye}%
1125 \def\xintiiSqrR {\romannumeral0\xintiisqrtr }%
1126 \def\xintiisqrtr {\expandafter\XINT_sqrtr_post\romannumeral0\xintiisquareroot }%

```

$N = (\#1)^2 - \#2$ 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-> $\#1$ si $\#2 \geq \#1$, $(\#1 - .5)^2 \geq N + .25 > N$, donc rounded-> $\#1 - 1$.

```

1127 \def\XINT_sqrtr_post #1#2%
1128   {\xintiiflt {#2}{#1}{ #1}{\XINT_dec #1\XINT_dec_bye234567890\xint_bye}}%
1129 \def\xintiSqr {\romannumeral0\XINT_signaldeprecated{xint}{xintiSqr}\xintisqr}%
1130 \def\xintisqr {\expandafter\XINT_sqrt_post\romannumeral0\xintisquareroot }%
1131 \def\xintiSqrR{\romannumeral0\XINT_signaldeprecated{xint}{xintiSqrtr}\xintisqrtr}%
1132 \def\xintisqrtr {\expandafter\XINT_sqrtr_post\romannumeral0\xintisquareroot }%

```

4.51 *\xintiiBinomial*

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.

1.2o deprecates `\xintiBinomial`. (which `xintfrac.sty` redefined to use `\xintNum`)

```
1133 \def\xintiBinomial {\romannumeral0\xintiibinomial }%
1134 \def\xintiibinomial #1#2%
1135 {%
1136   \expandafter\XINT_binom_pre\the\numexpr #1\expandafter.\the\numexpr #2.%
1137 }%
1138 \def\XINT_binom_pre #1.#2.%
1139 {%
1140   \expandafter\XINT_binom_fork \the\numexpr#1-#2.#2.#1.%
1141 }%
1142 \def\xintiBinomial{\romannumeral0\XINT_signaldeprecated{xint}{xintiBinomial}\xintibinomial}%
1143 \let\xintibinomial\xintiibinomial
```

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

```
1144 \def\XINT_binom_fork #1#2.#3#4.#5#6.%
1145 {%
1146   \if-#5\xint_dothis{\XINT_signalcondition{InvalidOperation}{Binomial with
1147     negative first arg: #5#6}{0}}\fi
1148   \if-#1\xint_dothis{ 0}\fi
1149   \if-#3\xint_dothis{ 0}\fi
1150   \if0#1\xint_dothis{ 1}\fi
1151   \if0#3\xint_dothis{ 1}\fi
1152   \ifnum #5#6>\xint_c_x^viii_mone\xint_dothis
1153     {\XINT_signalcondition{InvalidOperation}{Binomial with too
1154       large argument: 99999999 < #5#6}{0}}\fi
1155   \ifnum #1#2>#3#4 \xint_dothis{\XINT_binom_a #1#2.#3#4.}\fi
1156   \xint_orthat{\XINT_binom_a #3#4.#1#2.}%
1157 }%
```

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 `\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax` est le terminateur pour le `\XINT_unsep_cuzsmall` final.

```
1158 \def\XINT_binom_a #1.#2.%
1159 {%
1160   \expandafter\XINT_binom_b\the\numexpr \xint_c_i+#1.1.#2.1000000001!1!;!0!%
1161 }%
```

$y = x - k + 1$. $j = 1$. 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`).

4 Package *xint* implementation

```

1162 \def\XINT_binom_b #1.%
1163 {%
1164   \ifnum #1>9999 \xint_dothis\XINT_binom_vbigloop \fi
1165   \ifnum #1>463  \xint_dothis\XINT_binom_bigloop  \fi
1166   \ifnum #1>98   \xint_dothis\XINT_binom_medloop   \fi
1167   \ifnum #1>29   \xint_dothis\XINT_binom_smallloop \fi
1168               \xint_orthat\XINT_binom_vsmallloop #1.%
1169 }%

```

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;!0!. Dans smallloop on peut prendre 4 par 4.

```

1170 \def\XINT_binom_smallloop #1.#2.#3.%
1171 {%
1172   \ifcase\numexpr #3-#2\relax
1173     \expandafter\XINT_binom_end_
1174   \or \expandafter\XINT_binom_end_i
1175   \or \expandafter\XINT_binom_end_ii
1176   \or \expandafter\XINT_binom_end_iii
1177   \else\expandafter\XINT_binom_smallloop_a
1178   \fi #1.#2.#3.%
1179 }%

```

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.

```

1180 \def\XINT_binom_smallloop_a #1.#2.#3.%
1181 {%
1182   \expandafter\XINT_binom_smallloop_b
1183   \the\numexpr #1+\xint_c_iv\expandafter.%
1184   \the\numexpr #2+\xint_c_iv\expandafter.%
1185   \the\numexpr #3\expandafter.%
1186   \the\numexpr\expandafter\XINT_binom_div
1187   \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1188   !\romannumeral0\expandafter\XINT_binom_mul
1189   \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1190 }%
1191 \def\XINT_binom_smallloop_b #1.%
1192 {%
1193   \ifnum #1>98 \expandafter\XINT_binom_medloop \else
1194               \expandafter\XINT_binom_smallloop \fi #1.%
1195 }%

```

Ici on prend trois par trois.

```

1196 \def\XINT_binom_medloop #1.#2.#3.%
1197 {%
1198   \ifcase\numexpr #3-#2\relax
1199     \expandafter\XINT_binom_end_
1200   \or \expandafter\XINT_binom_end_i
1201   \or \expandafter\XINT_binom_end_ii
1202   \else\expandafter\XINT_binom_medloop_a
1203   \fi #1.#2.#3.%
1204 }%

```

4 Package *xint* implementation

```
1205 \def\XINT_binom_medloop_a #1.#2.#3.%
1206 {%
1207   \expandafter\XINT_binom_medloop_b
1208   \the\numexpr #1+\xint_c_iii\expandafter.%
1209   \the\numexpr #2+\xint_c_iii\expandafter.%
1210   \the\numexpr #3\expandafter.%
1211   \the\numexpr\expandafter\XINT_binom_div
1212     \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
1213     !\romannumeral0\expandafter\XINT_binom_mul
1214     \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1215 }%
1216 \def\XINT_binom_medloop_b #1.%
1217 {%
1218   \ifnum #1>463 \expandafter\XINT_binom_bigloop \else
1219     \expandafter\XINT_binom_medloop \fi #1.%
1220 }%
```

Ici on prend deux par deux.

```
1221 \def\XINT_binom_bigloop #1.#2.#3.%
1222 {%
1223   \ifcase\numexpr #3-#2\relax
1224     \expandafter\XINT_binom_end_
1225   \or \expandafter\XINT_binom_end_i
1226   \else\expandafter\XINT_binom_bigloop_a
1227   \fi #1.#2.#3.%
1228 }%
1229 \def\XINT_binom_bigloop_a #1.#2.#3.%
1230 {%
1231   \expandafter\XINT_binom_bigloop_b
1232   \the\numexpr #1+\xint_c_ii\expandafter.%
1233   \the\numexpr #2+\xint_c_ii\expandafter.%
1234   \the\numexpr #3\expandafter.%
1235   \the\numexpr\expandafter\XINT_binom_div
1236     \the\numexpr #2*(#2+\xint_c_i)\expandafter
1237     !\romannumeral0\expandafter\XINT_binom_mul
1238     \the\numexpr #1*(#1+\xint_c_i)!%
1239 }%
1240 \def\XINT_binom_bigloop_b #1.%
1241 {%
1242   \ifnum #1>9999 \expandafter\XINT_binom_vbigloop \else
1243     \expandafter\XINT_binom_bigloop \fi #1.%
1244 }%
```

Et finalement un par un.

```
1245 \def\XINT_binom_vbigloop #1.#2.#3.%
1246 {%
1247   \ifnum #3=#2
1248     \expandafter\XINT_binom_end_
1249   \else\expandafter\XINT_binom_vbigloop_a
1250   \fi #1.#2.#3.%
1251 }%
1252 \def\XINT_binom_vbigloop_a #1.#2.#3.%
```

4 Package *xint* implementation

```

1253 {%
1254   \expandafter\XINT_binom_vbigloop
1255   \the\numexpr #1+\xint_c_i\expandafter.%
1256   \the\numexpr #2+\xint_c_i\expandafter.%
1257   \the\numexpr #3\expandafter.%
1258   \the\numexpr\expandafter\XINT_binom_div\the\numexpr #2\expandafter
1259   !\romannumeral0\XINT_binom_mul #1!%
1260 }%

```

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.

```

1261 \def\XINT_binom_vsmallloop #1.#2.#3.%
1262 {%
1263   \ifcase\numexpr #3-#2\relax
1264     \expandafter\XINT_binom_vsmallend_
1265     \or \expandafter\XINT_binom_vsmallend_i
1266     \or \expandafter\XINT_binom_vsmallend_ii
1267     \or \expandafter\XINT_binom_vsmallend_iii
1268     \else\expandafter\XINT_binom_vsmallloop_a
1269     \fi #1.#2.#3.%
1270 }%
1271 \def\XINT_binom_vsmallloop_a #1.%
1272 {%
1273   \ifnum #1>26 \expandafter\XINT_binom_smallloop_a \else
1274     \expandafter\XINT_binom_vsmallloop_b \fi #1.%
1275 }%
1276 \def\XINT_binom_vsmallloop_b #1.#2.#3.%
1277 {%
1278   \expandafter\XINT_binom_vsmallloop
1279   \the\numexpr #1+\xint_c_iv\expandafter.%
1280   \the\numexpr #2+\xint_c_iv\expandafter.%
1281   \the\numexpr #3\expandafter.%
1282   \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1283   \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1284   !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1285 }%
1286 \def\XINT_binom_mul #1!#21!;!0!%
1287 {%
1288   \expandafter\XINT_rev_nounsep\expandafter{\expandafter}%
1289   \the\numexpr\expandafter\XINT_smallmul
1290   \the\numexpr\xint_c_x^viii+#1\expandafter
1291   !\romannumeral0\XINT_rev_nounsep {}1;!#2%
1292   \R!\R!\R!\R!\R!\R!\R!\R!\W
1293   \R!\R!\R!\R!\R!\R!\R!\R!\W
1294   1;!%
1295 }%
1296 \def\XINT_binom_div #1!1;!%
1297 {%
1298   \expandafter\XINT_smalldivx_a

```

4 Package *xint* implementation

```

1299 \the\numexpr #1/\xint_c_ii\expandafter\xint:
1300 \the\numexpr \xint_c_x^viii+#1!%
1301 }%

```

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

```

1302 \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.

```

1303 \def\xint_binom_end_iii #1.#2.#3.%
1304 {%
1305 \expandafter\xint_binom_finish
1306 \the\numexpr\expandafter\xint_binom_div
1307 \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1308 !\romannumeral0\expandafter\xint_binom_mul
1309 \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1310 }%
1311 \def\xint_binom_end_ii #1.#2.#3.%
1312 {%
1313 \expandafter\xint_binom_finish
1314 \the\numexpr\expandafter\xint_binom_div
1315 \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
1316 !\romannumeral0\expandafter\xint_binom_mul
1317 \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1318 }%
1319 \def\xint_binom_end_i #1.#2.#3.%
1320 {%
1321 \expandafter\xint_binom_finish
1322 \the\numexpr\expandafter\xint_binom_div
1323 \the\numexpr #2*(#2+\xint_c_i)\expandafter
1324 !\romannumeral0\expandafter\xint_binom_mul
1325 \the\numexpr #1*(#1+\xint_c_i)!%
1326 }%
1327 \def\xint_binom_end_ #1.#2.#3.%
1328 {%
1329 \expandafter\xint_binom_finish
1330 \the\numexpr\expandafter\xint_binom_div\the\numexpr #2\expandafter
1331 !\romannumeral0\xint_binom_mul #1!%
1332 }%
1333 \def\xint_binom_finish #1;!0!%
1334 {\xint_unsep_cuzsmall #1\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax}%

```

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).

```

1335 \def\xint_binom_vsmallend_iii #1.%
1336 {%
1337 \ifnum #1>26 \expandafter\xint_binom_end_iii \else
1338 \expandafter\xint_binom_vsmallend_iiib \fi #1.%
1339 }%
1340 \def\xint_binom_vsmallend_iiib #1.#2.#3.%
1341 {%

```

```

1342 \expandafter\XINT_binom_vsmallfinish
1343 \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1344 \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1345 !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1346 }%
1347 \def\XINT_binom_vsmallend_ii #1.%
1348 {%
1349 \ifnum #1>27 \expandafter\XINT_binom_end_ii \else
1350 \expandafter\XINT_binom_vsmallend_iib \fi #1.%
1351 }%
1352 \def\XINT_binom_vsmallend_iib #1.#2.#3.%
1353 {%
1354 \expandafter\XINT_binom_vsmallfinish
1355 \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1356 \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
1357 !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1358 }%
1359 \def\XINT_binom_vsmallend_i #1.%
1360 {%
1361 \ifnum #1>28 \expandafter\XINT_binom_end_i \else
1362 \expandafter\XINT_binom_vsmallend_ib \fi #1.%
1363 }%
1364 \def\XINT_binom_vsmallend_ib #1.#2.#3.%
1365 {%
1366 \expandafter\XINT_binom_vsmallfinish
1367 \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1368 \the\numexpr #2*(#2+\xint_c_i)\expandafter
1369 !\the\numexpr #1*(#1+\xint_c_i)!%
1370 }%
1371 \def\XINT_binom_vsmallend_ #1.%
1372 {%
1373 \ifnum #1>29 \expandafter\XINT_binom_end_ \else
1374 \expandafter\XINT_binom_vsmallend_b \fi #1.%
1375 }%
1376 \def\XINT_binom_vsmallend_b #1.#2.#3.%
1377 {%
1378 \expandafter\XINT_binom_vsmallfinish
1379 \the\numexpr\XINT_binom_vsmallmuldiv #2!#1!%
1380 }%
1381 \def\XINT_binom_vsmallfinish#1{%
1382 \def\XINT_binom_vsmallfinish1##1!1!;!0!{\expandafter#1\the\numexpr##1\relax}%
1383 }\XINT_binom_vsmallfinish{ }%

```

4.52 `\xintiiPFactorial`

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.

4 Package *xint* implementation

```

1384 \def\xintiipFactorial {\romannumeral0\xintiipfactorial }%
1385 \def\xintiipfactorial #1#2%
1386 {%
1387     \expandafter\XINT_pfac_fork\the\numexpr#1\expandafter.\the\numexpr #2.%
1388 }%
1389 \def\xintiPFactorial{\romannumeral0\XINT_signaldeprecated{xint}{xintiPFactorial}\xintipfactorial}%
1390 \let\xintipfactorial\xintiipfactorial
1391 \def\xintPFactorial{\romannumeral0\xintpfactorial}%
1392 \let\xintpfactorial\xintiipfactorial

```

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

```

1393 \def\XINT_pfac_fork #1#2.#3#4.%
1394 {%
1395     \unless\ifnum #1#2<#3#4 \xint_dothis\XINT_pfac_one\fi
1396     \if-#3\xint_dothis\XINT_pfac_neg\fi
1397     \if-#1\xint_dothis\XINT_pfac_zero\fi
1398     \ifnum #3#4>\xint_c_x^viii_mone\xint_dothis\XINT_pfac_outofrange\fi
1399     \xint_orthat \XINT_pfac_a #1#2.#3#4.%
1400 }%
1401 \def\XINT_pfac_outofrange #1.#2.%
1402     {\XINT_signalcondition{InvalidOperation}{PFactorial with
1403     too big second arg: 99999999 < #2}{\{0\}}}%
1404 \def\XINT_pfac_one #1.#2.{ 1}%
1405 \def\XINT_pfac_zero #1.#2.{ 0}%
1406 \def\XINT_pfac_neg -#1.-#2.%
1407 {%
1408     \ifnum #1>\xint_c_x^viii\xint_dothis\XINT_pfac_outofrange\fi
1409     \xint_orthat
1410     {\ifodd\numexpr#2-#1\relax\xint_afterfi{\expandafter-\romannumeral`&&@}\fi
1411     \expandafter\XINT_pfac_a }%
1412     \the\numexpr #2-\xint_c_i\expandafter.\the\numexpr#1-\xint_c_i.%
1413 }%
1414 \def\XINT_pfac_a #1.#2.%
1415 {%
1416     \expandafter\XINT_pfac_b\the\numexpr \xint_c_i+#1.#2.100000001!1;!%
1417     1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1418 }%
1419 \def\XINT_pfac_b #1.%
1420 {%
1421     \ifnum #1>9999 \xint_dothis\XINT_pfac_vbigloop \fi
1422     \ifnum #1>463 \xint_dothis\XINT_pfac_bigloop \fi
1423     \ifnum #1>98 \xint_dothis\XINT_pfac_medloop \fi
1424     \xint_orthat\XINT_pfac_smallloop #1.%
1425 }%
1426 \def\XINT_pfac_smallloop #1.#2.%
1427 {%
1428     \ifcase\numexpr #2-#1\relax
1429         \expandafter\XINT_pfac_end_
1430     \or \expandafter\XINT_pfac_end_i
1431     \or \expandafter\XINT_pfac_end_ii
1432     \or \expandafter\XINT_pfac_end_iii

```

4 Package *xint* implementation

```

1433 \else\expandafter\XINT_pfac_smallloop_a
1434 \fi #1.#2.%
1435 }%
1436 \def\XINT_pfac_smallloop_a #1.#2.%
1437 {%
1438 \expandafter\XINT_pfac_smallloop_b
1439 \the\numexpr #1+\xint_c_iv\expandafter.%
1440 \the\numexpr #2\expandafter.%
1441 \the\numexpr\expandafter\XINT_smallmul
1442 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1443 }%
1444 \def\XINT_pfac_smallloop_b #1.%
1445 {%
1446 \ifnum #1>98 \expandafter\XINT_pfac_medloop \else
1447 \expandafter\XINT_pfac_smallloop \fi #1.%
1448 }%
1449 \def\XINT_pfac_medloop #1.#2.%
1450 {%
1451 \ifcase\numexpr #2-#1\relax
1452 \expandafter\XINT_pfac_end_
1453 \or \expandafter\XINT_pfac_end_i
1454 \or \expandafter\XINT_pfac_end_ii
1455 \else\expandafter\XINT_pfac_medloop_a
1456 \fi #1.#2.%
1457 }%
1458 \def\XINT_pfac_medloop_a #1.#2.%
1459 {%
1460 \expandafter\XINT_pfac_medloop_b
1461 \the\numexpr #1+\xint_c_iii\expandafter.%
1462 \the\numexpr #2\expandafter.%
1463 \the\numexpr\expandafter\XINT_smallmul
1464 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1465 }%
1466 \def\XINT_pfac_medloop_b #1.%
1467 {%
1468 \ifnum #1>463 \expandafter\XINT_pfac_bigloop \else
1469 \expandafter\XINT_pfac_medloop \fi #1.%
1470 }%
1471 \def\XINT_pfac_bigloop #1.#2.%
1472 {%
1473 \ifcase\numexpr #2-#1\relax
1474 \expandafter\XINT_pfac_end_
1475 \or \expandafter\XINT_pfac_end_i
1476 \else\expandafter\XINT_pfac_bigloop_a
1477 \fi #1.#2.%
1478 }%
1479 \def\XINT_pfac_bigloop_a #1.#2.%
1480 {%
1481 \expandafter\XINT_pfac_bigloop_b
1482 \the\numexpr #1+\xint_c_ii\expandafter.%
1483 \the\numexpr #2\expandafter.%
1484 \the\numexpr\expandafter

```

4 Package `xint` implementation

```

1485 \XINT_smallmul\the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
1486 }%
1487 \def\XINT_pfac_bigloop_b #1.%
1488 {%
1489 \ifnum #1>9999 \expandafter\XINT_pfac_vbigloop \else
1490 \expandafter\XINT_pfac_bigloop \fi #1.%
1491 }%
1492 \def\XINT_pfac_vbigloop #1.#2.%
1493 {%
1494 \ifnum #2=#1
1495 \expandafter\XINT_pfac_end_
1496 \else\expandafter\XINT_pfac_vbigloop_a
1497 \fi #1.#2.%
1498 }%
1499 \def\XINT_pfac_vbigloop_a #1.#2.%
1500 {%
1501 \expandafter\XINT_pfac_vbigloop
1502 \the\numexpr #1+\xint_c_i\expandafter.%
1503 \the\numexpr #2\expandafter.%
1504 \the\numexpr\expandafter\XINT_smallmul\the\numexpr\xint_c_x^viii+#1!%
1505 }%
1506 \def\XINT_pfac_end_iii #1.#2.%
1507 {%
1508 \expandafter\XINT_mul_out
1509 \the\numexpr\expandafter\XINT_smallmul
1510 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1511 }%
1512 \def\XINT_pfac_end_ii #1.#2.%
1513 {%
1514 \expandafter\XINT_mul_out
1515 \the\numexpr\expandafter\XINT_smallmul
1516 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1517 }%
1518 \def\XINT_pfac_end_i #1.#2.%
1519 {%
1520 \expandafter\XINT_mul_out
1521 \the\numexpr\expandafter\XINT_smallmul
1522 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
1523 }%
1524 \def\XINT_pfac_end_ #1.#2.%
1525 {%
1526 \expandafter\XINT_mul_out
1527 \the\numexpr\expandafter\XINT_smallmul\the\numexpr \xint_c_x^viii+#1!%
1528 }%

```

4.53 `\xintBool`, `\xintToggle`

1.09c

```

1529 \def\xintBool #1{\romannumeral`&&@%
1530 \csname if#1\endcsname\expandafter1\else\expandafter0\fi }%
1531 \def\xintToggle #1{\romannumeral`&&@\iftoggle{#1}{1}{0}}%

```


At End of \LaTeX Document deprecation message

1.2o

```

1532 \ifdefined\documentclass\ifdefined\AtEndDocument
1533     \AtEndDocument{%
1534 \XINT_ifFlagRaised{Deprecated-xint}%
1535     {\PackageError{xint}
1536 {Usage of deprecated macros!}
1537 {These deprecated macros from xint.sty have been detected:\MessageBreak
1538 \XINT_useiimessage{xintEq}%
1539 \XINT_useiimessage{xintGeq}%
1540 \XINT_useiimessage{xintGt}%
1541 \XINT_useiimessage{xintLt}%
1542 \XINT_useiimessage{xintGtorEq}%
1543 \XINT_useiimessage{xintLtorEq}%
1544 \XINT_useiimessage{xintIsZero}%
1545 \XINT_useiimessage{xintIsNotZero}%
1546 \XINT_useiimessage{xintIsOne}%
1547 \XINT_useiimessage{xintOdd}%
1548 \XINT_useiimessage{xintEven}%
1549 \XINT_useiimessage{xintifSgn}%
1550 \XINT_useiimessage{xintifCmp}%
1551 \XINT_useiimessage{xintifEq}%
1552 \XINT_useiimessage{xintifGt}%
1553 \XINT_useiimessage{xintifLt}%
1554 \XINT_useiimessage{xintifZero}%
1555 \XINT_useiimessage{xintifNotZero}%
1556 \XINT_useiimessage{xintifOne}%
1557 \XINT_useiimessage{xintifOdd}%
1558 \XINT_ifFlagRaised{xintNeq}%
1559     {\string\xintNeq\space(use \string\xintiiNotEq\space or \string\xintNotEq\space from xintfrac!)\MessageBreak}
1560 \XINT_ifFlagRaised{xintNot}%
1561     {\string\xintNot\space(renamed to \string\xintNOT!)\MessageBreak}}}%
1562 \XINT_ifFlagRaised{xintMON}{\string\xintMON\MessageBreak}}}%
1563 \XINT_ifFlagRaised{xintMMON}{\string\xintMMON\MessageBreak}}}%
1564 \XINT_ifFlagRaised{xintiMax}{\string\xintiMax\MessageBreak}}}%
1565 \XINT_ifFlagRaised{xintiMin}{\string\xintiMin\MessageBreak}}}%
1566 \XINT_ifFlagRaised{xintiMaxof}{\string\xintiMaxof\MessageBreak}}}%
1567 \XINT_ifFlagRaised{xintiMinof}{\string\xintiMinof\MessageBreak}}}%
1568 \XINT_ifFlagRaised{xintiSquareRoot}{\string\xintiSquareRoot\MessageBreak}}}%
1569 \XINT_ifFlagRaised{xintiSqrt}{\string\xintiSqrt\MessageBreak}}}%
1570 \XINT_ifFlagRaised{xintiSqrtR}{\string\xintiSqrtR\MessageBreak}}}%
1571 \XINT_ifFlagRaised{xintiBinomial}{\string\xintiBinomial\MessageBreak}}}%
1572 \XINT_ifFlagRaised{xintiPFactorial}{\string\xintiPFactorial\MessageBreak}}}%
1573 They will get removed at some future release.}}%
1574     {% no deprecated macro used (at top level...)
1575     }%
1576 }\fi\fi
1577 \XINT_restorecatcodes_endinput%
```

5 Package *xintbinhex* implementation

.1	Catcodes, ε -TeX and reload detection . . .	154	.6	<code>\xintDecToBin</code>	159
.2	Package identification	155	.7	<code>\xintHexToDec</code>	160
.3	Constants, etc...	155	.8	<code>\xintBinToDec</code>	162
.4	Helper macros	156	.9	<code>\xintBinToHex</code>	163
.4.1	<code>\XINT_zeroes_foriv</code>	156	.10	<code>\xintHexToBin</code>	164
.5	<code>\xintDecToHex</code>	156	.11	<code>\xintCHexToBin</code>	164

The commenting is currently (2018/02/06) very sparse.

The macros from 1.08 (2013/06/07) remained unchanged until their complete rewrite at 1.2m (2012/07/07/31).

At 1.2n dependencies on *xintcore* were removed, so now the package loads only *xintkernel* (this could have been done earlier).

Also at 1.2n, macros evolved again, the main improvements being in the increased allowable sizes of the input for `\xintDecToHex`, `\xintDecToBin`, `\xintBinToHex`. Use of `\csname` governed expansion at some places rather than `\numexpr` with some clean-up after it.

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@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{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 xintkernel.sty not yet loaded.
28       \def\z{\endgroup\input xintkernel.sty\relax}%
29     \fi
30   \else

```

```

31 \def\empty {}%
32 \ifx\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 % 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 [2018/02/06 1.2q Expandable binary and hexadecimal conversions (JFB)]%

```

5.3 Constants, etc...

1.2n switches to \csname-governed expansion at various places.

```

47 \newcount\xint_c_ii^xv \xint_c_ii^xv 32768
48 \newcount\xint_c_ii^xvi \xint_c_ii^xvi 65536
49 \def\XINT_tmpa #1{\ifx\relax#1\else
50 \expandafter\edef\csname XINT_csdth_#1\endcsname
51 {\endcsname\ifcase #1 0\or 1\or 2\or 3\or 4\or 5\or 6\or 7\or
52 8\or 9\or A\or B\or C\or D\or E\or F\fi}%
53 \expandafter\XINT_tmpa\fi }%
54 \XINT_tmpa {0}{1}{2}{3}{4}{5}{6}{7}{8}{9}{10}{11}{12}{13}{14}{15}\relax
55 \def\XINT_tmpa #1{\ifx\relax#1\else
56 \expandafter\edef\csname XINT_csdtb_#1\endcsname
57 {\endcsname\ifcase #1
58 0000\or 0001\or 0010\or 0011\or 0100\or 0101\or 0110\or 0111\or
59 1000\or 1001\or 1010\or 1011\or 1100\or 1101\or 1110\or 1111\fi}%
60 \expandafter\XINT_tmpa\fi }%
61 \XINT_tmpa {0}{1}{2}{3}{4}{5}{6}{7}{8}{9}{10}{11}{12}{13}{14}{15}\relax
62 \let\XINT_tmpa\relax
63 \expandafter\def\csname XINT_csbth_0000\endcsname {\endcsname0}%
64 \expandafter\def\csname XINT_csbth_0001\endcsname {\endcsname1}%
65 \expandafter\def\csname XINT_csbth_0010\endcsname {\endcsname2}%
66 \expandafter\def\csname XINT_csbth_0011\endcsname {\endcsname3}%
67 \expandafter\def\csname XINT_csbth_0100\endcsname {\endcsname4}%
68 \expandafter\def\csname XINT_csbth_0101\endcsname {\endcsname5}%
69 \expandafter\def\csname XINT_csbth_0110\endcsname {\endcsname6}%
70 \expandafter\def\csname XINT_csbth_0111\endcsname {\endcsname7}%
71 \expandafter\def\csname XINT_csbth_1000\endcsname {\endcsname8}%
72 \expandafter\def\csname XINT_csbth_1001\endcsname {\endcsname9}%
73 \expandafter\def\csname XINT_csbth_1010\endcsname {\endcsname A}%
74 \expandafter\def\csname XINT_csbth_1011\endcsname {\endcsname B}%
75 \expandafter\def\csname XINT_csbth_1100\endcsname {\endcsname C}%
76 \expandafter\def\csname XINT_csbth_1101\endcsname {\endcsname D}%

```

```

77 \expandafter\def\csname XINT_csbth_1110\endcsname {\endcsname E}%
78 \expandafter\def\csname XINT_csbth_1111\endcsname {\endcsname F}%
79 \let\XINT_csbth_none \endcsname
80 \expandafter\def\csname XINT_cshtb_0\endcsname {\endcsname 0000}%
81 \expandafter\def\csname XINT_cshtb_1\endcsname {\endcsname 0001}%
82 \expandafter\def\csname XINT_cshtb_2\endcsname {\endcsname 0010}%
83 \expandafter\def\csname XINT_cshtb_3\endcsname {\endcsname 0011}%
84 \expandafter\def\csname XINT_cshtb_4\endcsname {\endcsname 0100}%
85 \expandafter\def\csname XINT_cshtb_5\endcsname {\endcsname 0101}%
86 \expandafter\def\csname XINT_cshtb_6\endcsname {\endcsname 0110}%
87 \expandafter\def\csname XINT_cshtb_7\endcsname {\endcsname 0111}%
88 \expandafter\def\csname XINT_cshtb_8\endcsname {\endcsname 1000}%
89 \expandafter\def\csname XINT_cshtb_9\endcsname {\endcsname 1001}%
90 \def\XINT_cshtb_A {\endcsname 1010}%
91 \def\XINT_cshtb_B {\endcsname 1011}%
92 \def\XINT_cshtb_C {\endcsname 1100}%
93 \def\XINT_cshtb_D {\endcsname 1101}%
94 \def\XINT_cshtb_E {\endcsname 1110}%
95 \def\XINT_cshtb_F {\endcsname 1111}%
96 \let\XINT_cshtb_none \endcsname

```

5.4 Helper macros

5.4.1 `\XINT_zeroes_foriv`

`\romannumeral0\XINT_zeroes_foriv #1\R{0\R}{00\R}{000\R}%
\ R{0\R}{00\R}{000\R}\R\W`

expands to the <empty> or 0 or 00 or 000 needed which when adjoined to #1 extend it to length 4N.

```

97 \def\XINT_zeroes_foriv #1#2#3#4#5#6#7#8%
98 {%
99   \xint_gob_til_R #8\XINT_zeroes_foriv_end\R\XINT_zeroes_foriv
100}%
101 \def\XINT_zeroes_foriv_end\R\XINT_zeroes_foriv #1#2\W
102   {\XINT_zeroes_foriv_done #1}%
103 \def\XINT_zeroes_foriv_done #1\R{ #1}%

```

5.5 `\xintDecToHex`

Complete rewrite at 1.2m in the 1.2 style. Also, 1.2m is robust against non terminated inputs.

Improvements of coding at 1.2n, increased maximal size. Again some coding improvement at 1.2o, about 6% speed gain.

An input without leading zeroes gives an output without leading zeroes.

```

104 \def\xintDecToHex {\romannumeral0\xintdectohex}%
105 \def\xintdectohex #1%
106 {%
107   \expandafter\XINT_dth_checkin\romannumeral`&&@#1\xint:
108}%
109 \def\XINT_dth_checkin #1%
110 {%
111   \xint_UDsignfork

```

5 Package *xintbinhex* implementation

```

112      #1\XINT_dth_neg
113      -{\XINT_dth_main #1}%
114      \krof
115 }%
116 \def\XINT_dth_neg {\expandafter-\romannumeral0\XINT_dth_main}%
117 \def\XINT_dth_main #1\xint:
118 {%
119   \expandafter\XINT_dth_finish
120   \romannumeral`&&\expandafter\XINT_dthb_start
121   \romannumeral0\XINT_zeroes_foriv
122   #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
123   #1\xint_bye\XINT_dth_tohex
124 }%
125 \def\XINT_dthb_start #1#2#3#4#5%
126 {%
127   \xint_bye#5\XINT_dthb_small\xint_bye\XINT_dthb_start_a #1#2#3#4#5%
128 }%
129 \def\XINT_dthb_small\xint_bye\XINT_dthb_start_a #1\xint_bye#2{#2#1!}%
130 \def\XINT_dthb_start_a #1#2#3#4#5#6#7#8#9%
131 {%
132   \expandafter\XINT_dthb_again\the\numexpr\expandafter\XINT_dthb_update
133   \the\numexpr#1#2#3#4%
134   \xint_bye#9\XINT_dthb_lastpass\xint_bye
135   #5#6#7#8!\XINT_dthb_exclam\relax\XINT_dthb_nextfour #9%
136 }%

```

The 1.2n inserted exclamations marks, which when bumping back from `\XINT_dthb_again` gave rise to a `\numexpr`-loop which gathered the ! delimited arguments and inserted `\expandafter\XINT_dthb_update\the\numexpr` dynamically. The 1.2o trick is to insert it here immediately. Then at `\XINT_dthb_again` the `\numexpr` will trigger an already prepared chain.

The crux of the thing is handling of #3 at `\XINT_dthb_update_a`.

```

137 \def\XINT_dthb_exclam {!\XINT_dthb_exclam\relax
138   \expandafter\XINT_dthb_update\the\numexpr}%
139 \def\XINT_dthb_update #1!%
140 {%
141   \expandafter\XINT_dthb_update_a
142   \the\numexpr (#1+\xint_c_ii^xv)/\xint_c_ii^xvi-\xint_c_i\xint:
143   #1\xint:%
144 }%
145 \def\XINT_dthb_update_a #1\xint:#2\xint:#3%
146 {%
147   0000+#1\expandafter#3\the\numexpr#2-#1*\xint_c_ii^xvi
148 }%

```

1.2m and 1.2n had some unduly complicated ending pattern for `\XINT_dthb_nextfour` as inheritance of a loop needing ! separators which was pruned out at 1.2o (see previous comment).

```

149 \def\XINT_dthb_nextfour #1#2#3#4#5%
150 {%
151   \xint_bye#5\XINT_dthb_lastpass\xint_bye
152   #1#2#3#4!\XINT_dthb_exclam\relax\XINT_dthb_nextfour#5%
153 }%
154 \def\XINT_dthb_lastpass\xint_bye #1!#2\xint_bye#3{#1!#3!}%

```

```

155 \def\XINT_dth_tohex
156 {%
157   \expandafter\expandafter\expandafter\XINT_dth_tohex_a\csname\XINT_tofourhex
158 }%
159 \def\XINT_dth_tohex_a\endcsname{!\XINT_dth_tohex!}%
160 \def\XINT_dthb_again #1!#2#3%
161 {%
162   \ifx#3\relax
163     \expandafter\xint_firstoftwo
164   \else
165     \expandafter\xint_secondoftwo
166   \fi
167   {\expandafter\XINT_dthb_again
168    \the\numexpr
169    \ifnum #1>\xint_c_
170      \xint_afterfi{\expandafter\XINT_dthb_update\the\numexpr#1}%
171    \fi}%
172   {\ifnum #1>\xint_c_ \xint_dothis{#2#1!}\fi\xint_orthat{!#2!}}%
173 }%
174 \def\XINT_tofourhex #1!%
175 {%
176   \expandafter\XINT_tofourhex_a
177   \the\numexpr (#1+\xint_c_ii^vii)/\xint_c_ii^viii-\xint_c_i\xint:
178   #1\xint:
179 }%
180 \def\XINT_tofourhex_a #1\xint:#2\xint:
181 {%
182   \expandafter\XINT_tofourhex_c
183   \the\numexpr (#1+\xint_c_viii)/\xint_c_xvi-\xint_c_i\xint:
184   #1\xint:
185   \the\numexpr #2-\xint_c_ii^viii*#1!%
186 }%
187 \def\XINT_tofourhex_c #1\xint:#2\xint:
188 {%
189   XINT_csdth_#1%
190   \csname XINT_csdth_\the\numexpr #2-\xint_c_xvi*#1\relax
191   \csname \expandafter\XINT_tofourhex_d
192 }%
193 \def\XINT_tofourhex_d #1!%
194 {%
195   \expandafter\XINT_tofourhex_e
196   \the\numexpr (#1+\xint_c_viii)/\xint_c_xvi-\xint_c_i\xint:
197   #1\xint:
198 }%
199 \def\XINT_tofourhex_e #1\xint:#2\xint:
200 {%
201   XINT_csdth_#1%
202   \csname XINT_csdth_\the\numexpr #2-\xint_c_xvi*#1\endcsname
203 }%

```

We only clean-up up to 3 zero hexadecimal digits, as output was produced in chunks of 4 hex digits. If input had no leading zero, output will have none either. If input had many leading zeroes, output will have some number (unspecified, but a recipe can be given...) of leading zeroes...

The coding is for varying a bit, I did not check if efficient, it does not matter.

```

204 \def\XINT_dth_finish !\XINT_dth_tohex!#1#2#3%
205 {%
206   \unless\if#10\xint_dothis{ #1#2#3}\fi
207   \unless\if#20\xint_dothis{ #2#3}\fi
208   \unless\if#30\xint_dothis{ #3}\fi
209   \xint_orthat{ }%
210 }%

```

5.6 \xintDecToBin

Complete rewrite at 1.2m in the 1.2 style. Also, 1.2m is robust against non terminated inputs.

Revisited at 1.2n like in \xintDecToHex: increased maximal size.

An input without leading zeroes gives an output without leading zeroes.

Most of the code canvas is shared with \xintDecToHex.

```

211 \def\xintDecToBin {\romannumeral0\xintdectobin }%
212 \def\xintdectobin #1%
213 {%
214   \expandafter\XINT_dtb_checkin\romannumeral`&&@#1\xint:
215 }%
216 \def\XINT_dtb_checkin #1%
217 {%
218   \xint_UDsignfork
219     #1\XINT_dtb_neg
220     -{\XINT_dtb_main #1}%
221   \krof
222 }%
223 \def\XINT_dtb_neg {\expandafter-\romannumeral0\XINT_dtb_main}%
224 \def\XINT_dtb_main #1\xint:
225 {%
226   \expandafter\XINT_dtb_finish
227   \romannumeral`&&\expandafter\XINT_dtb_start
228   \romannumeral0\XINT_zeroes_foriv
229   #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
230   #1\xint_bye\XINT_dtb_tobin
231 }%
232 \def\XINT_dtb_tobin
233 {%
234   \expandafter\expandafter\expandafter\XINT_dtb_tobin_a\cename\XINT_tosixteenbits
235 }%
236 \def\XINT_dtb_tobin_a\endcsname{!\XINT_dtb_tobin!}%
237 \def\XINT_tosixteenbits #1!%
238 {%
239   \expandafter\XINT_tosixteenbits_a
240   \the\numexpr (#1+\xint_c_ii^vii)/\xint_c_ii^viii-\xint_c_i\xint:
241   #1\xint:
242 }%
243 \def\XINT_tosixteenbits_a #1\xint:#2\xint:
244 {%
245   \expandafter\XINT_tosixteenbits_c
246   \the\numexpr (#1+\xint_c_viii)/\xint_c_xvi-\xint_c_i\xint:

```

```

247   #1\xint:
248   \the\numexpr #2-\xint_c_ii^viii*#1!%
249 }%
250 \def\XINT_tosixteenbits_c #1\xint:#2\xint:
251 {%
252   XINT_csdtb_#1%
253   \csname XINT_csdtb_\the\numexpr #2-\xint_c_xvi*#1\relax
254   \csname \expandafter\XINT_tosixteenbits_d
255 }%
256 \def\XINT_tosixteenbits_d #1!%
257 {%
258   \expandafter\XINT_tosixteenbits_e
259   \the\numexpr (#1+\xint_c_viii)/\xint_c_xvi-\xint_c_i\xint:
260   #1\xint:
261 }%
262 \def\XINT_tosixteenbits_e #1\xint:#2\xint:
263 {%
264   XINT_csdtb_#1%
265   \csname XINT_csdtb_\the\numexpr #2-\xint_c_xvi*#1\endcsname
266 }%
267 \def\XINT_dtb_finish !\XINT_dtb_tobin!#1#2#3#4#5#6#7#8%
268 {%
269   \expandafter\XINT_dtb_finish_a\the\numexpr #1#2#3#4#5#6#7#8\relax
270 }%
271 \def\XINT_dtb_finish_a #1{%
272 \def\XINT_dtb_finish_a ##1##2##3##4##5##6##7##8##9%
273 {%
274   \expandafter#1\the\numexpr ##1##2##3##4##5##6##7##8##9\relax
275 }}\XINT_dtb_finish_a { }%

```

5.7 \xintHexToDec

Completely (and belatedly) rewritten at 1.2m in the 1.2 style.

1.2m version robust against non terminated inputs, but there is no primitive from TeX which may generate hexadecimal digits and provoke expansion ahead, afaik, except of course if decimal digits are treated as hexadecimal. This robustness is not on purpose but from need to expand argument and then grab it again. So we do it safely.

Increased maximal size at 1.2n.

1.2m version robust against non terminated inputs.

An input without leading zeroes gives an output without leading zeroes.

```

276 \def\xintHexToDec {\romannumeral0\xinthextodec }%
277 \def\xinthextodec #1%
278 {%
279   \expandafter\XINT_htd_checkin\romannumeral`&&@#1\xint:
280 }%
281 \def\XINT_htd_checkin #1%
282 {%
283   \xint_UDsignfork
284   #1\XINT_htd_neg
285   -{\XINT_htd_main #1}%
286   \krof
287 }%

```


5 Package *xintbinhex* implementation

```

288 \def\XINT_htd_neg {\expandafter-\romannumeral0\XINT_htd_main}%
289 \def\XINT_htd_main #1\xint:
290 {%
291   \expandafter\XINT_htd_startb
292   \the\numexpr\expandafter\XINT_htd_starta
293   \romannumeral0\XINT_zeroes_foriv
294   #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
295   #1\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\relax
296 }%
297 \def\XINT_htd_starta #1#2#3#4{"#1#2#3#4+100000!}%
298 \def\XINT_htd_startb 1#1%
299 {%
300   \if#10\expandafter\XINT_htd_startba\else
301     \expandafter\XINT_htd_startbb
302   \fi 1#1%
303 }%
304 \def\XINT_htd_startba 10#1!\{\XINT_htd_again #1%
305   \xint_bye!2!3!4!5!6!7!8!9!\xint_bye\XINT_htd_nextfour}%
306 \def\XINT_htd_startbb 1#1#2!\{\XINT_htd_again #1!#2%
307   \xint_bye!2!3!4!5!6!7!8!9!\xint_bye\XINT_htd_nextfour}%

```

It is a bit annoying to grab all to the end here. I have a version, modeled on the 1.2n variant of `\xintDecToHex` which solved that problem there, but it did not prove enough if at all faster in my brief testing and it had the defect of a reduced maximal allowed size of the input.

```

308 \def\XINT_htd_again #1\XINT_htd_nextfour #2%
309 {%
310   \xint_bye #2\XINT_htd_finish\xint_bye
311   \expandafter\XINT_htd_A\the\numexpr
312   \XINT_htd_a #1\XINT_htd_nextfour #2%
313 }%
314 \def\XINT_htd_a #1!#2!#3!#4!#5!#6!#7!#8!#9!%
315 {%
316   #1\expandafter\XINT_htd_update
317   \the\numexpr #2\expandafter\XINT_htd_update
318   \the\numexpr #3\expandafter\XINT_htd_update
319   \the\numexpr #4\expandafter\XINT_htd_update
320   \the\numexpr #5\expandafter\XINT_htd_update
321   \the\numexpr #6\expandafter\XINT_htd_update
322   \the\numexpr #7\expandafter\XINT_htd_update
323   \the\numexpr #8\expandafter\XINT_htd_update
324   \the\numexpr #9\expandafter\XINT_htd_update
325   \the\numexpr \XINT_htd_a
326 }%
327 \def\XINT_htd_nextfour #1#2#3#4%
328 {%
329   *\xint_c_ii^xvi+"#1#2#3#4+10000000000\relax\xint_bye!%
330   2!3!4!5!6!7!8!9!\xint_bye\XINT_htd_nextfour
331 }%

```

If the innocent looking commented out #6 is left in the pattern as was the case at 1.2m, the maximal size becomes limited at 5538 digits, not 8298! (with parameter stack size = 10000.)

```

332 \def\XINT_htd_update 1#1#2#3#4#5#6!%

```

5 Package *xintbinhex* implementation

```
333 {%
334   *\xint_c_ii^xvi+10000#1#2#3#4#5!%#6!%
335}%
336 \def\xINT_htd_A 1#1%
337 {%
338   \if#10\expandafter\xINT_htd_Aa\else
339     \expandafter\xINT_htd_Ab
340   \fi 1#1%
341}%
342 \def\xINT_htd_Aa 10#1#2#3#4{\xINT_htd_again #1#2#3#4!}%
343 \def\xINT_htd_Ab 1#1#2#3#4#5{\xINT_htd_again #1!#2#3#4#5!}%
344 \def\xINT_htd_finish\xint_bye
345   \expandafter\xINT_htd_A\the\numexpr \xINT_htd_a #1\xINT_htd_nextfour
346 {%
347   \expandafter\xINT_htd_finish_cuz\the\numexpr0\xINT_htd_unsep_loop #1%
348}%
349 \def\xINT_htd_unsep_loop #1!#2!#3!#4!#5!#6!#7!#8!#9!%
350 {%
351   \expandafter\xINT_unsep_clean
352   \the\numexpr 1#1#2\expandafter\xINT_unsep_clean
353   \the\numexpr 1#3#4\expandafter\xINT_unsep_clean
354   \the\numexpr 1#5#6\expandafter\xINT_unsep_clean
355   \the\numexpr 1#7#8\expandafter\xINT_unsep_clean
356   \the\numexpr 1#9\xINT_htd_unsep_loop_a
357}%
358 \def\xINT_htd_unsep_loop_a #1!#2!#3!#4!#5!#6!#7!#8!#9!%
359 {%
360   #1\expandafter\xINT_unsep_clean
361   \the\numexpr 1#2#3\expandafter\xINT_unsep_clean
362   \the\numexpr 1#4#5\expandafter\xINT_unsep_clean
363   \the\numexpr 1#6#7\expandafter\xINT_unsep_clean
364   \the\numexpr 1#8#9\xINT_htd_unsep_loop
365}%
366 \def\xINT_unsep_clean 1{\relax}% also in xintcore
367 \def\xINT_htd_finish_cuz #1{%
368 \def\xINT_htd_finish_cuz ##1##2##3##4##5%
369   {\expandafter#1\the\numexpr ##1##2##3##4##5\relax}%
370 }\xINT_htd_finish_cuz{ }%
```

5.8 *\xintBinToDec*

Redone entirely for 1.2m. Starts by converting to hexadecimal first.

Increased maximal size at 1.2n.

An input without leading zeroes gives an output without leading zeroes.

Robust against non-terminated input.

```
371 \def\xintBinToDec {\romannumeral0\xintbintodec}%
372 \def\xintbintodec #1%
373 {%
374   \expandafter\xINT_btd_checkin\romannumeral`&&@#1\xint:
375}%
376 \def\xINT_btd_checkin #1%
377 {%
```

```

378 \xint_UDsignfork
379 #1\XINT_btd_N
380 -{\XINT_btd_main #1}%
381 \krof
382 }%
383 \def\XINT_btd_N {\expandafter-\romannumeral0\XINT_btd_main }%
384 \def\XINT_btd_main #1\xint:
385 {%
386 \csname XINT_btd_htd\csname\expandafter\XINT_bth_loop
387 \romannumeral0\XINT_zeroes_foriv
388 #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
389 #1\xint_bye2345678\xint_bye none\endcsname\xint:
390 }%
391 \def\XINT_btd_htd #1\xint:
392 {%
393 \expandafter\XINT_htd_startb
394 \the\numexpr\expandafter\XINT_htd_starta
395 \romannumeral0\XINT_zeroes_foriv
396 #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
397 #1\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\relax
398 }%

```

5.9 \xintBinToHex

Complete rewrite for 1.2m. But input for 1.2m version limited to about 13320 binary digits (expansion depth=10000).

Again redone for 1.2n for \csname governed expansion: increased maximal size.

Size of output is ceil(size(input)/4), leading zeroes in output (inherited from the input) are not trimmed.

An input without leading zeroes gives an output without leading zeroes.

Robust against non-terminated input.

```

399 \def\xintBinToHex {\romannumeral0\xintbinto hex }%
400 \def\xintbinto hex #1%
401 {%
402 \expandafter\XINT_bth_checkin\romannumeral`&&@#1\xint:
403 }%
404 \def\XINT_bth_checkin #1%
405 {%
406 \xint_UDsignfork
407 #1\XINT_bth_N
408 -{\XINT_bth_main #1}%
409 \krof
410 }%
411 \def\XINT_bth_N {\expandafter-\romannumeral0\XINT_bth_main }%
412 \def\XINT_bth_main #1\xint:
413 {%
414 \csname space\csname\expandafter\XINT_bth_loop
415 \romannumeral0\XINT_zeroes_foriv
416 #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
417 #1\xint_bye2345678\xint_bye none\endcsname
418 }%
419 \def\XINT_bth_loop #1#2#3#4#5#6#7#8%

```

```

420 {%
421     XINT_csbth_#1#2#3#4%
422     \csname XINT_csbth_#5#6#7#8%
423     \csname\XINT_bth_loop
424 }%
```

5.10 \xintHexToBin

Completely rewritten for 1.2m.

Attention this macro is not robust against arguments expanding after themselves.

Only up to three zeros are removed on front of output: if the input had a leading zero, there will be a leading zero (and then possibly 4n of them if inputs had more leading zeroes) on output.

Rewritten again at 1.2n for \csname governed expansion.

```

425 \def\xintHexToBin {\romannumeral0\xinthextobin }%
426 \def\xinthextobin #1%
427 {%
428     \expandafter\XINT_htb_checkin\romannumeral`&&@#1%
429     \xint_bye 23456789\xint_bye none\endcsname
430 }%
431 \def\XINT_htb_checkin #1%
432 {%
433     \xint_UDsignfork
434     #1\XINT_htb_N
435     -{\XINT_htb_main #1}%
436     \krof
437 }%
438 \def\XINT_htb_N {\expandafter-\romannumeral0\XINT_htb_main }%
439 \def\XINT_htb_main {\csname XINT_htb_cuz\csname\XINT_htb_loop}%
440 \def\XINT_htb_loop #1#2#3#4#5#6#7#8#9%
441 {%
442     XINT_cshtb_#1%
443     \csname XINT_cshtb_#2%
444     \csname XINT_cshtb_#3%
445     \csname XINT_cshtb_#4%
446     \csname XINT_cshtb_#5%
447     \csname XINT_cshtb_#6%
448     \csname XINT_cshtb_#7%
449     \csname XINT_cshtb_#8%
450     \csname XINT_cshtb_#9%
451     \csname \XINT_htb_loop
452 }%
453 \def\XINT_htb_cuz #1{%
454 \def\XINT_htb_cuz ##1##2##3##4%
455     {\expandafter#1\the\numexpr##1##2##3##4\relax}%
456 }\XINT_htb_cuz { }%
```

5.11 \xintCHexToBin

The 1.08 macro had same functionality as \xintHexToBin, and slightly different code, the 1.2m version has the same code as \xintHexToBin except that it does not remove leading zeros from output: if the input had N hexadecimal digits, the output will have exactly 4N binary digits.

Rewritten again at 1.2n for \csname governed expansion.

5 Package *xintbinhex* implementation

```
457 \def\xintCHexToBin {\romannumeral0\xintchextobin }%
458 \def\xintchextobin #1%
459 {%
460   \expandafter\XINT_chtb_checkin\romannumeral`&&@#1%
461   \xint_bye 23456789\xint_bye none\endcsname
462 }%
463 \def\XINT_chtb_checkin #1%
464 {%
465   \xint_UDsignfork
466   #1\XINT_chtb_N
467   -{\XINT_chtb_main #1}%
468   \krof
469 }%
470 \def\XINT_chtb_N {\expandafter-\romannumeral0\XINT_chtb_main }%
471 \def\XINT_chtb_main {\csname space\csname\XINT_htb_loop}%
472 \XINT_restorecatcodes_endinput%
```

6 Package *xintgcd* implementation

.1	Catcodes, ε -TeX and reload detection . . .	166	.7	<code>\xintBezoutAlgorithm</code>	174
.2	Package identification	167	.8	<code>\xintGCDof</code>	176
.3	<code>\xintGCD</code> , <code>\xintiiGCD</code>	167	.9	<code>\xintLCMof</code>	176
.4	<code>\xintLCM</code> , <code>\xintiiLCM</code>	168	.10	<code>\xintTypesetEuclideanAlgorithm</code>	176
.5	<code>\xintBezout</code>	168	.11	<code>\xintTypesetBezoutAlgorithm</code>	177
.6	<code>\xintEuclideanAlgorithm</code>	172			

The commenting is currently (2018/02/06) very sparse. Release 1.09h has modified a bit the `\xintTypesetEuclideanAlgorithm` and `\xintTypesetBezoutAlgorithm` layout with respect to line indentation in particular. And they use the `xinttools` `\xintloop` rather than the Plain TeX or \mathbb{X} 's `\loop`.

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

Breaking change at 1.2p: `\xintBezout{A}{B}` formerly had output $\{A\}{B}\{U\}{V}\{D\}$ with $AU-BV=D$, now it is $\{U\}{V}\{D\}$ with $AU+BV=D$.

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 [2018/02/06 1.2q Euclidean algorithm with xint package (JFB)]%

```

6.3 *\xintGCD*, *\xintiGCD*

1.09a added *\xintnum* filtering from *\xintiabs*. This is a bit overhead but makes it easier for the gcd function in *\xintexpr*.

1.1a defines *\xintiGCD* to avoid overhead in *\xintiexpr*.

1.2p: but 1.2o deprecated *\xintiAbs*, and in fact *\xintGCD* should not have any *\xintNum* overhead for consistency with other xint macros. But well, it would be breaking change to modify this now. We can not use *\xintAbs* which will create a fraction $A/1$, so we use *\xintNum* directly.

```

47 \def\xintGCD {\romannumeral0\xintgcd}%
48 \def\xintgcd #1#2{\xintiigcd {\xintNum{#1}}{\xintNum{#2}}}%
49 \def\xintiGCD {\romannumeral0\xintiigcd}%
50 \def\xintiigcd #1%
51 {%
52 \expandafter\XINT_iigcd\expandafter{\romannumeral0\xintiabs{#1}}%
53}%
54 \def\XINT_iigcd #1#2%
55 {%
56 \expandafter\XINT_gcd_fork\romannumeral0\xintiabs{#2}\Z #1\Z
57}%

```

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

```

58 \def\XINT_gcd_fork #1#2\Z #3#4\Z
59 {%
60 \xint_UDzerofork
61 #1\XINT_gcd_BisZero
62 #3\XINT_gcd_AisZero
63 0\XINT_gcd_loop
64 \krof
65 {#1#2}{#3#4}%
66}%
67 \def\XINT_gcd_AisZero #1#2{ #1}%
68 \def\XINT_gcd_BisZero #1#2{ #2}%

```

```

69 \def\XINT_gcd_CheckRem #1#2\Z
70 {%
71   \xint_gob_til_zero #1\XINT_gcd_end0\XINT_gcd_loop {#1#2}%
72 }%
73 \def\XINT_gcd_end0\XINT_gcd_loop #1#2{ #2}%

```

#1=B, #2=A. \XINT_div_prepare{#1}{#2} divides A by B.

```

74 \def\XINT_gcd_loop #1#2%
75 {%
76   \expandafter\expandafter\expandafter
77     \XINT_gcd_CheckRem
78   \expandafter\xint_seconddoftwo
79   \romannumeral0\XINT_div_prepare {#1}{#2}\Z
80   {#1}%
81 }%

```

6.4 \xintLCM, \xintiilCM

See comments of \xintGCD.

```

82 \def\xintLCM {\romannumeral0\xintlcm}%
83 \def\xintlcm #1#2{\xintiilcm{\xintNum{#1}}{\xintNum{#2}}}%
84 \def\xintiilCM {\romannumeral0\xintiilcm}%
85 \def\xintiilcm #1%
86 {%
87   \expandafter\XINT_iilcm\expandafter{\romannumeral0\xintiiabs{#1}}%
88 }%
89 \def\XINT_iilcm #1#2%
90 {%
91   \expandafter\XINT_lcm_fork\romannumeral0\xintiiabs{#2}\Z #1\Z
92 }%
93 \def\XINT_lcm_fork #1#2\Z #3#4\Z
94 {%
95   \xint_UDzerofork
96   #1\XINT_lcm_BisZero
97   #3\XINT_lcm_AisZero
98   0\expandafter
99   \krof
100   \XINT_lcm_notzero\expandafter{\romannumeral0\XINT_gcd_loop {#1#2}{#3#4}}%
101   {#1#2}{#3#4}%
102 }%
103 \def\XINT_lcm_AisZero #1#2#3#4#5{ 0}%
104 \def\XINT_lcm_BisZero #1#2#3#4#5{ 0}%
105 \def\XINT_lcm_notzero #1#2#3{\xintiimul {#2}{\xintiiQuo{#3}{#1}}}%

```

6.5 \xintBezout

\xintBezout{#1}{#2} produces $\{U\}\{V\}\{D\}$ with $UA+VB=D$, $D = \text{PGCD}(A,B)$ (non-positive), where #1 and #2 f-expand to big integers A and B.

I had not checked this macro for about three years when I realized in January 2017 that `\xintBezout{A}{B}` was buggy for the cases $A = 0$ or $B = 0$. I fixed that blemish in 1.21 but overlooked the

other blemish that $\backslash\text{xintBezout}\{A\}\{B\}$ with A multiple of B produced a coefficient U as -0 in place of 0.

Hence I rewrote again for 1.2p. On this occasion I modified the output of the macro to be $\{U\}\{V\}\{D\}$ with $AU+BV=D$, formerly it was $\{A\}\{B\}\{U\}\{V\}\{D\}$ with $AU - BV = D$. This is quite breaking change!

Note in particular change of sign of V.

I don't know why I had designed this macro to contain $\{A\}\{B\}$ in its output. Perhaps I initially intended to output $\{A//D\}\{B//D\}$ (but forgot), as this is actually possible from outcome of the last iteration, with no need of actually dividing. Current code however arranges to skip this last update, as U and V are already furnished by the iteration prior to realizing that the last non-zero remainder was found.

Also 1.2l raised `InvalidOperation` if both A and B vanished, but I removed this behaviour at 1.2p.

```

106 \def\xintBezout {\romannumeral0\xintbezout }%
107 \def\xintbezout #1%
108 {%
109   \expandafter\XINT_bezout\expandafter {\romannumeral0\xintnum{#1}}%
110 }%
111 \def\XINT_bezout #1#2%
112 {%
113   \expandafter\XINT_bezout_fork \romannumeral0\xintnum{#2}\Z #1\Z
114 }%

```

#3#4 = A, #1#2=B. Micro improvement for 1.2l.

```

115 \def\XINT_bezout_fork #1#2\Z #3#4\Z
116 {%
117   \xint_UDzerosfork
118   #1#3\XINT_bezout_botharezero
119   #10\XINT_bezout_secondiszero
120   #30\XINT_bezout_firstiszero
121   00\xint_UDsignsfork
122   \krof
123   #1#3\XINT_bezout_minusminus % A < 0, B < 0
124   #1-\XINT_bezout_minusplus % A > 0, B < 0
125   #3-\XINT_bezout_plusminus % A < 0, B > 0
126   --\XINT_bezout_plusplus % A > 0, B > 0
127   \krof
128   {#2}\{#4}\{#1#3} #1#2=B, #3#4=A
129 }%
130 \def\XINT_bezout_botharezero #1\krof#2#300{\{0\}\{0\}\{0\}}%
131 \def\XINT_bezout_firstiszero #1\krof#2#3#4#5%
132 {%
133   \xint_UDsignfork
134   #4{\{0\}\{-1\}\{#2\}}%
135   -{\{0\}\{1\}\{#4#2\}}%
136   \krof
137 }%
138 \def\XINT_bezout_secondiszero #1\krof#2#3#4#5%
139 {%
140   \xint_UDsignfork
141   #5{\{-1\}\{0\}\{#3\}}%
142   -{\{1\}\{0\}\{#5#3\}}%

```

```

143 \krof
144 }%

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

145 \def\XINT_bezout_minusminus #1#2#3#4%
146 {%
147 \expandafter\XINT_bezout_mm_post
148 \romannumeral0\expandafter\XINT_bezout_preloop_a
149 \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
150 }%
151 \def\XINT_bezout_mm_post #1#2%
152 {%
153 \expandafter\XINT_bezout_mm_postb\expandafter
154 {\romannumeral0\xintiopp{#2}}{\romannumeral0\xintiopp{#1}}%
155 }%
156 \def\XINT_bezout_mm_postb #1#2{\expandafter{#2}{#1}}%

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

157 \def\XINT_bezout_minusplus #1#2#3#4%
158 {%
159 \expandafter\XINT_bezout_mp_post
160 \romannumeral0\expandafter\XINT_bezout_preloop_a
161 \romannumeral0\XINT_div_prepare {#1}{#4#2}{#1}%
162 }%
163 \def\XINT_bezout_mp_post #1#2%
164 {%
165 \expandafter\xint_exchangetwo_keepbraces\expandafter
166 {\romannumeral0\xintiopp {#2}}{#1}%
167 }%

plusminus A < 0, B > 0

168 \def\XINT_bezout_plusminus #1#2#3#4%
169 {%
170 \expandafter\XINT_bezout_pm_post
171 \romannumeral0\expandafter\XINT_bezout_preloop_a
172 \romannumeral0\XINT_div_prepare {#3#1}{#2}{#3#1}%
173 }%
174 \def\XINT_bezout_pm_post #1{\expandafter{\romannumeral0\xintiopp{#1}}}%

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

175 \def\XINT_bezout_plusplus #1#2#3#4%
176 {%
177 \expandafter\XINT_bezout_preloop_a
178 \romannumeral0\XINT_div_prepare {#3#1}{#4#2}{#3#1}%
179 }%

n = 0: BA1001 (B, A, e=1, vv, uu, v, u)
r(1)=B, r(0)=A, après n étapes {r(n+1)}{r(n)}{vv}{uu}{v}{u}
q(n) quotient de r(n-1) par r(n)
si reste nul, exit et renvoie U = -e*uu, V = e*vv, A*U+B*V=D

```

```

sinon mise à jour
  vv, v = q * vv + v, vv
  uu, u = q * uu + u, uu
  e = -e
  puis calcul quotient reste et itération

```

We arrange for `\xintiiMul` sub-routine to be called only with positive arguments, thus skipping some un-needed sign parsing there. For that though we have to screen out the special cases A divides B, or B divides A. And we first want to exchange A and B if $A < B$. These special cases are the only one possibly leading to U or V zero (for A and B positive which is the case here.) Thus the general case always leads to non-zero U and V's and assigning a final sign is done simply adding a - to one of them, with no fear of producing -0.

```

180 \def\xINT_bezout_preloop_a #1#2#3%
181 {%
182   \if0#1\xint_dothis\xINT_bezout_preloop_exchange\fi
183   \if0#2\xint_dothis\xINT_bezout_preloop_exit\fi
184   \xint_orthat{\expandafter\xINT_bezout_loop_B}%
185   \romannumeral0\xINT_div_prepare {#2}{#3}{#2}{#1}110%
186 }%
187 \def\xINT_bezout_preloop_exit
188   \romannumeral0\xINT_div_prepare #1#2#3#4#5#6#7%
189 {%
190   {0}{1}{#2}%
191 }%
192 \def\xINT_bezout_preloop_exchange
193 {%
194   \expandafter\xint_exchangetwo_keepbraces
195   \romannumeral0\expandafter\xINT_bezout_preloop_A
196 }%
197 \def\xINT_bezout_preloop_A #1#2#3#4%
198 {%
199   \if0#2\xint_dothis\xINT_bezout_preloop_exit\fi
200   \xint_orthat{\expandafter\xINT_bezout_loop_B}%
201   \romannumeral0\xINT_div_prepare {#2}{#3}{#2}{#1}%
202 }%
203 \def\xINT_bezout_loop_B #1#2%
204 {%
205   \if0#2\expandafter\xINT_bezout_exitA
206   \else\expandafter\xINT_bezout_loop_C
207   \fi {#1}{#2}%
208 }%

```

We use the fact that the `\romannumeral-`0` (or equivalent) done by `\xintiiadd` will absorb the initial space token left by `\XINT_mul_plusplus` in its output.

We arranged for operands here to be always positive which is needed for `\XINT_mul_plusplus` entry point (last time I checked...). Admittedly this kind of optimization is not good for maintenance of code, but I can't resist temptation of limiting the shuffling around of tokens...

```

209 \def\xINT_bezout_loop_C #1#2#3#4#5#6#7%
210 {%
211   \expandafter\xINT_bezout_loop_D\expandafter
212   {\romannumeral0\xintiiadd{\XINT_mul_plusplus{}}{#1\xint:#4\xint:}{#6}}%
213   {\romannumeral0\xintiiadd{\XINT_mul_plusplus{}}{#1\xint:#5\xint:}{#7}}%

```

```

214     {#2}{#3}{#4}{#5}%
215 }%
216 \def\XINT_bezout_loop_D #1#2%
217 {%
218     \expandafter\XINT_bezout_loop_E\expandafter{#2}{#1}%
219 }%
220 \def\XINT_bezout_loop_E #1#2#3#4%
221 {%
222     \expandafter\XINT_bezout_loop_b
223     \romannumeral0\XINT_div_prepare {#3}{#4}{#3}{#2}{#1}%
224 }%
225 \def\XINT_bezout_loop_b #1#2%
226 {%
227     \if0#2\expandafter\XINT_bezout_exita
228     \else\expandafter\XINT_bezout_loop_c
229     \fi {#1}{#2}%
230 }%
231 \def\XINT_bezout_loop_c #1#2#3#4#5#6#7%
232 {%
233     \expandafter\XINT_bezout_loop_d\expandafter
234     {\romannumeral0\xintiadd{\XINT_mul_plusplus{}}{#1\xint:#4\xint:}{#6}}%
235     {\romannumeral0\xintiadd{\XINT_mul_plusplus{}}{#1\xint:#5\xint:}{#7}}%
236     {#2}{#3}{#4}{#5}%
237 }%
238 \def\XINT_bezout_loop_d #1#2%
239 {%
240     \expandafter\XINT_bezout_loop_e\expandafter{#2}{#1}%
241 }%
242 \def\XINT_bezout_loop_e #1#2#3#4%
243 {%
244     \expandafter\XINT_bezout_loop_B
245     \romannumeral0\XINT_div_prepare {#3}{#4}{#3}{#2}{#1}%
246 }%

```

sortir U, V, D mais on a travaillé avec vv, uu, v, u dans cet ordre.

The code is structured so that #4 and #5 are guaranteed non-zero if we exit here, hence we can not create a -0 in output.

```

247 \def\XINT_bezout_exita #1#2#3#4#5#6#7{{-#5}{#4}{#3}}%
248 \def\XINT_bezout_exita #1#2#3#4#5#6#7{{#5}{-#4}{#3}}%

```

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.

Formerly, used \xintiabs, but got deprecated at 1.2o.

```

249 \def\xintEuclideanAlgorithm {\romannumeral0\xinteucclideanalgorithm}%
250 \def\xinteucclideanalgorithm #1%
251 {%
252     \expandafter\XINT_euc\expandafter{\romannumeral0\xintiabs{\xintNum{#1}}}%
253 }%
254 \def\XINT_euc #1#2%

```

```

255 {%
256   \expandafter\XINT_euc_fork\romannumeral0\xinttiabs{\xintNum{#2}}\Z #1\Z
257 }%

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

258 \def\XINT_euc_fork #1#2\Z #3#4\Z
259 {%
260   \xint_UDzerofork
261   #1\XINT_euc_BisZero
262   #3\XINT_euc_AisZero
263   0\XINT_euc_a
264   \krof
265   {0}{#1#2}{#3#4}{#3#4}{#1#2}}\Z
266 }%

Le {} pour protéger {{A}{B}} si on s'arrête après une étape (B divise A). On va renvoyer:
{N}{A}{D=r(n)}{B}{q1}{r1}{q2}{r2}{q3}{r3}...{qN}{rN=0}

267 \def\XINT_euc_AisZero #1#2#3#4#5#6{{1}{0}{#2}{#2}{0}{0}}%
268 \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

269 \def\XINT_euc_a #1#2#3%
270 {%
271   \expandafter\XINT_euc_b\the\numexpr #1+\xint_c_i\expandafter.%
272   \romannumeral0\XINT_div_prepare {#2}{#3}{#2}%
273 }%

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

274 \def\XINT_euc_b #1.#2#3#4%
275 {%
276   \XINT_euc_c #3\Z {#1}{#3}{#4}{{#2}{#3}}%
277 }%

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.

278 \def\XINT_euc_c #1#2\Z
279 {%
280   \xint_gob_til_zero #1\XINT_euc_end0\XINT_euc_a
281 }%

{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}

282 \def\XINT_euc_end0\XINT_euc_a #1#2#3#4\Z%
283 {%
284   \expandafter\XINT_euc_end_a
285   \romannumeral0%
286   \XINT_rord_main {#4}{#1}{#3}}%

```

```

287 \xint:
288 \xint_bye\xint_bye\xint_bye\xint_bye
289 \xint_bye\xint_bye\xint_bye\xint_bye
290 \xint:
291 }%
292 \def\xINT_euc_end_a #1#2#3{{#1}{{#3}}{{#2}}}%

```

6.7 \xintBezoutAlgorithm

Pour Bezout: objectif, renvoyer

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

```

293 \def\xintBezoutAlgorithm {\romannumeral0\xintbezoutalgorithm }%
294 \def\xintbezoutalgorithm #1%
295 {%
296 \expandafter \XINT_bezalg
297 \expandafter{\romannumeral0\xinttiabs{\xintNum{#1}}}%
298 }%
299 \def\XINT_bezalg #1#2%
300 {%
301 \expandafter\XINT_bezalg_fork\romannumeral0\xinttiabs{\xintNum{#2}}\Z #1\Z
302 }%

```

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

```

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

```

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

```

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

```

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

```

319 \def\XINT_bezalg_b #1.#2#3#4#5#6#7#8%
320 {%
321 \expandafter\XINT_bezalg_c\expandafter
322 {\romannumeral0\xinttiadd {\xinttiMul {#6}{{#2}}{{#8}}}%

```

```

323      {\romannumeral0\xintiiadd {\xintiiMul {#5}{#2}}{#7}}%
324      {#1}{#2}{#3}{#4}{#5}{#6}%
325 }%

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

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

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

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

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

334 \def\XINT_bezalg_e #1#2\Z
335 {%
336   \xint_gob_til_zero #1\XINT_bezalg_end0\XINT_bezalg_a
337 }%

Ici r(n+1) = 0. On arrête on se prépare à inverser.
{n+1}{r(n+1)}{r(n)}{alpha(n+1)}{beta(n+1)}{alpha(n)}{beta(n)}
{q,r,alpha,beta(n+1)}...{A}{B}}{\Z
On veut renvoyer
{N}{A}{0}{1}{D=r(n)}{B}{1}{0}{q1}{r1}{alpha1=q1}{beta1=1}
{q2}{r2}{alpha2}{beta2}...{qN}{rN=0}{alphaN=A/D}{betaN=B/D}

338 \def\XINT_bezalg_end0\XINT_bezalg_a #1#2#3#4#5#6#7#8\Z
339 {%
340   \expandafter\XINT_bezalg_end_a
341   \romannumeral0%
342   \XINT_rord_main {}#8{{#1}{#3}}%
343   \xint:
344     \xint_bye\xint_bye\xint_bye\xint_bye
345     \xint_bye\xint_bye\xint_bye\xint_bye
346   \xint:
347 }%

{N}{D}{A}{B}{q1}{r1}{alpha1=q1}{beta1=1}{q2}{r2}{alpha2}{beta2}
...{qN}{rN=0}{alphaN=A/D}{betaN=B/D}
On veut renvoyer
{N}{A}{0}{1}{D=r(n)}{B}{1}{0}{q1}{r1}{alpha1=q1}{beta1=1}
{q2}{r2}{alpha2}{beta2}...{qN}{rN=0}{alphaN=A/D}{betaN=B/D}

348 \def\XINT_bezalg_end_a #1#2#3#4{{#1}{#3}{0}{1}{#2}{#4}{1}{0}}%

```

6.8 `\xintGCDof`

1.21 adds protection against items being non-terminated `\the\numexpr...`

```

349 \def\xintGCDof      {\romannumeral0\xintgcdof }%
350 \def\xintgcdof      #1{\expandafter\XINT_gcdof_a\romannumeral`&&@#1\xint:}%
351 \def\XINT_gcdof_a    #1{\expandafter\XINT_gcdof_b\romannumeral`&&@#1!}%
352 \def\XINT_gcdof_b    #1!#2{\expandafter\XINT_gcdof_c\romannumeral`&&@#2!{#1}!}%
353 \def\XINT_gcdof_c    #1{\xint_gob_til_xint: #1\XINT_gcdof_e\xint:\XINT_gcdof_d #1}%
354 \def\XINT_gcdof_d    #1!{\expandafter\XINT_gcdof_b\romannumeral0\xintgcd {#1}}%
355 \def\XINT_gcdof_e    #1!#2!{ #2}%

```

6.9 `\xintLCMof`

New with 1.09a

1.21 adds protection against items being non-terminated `\the\numexpr...`

```

356 \def\xintLCMof      {\romannumeral0\xintlcmof }%
357 \def\xintlcmof      #1{\expandafter\XINT_lcmof_a\romannumeral`&&@#1\xint:}%
358 \def\XINT_lcmof_a    #1{\expandafter\XINT_lcmof_b\romannumeral`&&@#1!}%
359 \def\XINT_lcmof_b    #1!#2{\expandafter\XINT_lcmof_c\romannumeral`&&@#2!{#1}!}%
360 \def\XINT_lcmof_c    #1{\xint_gob_til_xint: #1\XINT_lcmof_e\xint:\XINT_lcmof_d #1}%
361 \def\XINT_lcmof_d    #1!{\expandafter\XINT_lcmof_b\romannumeral0\xintlcm {#1}}%
362 \def\XINT_lcmof_e    #1!#2!{ #2}%

```

6.10 `\xintTypesetEuclideanAlgorithm`

TYPESETTING

Organisation:

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

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

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

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

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

```

363 \def\xintTypesetEuclideanAlgorithm {%
364   \unless\ifdefined\xintAssignArray
365     \errmessage
366     {xintgcd: package xinttools is required for \string\xintTypesetEuclideanAlgorithm}%
367     \expandafter\xint_gobble_iii
368   \fi
369   \XINT_TypesetEuclideanAlgorithm
370 }%
371 \def\XINT_TypesetEuclideanAlgorithm #1#2%
372 {% l'algo remplace #1 et #2 par |#1| et |#2|
373   \par
374   \begingroup
375     \xintAssignArray\xintEuclideanAlgorithm {#1}{#2}\to\U
376     \edef\A{\U2}\edef\B{\U4}\edef\N{\U1}%
377     \setbox 0 \vbox{\halign {$##$\cr \A\cr \B \cr}}%
378     \count 255 1

```



```

379 \xintloop
380 \indent\hbox to \wd 0 {\hfil$U{\numexpr 2*\count255\relax}$}%
381 ${} = \U{\numexpr 2*\count255 + 3\relax}
382 \times \U{\numexpr 2*\count255 + 2\relax}
383 + \U{\numexpr 2*\count255 + 4\relax}$%
384 \ifnum \count255 < \N
385 \par
386 \advance \count255 1
387 \repeat
388 \endgroup
389 }%

```

6.11 \xintTypesetBezoutAlgorithm

Pour Bezout on a: $\{N\}\{A\}\{0\}\{1\}\{D=r(n)\}\{B\}\{1\}\{0\}\{q_1\}\{r_1\}\{\alpha_1=q_1\}\{\beta_1=1\}$
 $\{q_2\}\{r_2\}\{\alpha_2\}\{\beta_2\}\dots\{q_N\}\{r_N=0\}\{\alpha_N=A/D\}\{\beta_N=B/D\}$
 Donc $4N+8$ termes: $U_1 = N$, $U_2 = A$, $U_5 = D$, $U_6 = B$, $q_1 = U_9$, $q_n = U_{4n+5}$, n au moins 1
 $r_n = U_{4n+6}$, n au moins -1
 $\alpha(n) = U_{4n+7}$, n au moins -1
 $\beta(n) = U_{4n+8}$, n au moins -1

1.09h uses \xintloop, and \par rather than \endgraf; and no more \parindent0pt

```

390 \def\xintTypesetBezoutAlgorithm {%
391 \unless\ifdefined\xintAssignArray
392 \errmessage
393 {xintgcd: package xinttools is required for \string\xintTypesetBezoutAlgorithm}%
394 \expandafter\xint_gobble_iii
395 \fi
396 \XINT_TypesetBezoutAlgorithm
397 }%
398 \def\XINT_TypesetBezoutAlgorithm #1#2%
399 {%
400 \par
401 \begingroup
402 \xintAssignArray\xintBezoutAlgorithm {#1}{#2}\to\BEZ
403 \edef\A{\BEZ2}\edef\B{\BEZ6}\edef\N{\BEZ1}% A = |#1|, B = |#2|
404 \setbox 0 \vbox{\halign {$##$\cr \A\cr \B \cr}}%
405 \count255 1
406 \xintloop
407 \indent\hbox to \wd 0 {\hfil$\BEZ{4*\count255 - 2}$}%
408 ${} = \BEZ{4*\count255 + 5}
409 \times \BEZ{4*\count255 + 2}
410 + \BEZ{4*\count255 + 6}$\hfill\break
411 \hbox to \wd 0 {\hfil$\BEZ{4*\count255 + 7}$}%
412 ${} = \BEZ{4*\count255 + 5}
413 \times \BEZ{4*\count255 + 3}
414 + \BEZ{4*\count255 - 1}$\hfill\break
415 \hbox to \wd 0 {\hfil$\BEZ{4*\count255 + 8}$}%
416 ${} = \BEZ{4*\count255 + 5}
417 \times \BEZ{4*\count255 + 4}
418 + \BEZ{4*\count255 }$
419 \par
420 \ifnum \count255 < \N

```

```

421   \advance \count255 1
422   \repeat
423   \edef\U{\BEZ{4*\N + 4}}%
424   \edef\V{\BEZ{4*\N + 3}}%
425   \edef\D{\BEZ5}%
426   \ifodd\N
427     $\U\times A - \V\times B = -\D$%
428   \else
429     $\U\times A - \V\times B = \D$%
430   \fi
431   \par
432   \endgroup
433 }%
434 \XINT_restorecatcodes_endinput%

```

7 Package *xintfrac* implementation

.1	Catcodes, ε -T _E X and reload detection . . .	179	.42	<code>\xintPrd</code>	213
.2	Package identification	180	.43	<code>\xintDiv</code>	213
.3	Macros now deprecated in <i>xintcore</i> or <i>xinti</i> 180		.44	<code>\xintDivFloor</code>	213
.4	<code>\XINT_cntSgnFork</code>	181	.45	<code>\xintDivTrunc</code>	214
.5	<code>\xintLen</code>	181	.46	<code>\xintDivRound</code>	214
.6	<code>\XINT_outfrac</code>	182	.47	<code>\xintModTrunc</code>	214
.7	<code>\XINT_inFrac</code>	182	.48	<code>\xintDivMod</code>	215
.8	<code>\XINT_frac_gen</code>	184	.49	<code>\xintMod</code>	216
.9	<code>\XINT_factoritens</code> , <code>\XINT_cuz_cnt</code> . . .	186	.50	<code>\xintIsOne</code>	216
.10	<code>\xintRaw</code>	188	.51	<code>\xintGeq</code>	217
.11	<code>\xintPRaw</code>	188	.52	<code>\xintMax</code>	218
.12	<code>\xintRawWithZeros</code>	189	.53	<code>\xintMaxof</code>	218
.13	<code>\xintFloor</code> , <code>\xintiFloor</code>	189	.54	<code>\xintMin</code>	219
.14	<code>\xintCeil</code> , <code>\xintiCeil</code>	190	.55	<code>\xintMinof</code>	219
.15	<code>\xintNumerator</code>	190	.56	<code>\xintCmp</code>	220
.16	<code>\xintDenominator</code>	190	.57	<code>\xintAbs</code>	221
.17	<code>\xintFrac</code>	190	.58	<code>\xintOpp</code>	221
.18	<code>\xintSignedFrac</code>	191	.59	<code>\xintSgn</code>	221
.19	<code>\xintFwOver</code>	192	.60	Floating point macros	221
.20	<code>\xintSignedFwOver</code>	192	.61	<code>\xintFloat</code>	222
.21	<code>\xintREZ</code>	193	.62	<code>\XINTinFloat</code> , <code>\XINTinFloatS</code>	224
.22	<code>\xintE</code>	193	.63	<code>\xintPFloat</code>	230
.23	<code>\xintIrr</code>	194	.64	<code>\XINTinFloatFracdigits</code>	231
.24	<code>\xintifInt</code>	195	.65	<code>\xintFloatAdd</code> , <code>\XINTinFloatAdd</code> . . .	232
.25	<code>\xintJrr</code>	195	.66	<code>\xintFloatSub</code> , <code>\XINTinFloatSub</code> . . .	233
.26	<code>\xintTFrac</code>	196	.67	<code>\xintFloatMul</code> , <code>\XINTinFloatMul</code> . . .	234
.27	<code>\xintTrunc</code> , <code>\xintiTrunc</code>	197	.68	<code>\xintFloatDiv</code> , <code>\XINTinFloatDiv</code> . . .	234
.28	<code>\xintTTrunc</code>	200	.69	<code>\xintFloatPow</code> , <code>\XINTinFloatPow</code> . . .	235
.29	<code>\xintNum</code>	200	.70	<code>\xintFloatPower</code> , <code>\XINTinFloatPower</code> .	239
.30	<code>\xintRound</code> , <code>\xintiRound</code>	200	.71	<code>\xintFloatFac</code> , <code>\XINTFloatFac</code>	243
.31	<code>\xintXTrunc</code>	200	.72	<code>\xintFloatPFactorial</code> , <code>\XINTinFloatP-</code> <code>Factorial</code>	248
.32	<code>\xintDigits</code>	207	.73	<code>\xintFloatBinomial</code> , <code>\XINTinFloatBino-</code> <code>mial</code>	252
.33	<code>\xintAdd</code>	208	.74	<code>\xintFloatSqrt</code> , <code>\XINTinFloatSqrt</code> . . .	254
.34	<code>\xintSub</code>	209	.75	<code>\xintFloatE</code> , <code>\XINTinFloatE</code>	256
.35	<code>\xintSum</code>	210	.76	<code>\XINTinFloatMod</code>	257
.36	<code>\xintMul</code>	210	.77	<code>\XINTinFloatDivFloor</code>	257
.37	<code>\xintSqr</code>	210	.78	<code>\XINTinFloatDivMod</code>	258
.38	<code>\xintPow</code>	211		At End of L ^A T _E X Document deprecation message	258
.39	<code>\xintFac</code>	212			
.40	<code>\xintBinomial</code>	212			
.41	<code>\xintPFactorial</code>	212			

The commenting is currently (2018/02/06) 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 [2018/02/06 1.2q Expandable operations on fractions (JFB)]%

```

7.3 Macros now deprecated in *xintcore* or *xint*

1.2o

```

47 \def\xintSgn {\romannumeral0\xintsgn}% deprecated from xintcore
48 \def\xintCmp {\romannumeral0\xintcmp}% deprecated from xintcore

```

```

49 \def\xintEq {\romannumeral0\xinteq }% deprecated from xint
50 \def\xintNeq #1#2% renamed into \xintNotEq
51   {\romannumeral0\XINT_signaldeprecated{xintfrac}{xintNeq}\xintifeq {#1}{#2}{0}{1}}%
52 \def\xintNotEq #1#2{\romannumeral0\xintifeq {#1}{#2}{0}{1}}%
53 %\def\xintGeq {\romannumeral0\xintgeq }% further down
54 \def\xintGt {\romannumeral0\xintgt }%
55 \def\xintLt {\romannumeral0\xintlt }%
56 \def\xintGtorEq #1#2{\romannumeral0\xintiflt {#1}{#2}{0}{1}}%
57 \def\xintLtorEq #1#2{\romannumeral0\xintifgt {#1}{#2}{0}{1}}%
58 \def\xintIsZero {\romannumeral0\xintiszero }%
59 \def\xintIsNotZero{\romannumeral0\xintisnotzero }%
60 \def\xintIsOne {\romannumeral0\xintisone }%
61 \def\xintOdd {\romannumeral0\xintodd }%
62 \def\xintEven {\romannumeral0\xinteven }%
63 \def\xintifSgn{\romannumeral0\xintifsgn }%
64 \def\xintifCmp{\romannumeral0\xintifcmp }%
65 \def\xintifEq {\romannumeral0\xintifeq }%
66 \def\xintifGt {\romannumeral0\xintifgt }%
67 \def\xintifLt {\romannumeral0\xintiflt }%
68 \def\xintifZero {\romannumeral0\xintifzero }%
69 \def\xintifNotZero{\romannumeral0\xintifnotzero }%
70 \def\xintifOne {\romannumeral0\xintifone }%
71 \def\xintifOdd {\romannumeral0\xintifodd }%

```

7.4 \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.

```

72 \def\XINT_cntSgnFork #1%
73 {%
74   \ifcase #1\expandafter\xint_secondofthree
75     \or\expandafter\xint_thirdofthree
76     \else\expandafter\xint_firstofthree
77   \fi
78 }%

```

7.5 \xintLen

The used formula is disputable, the idea is that A/1 and A should have same length. Venerable code rewritten for 1.2i, following updates to \xintLength in xintkernel.sty. And sadly, I forgot on this occasion that this macro is not supposed to count the sign... Fixed in 1.2k.

```

79 \def\xintLen {\romannumeral0\xintlen }%
80 \def\xintlen #1%
81 {%
82   \expandafter\XINT_flen\romannumeral0\XINT_infrac {#1}%
83 }%
84 \def\XINT_flen#1{\def\XINT_flen ##1##2##3%
85 {%
86   \expandafter#1%

```

```

87 \the\numexpr \XINT_abs##1+%
88 \XINT_len_fork ##2##3\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
89 \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
90 \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye-\xint_c_i
91 \relax
92 }}\XINT_flen{ }%

```

7.6 \XINT_outfrac

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

```

93 \def\XINT_outfrac #1#2#3%
94 {%
95   \ifcase\XINT_cntSgn #3\xint:
96     \expandafter \XINT_outfrac_divisionbyzero
97   \or
98     \expandafter \XINT_outfrac_P
99   \else
100    \expandafter \XINT_outfrac_N
101   \fi
102   {#2}\{#3}\{#1}%
103 }%
104 \def\XINT_outfrac_divisionbyzero #1#2%
105 {%
106   \XINT_signalcondition{DivisionByZero}{Division of #1 by #2}\{0/1[0]}%
107 }%
108 \def\XINT_outfrac_P#1{%
109 \def\XINT_outfrac_P ##1##2%
110   {\if0\XINT_Sgn ##1\xint:\expandafter\XINT_outfrac_Zero\fi#1##1/##2}%
111 }\XINT_outfrac_P{ }%
112 \def\XINT_outfrac_Zero #1[#2]{ 0/1[0]}%
113 \def\XINT_outfrac_N #1#2%
114 {%
115   \expandafter\XINT_outfrac_N_a\expandafter
116   {\romannumeral0\XINT_opp #2}\{\romannumeral0\XINT_opp #1}%
117 }%
118 \def\XINT_outfrac_N_a #1#2%
119 {%
120   \expandafter\XINT_outfrac_P\expandafter {#2}\{#1}%
121 }%

```

7.7 \XINT_inFrac

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 $\{\text{exponent}\}\{\text{Numerator}\}\{\text{Denominator}\}$ 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.

1.2l fixes frailty of `\XINT_infrac` (hence basically of all `xintfrac` macros) respective to non terminated `\numexpr` input: `\xintRaw{\the\numexpr1}` for example. The issue was that `\numexpr` sees the `/` and expands what's next. But even `\numexpr 1//` for example creates an error, and to my mind this is a defect of `\numexpr`. It should be able to trace back and see that `/` was used as delimiter not as operator. Anyway, I thus fixed this problem belatedly here regarding `\XINT_infrac`.

```

122 \def\XINT_inFrac {\romannumeral0\XINT_infrac }%
123 \def\XINT_infrac #1%
124 {%
125   \expandafter\XINT_infrac_fork\romannumeral`&&@#1\xint:/\XINT_W[\XINT_W\XINT_T
126 }%
127 \def\XINT_infrac_fork #1[#2%
128 {%
129   \xint_UDXINTWfork
130   #2\XINT_frac_gen           % input has no brackets [N]
131   \XINT_W\XINT_infrac_res_a % there is some [N], must be strict A[N] or A/B[N] input
132   \krof
133   #1[#2%
134 }%
135 \def\XINT_infrac_res_a #1%
```

```

136 {%
137   \xint_gob_til_zero #1\XINT_infrac_res_zero 0\XINT_infrac_res_b #1%
138 }%

```

Note that input exponent is here ignored and forced to be zero.

```

139 \def\XINT_infrac_res_zero 0\XINT_infrac_res_b #1\XINT_T {{0}{0}{1}}%
140 \def\XINT_infrac_res_b #1/#2%
141 {%
142   \xint_UDXINTWfork
143   #2\XINT_infrac_res_ca      % it was A[N] input
144   \XINT_W\XINT_infrac_res_cb % it was A/B[N] input
145   \krof
146   #1/#2%
147 }%

```

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.

```

148 \def\XINT_infrac_res_ca #1[#2]\xint:/\XINT_W[\XINT_W\XINT_T
149   {\expandafter{\the\numexpr #2}{#1}{1}}%
150 \def\XINT_infrac_res_cb #1/#2[%
151   {\expandafter\XINT_infrac_res_cc\romannumeral`&&@#2~#1[%
152 \def\XINT_infrac_res_cc #1~#2[#3]\xint:/\XINT_W[\XINT_W\XINT_T
153   {\expandafter{\the\numexpr #3}{#2}{#1}}%

```

7.8 \XINT_frac_gen

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

```

154 \def\XINT_frac_gen #1/#2%
155 {%
156   \xint_UDXINTWfork
157   #2\XINT_frac_gen_A      % there was no /
158   \XINT_W\XINT_frac_gen_B % there was a /
159   \krof
160   #1/#2%
161 }%

```

Note that #1 is only expanded so far up to decimal mark or "e".

```

162 \def\XINT_frac_gen_A #1\xint:/\XINT_W [\XINT_W {\XINT_frac_gen_C 0~1!#1ee.\XINT_W }%
163 \def\XINT_frac_gen_B #1/#2\xint:/\XINT_W[%\XINT_W
164 {%
165   \expandafter\XINT_frac_gen_Ba
166   \romannumeral`&&@#2ee.\XINT_W\XINT_Z #1ee.%\XINT_W
167 }%
168 \def\XINT_frac_gen_Ba #1.#2%

```



```

169 {%
170   \xint_UDXINTWfork
171     #2\XINT_frac_gen_Bb
172     \XINT_W\XINT_frac_gen_Bc
173   \krof
174   #1.#2%
175 }%
176 \def\XINT_frac_gen_Bb #1e#2e#3\XINT_Z
177       {\expandafter\XINT_frac_gen_C\the\numexpr #2+\xint_c_~#1!}%
178 \def\XINT_frac_gen_Bc #1.#2e%
179 {%
180   \expandafter\XINT_frac_gen_Bd\romannumeral`&&@#2.#1e%
181 }%
182 \def\XINT_frac_gen_Bd #1.#2e#3e#4\XINT_Z
183 {%
184   \expandafter\XINT_frac_gen_C\the\numexpr #3-%
185   \numexpr\XINT_length_loop
186   #1\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
187   \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
188   \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
189   ~#2#1!%
190 }%
191 \def\XINT_frac_gen_C #1!#2.#3%
192 {%
193   \xint_UDXINTWfork
194     #3\XINT_frac_gen_Ca
195     \XINT_W\XINT_frac_gen_Cb
196   \krof
197   #1!#2.#3%
198 }%
199 \def\XINT_frac_gen_Ca #1~#2!#3e#4e#5\XINT_T
200 {%
201   \expandafter\XINT_frac_gen_F\the\numexpr #4-#1\expandafter
202   ~\romannumeral0\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
203   #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z~#3~%
204 }%
205 \def\XINT_frac_gen_Cb #1.#2e%
206 {%
207   \expandafter\XINT_frac_gen_Cc\romannumeral`&&@#2.#1e%
208 }%
209 \def\XINT_frac_gen_Cc #1.#2~#3!#4e#5e#6\XINT_T
210 {%
211   \expandafter\XINT_frac_gen_F\the\numexpr #5-#2-%
212
213   \numexpr\XINT_length_loop
214   #1\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
215   \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
216   \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye

```

```

216 \relax\expandafter~%
217 \romannumeral0\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
218 #3\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z
219 ~#4#1~%
220 }%
221 \def\XINT_frac_gen_F #1~#2%
222 {%
223 \xint_UDzerominusfork
224 #2-\XINT_frac_gen_Gdivbyzero
225 0#2{\XINT_frac_gen_G -{}}%
226 0-{\XINT_frac_gen_G {}#2}%
227 \krof #1~%
228 }%
229 \def\XINT_frac_gen_Gdivbyzero #1~~#2~%
230 {%
231 \expandafter\XINT_frac_gen_Gdivbyzero_a
232 \romannumeral0\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
233 #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z~#1~%
234 }%
235 \def\XINT_frac_gen_Gdivbyzero_a #1~#2~%
236 {%
237 \XINT_signalcondition{DivisionByZero}{Division of #1 by zero}{\{#2\}{#1\}{0}}%
238 }%
239 \def\XINT_frac_gen_G #1#2#3~#4~#5~%
240 {%
241 \expandafter\XINT_frac_gen_Ga
242 \romannumeral0\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
243 #1#5\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z~#3~#2#4}%
244 }%
245 \def\XINT_frac_gen_Ga #1#2~#3~%
246 {%
247 \xint_gob_til_zero #1\XINT_frac_gen_zero 0%
248 {\#3\}{#1#2}%
249 }%
250 \def\XINT_frac_gen_zero 0#1#2#3{\{0\}{0\}{1}}%

```

7.9 \XINT_factortens, \XINT_cuz_cnt

Old routines.

```

251 \def\XINT_factortens #1%
252 {%
253 \expandafter\XINT_cuz_cnt_loop\expandafter
254 {\expandafter}\romannumeral0\XINT_rord_main {}#1%
255 \xint:
256 \xint_bye\xint_bye\xint_bye\xint_bye
257 \xint_bye\xint_bye\xint_bye\xint_bye
258 \xint:
259 \R\R\R\R\R\R\R\R\Z
260 }%
261 \def\XINT_cuz_cnt #1%
262 {%
263 \XINT_cuz_cnt_loop {}#1\R\R\R\R\R\R\R\R\Z

```

```

264 }%
265 \def\XINT_cuz_cnt_loop #1#2#3#4#5#6#7#8#9%
266 {%
267   \xint_gob_til_R #9\XINT_cuz_cnt_toofara \R
268   \expandafter\XINT_cuz_cnt_checka\expandafter
269   {\the\numexpr #1+8\relax}{#2#3#4#5#6#7#8#9}%
270 }%
271 \def\XINT_cuz_cnt_toofara\R
272   \expandafter\XINT_cuz_cnt_checka\expandafter #1#2%
273 {%
274   \XINT_cuz_cnt_toofarb {#1}#2%
275 }%
276 \def\XINT_cuz_cnt_toofarb #1#2\Z {\XINT_cuz_cnt_toofarc #2\Z {#1}}%
277 \def\XINT_cuz_cnt_toofarc #1#2#3#4#5#6#7#8%
278 {%
279   \xint_gob_til_R #2\XINT_cuz_cnt_toofard 7%
280     #3\XINT_cuz_cnt_toofard 6%
281     #4\XINT_cuz_cnt_toofard 5%
282     #5\XINT_cuz_cnt_toofard 4%
283     #6\XINT_cuz_cnt_toofard 3%
284     #7\XINT_cuz_cnt_toofard 2%
285     #8\XINT_cuz_cnt_toofard 1%
286     \Z #1#2#3#4#5#6#7#8%
287 }%
288 \def\XINT_cuz_cnt_toofard #1#2\Z #3\R #4\Z #5%
289 {%
290   \expandafter\XINT_cuz_cnt_toofare
291   \the\numexpr #3\relax \R\R\R\R\R\R\R\R\Z
292   {\the\numexpr #5-#1\relax}\R\Z
293 }%
294 \def\XINT_cuz_cnt_toofare #1#2#3#4#5#6#7#8%
295 {%
296   \xint_gob_til_R #2\XINT_cuz_cnt_stopc 1%
297     #3\XINT_cuz_cnt_stopc 2%
298     #4\XINT_cuz_cnt_stopc 3%
299     #5\XINT_cuz_cnt_stopc 4%
300     #6\XINT_cuz_cnt_stopc 5%
301     #7\XINT_cuz_cnt_stopc 6%
302     #8\XINT_cuz_cnt_stopc 7%
303     \Z #1#2#3#4#5#6#7#8%
304 }%
305 \def\XINT_cuz_cnt_checka #1#2%
306 {%
307   \expandafter\XINT_cuz_cnt_checkb\the\numexpr #2\relax \Z {#1}%
308 }%
309 \def\XINT_cuz_cnt_checkb #1%
310 {%
311   \xint_gob_til_zero #1\expandafter\XINT_cuz_cnt_loop\xint_gob_til_Z
312   0\XINT_cuz_cnt_stopa #1%
313 }%
314 \def\XINT_cuz_cnt_stopa #1\Z
315 {%

```

```

316 \XINT_cuz_cnt_stopb #1\R\R\R\R\R\R\R\Z %
317 }%
318 \def\XINT_cuz_cnt_stopb #1#2#3#4#5#6#7#8#9%
319 {%
320 \xint_gob_til_R #2\XINT_cuz_cnt_stopc 1%
321 #3\XINT_cuz_cnt_stopc 2%
322 #4\XINT_cuz_cnt_stopc 3%
323 #5\XINT_cuz_cnt_stopc 4%
324 #6\XINT_cuz_cnt_stopc 5%
325 #7\XINT_cuz_cnt_stopc 6%
326 #8\XINT_cuz_cnt_stopc 7%
327 #9\XINT_cuz_cnt_stopc 8%
328 \Z #1#2#3#4#5#6#7#8#9%
329 }%
330 \def\XINT_cuz_cnt_stopc #1#2\Z #3\R #4\Z #5%
331 {%
332 \expandafter\XINT_cuz_cnt_stopd\expandafter
333 {\the\numexpr #5-#1}\#3%
334 }%
335 \def\XINT_cuz_cnt_stopd #1#2\R #3\Z
336 {%
337 \expandafter\space\expandafter
338 {\romannumeral0\XINT_rord_main }\#2%
339 \xint:
340 \xint_bye\xint_bye\xint_bye\xint_bye
341 \xint_bye\xint_bye\xint_bye\xint_bye
342 \xint:}\#1}%
343 }%

```

7.10 \xintRaw

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

```

344 \def\xintRaw {\romannumeral0\xinraw }%
345 \def\xinraw
346 {%
347 \expandafter\XINT_raw\romannumeral0\XINT_infrac
348 }%
349 \def\XINT_raw #1#2#3{ #2/#3[#1]}%

```

7.11 \xintPRaw

1.09b

```

350 \def\xintPRaw {\romannumeral0\xintpraw }%
351 \def\xintpraw
352 {%
353 \expandafter\XINT_praw\romannumeral0\XINT_infrac
354 }%
355 \def\XINT_praw #1%
356 {%
357 \ifnum #1=\xint_c_ \expandafter\XINT_praw_a\fi \XINT_praw_A {#1}%

```

```

358 }%
359 \def\XINT_praw_A #1#2#3%
360 {%
361     \if\XINT_isOne{#3}1\expandafter\xint_firstoftwo
362         \else\expandafter\xint_secondoftwo
363     \fi { #2[#1]}{ #2/#3[#1]}%
364 }%
365 \def\XINT_praw_a\XINT_praw_A #1#2#3%
366 {%
367     \if\XINT_isOne{#3}1\expandafter\xint_firstoftwo
368         \else\expandafter\xint_secondoftwo
369     \fi { #2}{ #2/#3}%
370 }%

```

7.12 \xintRawWithZeros

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

```

371 \def\xintRawWithZeros {\romannumeral0\xintrawwithzeros }%
372 \def\xintrawwithzeros
373 {%
374     \expandafter\XINT_rawz_fork\romannumeral0\XINT_infrac
375 }%

376 \def\XINT_rawz_fork #1%
377 {%
378     \ifnum#1<\xint_c_
379         \expandafter\XINT_rawz_Ba
380     \else
381         \expandafter\XINT_rawz_A
382     \fi
383     #1.%
384 }%
385 \def\XINT_rawz_A #1.#2#3{\XINT_dsx_addzeros{#1}#2;/#3}%
386 \def\XINT_rawz_Ba -#1.#2#3{\expandafter\XINT_rawz_Bb
387     \expandafter{\romannumeral0\XINT_dsx_addzeros{#1}#3;}{#2}}%
388 \def\XINT_rawz_Bb #1#2{ #2/#1}%

```

7.13 \xintFloor, \xintiFloor

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

```

389 \def\xintFloor {\romannumeral0\xintfloor }%
390 \def\xintfloor #1% devrais-je faire \xintREZ?
391     {\expandafter\XINT_ifloor \romannumeral0\xintrawwithzeros {#1}./1[0]}%
392 \def\xintiFloor {\romannumeral0\xintifloor }%
393 \def\xintifloor #1%
394     {\expandafter\XINT_ifloor \romannumeral0\xintrawwithzeros {#1}.)}%
395 \def\XINT_ifloor #1/#2.{\xinti quo {#1}{#2}}%

```

7.14 `\xintCeil`, `\xintiCeil`

1.09a

```

396 \def\xintCeil {\romannumeral0\xintceil }%
397 \def\xintceil #1{\xintiopp {\xintFloor {\xintOpp{#1}}}}%
398 \def\xintiCeil {\romannumeral0\xinticeil }%
399 \def\xinticeil #1{\xintiopp {\xintiFloor {\xintOpp{#1}}}}%

```

7.15 `\xintNumerator`

```

400 \def\xintNumerator {\romannumeral0\xintnumerator }%
401 \def\xintnumerator
402 {%
403   \expandafter\XINT_numer\romannumeral0\XINT_infrac
404 }%
405 \def\XINT_numer #1%
406 {%
407   \ifcase\XINT_cntSgn #1\xint:
408     \expandafter\XINT_numer_B
409   \or
410     \expandafter\XINT_numer_A
411   \else
412     \expandafter\XINT_numer_B
413   \fi
414   {#1}%
415 }%
416 \def\XINT_numer_A #1#2#3{\XINT_dsx_addzeros{#1}#2;}%
417 \def\XINT_numer_B #1#2#3{ #2}%

```

7.16 `\xintDenominator`

```

418 \def\xintDenominator {\romannumeral0\xintdenominator }%
419 \def\xintdenominator
420 {%
421   \expandafter\XINT_denom_fork\romannumeral0\XINT_infrac
422 }%
423 \def\XINT_denom_fork #1%
424 {%
425   \ifnum#1<\xint_c_
426     \expandafter\XINT_denom_B
427   \else
428     \expandafter\XINT_denom_A
429   \fi
430   #1.%
431 }%
432 \def\XINT_denom_A #1.#2#3{ #3}%
433 \def\XINT_denom_B -#1.#2#3{\XINT_dsx_addzeros{#1}#3;}%

```

7.17 `\xintFrac`

Useless typesetting macro.

```

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

```

7.18 *\xintSignedFrac*

```

460 \def\xintSignedFrac {\romannumeral0\xintsignedfrac }%
461 \def\xintsignedfrac #1%
462 {%
463   \expandafter\XINT_sgnfrac_a\romannumeral0\XINT_infrac {#1}%
464 }%
465 \def\XINT_sgnfrac_a #1#2%
466 {%
467   \XINT_sgnfrac_b #2\Z {#1}%
468 }%
469 \def\XINT_sgnfrac_b #1%
470 {%
471   \xint_UDsignfork
472   #1\XINT_sgnfrac_N
473   -{\XINT_sgnfrac_P #1}%
474   \krof
475 }%
476 \def\XINT_sgnfrac_P #1\Z #2%
477 {%
478   \XINT_fracfrac_A {#2}{#1}%
479 }%
480 \def\XINT_sgnfrac_N
481 {%
482   \expandafter-\romannumeral0\XINT_sgnfrac_P
483 }%

```

7.19 \backslash xintFwOver

```

484 \def\xintFwOver {\romannumeral0\xintfwover }%
485 \def\xintfwover #1%
486 {%
487   \expandafter\XINT_fwover_A\romannumeral0\XINT_infrac {#1}%
488 }%
489 \def\XINT_fwover_A #1{\XINT_fwover_B #1\Z }%
490 \def\XINT_fwover_B #1#2\Z
491 {%
492   \xint_gob_til_zero #1\XINT_fwover_C 0\XINT_fwover_D {10^{#1#2}}%
493 }%
494 \catcode`^=11
495 \def\XINT_fwover_C #1#2#3#4#5%
496 {%
497   \if0\XINT_isOne {#5}\xint_afterfi { {#4\over #5}}%
498   \else\xint_afterfi { #4}%
499   \fi
500 }%
501 \def\XINT_fwover_D #1#2#3%
502 {%
503   \if0\XINT_isOne {#3}\xint_afterfi { {#2\over #3}}%
504   \else\xint_afterfi { #2\cdot }%
505   \fi
506   #1%
507 }%

```

7.20 \backslash xintSignedFwOver

```

508 \def\xintSignedFwOver {\romannumeral0\xintsignedfwover }%
509 \def\xintsignedfwover #1%
510 {%
511   \expandafter\XINT_sgnfwover_a\romannumeral0\XINT_infrac {#1}%
512 }%
513 \def\XINT_sgnfwover_a #1#2%
514 {%
515   \XINT_sgnfwover_b #2\Z {#1}%
516 }%
517 \def\XINT_sgnfwover_b #1%
518 {%
519   \xint_UDsignfork
520   #1\XINT_sgnfwover_N
521   -{\XINT_sgnfwover_P #1}%
522   \krof
523 }%
524 \def\XINT_sgnfwover_P #1\Z #2%
525 {%
526   \XINT_fwover_A {#2}{#1}%
527 }%
528 \def\XINT_sgnfwover_N
529 {%
530   \expandafter-\romannumeral0\XINT_sgnfwover_P
531 }%

```


7.21 `\xintREZ`

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

```

532 \def\xintREZ {\romannumeral0\xintrez }%
533 \def\xintrez
534 {%
535   \expandafter\XINT_rez_A\romannumeral0\XINT_infrac
536 }%
537 \def\XINT_rez_A #1#2%
538 {%
539   \XINT_rez_AB #2\Z {#1}%
540 }%
541 \def\XINT_rez_AB #1%
542 {%
543   \xint_UDzerominusfork
544   #1-\XINT_rez_zero
545   0#1\XINT_rez_neg
546   0-{\XINT_rez_B #1}%
547   \krof
548 }%
549 \def\XINT_rez_zero #1\Z #2#3{ 0/1[0]}%
550 \def\XINT_rez_neg {\expandafter-\romannumeral0\XINT_rez_B }%
551 \def\XINT_rez_B #1\Z
552 {%
553   \expandafter\XINT_rez_C\romannumeral0\XINT_factortens {#1}%
554 }%
555 \def\XINT_rez_C #1#2#3#4%
556 {%
557   \expandafter\XINT_rez_D\romannumeral0\XINT_factortens {#4}{#3}{#2}{#1}%
558 }%
559 \def\XINT_rez_D #1#2#3#4#5%
560 {%
561   \expandafter\XINT_rez_E\expandafter
562   {\the\numexpr #3+#4-#2}{#1}{#5}%
563 }%
564 \def\XINT_rez_E #1#2#3{ #3/#2[#1]}%
```

7.22 `\xintE`

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

```

565 \def\xintE {\romannumeral0\xinte }%
566 \def\xinte #1%
567 {%
568   \expandafter\XINT_e \romannumeral0\XINT_infrac {#1}%
569 }%
570 \def\XINT_e #1#2#3#4%
571 {%
572   \expandafter\XINT_e_end\the\numexpr #1+#4.{#2}{#3}%
573 }%
574 \def\XINT_e_end #1.#2#3{ #2/#3[#1]}%
```

7.23 *\xintIrr*

```

575 \def\xintIrr {\romannumeral0\xintirr }%
576 \def\xintirr #1%
577 {%
578   \expandafter\XINT_irr_start\romannumeral0\xintraewithzeros {#1}\Z
579 }%
580 \def\XINT_irr_start #1#2/#3\Z
581 {%
582   \if0\XINT_isOne {#3}%
583     \xint_afterfi
584       {\xint_UDsignfork
585         #1\XINT_irr_negative
586         -{\XINT_irr_nonneg #1}%
587         \krof}%
588   \else
589     \xint_afterfi{\XINT_irr_denomisine #1}%
590   \fi
591   #2\Z {#3}%
592 }%
593 \def\XINT_irr_denomisine #1\Z #2{ #1/1}% changed in 1.08
594 \def\XINT_irr_negative #1\Z #2{\XINT_irr_D #1\Z #2\Z -}%
595 \def\XINT_irr_nonneg #1\Z #2{\XINT_irr_D #1\Z #2\Z \space}%
596 \def\XINT_irr_D #1#2\Z #3#4\Z
597 {%
598   \xint_UDzerosfork
599     #3#1\XINT_irr_indeterminate
600     #30\XINT_irr_divisionbyzero
601     #10\XINT_irr_zero
602     00\XINT_irr_loop_a
603   \krof
604   {#3#4}{#1#2}{#3#4}{#1#2}%
605 }%
606 \def\XINT_irr_indeterminate #1#2#3#4#5%
607 {%
608   \XINT_signalcondition{DivisionUndefined}{indeterminate: 0/0}{0/1}%
609 }%
610 \def\XINT_irr_divisionbyzero #1#2#3#4#5%
611 {%
612   \XINT_signalcondition{DivisionByZero}{vanishing denominator: #5#2/0}{0/1}%
613 }%
614 \def\XINT_irr_zero #1#2#3#4#5{ 0/1}% changed in 1.08
615 \def\XINT_irr_loop_a #1#2%
616 {%
617   \expandafter\XINT_irr_loop_d
618   \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
619 }%
620 \def\XINT_irr_loop_d #1#2%
621 {%
622   \XINT_irr_loop_e #2\Z
623 }%
624 \def\XINT_irr_loop_e #1#2\Z
625 {%

```

```

626 \xint_gob_til_zero #1\XINT_irr_loop_exit0\XINT_irr_loop_a {#1#2}%
627 }%
628 \def\XINT_irr_loop_exit0\XINT_irr_loop_a #1#2#3#4%
629 {%
630 \expandafter\XINT_irr_loop_exitb\expandafter
631 {\romannumeral0\xintiique {#3}{#2}}%
632 {\romannumeral0\xintiique {#4}{#2}}%
633 }%
634 \def\XINT_irr_loop_exitb #1#2%
635 {%
636 \expandafter\XINT_irr_finish\expandafter {#2}{#1}%
637 }%
638 \def\XINT_irr_finish #1#2#3{#3#1/#2}% changed in 1.08

```

7.24 \xintifInt

```

639 \def\xintifInt {\romannumeral0\xintifint }%
640 \def\xintifint #1{\expandafter\XINT_ifint\romannumeral0\xintraawithzeros {#1}.}%
641 \def\XINT_ifint #1/#2.%
642 {%
643 \if 0\xintiiRem {#1}{#2}%
644 \expandafter\xint_firstoftwo_thenstop
645 \else
646 \expandafter\xint_secondoftwo_thenstop
647 \fi
648 }%

```

7.25 \xintJrr

```

649 \def\xintJrr {\romannumeral0\xintjrr }%
650 \def\xintjrr #1%
651 {%
652 \expandafter\XINT_jrr_start\romannumeral0\xintraawithzeros {#1}\Z
653 }%
654 \def\XINT_jrr_start #1#2/#3\Z
655 {%
656 \if0\XINT_isOne {#3}\xint_afterfi
657 {\xint_UDsignfork
658 #1\XINT_jrr_negative
659 -{\XINT_jrr_nonneg #1}%
660 \krof}%
661 \else
662 \xint_afterfi{\XINT_jrr_denomiseone #1}%
663 \fi
664 #2\Z {#3}%
665 }%
666 \def\XINT_jrr_denomiseone #1\Z #2{ #1/1}% changed in 1.08
667 \def\XINT_jrr_negative #1\Z #2{\XINT_jrr_D #1\Z #2\Z -}%
668 \def\XINT_jrr_nonneg #1\Z #2{\XINT_jrr_D #1\Z #2\Z \space}%
669 \def\XINT_jrr_D #1#2\Z #3#4\Z
670 {%
671 \xint_UDzerosfork
672 #3#1\XINT_jrr_indeterminate
673 #30\XINT_jrr_divisionbyzero
674 #10\XINT_jrr_zero

```

```

675      00\XINT_jrr_loop_a
676      \krof
677      {#3#4}{#1#2}1001%
678 }%
679 \def\XINT_jrr_indeterminate #1#2#3#4#5#6#7%
680 {%
681      \XINT_signalcondition{DivisionUndefined}{indeterminate: 0/0}{0/1}%
682 }%
683 \def\XINT_jrr_divisionbyzero #1#2#3#4#5#6#7%
684 {%
685      \XINT_signalcondition{DivisionByZero}{Vanishing denominator: #7#2/0}{0/1}%
686 }%
687 \def\XINT_jrr_zero #1#2#3#4#5#6#7{ 0/1}% changed in 1.08
688 \def\XINT_jrr_loop_a #1#2%
689 {%
690      \expandafter\XINT_jrr_loop_b
691      \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
692 }%
693 \def\XINT_jrr_loop_b #1#2#3#4#5#6#7%
694 {%
695      \expandafter \XINT_jrr_loop_c \expandafter
696      {\romannumeral0\xintiiadd{\XINT_mul_fork #4\xint:#1\xint:}{#6}}%
697      {\romannumeral0\xintiiadd{\XINT_mul_fork #5\xint:#1\xint:}{#7}}%
698      {#2}{#3}{#4}{#5}%
699 }%
700 \def\XINT_jrr_loop_c #1#2%
701 {%
702      \expandafter \XINT_jrr_loop_d \expandafter{#2}{#1}%
703 }%
704 \def\XINT_jrr_loop_d #1#2#3#4%
705 {%
706      \XINT_jrr_loop_e #3\Z {#4}{#2}{#1}%
707 }%
708 \def\XINT_jrr_loop_e #1#2\Z
709 {%
710      \xint_gob_til_zero #1\XINT_jrr_loop_exit0\XINT_jrr_loop_a {#1#2}%
711 }%
712 \def\XINT_jrr_loop_exit0\XINT_jrr_loop_a #1#2#3#4#5#6%
713 {%
714      \XINT_irr_finish {#3}{#4}%
715 }%

```

7.26 \xintTFrac

1.09i, for `frac` in `\xintexpr`. And `\xintFrac` is already assigned. T for truncation. However, potentially not very efficient with numbers in scientific notations, with big exponents. Will have to think it again some day. I hesitated how to call the macro. Same convention as in maple, but some people reserve fractional part to $x - \text{floor}(x)$. Also, not clear if I had to make it negative (or zero) if $x < 0$, or rather always positive. There should be in fact such a thing for each rounding function, `trunc`, `round`, `floor`, `ceil`.

```

716 \def\xintTFrac {\romannumeral0\xinttfrac }%
717 \def\xinttfrac #1{\expandafter\XINT_tfrac_fork\romannumeral0\xintrawwithzeros {#1}\Z }%
718 \def\XINT_tfrac_fork #1%

```

```

719 {%
720   \xint_UDzerominusfork
721     #1-\XINT_tfrac_zero
722     0#1{\xintiopp\XINT_tfrac_P }%
723     0-{\XINT_tfrac_P #1}%
724   \krof
725 }%
726 \def\XINT_tfrac_zero #1\Z { 0/1[0]}%
727 \def\XINT_tfrac_P #1/#2\Z {\expandafter\XINT_rez_AB
728                               \romannumeral0\xintiirem{#1}{#2}\Z {0}{#2}}%

```

7.27 \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.

```

729 \def\xintTrunc {\romannumeral0\xinttrunc }%
730 \def\xintiTrunc {\romannumeral0\xintitrunc}%
731 \def\xinttrunc #1{\expandafter\XINT_trunc\the\numexpr#1.\XINT_trunc_G}%
732 \def\xintitrunc #1{\expandafter\XINT_trunc\the\numexpr#1.\XINT_itrunc_G}%
733 \def\XINT_trunc #1.#2#3%
734 {%
735   \expandafter\XINT_trunc_a\romannumeral0\XINT_infrac{#3}#1.#2%
736 }%
737 \def\XINT_trunc_a #1#2#3#4.#5%
738 {%
739   \if0\XINT_Sgn#2\xint:\xint_dothis\XINT_trunc_zero\fi
740   \if1\XINT_is_One#3XY\xint_dothis\XINT_trunc_sp_b\fi
741   \xint_orthat\XINT_trunc_b #1+#4.{#2}{#3}#5#4.%
742 }%
743 \def\XINT_trunc_zero #1.#2.{ 0}%

744 \def\XINT_trunc_b {\expandafter\XINT_trunc_B\the\numexpr}%
745 \def\XINT_trunc_sp_b {\expandafter\XINT_trunc_sp_B\the\numexpr}%

746 \def\XINT_trunc_B #1%
747 {%
748   \xint_UDsignfork
749     #1\XINT_trunc_C
750     -\XINT_trunc_D
751   \krof #1%
752 }%

```

7 Package *xintfrac* implementation

```

753 \def\XINT_trunc_sp_B #1%
754 {%
755     \xint_UDsignfork
756     #1\XINT_trunc_sp_C
757     -\XINT_trunc_sp_D
758     \krof #1%
759 }%

760 \def\XINT_trunc_C -#1.#2#3%
761 {%
762     \expandafter\XINT_trunc_CE
763     \romannumeral0\XINT_dsx_addzeros{#1}#3;. {#2}%
764 }%
765 \def\XINT_trunc_CE #1.#2{\XINT_trunc_E #2.{#1}}%

766 \def\XINT_trunc_sp_C -#1.#2#3{\XINT_trunc_sp_Ca #2.#1.%}
767 \def\XINT_trunc_sp_Ca #1%
768 {%
769     \xint_UDsignfork
770     #1{\XINT_trunc_sp_Cb -}%
771     -{\XINT_trunc_sp_Cb \space#1}%
772     \krof
773 }%
774 \def\XINT_trunc_sp_Cb #1#2.#3.%
775 {%
776     \expandafter\XINT_trunc_sp_Cc

777     \romannumeral0\expandafter\XINT_split_fromright_a
778     \the\numexpr#3-\numexpr\XINT_length_loop
779     #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
780     \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
781     \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
782     .#2\xint_bye2345678\xint_bye..#1%
783 }%

784 \def\XINT_trunc_sp_Cc #1%
785 {%
786     \if.#1\xint_dothis{\XINT_trunc_sp_Cd 0.}\fi
787     \xint_orthat {\XINT_trunc_sp_Cd #1}%
788 }%
789 \def\XINT_trunc_sp_Cd #1.#2.#3%
790 {%
791     \XINT_trunc_sp_F #3#1.%
792 }%
793 \def\XINT_trunc_D #1.#2%
794 {%
795     \expandafter\XINT_trunc_E

```

7 Package *xintfrac* implementation

```

796 \romannumeral0\XINT_dsx_addzeros {#1}#2;.%
797 }%
798 \def\XINT_trunc_sp_D #1.#2#3%
799 {%
800 \expandafter\XINT_trunc_sp_E
801 \romannumeral0\XINT_dsx_addzeros {#1}#2;.%
802 }%
803 \def\XINT_trunc_E #1%
804 {%
805 \xint_UDsignfork
806 #1{\XINT_trunc_F -}%
807 -{\XINT_trunc_F \space#1}%
808 \krof
809 }%
810 \def\XINT_trunc_sp_E #1%
811 {%
812 \xint_UDsignfork
813 #1{\XINT_trunc_sp_F -}%
814 -{\XINT_trunc_sp_F\space#1}%
815 \krof
816 }%
817 \def\XINT_trunc_F #1#2.#3#4%
818 {\expandafter#4\romannumeral`&&\expandafter\xint_firstoftwo
819 \romannumeral0\XINT_div_prepare {#3}{#2}.#1}%
820 \def\XINT_trunc_sp_F #1#2.#3{#3#2.#1}%
821 \def\XINT_itrunc_G #1#2.#3#4.{\if#10\xint_dothis{ 0}\fi\xint_orthat{#3#1}#2}%
822 \def\XINT_trunc_G #1.#2#3.%
823 {%
824 \expandafter\XINT_trunc_H
825 \the\numexpr\romannumeral0\xintlength {#1}-#3.#3.{#1}#2%
826 }%
827 \def\XINT_trunc_H #1.#2.%
828 {%
829 \ifnum #1 > \xint_c_
830 \xint_afterfi {\XINT_trunc_Ha {#2}}%
831 \else
832 \xint_afterfi {\XINT_trunc_Hb {-#1}}% -0,--1,--2, ....
833 \fi
834 }%
835 \def\XINT_trunc_Ha{\expandafter\XINT_trunc_Haa\romannumeral0\xintdecsplit}%
836 \def\XINT_trunc_Haa #1#2#3{#3#1.#2}%
837 \def\XINT_trunc_Hb #1#2#3%
838 {%
839 \expandafter #3\expandafter0\expandafter.%

840 \romannumeral\xintreplicate{#1}0#2%
841 }%

```

7.28 `\xintTTrunc`

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

```
842 \def\xintTTrunc {\romannumeral0\xintttrunc }%
843 \def\xintttrunc {\xintitrunc\xint_c_}%
```

7.29 `\xintNum`

```
844 \let\xintnum \xintttrunc
```

7.30 `\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`.

```
845 \def\xintRound {\romannumeral0\xintround }%
846 \def\xintiRound {\romannumeral0\xintiround }%
847 \def\xintround #1{\expandafter\XINT_round\the\numexpr #1.\XINT_round_A}%
848 \def\xintiround #1{\expandafter\XINT_round\the\numexpr #1.\XINT_iround_A}%
849 \def\XINT_round #1.{\expandafter\XINT_round_aa\the\numexpr #1+\xint_c_i.#1.}%
850 \def\XINT_round_aa #1.#2.#3#4%
851 {%
852   \expandafter\XINT_round_a\romannumeral0\XINT_infrac{#4}#1.#3#2.%
853 }%
854 \def\XINT_round_a #1#2#3#4.%
855 {%
856   \if0\XINT_Sgn#2\xint:\xint_dothis\XINT_trunc_zero\fi
857   \if1\XINT_is_One#3XY\xint_dothis\XINT_trunc_sp_b\fi
858   \xint_orthat\XINT_trunc_b #1+#4.{#2}{#3}%
859 }%
860 \def\XINT_round_A{\expandafter\XINT_trunc_G\romannumeral0\XINT_round_B}%
861 \def\XINT_iround_A{\expandafter\XINT_itrunc_G\romannumeral0\XINT_round_B}%
862 \def\XINT_round_B #1.%
863   {\XINT_dsrr #1\xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax.}%
```

7.31 `\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 `\xintiloop` from `xinttools.sty` (replaced by `\xintreplicate...` from `xintkernel.sty`),
- no more use in $O>N>-D$ 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.

```
864 \def\xintXTrunc #1%#2%
865 {%
866   \expandafter\XINT_xtrunc_a
867   \the\numexpr #1\expandafter.\romannumeral0\xintraw
```


7 Package *xintfrac* implementation

```

868 }%
869 \def\XINT_xtrunc_a #1.% ?? faire autre chose
870 {%
871   \expandafter\XINT_xtrunc_b\the\numexpr\ifnum#1<\xint_c_i \xint_c_i-\fi #1.%
872 }%

873 \def\XINT_xtrunc_b #1.#2{\XINT_xtrunc_c #2{#1}}%

874 \def\XINT_xtrunc_c #1%
875 {%
876   \xint_UDzerominusfork
877     #1-\XINT_xtrunc_zero
878     0#1{-\XINT_xtrunc_d {}}%
879     0-{\XINT_xtrunc_d #1}%
880   \krof
881 }%[
882 \def\XINT_xtrunc_zero #1#2]{0.\romannumeral\xintreplicate{#1}0}%

883 \def\XINT_xtrunc_d #1#2#3/#4[#5]%
884 {%
885   \XINT_xtrunc_prepare_a#4\R\R\R\R\R\R\R {10}0000001\W
886   !{#4};{#5}{#2}{#1#3}%
887 }%
888 \def\XINT_xtrunc_prepare_a #1#2#3#4#5#6#7#8#9%
889 {%
890   \xint_gob_til_R #9\XINT_xtrunc_prepare_small\R
891   \XINT_xtrunc_prepare_b #9%
892 }%
893 \def\XINT_xtrunc_prepare_small\R #1!#2;%
894 {%
895   \ifcase #2
896   \or\expandafter\XINT_xtrunc_BisOne
897   \or\expandafter\XINT_xtrunc_BisTwo
898   \or
899   \or\expandafter\XINT_xtrunc_BisFour
900   \or\expandafter\XINT_xtrunc_BisFive
901   \or
902   \or
903   \or\expandafter\XINT_xtrunc_BisEight
904   \fi\XINT_xtrunc_BisSmall {#2}%
905 }%

906 \def\XINT_xtrunc_BisOne\XINT_xtrunc_BisSmall #1#2#3#4%
907   {\XINT_xtrunc_sp_e {#2}{#4}{#3}}%
908 \def\XINT_xtrunc_BisTwo\XINT_xtrunc_BisSmall #1#2#3#4%
909 {%
910   \expandafter\XINT_xtrunc_sp_e\expandafter

```

7 Package *xintfrac* implementation

```

911     {\the\numexpr #2-\xint_c_i\expandafter}\expandafter
912     {\romannumeral0\xintiimul 5{#4}}{#3}%
913 }%
914 \def\xINT_xtrunc_BisFour\xINT_xtrunc_BisSmall #1#2#3#4%
915 {%
916     \expandafter\xINT_xtrunc_sp_e\expandafter
917     {\the\numexpr #2-\xint_c_ii\expandafter}\expandafter
918     {\romannumeral0\xintiimul {25}{#4}}{#3}%
919 }%
920 \def\xINT_xtrunc_BisFive\xINT_xtrunc_BisSmall #1#2#3#4%
921 {%
922     \expandafter\xINT_xtrunc_sp_e\expandafter
923     {\the\numexpr #2-\xint_c_i\expandafter}\expandafter
924     {\romannumeral0\xintdouble {#4}}{#3}%
925 }%
926 \def\xINT_xtrunc_BisEight\xINT_xtrunc_BisSmall #1#2#3#4%
927 {%
928     \expandafter\xINT_xtrunc_sp_e\expandafter
929     {\the\numexpr #2-\xint_c_iii\expandafter}\expandafter
930     {\romannumeral0\xintiimul {125}{#4}}{#3}%
931 }%
932 \def\xINT_xtrunc_BisSmall #1%
933 {%
934     \expandafter\xINT_xtrunc_e\expandafter
935     {\expandafter\xINT_xtrunc_small_a
936     \the\numexpr #1/\xint_c_ii\expandafter
937     .\the\numexpr \xint_c_x^viii+#1!}%
938 }%

939 \def\xINT_xtrunc_small_a #1.#2!#3%
940 {%
941     \expandafter\xINT_div_small_b\the\numexpr #1\expandafter
942     \xint:\the\numexpr #2\expandafter!%
943     \romannumeral0\xINT_div_small_ba #3\R\R\R\R\R\R\R\R{10}0000001\W
944     #3\xINT_sepyviii_Z_end 2345678\relax
945 }%

946 \def\xINT_xtrunc_prepare_b
947     {\expandafter\xINT_xtrunc_prepare_c\romannumeral0\xINT_zeroes_forviii }%
948 \def\xINT_xtrunc_prepare_c #1!%
949 {%
950     \xINT_xtrunc_prepare_d #1.00000000!{#1}%
951 }%
952 \def\xINT_xtrunc_prepare_d #1#2#3#4#5#6#7#8#9%
953 {%
954     \expandafter\xINT_xtrunc_prepare_e
955     \xint_gob_til_dot #1#2#3#4#5#6#7#8#9!%
956 }%
957 \def\xINT_xtrunc_prepare_e #1!#2!#3#4%
958 {%

```

7 Package *xintfrac* implementation

```

959 \XINT_xtrunc_prepare_f #4#3\X {#1}{#3}%
960 }%
961 \def\XINT_xtrunc_prepare_f #1#2#3#4#5#6#7#8#9\X
962 {%
963 \expandafter\XINT_xtrunc_prepare_g\expandafter
964 \XINT_div_prepare_g
965 \the\numexpr #1#2#3#4#5#6#7#8+\xint_c_i\expandafter
966 \xint:\the\numexpr (#1#2#3#4#5#6#7#8+\xint_c_i)/\xint_c_ii\expandafter
967 \xint:\the\numexpr #1#2#3#4#5#6#7#8\expandafter
968 \xint:\romannumeral0\XINT_sepandrev_andcount
969 #1#2#3#4#5#6#7#8#9\XINT_rsepbyviii_end_A 2345678%
970 \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
971 \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
972 \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
973 \X
974 }%

975 \def\XINT_xtrunc_prepare_g #1;{\XINT_xtrunc_e {#1}}%

976 \def\XINT_xtrunc_e #1#2%
977 {%
978 \ifnum #2<\xint_c_
979 \expandafter\XINT_xtrunc_I
980 \else
981 \expandafter\XINT_xtrunc_II
982 \fi #2\xint:{#1}%
983 }%

984 \def\XINT_xtrunc_I -#1\xint:#2#3#4%
985 {%
986 \expandafter\XINT_xtrunc_I_a\romannumeral0#2{#4}{#2}{#1}{#3}%
987 }%

988 \def\XINT_xtrunc_I_a #1#2#3#4#5%
989 {%
990 \expandafter\XINT_xtrunc_I_b\the\numexpr #4-#5\xint:#4\xint:{#5}{#2}{#3}{#1}%
991 }%

992 \def\XINT_xtrunc_I_b #1%
993 {%
994 \xint_UDsignfork
995 #1\XINT_xtrunc_IA_c
996 -\XINT_xtrunc_IB_c
997 \krof #1%
998 }%

```

7 Package *xintfrac* implementation

```

999 \def\XINT_xtrunc_IA_c -#1\xint:#2\xint:#3#4#5#6%
1000 {%
1001   \expandafter\XINT_xtrunc_IA_d
1002   \the\numexpr#2-\xintlength{#6}\xint:{#6}%
1003   \expandafter\XINT_xtrunc_IA_xd
1004   \the\numexpr (#1+\xint_c_ii^v)/\xint_c_ii^vi-\xint_c_i\xint:#1\xint:{#5}{#4}%
1005 }%

1006 \def\XINT_xtrunc_IA_d #1%
1007 {%
1008   \xint_UDsignfork
1009     #1\XINT_xtrunc_IAA_e
1010     -\XINT_xtrunc_IAB_e
1011   \krof #1%
1012 }%

1013 \def\XINT_xtrunc_IAA_e -#1\xint:#2%
1014 {%
1015   \romannumeral0\XINT_split_fromleft
1016   #1.#2\xint_gobble_i\xint_bye2345678\xint_bye..%
1017 }%

1018 \def\XINT_xtrunc_IAB_e #1\xint:#2%
1019 {%
1020   0.\romannumeral\XINT_rep#1\endcsname0#2%
1021 }%

1022 \def\XINT_xtrunc_IA_xd #1\xint:#2\xint:%
1023 {%
1024   \expandafter\XINT_xtrunc_IA_xe\the\numexpr #2-\xint_c_ii^vi*#1\xint:#1\xint:%
1025 }%

1026 \def\XINT_xtrunc_IA_xe #1\xint:#2\xint:#3#4%
1027 {%
1028   \XINT_xtrunc_loop {#2}{#4}{#3}{#1}%
1029 }%

1030 \def\XINT_xtrunc_IB_c #1\xint:#2\xint:#3#4#5#6%
1031 {%
1032   \expandafter\XINT_xtrunc_IB_d
1033   \romannumeral0\XINT_split_xfork #1.#6\xint_bye2345678\xint_bye..{#3}%
1034 }%

```

7 Package `xintfrac` implementation

[illegible]

7 Package *xintfrac* implementation

```

1076 \ifnum #4=\xint_c_ \expandafter\xint_gobble_vi\fi
1077 \expandafter\XINT_xtrunc_finish\expandafter
1078 {\romannumeral0\XINT_dsx_addzeros{#4}#2;}{#3}{#4}%
1079 }%
1080 \def\XINT_xtrunc_finish #1#2%
1081 {%
1082 \expandafter\XINT_xtrunc_finish_a\romannumeral0#2{#1}%
1083 }%
1084 \def\XINT_xtrunc_finish_a #1#2#3%
1085 {%
1086 \romannumeral\xintreplicate{#3-\xintLength{#1}}0#1%
1087 }%

1088 \def\XINT_xtrunc_sp_e #1%
1089 {%
1090 \ifnum #1<\xint_c_
1091 \expandafter\XINT_xtrunc_sp_I
1092 \else
1093 \expandafter\XINT_xtrunc_sp_II
1094 \fi #1\xint:%
1095 }%

1096 \def\XINT_xtrunc_sp_I -#1\xint:#2#3%
1097 {%
1098 \expandafter\XINT_xtrunc_sp_I_a\the\numexpr #1-#3\xint:#1\xint:{#3}{#2}%
1099 }%

1100 \def\XINT_xtrunc_sp_I_a #1%
1101 {%
1102 \xint_UDsignfork
1103 #1\XINT_xtrunc_sp_IA_b
1104 -\XINT_xtrunc_sp_IB_b
1105 \krof #1%
1106 }%

1107 \def\XINT_xtrunc_sp_IA_b -#1\xint:#2\xint:#3#4%
1108 {%
1109 \expandafter\XINT_xtrunc_sp_IA_c
1110 \the\numexpr#2-\xintLength{#4}\xint:{#4}\romannumeral\XINT_rep#1\endcsname0%
1111 }%

1112 \def\XINT_xtrunc_sp_IA_c #1%
1113 {%
1114 \xint_UDsignfork
1115 #1\XINT_xtrunc_sp_IAA

```

7 Package *xintfrac* implementation

```
1116      -\XINT_xtrunc_sp_IAB
1117      \krof #1%
1118 }%

1119 \def\XINT_xtrunc_sp_IAA -#1\xint:#2%
1120 {%
1121      \romannumeral0\XINT_split_fromleft
1122      #1.#2\xint_gobble_i\xint_bye2345678\xint_bye..%
1123 }%

1124 \def\XINT_xtrunc_sp_IAB #1\xint:#2%
1125 {%
1126      0.\romannumeral\XINT_rep#1\endcsname0#2%
1127 }%

1128 \def\XINT_xtrunc_sp_IB_b #1\xint:#2\xint:#3#4%
1129 {%
1130      \expandafter\XINT_xtrunc_sp_IB_c
1131      \romannumeral0\XINT_split_xfork #1.#4\xint_bye2345678\xint_bye..{#3}%
1132 }%

1133 \def\XINT_xtrunc_sp_IB_c #1.#2.#3%
1134 {%
1135      \expandafter\XINT_xtrunc_sp_IA_c\the\numexpr#3-\xintLength {#1}\xint:{#1}%
1136 }%

1137 \def\XINT_xtrunc_sp_II #1\xint:#2#3%
1138 {%
1139      #2\romannumeral\XINT_rep#1\endcsname0.\romannumeral\XINT_rep#3\endcsname0%
1140 }%
```

7.32 *\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.

```
1141 \mathchardef\XINTdigits 16
1142 \def\xintDigits #1#2%
1143      {\afterassignment \xint_gobble_i \mathchardef\XINTdigits=}%
1144 \def\xinttheDigits {\number\XINTdigits }%
```

7.33 *\xintAdd*

```

1145 \def\xintAdd {\romannumeral0\xintadd }%
1146 \def\xintadd #1{\expandafter\XINT_fadd\romannumeral0\xintraw {#1}}%
1147 \def\XINT_fadd #1{\xint_gob_til_zero #1\XINT_fadd_Azero 0\XINT_fadd_a #1}%
1148 \def\XINT_fadd_Azero #1{\xintraw }%
1149 \def\XINT_fadd_a #1/#2[#3]#4%
1150   {\expandafter\XINT_fadd_b\romannumeral0\xintraw {#4}{#3}{#1}{#2}}%
1151 \def\XINT_fadd_b #1{\xint_gob_til_zero #1\XINT_fadd_Bzero 0\XINT_fadd_c #1}%
1152 \def\XINT_fadd_Bzero #1#2#3#4{ #3/#4[#2]}%
1153 \def\XINT_fadd_c #1/#2[#3]#4%
1154 {%
1155   \expandafter\XINT_fadd_Aa\the\numexpr #4-#3.{#3}{#4}{#1}{#2}%
1156 }%
1157 \def\XINT_fadd_Aa #1%
1158 {%
1159   \xint_UDzerominusfork
1160     #1-\XINT_fadd_B
1161     0#1\XINT_fadd_Bb
1162     0-\XINT_fadd_Ba
1163   \krof #1%
1164 }%
1165 \def\XINT_fadd_B   #1.#2#3#4#5#6#7{\XINT_fadd_C {#4}{#5}{#7}{#6}[#3]}%
1166 \def\XINT_fadd_Ba  #1.#2#3#4#5#6#7%
1167 {%
1168   \expandafter\XINT_fadd_C\expandafter
1169     {\romannumeral0\XINT_dsx_addzeros {#1}#6;}%
1170   {#7}{#5}{#4}[#2]}%
1171 }%
1172 \def\XINT_fadd_Bb  -#1.#2#3#4#5#6#7%
1173 {%
1174   \expandafter\XINT_fadd_C\expandafter
1175     {\romannumeral0\XINT_dsx_addzeros {#1}#4;}%
1176   {#5}{#7}{#6}[#3]}%
1177 }%
1178 \def\XINT_fadd_C #1#2#3%
1179 {%
1180   \ifcase\romannumeral0\xintiicomp {#2}{#3} %<- intentional space here.
1181     \expandafter\XINT_fadd_eq
1182   \or\expandafter\XINT_fadd_D
1183   \else\expandafter\XINT_fadd_Da
1184   \fi {#2}{#3}{#1}%
1185 }%
1186 \def\XINT_fadd_eq #1#2#3#4#5%
1187 {%
1188   \expandafter\XINT_fadd_G
1189   \romannumeral0\xintiiadd {#3}{#4}/#1[#5]%
1190 }%
1191 \def\XINT_fadd_D #1#2%
1192 {%
1193   \expandafter\XINT_fadd_E\romannumeral0\XINT_div_prepare {#2}{#1}{#1}{#2}%
1194 }%
1195 \def\XINT_fadd_E #1#2%

```



```

1196 {%
1197   \if0\XINT_Sgn #2\xint:
1198     \expandafter\XINT_fadd_F
1199   \else\expandafter\XINT_fadd_K
1200   \fi {#1}%
1201 }%
1202 \def\XINT_fadd_F #1#2#3#4#5#6%
1203 {%
1204   \expandafter\XINT_fadd_G
1205   \romannumeral0\xintiiadd {\xintiiMul {#5}{#1}}{#4}/#2%[#6]%
1206 }%
1207 \def\XINT_fadd_Da #1#2%
1208 {%
1209   \expandafter\XINT_fadd_Ea\romannumeral0\XINT_div_prepare {#1}{#2}{#1}{#2}%
1210 }%
1211 \def\XINT_fadd_Ea #1#2%
1212 {%
1213   \if0\XINT_Sgn #2\xint:
1214     \expandafter\XINT_fadd_Fa
1215   \else\expandafter\XINT_fadd_K
1216   \fi {#1}%
1217 }%
1218 \def\XINT_fadd_Fa #1#2#3#4#5#6%
1219 {%
1220   \expandafter\XINT_fadd_G
1221   \romannumeral0\xintiiadd {\xintiiMul {#4}{#1}}{#5}/#3%[#6]%
1222 }%
1223 \def\XINT_fadd_G #1{%
1224 \def\XINT_fadd_G #1{\if0##1\expandafter\XINT_fadd_iszero\fi#1##1}%
1225 }\XINT_fadd_G{ }%
1226 \def\XINT_fadd_K #1#2#3#4#5%
1227 {%
1228   \expandafter\XINT_fadd_L
1229   \romannumeral0\xintiiadd {\xintiiMul {#2}{#5}}{\xintiiMul {#3}{#4}}.%
1230   {{#2}{#3}}}%
1231 }%
1232 \def\XINT_fadd_L #1{\if0#1\expandafter\XINT_fadd_iszero\fi\XINT_fadd_M #1}%
1233 \def\XINT_fadd_M #1.#2{\expandafter\XINT_fadd_N \expandafter
1234   {\romannumeral0\xintiimul #2}{#1}}%
1235 \def\XINT_fadd_N #1#2{ #2/#1}%
1236 \def\XINT_fadd_iszero #1[#2]{ 0/1[0]}% ou [#2] originel?

```

7.34 \xintSub

```

1237 \def\xintSub {\romannumeral0\xintsub }%
1238 \def\xintsub #1{\expandafter\XINT_fsub\romannumeral0\xintra #1}%
1239 \def\XINT_fsub #1{\xint_gob_til_zero #1\XINT_fsub_Azero 0\XINT_fsub_a #1}%
1240 \def\XINT_fsub_Azero #1]{\xintopp }%
1241 \def\XINT_fsub_a #1/#2[#3]#4%
1242   {\expandafter\XINT_fsub_b\romannumeral0\xintra #4}{#3}{#1}{#2}%
1243 \def\XINT_fsub_b #1{\xint_UDzerominusfork
1244   #1-\XINT_fadd_Bzero
1245   0#1\XINT_fadd_c
1246   0-{\XINT_fadd_c -#1}%

```

1247 \krof }%

7.35 \xintSum

There was (not documented anymore since 1.09d, 2013/10/22) a macro `\xintSumExpr`, but it has been deleted at 1.21.

Empty items are not accepted by this macro.

```

1248 \def\xintSum {\romannumeral0\xintsum }%
1249 \def\xintsum #1{\expandafter\XINT_fsumexpr\romannumeral`&&@#1\xint:}%
1250 \def\XINT_fsumexpr {\XINT_fsum_loop_a {0/1[0]}}%
1251 \def\XINT_fsum_loop_a #1#2%
1252 {%
1253   \expandafter\XINT_fsum_loop_b \romannumeral`&&@#2\xint:{#1}%
1254 }%
1255 \def\XINT_fsum_loop_b #1%
1256 {%
1257   \xint_gob_til_xint: #1\XINT_fsum_finished\xint:\XINT_fsum_loop_c #1%
1258 }%
1259 \def\XINT_fsum_loop_c #1\xint:#2%
1260 {%
1261   \expandafter\XINT_fsum_loop_a\expandafter{\romannumeral0\xintadd {#2}{#1}}%
1262 }%
1263 \def\XINT_fsum_finished #1\xint:\xint:#2{ #2}%

```

7.36 \xintMul

```

1264 \def\xintMul {\romannumeral0\xintmul }%
1265 \def\xintmul #1{\expandafter\XINT_fmula\romannumeral0\xintraw {#1}.}%
1266 \def\XINT_fmula #1{\xint_gob_til_zero #1\XINT_fmula_zero 0\XINT_fmula_a #1}%
1267 \def\XINT_fmula_a #1[#2].#3%
1268   {\expandafter\XINT_fmula_b\romannumeral0\xintraw {#3}#1[#2.]}%
1269 \def\XINT_fmula_b #1{\xint_gob_til_zero #1\XINT_fmula_zero 0\XINT_fmula_c #1}%
1270 \def\XINT_fmula_c #1/#2[#3]#4/#5[#6.]%
1271 {%
1272   \expandafter\XINT_fmula_d
1273   \expandafter{\the\numexpr #3+#6\expandafter}%
1274   \expandafter{\romannumeral0\xintiimul {#5}{#2}}%
1275   {\romannumeral0\xintiimul {#4}{#1}}%
1276 }%
1277 \def\XINT_fmula_d #1#2#3%
1278 {%
1279   \expandafter \XINT_fmula_e \expandafter{#3}{#1}{#2}%
1280 }%
1281 \def\XINT_fmula_e #1#2{\XINT_outfrac {#2}{#1}}%
1282 \def\XINT_fmula_zero #1.#2{ 0/1[0]}%

```

7.37 \xintSqr

1.1 modifs comme `xintMul`.

```

1283 \def\xintSqr {\romannumeral0\xintsqr }%
1284 \def\xintsqr #1{\expandafter\XINT_fsqr\romannumeral0\xintraw {#1}}%
1285 \def\XINT_fsqr #1{\xint_gob_til_zero #1\XINT_fsqr_zero 0\XINT_fsqr_a #1}%

```

```

1286 \def\XINT_fsqr_a #1/#2[#3]%
1287 {%
1288   \expandafter\XINT_fsqr_b
1289   \expandafter{\the\numexpr #3+#3\expandafter}%
1290   \expandafter{\romannumeral0\xintiisqr {#2}}%
1291   {\romannumeral0\xintiisqr {#1}}%
1292 }%
1293 \def\XINT_fsqr_b #1#2#3{\expandafter \XINT_fmulo \expandafter{#3}{#1}{#2}}%
1294 \def\XINT_fsqr_zero #1{\ 0/1[0]}%

```

7.38 \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.

```

1295 \def\xintipow #1#2%
1296 {%
1297   \expandafter\xint_pow\the\numexpr \xintNum{#2}\expandafter
1298   .\romannumeral0\xintnum{#1}\xint:
1299 }%
1300 \def\xintPow {\romannumeral0\xintpow }%
1301 \def\xintpow #1%
1302 {%
1303   \expandafter\XINT_fpow\expandafter {\romannumeral0\XINT_infrac {#1}}%
1304 }%
1305 \def\XINT_fpow #1#2%
1306 {%
1307   \expandafter\XINT_fpow_fork\the\numexpr \xintNum{#2}\relax\Z #1%
1308 }%
1309 \def\XINT_fpow_fork #1#2\Z
1310 {%
1311   \xint_UDzerominusfork
1312   #1-\XINT_fpow_zero
1313   0#1\XINT_fpow_neg
1314   0-{\XINT_fpow_pos #1}%
1315   \krof
1316   {#2}%
1317 }%
1318 \def\XINT_fpow_zero #1#2#3#4{ 1/1[0]}%
1319 \def\XINT_fpow_pos #1#2#3#4#5%
1320 {%
1321   \expandafter\XINT_fpow_pos_A\expandafter
1322   {\the\numexpr #1#2*#3\expandafter}\expandafter
1323   {\romannumeral0\xintiipow {#5}{#1#2}}%
1324   {\romannumeral0\xintiipow {#4}{#1#2}}%
1325 }%
1326 \def\XINT_fpow_neg #1#2#3#4%
1327 {%
1328   \expandafter\XINT_fpow_pos_A\expandafter
1329   {\the\numexpr -#1*#2\expandafter}\expandafter
1330   {\romannumeral0\xintiipow {#3}{#1}}%

```

```

1331    {\romannumeral0\xintiipow {#4}{#1}}%
1332 }%
1333 \def\XINT_fpow_pos_A #1#2#3%
1334 {%
1335    \expandafter\XINT_fpow_pos_B\expandafter {#3}{#1}{#2}%
1336 }%
1337 \def\XINT_fpow_pos_B #1#2{\XINT_outfrac {#2}{#1}}%

```

7.39 `\xintFac`

Factorial coefficients: variant which can be chained with other *xintfrac* macros. `\xintiFac` deprecated at 1.2o; `\xintFac` used by *xintexpr.sty*.

```

1338 \def\xintifac #1{\expandafter\XINT_fac_fork\the\numexpr\xintNum{#1}.}%
1339 \def\xintFac {\romannumeral0\xintfac}%
1340 \def\xintfac #1{\expandafter\XINT_fac_fork\the\numexpr\xintNum{#1}.[0]}%

```

7.40 `\xintBinomial`

1.2f. Binomial coefficients. `\xintiBinomial` deprecated at 1.2o; `\xintBinomial` needed by *xintexpr.sty*.

```

1341 \def\xintibinomial #1#2%
1342 {%
1343    \expandafter\XINT_binom_pre
1344    \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.%
1345 }%
1346 \def\xintBinomial {\romannumeral0\xintbinomial}%
1347 \def\xintbinomial #1#2%
1348 {%
1349    \expandafter\XINT_binom_pre
1350    \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.[0]%
1351 }%

```

7.41 `\xintPFactorial`

1.2f. Partial factorial. For needs of *xintexpr.sty*.

```

1352 \def\xintipfactorial #1#2%
1353 {%
1354    \expandafter\XINT_pfac_fork
1355    \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.%
1356 }%
1357 \def\xintPFactorial {\romannumeral0\xintpfactorial}%
1358 \def\xintpfactorial #1#2%
1359 {%
1360    \expandafter\XINT_pfac_fork
1361    \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.[0]%
1362 }%

```

7.42 `\xintPrd`

There was (not documented anymore since 1.09d, 2013/10/22) a macro `\xintPrdExpr`, but it has been deleted at 1.2l

```

1363 \def\xintPrd {\romannumeral0\xintprd }%
1364 \def\xintprd #1{\expandafter\XINT_fprdexpr \romannumeral`&&@#1\xint:}%
1365 \def\XINT_fprdexpr {\XINT_fprod_loop_a {1/1[0]}}%
1366 \def\XINT_fprod_loop_a #1#2%
1367 {%
1368   \expandafter\XINT_fprod_loop_b \romannumeral`&&@#2\xint:{#1}%
1369 }%
1370 \def\XINT_fprod_loop_b #1%
1371 {%
1372   \xint_gob_til_xint: #1\XINT_fprod_finished\xint:\XINT_fprod_loop_c #1%
1373 }%
1374 \def\XINT_fprod_loop_c #1\xint:#2%
1375 {%
1376   \expandafter\XINT_fprod_loop_a\expandafter{\romannumeral0\xintmul {#1}{#2}}%
1377 }%
1378 \def\XINT_fprod_finished#1\xint:\xint:#2{ #2}%

```

7.43 `\xintDiv`

```

1379 \def\xintDiv {\romannumeral0\xintdiv }%
1380 \def\xintdiv #1%
1381 {%
1382   \expandafter\XINT_fdiv\expandafter {\romannumeral0\XINT_infrac {#1}}%
1383 }%
1384 \def\XINT_fdiv #1#2%
1385   {\expandafter\XINT_fdiv_A\romannumeral0\XINT_infrac {#2}#1}%
1386 \def\XINT_fdiv_A #1#2#3#4#5#6%
1387 {%
1388   \expandafter\XINT_fdiv_B
1389   \expandafter{\the\numexpr #4-#1\expandafter}%
1390   \expandafter{\romannumeral0\xintiimul {#2}{#6}}%
1391   {\romannumeral0\xintiimul {#3}{#5}}%
1392 }%
1393 \def\XINT_fdiv_B #1#2#3%
1394 {%
1395   \expandafter\XINT_fdiv_C
1396   \expandafter{#3}{#1}{#2}%
1397 }%
1398 \def\XINT_fdiv_C #1#2{\XINT_outfrac {#2}{#1}}%

```

7.44 `\xintDivFloor`

1.1. Changed at 1.2p to not append `/1[0]` ending but rather output a big integer in strict format, like `\xintDivTrunc` and `\xintDivRound`.

```

1399 \def\xintDivFloor {\romannumeral0\xintdivfloor }%
1400 \def\xintdivfloor #1#2{\xintifloor{\xintDiv {#1}{#2}}}

```

7.45 `\xintDivTrunc`

1.1. `\xintttrunc` rather than `\xintitrunc0` in 1.1a

```
1401 \def\xintDivTrunc      {\romannumeral0\xintdivtrunc }%
1402 \def\xintdivtrunc #1#2{\xintttrunc {\xintDiv {#1}{#2}}}%
```

7.46 `\xintDivRound`

1.1

```
1403 \def\xintDivRound      {\romannumeral0\xintdivround }%
1404 \def\xintdivround #1#2{\xintiround 0{\xintDiv {#1}{#2}}}%
```

7.47 `\xintModTrunc`

1.1. `\xintModTrunc {q1}{q2}` computes $q1 - q2 * t(q1/q2)$ with $t(q1/q2)$ equal to the truncated division of two fractions $q1$ and $q2$.

Its former name, prior to 1.2p, was `\xintMod`.

```
1405 \def\xintModTrunc {\romannumeral0\xintmodtrunc }%
1406 \def\xintmodtrunc #1{\expandafter\XINT_modtrunc_a\romannumeral0\xintra{#1}.}%
1407 \def\XINT_modtrunc_a #1#2.#3%
1408   {\expandafter\XINT_modtrunc_b\expandafter #1\romannumeral0\xintra{#3}#2.}%
1409 \def\XINT_modtrunc_b #1#2% #1 de A, #2 de B.
1410 {%
1411   \if0#2\xint_dothis{\XINT_modtrunc_divbyzero #1#2}\fi
1412   \if0#1\xint_dothis\XINT_modtrunc_aiszero\fi
1413   \if-#2\xint_dothis{\XINT_modtrunc_bneg #1}\fi
1414   \xint_orthat{\XINT_modtrunc_bpos #1#2}%
1415 }%
1416 \def\XINT_modtrunc_divbyzero #1#2[#3]#4.%
1417 {%
1418   \XINT_signalcondition{DivisionByZero}{Division by #2[#3] of #1#4}{\{0/1[0]}}%
1419 }%
1420 \def\XINT_modtrunc_aiszero #1.{ 0/1[0]}%
1421 \def\XINT_modtrunc_bneg #1%
1422 {%
1423   \xint_UDsignfork
1424     #1{\xintiiopp\XINT_modtrunc_pos {}}%
1425     -{\XINT_modtrunc_pos #1}%
1426   \krof
1427 }%
1428 \def\XINT_modtrunc_bpos #1%
1429 {%
1430   \xint_UDsignfork
1431     #1{\xintiiopp\XINT_modtrunc_pos {}}%
1432     -{\XINT_modtrunc_pos #1}%
1433   \krof
1434 }%
```

Attention. This crucially uses that `xint`'s `\xintiiE{x}{e}` is defined to return x unchanged if e is negative (and x extended by e zeroes if $e \geq 0$).

```

1435 \def\XINT_modtrunc_pos #1#2/#3[#4]#5/#6[#7].%
1436 {%
1437   \expandafter\XINT_modtrunc_pos_a
1438   \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.\expandafter
1439   {\romannumeral0\xintiimul {#6}{#3}}%
1440   {\xintiiE{\xintiiMul {#1#5}{#3}}{#7-#4}}%
1441   {\xintiiE{\xintiiMul {#2}{#6}}{#4-#7}}%
1442 }%
1443 \def\XINT_modtrunc_pos_a #1.#2#3#4{\xintiirem {#3}{#4}/#2[#1]}%

```

7.48 \xintDivMod

1.2p. `\xintDivMod{q1}{q2}` outputs `{floor(q1/q2)}{q1 - q2*floor(q1/q2)}`. Attention that it relies on `\xintiiE{x}{e}` returning `x` if `e < 0`.

```

1444 \def\xintDivMod {\romannumeral0\xintdivmod }%
1445 \def\xintdivmod #1{\expandafter\XINT_divmod_a\romannumeral0\xintra{#1}.}%
1446 \def\XINT_divmod_a #1#2.#3%
1447   {\expandafter\XINT_divmod_b\expandafter #1\romannumeral0\xintra{#3}#2.}%
1448 \def\XINT_divmod_b #1#2% #1 de A, #2 de B.
1449 {%
1450   \if0#2\xint_dothis{\XINT_divmod_divbyzero #1#2}\fi
1451   \if0#1\xint_dothis\XINT_divmod_aiszero\fi
1452   \if-#2\xint_dothis{\XINT_divmod_bneg #1}\fi
1453   \xint_orthat{\XINT_divmod_bpos #1#2}%
1454 }%
1455 \def\XINT_divmod_divbyzero #1#2[#3]#4.%
1456 {%
1457   \XINT_signalcondition{DivisionByZero}{Division by #2[#3] of #1#4}{}%
1458   {{0}{0/1[0]}}% à revoir...
1459 }%
1460 \def\XINT_divmod_aiszero #1.{{0}{0/1[0]}}%
1461 \def\XINT_divmod_bneg #1% f // -g = (-f) // g, f % -g = -((-f) % g)
1462 {%
1463   \expandafter\XINT_divmod_bneg_finish
1464   \romannumeral0\xint_UDsignfork
1465   #1{\XINT_divmod_bpos {}}%
1466   -{\XINT_divmod_bpos {-#1}}%
1467   \krof
1468 }%
1469 \def\XINT_divmod_bneg_finish#1#2%
1470 {%
1471   \expandafter\xint_exchangetwo_keepbraces\expandafter
1472   {\romannumeral0\xintiiopp#2}{#1}%
1473 }%
1474 \def\XINT_divmod_bpos #1#2/#3[#4]#5/#6[#7].%
1475 {%
1476   \expandafter\XINT_divmod_bpos_a
1477   \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.\expandafter
1478   {\romannumeral0\xintiimul {#6}{#3}}%
1479   {\xintiiE{\xintiiMul {#1#5}{#3}}{#7-#4}}%
1480   {\xintiiE{\xintiiMul {#2}{#6}}{#4-#7}}%
1481 }%

```

```

1482 \def\XINT_divmod_bpos_a #1.#2#3#4%
1483 {%
1484   \expandafter\XINT_divmod_bpos_finish
1485   \romannumeral0\xintidivision{#3}{#4}{/#2[#1]}%
1486 }%
1487 \def\XINT_divmod_bpos_finish #1#2#3{{#1}{#2#3}}%

```

7.49 \xintMod

1.2p. `\xintMod{q1}{q2}` computes $q1 - q2 \cdot \text{floor}(q1/q2)$. Attention that it relies on `\xintiiE{x}{e}` returning x if $e < 0$.

Prior to 1.2p, that macro had the meaning now attributed to `\xintModTrunc`.

```

1488 \def\xintMod {\romannumeral0\xintmod }%
1489 \def\xintmod #1{\expandafter\XINT_mod_a\romannumeral0\xintra{#1}.}%
1490 \def\XINT_mod_a #1#2.#3%
1491   {\expandafter\XINT_mod_b\expandafter #1\romannumeral0\xintra{#3}#2.}%
1492 \def\XINT_mod_b #1#2% #1 de A, #2 de B.
1493 {%
1494   \if0#2\xint_dothis{\XINT_mod_divbyzero #1#2}\fi
1495   \if0#1\xint_dothis\XINT_mod_aiszero\fi
1496   \if-#2\xint_dothis{\XINT_mod_bneg #1}\fi
1497   \xint_orthat{\XINT_mod_bpos #1#2}%
1498 }%

```

Attention to not move `ModTrunc` code beyond that point.

```

1499 \let\XINT_mod_divbyzero\XINT_modtrunc_divbyzero
1500 \let\XINT_mod_aiszero \XINT_modtrunc_aiszero
1501 \def\XINT_mod_bneg #1% f % -g = - ((-f) % g), for g > 0
1502 {%
1503   \xintiiopp\xint_UDsignfork
1504   #1{\XINT_mod_bpos {}}%
1505   -{\XINT_mod_bpos {-#1}}%
1506   \krof
1507 }%
1508 \def\XINT_mod_bpos #1#2/#3[#4]#5/#6[#7].%
1509 {%
1510   \expandafter\XINT_mod_bpos_a
1511   \the\numexprifnum#7>#4 #4\else #7\fi\expandafter.\expandafter
1512   {\romannumeral0\xintiimul {#6}{#3}}%
1513   {\xintiiE{\xintiiMul {#1#5}{#3}}{#7-#4}}%
1514   {\xintiiE{\xintiiMul {#2}{#6}}{#4-#7}}%
1515 }%
1516 \def\XINT_mod_bpos_a #1.#2#3#4{\xintiiirem {#3}{#4}/#2[#1]}%

```

7.50 \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.

```

1517 \def\xintIsOne {\romannumeral0\xintisone }%
1518 \def\xintisone #1{\expandafter\XINT_fracisone

```



```

1519          \romannumeral0\xintraewithzeros{#1}\Z }%
1520 \def\XINT_fracisone #1/#2\Z
1521   {\if0\xintiiCmp {#1}{#2}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%

```

7.51 \xintGeq

```

1522 \def\xintGeq {\romannumeral0\xintgeq }%
1523 \def\xintgeq #1%
1524 {%
1525   \expandafter\XINT_fgeq\expandafter {\romannumeral0\xintabs {#1}}%
1526 }%
1527 \def\XINT_fgeq #1#2%
1528 {%
1529   \expandafter\XINT_fgeq_A \romannumeral0\xintabs {#2}#1%
1530 }%
1531 \def\XINT_fgeq_A #1%
1532 {%
1533   \xint_gob_til_zero #1\XINT_fgeq_Zii 0%
1534   \XINT_fgeq_B #1%
1535 }%
1536 \def\XINT_fgeq_Zii 0\XINT_fgeq_B #1[#2]#3[#4]{ 1}%
1537 \def\XINT_fgeq_B #1/#2[#3]#4#5/#6[#7]%
1538 {%
1539   \xint_gob_til_zero #4\XINT_fgeq_Zi 0%
1540   \expandafter\XINT_fgeq_C\expandafter
1541   {\the\numexpr #7-#3\expandafter}\expandafter
1542   {\romannumeral0\xintiimul {#4#5}{#2}}%
1543   {\romannumeral0\xintiimul {#6}{#1}}%
1544 }%
1545 \def\XINT_fgeq_Zi 0#1#2#3#4#5#6#7{ 0}%
1546 \def\XINT_fgeq_C #1#2#3%
1547 {%
1548   \expandafter\XINT_fgeq_D\expandafter
1549   {#3}{#1}{#2}%
1550 }%
1551 \def\XINT_fgeq_D #1#2#3%
1552 {%
1553   \expandafter\XINT_cntSgnFork\romannumeral`&&\expandafter\XINT_cntSgn
1554   \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\xint:
1555   { 0}{\XINT_fgeq_E #2\Z {#3}{#1}}{ 1}%
1556 }%
1557 \def\XINT_fgeq_E #1%
1558 {%
1559   \xint_UDsignfork
1560   #1\XINT_fgeq_Fd
1561   -{\XINT_fgeq_Fn #1}%
1562   \krof
1563 }%

1564 \def\XINT_fgeq_Fd #1\Z #2#3%
1565 {%
1566   \expandafter\XINT_fgeq_Fe

```

```

1567 \romannumeral0\XINT_dsx_addzeros {#1}#3;\xint:#2\xint:
1568 }%
1569 \def\XINT_fgeq_Fe #1\xint:#2#3\xint:{\XINT_geq_plusplus #2#1\xint:#3\xint:}%
1570 \def\XINT_fgeq_Fn #1\Z #2#3%
1571 {%
1572 \expandafter\XINT_fgeq_Fo
1573 \romannumeral0\XINT_dsx_addzeros {#1}#2;\xint:#3\xint:
1574 }%
1575 \def\XINT_fgeq_Fo #1#2\xint:#3\xint:{\XINT_geq_plusplus #1#3\xint:#2\xint:}%

```

7.52 \xintMax

```

1576 \def\xintMax {\romannumeral0\xintmax }%
1577 \def\xintmax #1%
1578 {%
1579 \expandafter\XINT_fmax\expandafter {\romannumeral0\xintra #1}}%
1580 }%
1581 \def\XINT_fmax #1#2%
1582 {%
1583 \expandafter\XINT_fmax_A\romannumeral0\xintra #2}%1%
1584 }%
1585 \def\XINT_fmax_A #1#2/#3[#4]#5#6/#7[#8]%
1586 {%
1587 \xint_UDsignsfork
1588 #1#5\XINT_fmax_minusminus
1589 -#5\XINT_fmax_firstneg
1590 #1-\XINT_fmax_secondneg
1591 --\XINT_fmax_nonneg_a
1592 \krof
1593 #1#5{#2/#3[#4]}{#6/#7[#8]}%
1594 }%
1595 \def\XINT_fmax_minusminus --%
1596 {\expandafter-\romannumeral0\XINT_fmin_nonneg_b }%
1597 \def\XINT_fmax_firstneg #1-#2#3{ #1#2}%
1598 \def\XINT_fmax_secondneg -#1#2#3{ #1#3}%
1599 \def\XINT_fmax_nonneg_a #1#2#3#4%
1600 {%
1601 \XINT_fmax_nonneg_b {#1#3}{#2#4}%
1602 }%
1603 \def\XINT_fmax_nonneg_b #1#2%
1604 {%
1605 \if0\romannumeral0\XINT_fgeq_A #1#2%
1606 \xint_afterfi{ #1}%
1607 \else \xint_afterfi{ #2}%
1608 \fi
1609 }%

```

7.53 \xintMaxof

1.21 protects \xintMaxof against items with non terminated \the\numexpr expressions.
The macro is not compatible with an empty list.

```

1610 \def\xintMaxof {\romannumeral0\xintmaxof }%
1611 \def\xintmaxof #1{\expandafter\XINT_maxof_a\romannumeral`&&@#1\xint:}%

```

```

1612 \def\XINT_maxof_a #1{\expandafter\XINT_maxof_b\romannumeral0\xintra{#1}!}%
1613 \def\XINT_maxof_b #1!#2%
1614     {\expandafter\XINT_maxof_c\romannumeral`&&@#2!{#1}!}%
1615 \def\XINT_maxof_c #1%
1616     {\xint_gob_til_xint: #1\XINT_maxof_e\xint:\XINT_maxof_d #1}%
1617 \def\XINT_maxof_d #1!%
1618     {\expandafter\XINT_maxof_b\romannumeral0\xintmax {#1}}%
1619 \def\XINT_maxof_e #1!#2!{ #2}%

```

7.54 \xintMin

```

1620 \def\xintMin {\romannumeral0\xintmin }%
1621 \def\xintmin #1%
1622 {%
1623     \expandafter\XINT_fmin\expandafter {\romannumeral0\xintra {#1}}%
1624 }%
1625 \def\XINT_fmin #1#2%
1626 {%
1627     \expandafter\XINT_fmin_A\romannumeral0\xintra {#2}#1%
1628 }%
1629 \def\XINT_fmin_A #1#2/#3[#4]#5#6/#7[#8]%
1630 {%
1631     \xint_UDsignsfork
1632     #1#5\XINT_fmin_minusminus
1633     -#5\XINT_fmin_firstneg
1634     #1-\XINT_fmin_secondneg
1635     --\XINT_fmin_nonneg_a
1636     \krof
1637     #1#5{#2/#3[#4]}{#6/#7[#8]}%
1638 }%
1639 \def\XINT_fmin_minusminus --%
1640     {\expandafter-\romannumeral0\XINT_fmax_nonneg_b }%
1641 \def\XINT_fmin_firstneg #1-#2#3{ -#3}%
1642 \def\XINT_fmin_secondneg -#1#2#3{ -#2}%
1643 \def\XINT_fmin_nonneg_a #1#2#3#4%
1644 {%
1645     \XINT_fmin_nonneg_b {#1#3}{#2#4}%
1646 }%
1647 \def\XINT_fmin_nonneg_b #1#2%
1648 {%
1649     \if0\romannumeral0\XINT_fgeq_A #1#2%
1650         \xint_afterfi{ #2}%
1651     \else \xint_afterfi{ #1}%
1652     \fi
1653 }%

```

7.55 \xintMinof

1.21 protects \xintMinof against items with non terminated \the\numexpr expressions.
The macro is not compatible with an empty list.

```

1654 \def\xintMinof     {\romannumeral0\xintminof }%
1655 \def\xintminof     #1{\expandafter\XINT_minof_a\romannumeral`&&@#1\xint:}%
1656 \def\XINT_minof_a #1{\expandafter\XINT_minof_b\romannumeral0\xintra{#1}!}%

```

```

1657 \def\XINT_minof_b #1!#2%
1658     {\expandafter\XINT_minof_c\romannumeral`&&@#2!{#1}!}%
1659 \def\XINT_minof_c #1%
1660     {\xint_gob_til_xint: #1\XINT_minof_e\xint:\XINT_minof_d #1}%
1661 \def\XINT_minof_d #1!%
1662     {\expandafter\XINT_minof_b\romannumeral0\xintmin {#1}}%
1663 \def\XINT_minof_e #1!#2!{ #2}%

```

7.56 \xintCmp

```

1664 \def\xintCmp {\romannumeral0\xintcmp }%
1665 \def\xintcmp #1%
1666 {%
1667     \expandafter\XINT_fcmp\expandafter {\romannumeral0\xintraw {#1}}%
1668 }%
1669 \def\XINT_fcmp #1#2%
1670 {%
1671     \expandafter\XINT_fcmp_A\romannumeral0\xintraw {#2}#1%
1672 }%
1673 \def\XINT_fcmp_A #1#2/#3[#4]#5#6/#7[#8]%
1674 {%
1675     \xint_UDsignsfork
1676     #1#5\XINT_fcmp_minusminus
1677     -#5\XINT_fcmp_firstneg
1678     #1-\XINT_fcmp_secondneg
1679     --\XINT_fcmp_nonneg_a
1680     \krof
1681     #1#5{#2/#3[#4]}{#6/#7[#8]}%
1682 }%
1683 \def\XINT_fcmp_minusminus --#1#2{\XINT_fcmp_B #2#1}%
1684 \def\XINT_fcmp_firstneg #1-#2#3{ -1}%
1685 \def\XINT_fcmp_secondneg -#1#2#3{ 1}%
1686 \def\XINT_fcmp_nonneg_a #1#2%
1687 {%
1688     \xint_UDzerosfork
1689     #1#2\XINT_fcmp_zerozero
1690     0#2\XINT_fcmp_firstzero
1691     #10\XINT_fcmp_secondzero
1692     00\XINT_fcmp_pos
1693     \krof
1694     #1#2%
1695 }%
1696 \def\XINT_fcmp_zerozero #1#2#3#4{ 0}%
1697 \def\XINT_fcmp_firstzero #1#2#3#4{ -1}%
1698 \def\XINT_fcmp_secondzero #1#2#3#4{ 1}%
1699 \def\XINT_fcmp_pos #1#2#3#4%
1700 {%
1701     \XINT_fcmp_B #1#3#2#4%
1702 }%
1703 \def\XINT_fcmp_B #1/#2[#3]#4/#5[#6]%
1704 {%
1705     \expandafter\XINT_fcmp_C\expandafter
1706     {\the\numexpr #6-#3\expandafter}\expandafter

```

7 Package *xintfrac* implementation

```
1707 {\romannumeral0\xintiimul {#4}{#2}}%
1708 {\romannumeral0\xintiimul {#5}{#1}}%
1709 }%
1710 \def\XINT_fcmp_C #1#2#3%
1711 {%
1712   \expandafter\XINT_fcmp_D\expandafter
1713   {#3}{#1}{#2}%
1714 }%
1715 \def\XINT_fcmp_D #1#2#3%
1716 {%
1717   \expandafter\XINT_cntSgnFork\romannumeral`&&\expandafter\XINT_cntSgn
1718   \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\xint:
1719   { -1}{\XINT_fcmp_E #2\Z {#3}{#1}}{ 1}%
1720 }%
1721 \def\XINT_fcmp_E #1%
1722 {%
1723   \xint_UDsignfork
1724   #1\XINT_fcmp_Fd
1725   -{\XINT_fcmp_Fn #1}%
1726   \krof
1727 }%

1728 \def\XINT_fcmp_Fd #1\Z #2#3%
1729 {%
1730   \expandafter\XINT_fcmp_Fe
1731   \romannumeral0\XINT_dsx_addzeros {#1}#3;\xint:#2\xint:
1732 }%
1733 \def\XINT_fcmp_Fe #1\xint:#2#3\xint:{\XINT_cmp_plusplus #2#1\xint:#3\xint:}%
1734 \def\XINT_fcmp_Fn #1\Z #2#3%
1735 {%
1736   \expandafter\XINT_fcmp_Fo
1737   \romannumeral0\XINT_dsx_addzeros {#1}#2;\xint:#3\xint:
1738 }%
1739 \def\XINT_fcmp_Fo #1#2\xint:#3\xint:{\XINT_cmp_plusplus #1#3\xint:#2\xint:}%

```

7.57 *\xintAbs*

```
1740 \def\xintAbs {\romannumeral0\xintabs }%
1741 \def\xintabs #1{\expandafter\XINT_abs\romannumeral0\xintraw {#1}}%

```

7.58 *\xintOpp*

```
1742 \def\xintOpp {\romannumeral0\xintopp }%
1743 \def\xintopp #1{\expandafter\XINT_opp\romannumeral0\xintraw {#1}}%

```

7.59 *\xintSgn*

```
1744 \def\xintSgn {\romannumeral0\xintsgn }%
1745 \def\xintsgn #1{\expandafter\XINT_sgn\romannumeral0\xintraw {#1}\xint:}%

```

7.60 Floating point macros

For a long time the float routines dating back to releases [1.07/1.08a](#) (May–June 2013) were not modified.

Since 1.2f (March 2016) the four operations first round their arguments to `\xinttheDigits`-floats (or P -floats), not `(\xinttheDigits+2)`-floats or $(P+2)$ -floats as was the case with earlier releases.

The four operations addition, subtraction, multiplication, division have always produced the correct rounding of the theoretical exact value to P or `\xinttheDigits` digits when the inputs are decimal numbers with at most P digits, and arbitrary decimal exponent part.

From 1.08a to 1.2j, `\xintFloat` (and `\XINTinFloat` which is used to parse inputs to other float macros) handled a fractional input A/B via an initial replacement to A'/B' where A' and B' were A and B truncated to $Q+2$ digits (where asked-for precision is Q), and then they correctly rounded A/B to Q digits. But this meant that this rounding of the input could differ (by up to one unit in the last place) from the correct rounding of the original A/B to the asked-for number of digits (which until 1.2f in uses as auxiliary to the macros for the basic operations was 2 more than the prevailing precision).

Since 1.2k all inputs are correctly rounded to the asked-for number of digits (this was, I think, the case in the 1.07 release -- there are no code comments -- but was, afaicr, not very efficiently done, and this is why the 1.08a release opted for truncation of the numerator and denominator.)

Notice that in float expressions, the `/` is treated as operator, hence the above discussion makes a difference only for the special input form `qfloat(A/B)` or for an `\xintexpr A/B\relax` embedded in the float expression, with A or B having more digits than the prevailing float precision.

Internally there is no inner representation of P -floats as such !!!!!

The input parser will again compute the length of the mantissa on each use !!! This is obviously something that must be improved upon before implementation of higher functions.

Currently, special tricks are used to quickly recognize inputs having no denominators, or fractions whose numerators and denominators are not too long compared to the target precision P , and in particular P -floats or quotients of two such.

Another long-standing issue is that float multiplication will first compute the $2P$ or $2P-2$ digits of the exact product, and then round it to P digits. This is sub-optimal for large P particularly as the multiplication algorithm is basically the schoolbook one, hence worse than quadratic in the $\text{T}_{\text{E}}\text{X}$ implementation which has extra cost of fetching long sequences of tokens.

7.61 `\xintFloat`

1.2f and 1.2g brought some refactoring which resulted in faster treatment of decimal inputs. 1.2i dropped use of some old routines dating back to pre 1.2 era in favor of more modern `\xintDSRr` for rounding. Then 1.2k improves again the handling of denominators B with few digits.

But the main change with 1.2k is a complete rewrite of the $B>1$ case in order to achieve again correct rounding in all cases.

The original version from 1.07 (May 2013) computed the exact rounding to P digits for all inputs. But from 1.08 on (June 2013), the macro handled A/B input by first truncating both A and B to at most $P+2$ digits. This meant that decimal input (arbitrarily long, with scientific part) was correctly rounded, but in case of fractional input there could be up to 0.6 unit in the last place difference of the produced rounding to the input, hence the output could differ from the correct rounding.

Example with 16 digits (the default): `\xintFloat {1/17597472569900621233}`

with `xintfrac 1.07`: 5.682634230727187e-20

with `xintfrac 1.08b--1.2j`: 5.682634230727188e-20

with `xintfrac 1.2k`: 5.682634230727187e-20

The exact value is 5.682634230727187499924124...e-20, showing that 1.07 and 1.2k produce the correct rounding.

Currently the code ends in a more costly branch in about 1 case among 500, where it does some extra operations (a multiplication in particular). There is a free parameter `delta` (here set at

4), I have yet to make some numerical explorations, to see if it could be favorable to set it to a higher value (with $\delta=5$, there is only 1 exceptional case in 5000, etc...).

I have always hesitated about the policy of printing $10.00\dots 0$ in case of rounding upwards to the next power of ten. Already since 1.2f `\XINTinFloat` always produced a mantissa with exactly P digits (except for the zero value). Starting with 1.2k, `\xintFloat` drops this habit of printing $10.00\dots 0$ in such cases. Side note: the rounding-up detection worked when the input A/B was with numerator A and denominator B having each less than $P+2$ digits, or with $B=1$, else, it could happen that the output was a power of ten but not detected to be a rounding up of the original fraction. The value was ok, but printed $1.0\dots 0eN$ with $P-1$ zeroes, not $10.0\dots 0e(N-1)$.

I decided it was not worth the effort to enhance the algorithm to detect with 100% fiability all cases of rounding up to next power of ten, hence 1.2k dropped this.

To avoid duplication of code, and any extra burden on `\XINTinFloat`, which is the macro used internally by the float macros for parsing their inputs, we simply make now `\xintFloat` a wrapper of `\XINTinFloat`.

```

1746 \def\xintFloat    {\romannumeral0\xintfloat }%
1747 \def\xintfloat #1{\XINT_float_chkopt #1\xint:}%
1748 \def\XINT_float_chkopt #1%
1749 {%
1750   \ifx [#1\expandafter\XINT_float_opt
1751     \else\expandafter\XINT_float_noopt
1752   \fi #1%
1753 }%
1754 \def\XINT_float_noopt #1\xint:%
1755 {%
1756   \expandafter\XINT_float_post
1757   \romannumeral0\XINTinfloat[\XINTdigits]{#1}\XINTdigits.%
1758 }%
1759 \def\XINT_float_opt [\xint:#1]%
1760 {%
1761   \expandafter\XINT_float_opt_a\the\numexpr #1.%
1762 }%
1763 \def\XINT_float_opt_a #1.#2%
1764 {%
1765   \expandafter\XINT_float_post
1766   \romannumeral0\XINTinfloat[#1]{#2}#1.%
1767 }%
1768 \def\XINT_float_post #1%
1769 {%
1770   \xint_UDzerominusfork
1771   #1-\XINT_float_zero
1772   0#1\XINT_float_neg
1773   0-\XINT_float_pos
1774   \krof #1%
1775 }%[
1776 \def\XINT_float_zero #1]#2.{ 0.e0}%
1777 \def\XINT_float_neg-{\expandafter-\romannumeral0\XINT_float_pos}%
1778 \def\XINT_float_pos #1#2[#3]#4.%
1779 {%
1780   \expandafter\XINT_float_pos_done\the\numexpr#3+#4-\xint_c_i.#1.#2;%
1781 }%
1782 \def\XINT_float_pos_done #1.#2;{ #2e#1}%

```

7.62 \XINTinFloat, \XINTinFloatS

This routine is like `\xintFloat` but produces an output of the shape `A[N]` which is then parsed faster as input to other float macros. Float operations in `\xintfloatexpr...` relax use internally this format.

It must be used in form `\XINTinFloat[P]{f}`: the optional `[P]` is mandatory.

Since 1.2f, the mantissa always has exactly `P` digits even in case of rounding up to next power of ten. This simplifies other routines.

1.2g added a variant `\XINTinFloatS` which, in case of decimal input with less than the asked for precision `P` will not add extra zeros to the mantissa. For example it may output `2[0]` even if `P=500`, rather than the canonical representation `200...000[-499]`. This is how `\xintFloatMul` and `\xintFloatDiv` parse their inputs, which speeds-up follow-up processing. But `\xintFloatAdd` and `\xintFloatSub` still use `\XINTinFloat` for parsing their inputs; anyway this will have to be changed again when inner structure will carry upfront at least the length of mantissa as data.

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.

Since 1.2k, `\XINTinFloat` always correctly rounds its argument, even if it is a fraction with very big numerator and denominator. See the discussion of `\xintFloat`.

```
1783 \def\xINTinFloat {\romannumeral0\xINTinfloat}%
1784 \def\xINTinfloat
1785   {\expandafter\xINT_infloat_clean\romannumeral0\xINT_infloat}%
1786 \def\xINT_infloat_clean #1%
1787   {\if #1!\xint_dothis\xINT_infloat_clean_a\fi\xint_orthat{ }#1}%

```

Ici on ajoute les zeros pour faire exactement avec `P` chiffres. Car le `#1 = P - L` avec `L` la longueur de `#2`, (ou de `abs(#2)`, ici le `#2` peut avoir un signe) qui est `< P`

```
1788 \def\xINT_infloat_clean_a !#1.#2[#3]%
1789 {%
1790   \expandafter\xINT_infloat_done
1791   \the\numexpr #3-#1\expandafter.%
1792   \romannumeral0\xINT_dsx_addzeros {#1}#2;%
1793}%
1794 \def\xINT_infloat_done #1.#2;{ #2[#1]}%

```

variant which allows output with shorter mantissas.

```
1795 \def\xINTinFloatS {\romannumeral0\xINTinfloatS}%
1796 \def\xINTinfloatS
1797   {\expandafter\xINT_infloatS_clean\romannumeral0\xINT_infloat}%
1798 \def\xINT_infloatS_clean #1%
1799   {\if #1!\xint_dothis\xINT_infloatS_clean_a\fi\xint_orthat{ }#1}%
1800 \def\xINT_infloatS_clean_a !#1.{ }%

```

début de la routine proprement dite, l'argument optionnel est obligatoire.

```
1801 \def\xINT_infloat [#1]#2%
1802 {%
1803   \expandafter\xINT_infloat_a\the\numexpr #1\expandafter.%
1804   \romannumeral0\xINT_infrac {#2}%
1805}%

```


7 Package *xintfrac* implementation

#1=P, #2=n, #3=A, #4=B.

```
1806 \def\XINT_infloat_a #1.#2#3#4%
1807 {%
```

micro boost au lieu d'utiliser \XINT_isOne{#4}, mais pas bon style.

```
1808 \if1\XINT_is_One#4XY%
1809 \expandafter\XINT_infloat_sp
1810 \else\expandafter\XINT_infloat_fork
1811 \fi #3.{#1}{#2}{#4}%
1812 }%
```

Special quick treatment of B=1 case (1.2f then again 1.2g.)
maintenant: A.{P}{N}{1} Il est possible que A soit nul.

```
1813 \def\XINT_infloat_sp #1%
1814 {%
1815 \xint_UDzerominusfork
1816 #1-\XINT_infloat_spzero
1817 0#1\XINT_infloat_spneg
1818 0-\XINT_infloat_sppos
1819 \krof #1%
1820 }%
```

Attention surtout pas 0/1[0] ici.

```
1821 \def\XINT_infloat_spzero 0.#1#2#3{ 0[0]}%
1822 \def\XINT_infloat_spneg-%
1823 {\expandafter\XINT_infloat_spnegend\romannumeral0\XINT_infloat_sppos}%
1824 \def\XINT_infloat_spnegend #1%
1825 {\if#1!\expandafter\XINT_infloat_spneg_needzeros\fi -#1}%
1826 \def\XINT_infloat_spneg_needzeros -!#1.{!#1.-}%%
```

in: A.{P}{N}{1}
out: P-L.A.P.N.

```
1827 \def\XINT_infloat_sppos #1.#2#3#4%
1828 {%
1829 \expandafter\XINT_infloat_sp_b\the\numexpr#2-\xintLength{#1}.#1.#2.#3.%
1830 }%
```

#1= P-L. Si c'est positif ou nul il faut retrancher #1 à l'exposant, et ajouter autant de zéros.
On regarde premier token. P-L.A.P.N.

```
1831 \def\XINT_infloat_sp_b #1%
1832 {%
1833 \xint_UDzerominusfork
1834 #1-\XINT_infloat_sp_quick
1835 0#1\XINT_infloat_sp_c
1836 0-\XINT_infloat_sp_needzeros
1837 \krof #1%
1838 }%
```

Ici P=L. Le cas usuel dans \xintfloatexpr.

```
1839 \def\XINT_infloat_sp_quick 0.#1.#2.#3.{ #1[#3]}%
```

7 Package *xintfrac* implementation

Ici $\#1=P-L$ est >0 . L'exposant sera $N-(P-L)$. $\#2=A$. $\#3=P$. $\#4=N$.

18 mars 2016. En fait dans certains contextes il est sous-optimal d'ajouter les zéros. Par exemple quand c'est appelé par la multiplication ou la division, c'est idiot de convertir 2 en 200000...00000[-499]. Donc je redéfinis `addzeros` en `needzeroes`. Si on appelle sous la forme `\XINTinFloatS`, on ne fait pas l'addition de zeros.

```
1840 \def\XINT_infloat_sp_needzeros #1.#2.#3.#4.{!#1.#2[#4]}%
```

$L-P=\#1$. $A=\#2$. $P=\#3$. $N=\#4$.

Ici $P < L$. Il va falloir arrondir. Attention si on va à la puissance de 10 suivante. En $\#1$ on a $L-P$ qui est >0 . L'exposant final sera $N+L-P$, sauf dans le cas spécial, il sera alors $N+L-P+1$. L'ajustement final est fait par `\XINT_infloat_Y`.

```
1841 \def\XINT_infloat_sp_c -#1.#2.#3.#4.#5.%
```

```
1842 {%
```

```
1843   \expandafter\XINT_infloat_Y
```

```
1844   \the\numexpr #5+#1\expandafter.%
```

```
1845   \romannumeral0\expandafter\XINT_infloat_sp_round
```

```
1846   \romannumeral0\XINT_split_fromleft
```

```
1847   (\xint_c_i+#4).#2#3\xint_bye2345678\xint_bye..#2%
```

```
1848 }%
```

```
1849 \def\XINT_infloat_sp_round #1.#2.%
```

```
1850 {%
```

```
1851   \XINT_dsrr#1\xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax.%
```

```
1852 }%
```

General branch for A/B with $B > 1$ inputs. It achieves correct rounding always since 1.2k (done January 2, 2017.) This branch is never taken for $A=0$ because `\XINT_infrac` will have returned $B=1$ then.

```
1853 \def\XINT_infloat_fork #1%
```

```
1854 {%
```

```
1855   \xint_UDsignfork
```

```
1856   #1\XINT_infloat_J
```

```
1857   -\XINT_infloat_K
```

```
1858   \krof #1%
```

```
1859 }%
```

```
1860 \def\XINT_infloat_J-{\expandafter-\romannumeral0\XINT_infloat_K }%
```

$A.\{P\}\{n\}\{B\}$ avec $B > 1$.

```
1861 \def\XINT_infloat_K #1.#2%
```

```
1862 {%
```

```
1863   \expandafter\XINT_infloat_L
```

```
1864   \the\numexpr\xintLength{#1}\expandafter.\the\numexpr #2+\xint_c_iv.{#1}\{#2}%
```

```
1865 }%
```

$|A|.P+4.\{A\}\{P\}\{n\}\{B\}$. We check if A already has length $\leq P+4$.

```
1866 \def\XINT_infloat_L #1.#2.%
```

```
1867 {%
```

```
1868   \ifnum #1>#2
```

```
1869     \expandafter\XINT_infloat_Ma
```

```
1870   \else
```

```
1871     \expandafter\XINT_infloat_Mb
```

```
1872   \fi #1.#2.%
```

```
1873 }%
```

7 Package *xintfrac* implementation

$|A|.P+4.\{A\}\{P\}\{n\}\{B\}$. We will keep only the first $P+4$ digits of A , denoted A' in what follows.
output: $u=-0.A'.junk.P+4.|A|. \{A\}\{P\}\{n\}\{B\}$

```
1874 \def\XINT_infloat_Ma #1.#2.#3%
1875 {%
1876   \expandafter\XINT_infloat_MtoN\expandafter-\expandafter0\expandafter.%
1877   \romannumeral0\XINT_split_fromleft#2.#3\xint_bye2345678\xint_bye..%
1878   #2.#1.{#3}%
1879 }%
```

$|A|.P+4.\{A\}\{P\}\{n\}\{B\}$.
Here A is short. We set $u = P+4 - |A|$, and $A' = A$ ($A' = 10^u A$)
output: $u.A'..P+4.|A|. \{A\}\{P\}\{n\}\{B\}$

```
1880 \def\XINT_infloat_Mb #1.#2.#3%
1881 {%
1882   \expandafter\XINT_infloat_MtoN\the\numexpr#2-#1.%
1883   #3..#2.#1.{#3}%
1884 }%
```

input $u.A'..junk.P+4.|A|. \{A\}\{P\}\{n\}\{B\}$
output $|B|.P+4.\{B\}u.A'..P.|A|.n.\{A\}\{B\}$

```
1885 \def\XINT_infloat_MtoN #1.#2.#3.#4.#5.#6#7#8#9%
1886 {%
1887   \expandafter\XINT_infloat_N
1888   \the\numexpr\xintLength{#9}.#4.{#9}#1.#2.#7.#5.#8.{#6}{#9}%
1889 }%
1890 \def\XINT_infloat_N #1.#2.%
1891 {%
1892   \ifnum #1>#2
1893     \expandafter\XINT_infloat_Oa
1894   \else
1895     \expandafter\XINT_infloat_Ob
1896   \fi #1.#2.%
1897 }%
```

input $|B|.P+4.\{B\}u.A'..P.|A|.n.\{A\}\{B\}$
output $v=-0.B'..junk.|B|.u.A'..P.|A|.n.\{A\}\{B\}$

```
1898 \def\XINT_infloat_Oa #1.#2.#3%
1899 {%
1900   \expandafter\XINT_infloat_P\expandafter-\expandafter0\expandafter.%
1901   \romannumeral0\XINT_split_fromleft#2.#3\xint_bye2345678\xint_bye..%
1902   #1.%
1903 }%
```

output $v=P+4-|B|>=0.B'..junk.|B|.u.A'..P.|A|.n.\{A\}\{B\}$

```
1904 \def\XINT_infloat_Ob #1.#2.#3%
1905 {%
1906   \expandafter\XINT_infloat_P\the\numexpr#2-#1.#3..#1.%
1907 }%
```

7 Package *xintfrac* implementation

```

input v.B''.junk.|B|.u.A''.P.|A|.n.{A}{B}
output Q1.P.|B|. |A|.n.{A}{B}
Q1 = division euclidienne de A''.10^{u-v+P+3} par B''.
  Special detection of cases with A and B both having length at most P+4: this will happen when
  called from \xintFloatDiv as A and B (produced then via \XINTinFloatS) will have at most P digits.
  We then only need integer division with P+1 extra zeros, not P+3.

1908 \def\xINT_infloat_P #1#2.#3.#4.#5.#6#7.#8.#9.%
1909 {%
1910   \csname XINT_infloat_Q\if-#1\else\if-#6\else q\fi\fi\expandafter\endcsname
1911   \romannumeral0\xintiique
1912   {\romannumeral0\xINT_dsx_addzerosnofuss
1913    {#6#7-#1#2+#9+\xint_c_iii\if-#1\else\if-#6\else-\xint_c_ii\fi\fi}#8;}%
1914   {#3}.#9.#5.%
1915 }%

  «quick» branch.

1916 \def\xINT_infloat_Qq #1.#2.%
1917 {%
1918   \expandafter\xINT_infloat_Rq
1919   \romannumeral0\xINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..#2.%
1920 }%
1921 \def\xINT_infloat_Rq #1.#2#3.%
1922 {%
1923   \ifnum#2<\xint_c_v
1924     \expandafter\xINT_infloat_SEq
1925   \else\expandafter\xINT_infloat_SUP
1926   \fi
1927   {\if.#3.\xint_c_\else\xint_c_i\fi}#1.%
1928 }%

  standard branch which will have to handle undecided rounding, if too close to a mid-value.

1929 \def\xINT_infloat_Q #1.#2.%
1930 {%
1931   \expandafter\xINT_infloat_R
1932   \romannumeral0\xINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..#2.%
1933 }%
1934 \def\xINT_infloat_R #1.#2#3#4#5.%
1935 {%
1936   \if.#5.\expandafter\xINT_infloat_Sa\else\expandafter\xINT_infloat_Sb\fi
1937   #2#3#4#5.#1.%
1938 }%

  trailing digits.Q.P.|B|. |A|.n.{A}{B}
  #1=trailing digits (they may have leading zeros.)

1939 \def\xINT_infloat_Sa #1.%
1940 {%
1941   \ifnum#1>500 \xint_dothis\xINT_infloat_SUP\fi
1942   \ifnum#1<499 \xint_dothis\xINT_infloat_SEq\fi
1943   \xint_orthat\xINT_infloat_X\xint_c_
1944 }%
```

7 Package *xintfrac* implementation

```

1945 \def\XINT_infloat_Sb #1.%
1946 {%
1947   \ifnum#1>5009 \xint_dothis\XINT_infloat_SUP\fi
1948   \ifnum#1<4990 \xint_dothis\XINT_infloat_SEq\fi
1949   \xint_orthat\XINT_infloat_X\xint_c_i
1950 }%

```

$\epsilon = Q.P.B.A.n.A.B$
 $\text{exposant final est } n+|A|-|B|-P+\epsilon$

```

1951 \def\XINT_infloat_SEq #1#2.#3.#4.#5.#6.#7#8%
1952 {%
1953   \expandafter\XINT_infloat_SY
1954   \the\numexpr #6+#5-#4-#3+#1.#2.%
1955 }%
1956 \def\XINT_infloat_SY #1.#2.{ #2[#1]}%

```

initial digit #2 put aside to check for case of rounding up to next power of ten, which will need adjustment of mantissa and exponent.

```

1957 \def\XINT_infloat_SUP #1#2#3.#4.#5.#6.#7.#8#9%
1958 {%
1959   \expandafter\XINT_infloat_Y
1960   \the\numexpr#7+#6-#5-#4+#1\expandafter.%
1961   \romannumeral0\xintinc{#2#3}.#2%
1962 }%

```

$\epsilon = Q.P.B.A.n.A.B$

$\text{\xintDSH}\{-x\}\{U\}$ multiplies U by 10^x . When x is negative, this means it truncates (i.e. it drops the last $-x$ digits).

We don't try to optimize too much macro calls here, the odds are 2 per 1000 for this branch to be taken. Perhaps in future I will use higher free parameter d , which currently is set at 4.

$\#1=\epsilon, \#2\#3=Q, \#4=P, \#5=|B|, \#6=|A|, \#7=n, \#8=A, \#9=B$

```

1963 \def\XINT_infloat_X #1#2#3.#4.#5.#6.#7.#8#9%
1964 {%
1965   \expandafter\XINT_infloat_Y
1966   \the\numexpr #7+#6-#5-#4+#1\expandafter.%
1967   \romannumeral`&&\romannumeral0\xintiiiflt
1968   {\xintDSH{#6-#5-#4+#1}\xintDouble{#8}}}%
1969   {\xintiiMul{\xintInc{\xintDouble{#2#3}}}\{#9}}}%
1970   \xint_firstofone
1971   \xintinc{#2#3}.#2%
1972 }%

```

check for rounding up to next power of ten.

```

1973 \def\XINT_infloat_Y #1{%
1974 \def\XINT_infloat_Y ##1.##2##3.##4%
1975 {%
1976   \if##49\if##21\expandafter\expandafter\expandafter\XINT_infloat_Z\fi\fi
1977   #1##2##3[##1]%
1978 }}\XINT_infloat_Y{ }%

```

#1=1, #2=0.

```
1979 \def\XINT_infloat_Z #1#2#3[#4]%
1980 {%
1981   \expandafter\XINT_infloat_ZZ\the\numexpr#4+\xint_c_i.#3.%
1982 }%
1983 \def\XINT_infloat_ZZ #1.#2.{ 1#2[#1]}%
```

7.63 \xintPFloat

1.1. This is a prettifying printing macro for floats.

The macro applies one simple rule: $x.yz\dots eN$ 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`.

Special case: the zero value is printed 0. (with a dot)

The coding got simpler with 1.2k as its `\xintFloat` always produces a mantissa with exactly P digits (no more $10.0\dots 0eN$ annoying exception).

```
1984 \def\xintPFloat {\romannumeral0\xintpfloat }%
1985 \def\xintpfloat #1{\XINT_pfloat_chkopt #1\xint:}%
1986 \def\XINT_pfloat_chkopt #1%
1987 {%
1988   \ifx [#1\expandafter\XINT_pfloat_opt
1989     \else\expandafter\XINT_pfloat_noopt
1990   \fi #1%
1991 }%
1992 \def\XINT_pfloat_noopt #1\xint:%
1993 {%
1994   \expandafter\XINT_pfloat_a
1995   \romannumeral0\xintfloat [\XINTdigits]{#1};\XINTdigits.%
1996 }%

1997 \def\XINT_pfloat_opt [\xint:#1]%
1998 {%
1999   \expandafter\XINT_pfloat_opt_a \the\numexpr #1.%
2000 }%
2001 \def\XINT_pfloat_opt_a #1.#2%
2002 {%
2003   \expandafter\XINT_pfloat_a\romannumeral0\xintfloat [#1]{#2};#1.%
2004 }%
2005 \def\XINT_pfloat_a #1%
2006 {%
2007   \xint_UDzerominusfork
2008     #1-\XINT_pfloat_zero
2009     0#1\XINT_pfloat_neg
2010     0-\XINT_pfloat_pos
2011   \krof #1%
2012 }%

2013 \def\XINT_pfloat_zero #1;#2.{ 0.}%
2014 \def\XINT_pfloat_neg-{\expandafter-\romannumeral0\XINT_pfloat_pos }%
```

```

2015 \def\XINT_pfloat_pos #1.#2e#3;#4.%
2016 {%
2017   \ifnum #3>\xint_c_v \xint_dothis\XINT_pfloat_no\fi
2018   \ifnum #3<-\xint_c_v \xint_dothis\XINT_pfloat_no\fi
2019   \ifnum #3<\xint_c_ \xint_dothis\XINT_pfloat_N\fi
2020   \ifnum #3>\numexpr #4-\xint_c_i\relax \xint_dothis\XINT_pfloat_Ps\fi
2021   \xint_orthat\XINT_pfloat_P #1#2e#3;%
2022 }%
2023 \def\XINT_pfloat_no #1#2;{ #1.#2}%

```

This is all simpler coded, now that 1.2k's `\xintFloat` always outputs a mantissa with exactly one digits before decimal mark always.

```

2024 \def\XINT_pfloat_N #1e-#2;%
2025 {%
2026   \csname XINT_pfloat_N\romannumeral#2\endcsname #1%
2027 }%
2028 \def\XINT_pfloat_N_i { 0.}%
2029 \def\XINT_pfloat_N_ii { 0.0}%
2030 \def\XINT_pfloat_N_iii{ 0.00}%
2031 \def\XINT_pfloat_N_iv { 0.000}%
2032 \def\XINT_pfloat_N_v { 0.0000}%

```

```

2033 \def\XINT_pfloat_P #1e#2;%
2034 {%
2035   \csname XINT_pfloat_P\romannumeral#2\endcsname #1%
2036 }%
2037 \def\XINT_pfloat_P_ #1{ #1.}%
2038 \def\XINT_pfloat_P_i #1#2{ #1#2.}%
2039 \def\XINT_pfloat_P_ii #1#2#3{ #1#2#3.}%
2040 \def\XINT_pfloat_P_iii#1#2#3#4{ #1#2#3#4.}%
2041 \def\XINT_pfloat_P_iv #1#2#3#4#5{ #1#2#3#4#5.}%
2042 \def\XINT_pfloat_P_v #1#2#3#4#5#6{ #1#2#3#4#5#6.}%

```

```

2043 \def\XINT_pfloat_Ps #1e#2;%
2044 {%
2045   \csname XINT_pfloat_Ps\romannumeral#2\endcsname #100000;%
2046 }%
2047 \def\XINT_pfloat_Psi #1#2#3;{ #1#2.}%
2048 \def\XINT_pfloat_Psii #1#2#3#4;{ #1#2#3.}%
2049 \def\XINT_pfloat_Psiii#1#2#3#4#5;{ #1#2#3#4.}%
2050 \def\XINT_pfloat_Psiv #1#2#3#4#5#6;{ #1#2#3#4#5.}%
2051 \def\XINT_pfloat_Psv #1#2#3#4#5#6#7;{ #1#2#3#4#5#6.}%

```

7.64 `\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.

```
2052 \def\XINTinFloatFracdigits {\romannumeral0\XINTinfloatfracdigits }%
2053 \def\XINTinfloatfracdigits #1%
2054 {%
2055   \expandafter\XINT_infloatfracdg_a\expandafter {\romannumeral0\xintfrac{#1}}%
2056 }%
2057 \def\XINT_infloatfracdg_a {\XINTinfloat [\XINTdigits]]%
```

7.65 `\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.

```
2058 \def\xintFloatAdd      {\romannumeral0\xintfloatadd }%
2059 \def\xintfloatadd      #1{\XINT_fladd_chkopt \xintfloat #1\xint:}%
2060 \def\XINTinFloatAdd     {\romannumeral0\XINTinfloatadd }%
2061 \def\XINTinfloatadd     #1{\XINT_fladd_chkopt \XINTinfloatS #1\xint:}%
2062 \def\XINT_fladd_chkopt #1#2%
2063 {%
2064   \ifx [#2\expandafter\XINT_fladd_opt
2065     \else\expandafter\XINT_fladd_noopt
2066   \fi #1#2%
2067 }%
2068 \def\XINT_fladd_noopt #1#2\xint:#3%
2069 {%
2070   #1[\XINTdigits]%
2071   {\expandafter\XINT_FL_add_a
2072     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{#3}}%
2073 }%
2074 \def\XINT_fladd_opt #1[\xint:#2]%#3#4%
2075 {%
2076   \expandafter\XINT_fladd_opt_a\the\numexpr #2.#1%
2077 }%
2078 \def\XINT_fladd_opt_a #1.#2#3#4%
2079 {%
2080   #2[#1]{\expandafter\XINT_FL_add_a\romannumeral0\XINTinfloat[#1]{#3}#1.{#4}}%
2081 }%
2082 \def\XINT_FL_add_a #1%
2083 {%
2084   \xint_gob_til_zero #1\XINT_FL_add_zero 0\XINT_FL_add_b #1%
2085 }%

2086 \def\XINT_FL_add_zero #1.#2{#2}%[
```



```

2087 \def\XINT_FL_add_b #1]#2.#3%
2088 {%
2089   \expandafter\XINT_FL_add_c\romannumeral0\XINTinfloat[#2]{#3}#2.#1%
2090 }%

```

```

2091 \def\XINT_FL_add_c #1%
2092 {%
2093   \xint_gob_til_zero #1\XINT_FL_add_zero 0\XINT_FL_add_d #1%
2094 }%

```

```

2095 \def\XINT_FL_add_d #1[#2]#3.#4[#5]%
2096 {%
2097   \ifnum\numexpr #2-#3-#5>\xint_c_\xint_dothis\xint_firstoftwo\fi
2098   \ifnum\numexpr #5-#3-#2>\xint_c_\xint_dothis\xint_secondoftwo\fi
2099   \xint_orthat\xintAdd {#1[#2]}{#4[#5]}%
2100 }%

```

7.66 \xintFloatSub, \XINTinFloatSub

First done 1.07.

Starting with 1.2f the arguments undergo an initial rounding to the target precision P not P+2.

```

2101 \def\xintFloatSub      {\romannumeral0\xintfloatsub }%
2102 \def\xintfloatsub      #1{\XINT_flsub_chkopt \xintfloat #1\xint:}%
2103 \def\XINTinFloatSub    {\romannumeral0\XINTinfloatsub }%
2104 \def\XINTinfloatsub    #1{\XINT_flsub_chkopt \XINTinfloatS #1\xint:}%
2105 \def\XINT_flsub_chkopt #1#2%
2106 {%
2107   \ifx [#2\expandafter\XINT_flsub_opt
2108     \else\expandafter\XINT_flsub_noopt
2109   \fi #1#2%
2110 }%
2111 \def\XINT_flsub_noopt #1#2\xint:#3%
2112 {%
2113   #1[\XINTdigits]%
2114   {\expandafter\XINT_FL_add_a
2115     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{\xintOpp{#3}}}%
2116 }%
2117 \def\XINT_flsub_opt #1[\xint:#2]#3#4%
2118 {%
2119   \expandafter\XINT_flsub_opt_a\the\numexpr #2.#1%
2120 }%
2121 \def\XINT_flsub_opt_a #1.#2#3#4%
2122 {%
2123   #2[#1]{\expandafter\XINT_FL_add_a\romannumeral0\XINTinfloat[#1]{#3}#1.{\xintOpp{#4}}}%
2124 }%

```

7.67 \xintFloatMul, \XINTinFloatMul

1.07.

Starting with 1.2f the arguments are rounded to the target precision P not P+2.

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.

1.2k does a micro improvement to the way the macro passes over control to its output routine (former version used a higher level \xintE causing some extra un-needed processing with two calls to \XINT_infrac where one was amply enough).

```

2125 \def\xintFloatMul    {\romannumeral0\xintfloatmul    }%
2126 \def\xintfloatmul    #1{\XINT_flmul_chkopt \xintfloat #1\xint:}%
2127 \def\XINTinFloatMul  {\romannumeral0\XINTinfloatmul }%
2128 \def\XINTinfloatmul  #1{\XINT_flmul_chkopt \XINTinfloatS #1\xint:}%
2129 \def\XINT_flmul_chkopt #1#2%
2130 {%
2131   \ifx [#2\expandafter\XINT_flmul_opt
2132     \else\expandafter\XINT_flmul_noopt
2133   \fi  #1#2%
2134 }%
2135 \def\XINT_flmul_noopt #1#2\xint:#3%
2136 {%
2137   #1[\XINTdigits]%
2138   {\expandafter\XINT_FL_mul_a
2139     \romannumeral0\XINTinfloatS[\XINTdigits]{#2}\XINTdigits.{#3}}%
2140 }%
2141 \def\XINT_flmul_opt #1[\xint:#2]%#3#4%
2142 {%
2143   \expandafter\XINT_flmul_opt_a\the\numexpr #2.#1%
2144 }%
2145 \def\XINT_flmul_opt_a #1.#2#3#4%
2146 {%
2147   #2[#1]{\expandafter\XINT_FL_mul_a\romannumeral0\XINTinfloatS[#1]{#3}#1.#4}}%
2148 }%
2149 \def\XINT_FL_mul_a #1[#2]#3.#4%
2150 {%
2151   \expandafter\XINT_FL_mul_b\romannumeral0\XINTinfloatS[#3]{#4}#1[#2]%
2152 }%

2153 \def\XINT_FL_mul_b #1[#2]#3[#4]{\xintiiMul{#3}{#1}/1[#4+#2]}%

```

7.68 \xintFloatDiv, \XINTinFloatDiv

1.07.

Starting with 1.2f the arguments are rounded to the target precision P not P+2.

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.

The actual rounding of the quotient is handled via \xintfloat (or \XINTinfloatS).

1.2k does the same kind of improvement in \XINT_FL_div_b as for multiplication: earlier code was unnecessarily high level.

```

2154 \def\xintFloatDiv {\romannumeral0\xintfloatdiv }%
2155 \def\xintfloatdiv #1{\XINT_fldiv_chkopt \xintfloat #1\xint:}%
2156 \def\XINTinFloatDiv {\romannumeral0\XINTinfloatdiv }%
2157 \def\XINTinfloatdiv #1{\XINT_fldiv_chkopt \XINTinfloatS #1\xint:}%
2158 \def\XINT_fldiv_chkopt #1#2%
2159 {%
2160   \ifx [#2\expandafter\XINT_fldiv_opt
2161     \else\expandafter\XINT_fldiv_noopt
2162   \fi #1#2%
2163 }%

2164 \def\XINT_fldiv_noopt #1#2\xint:#3%
2165 {%
2166   #1[\XINTdigits]%
2167   {\expandafter\XINT_FL_div_a
2168     \romannumeral0\XINTinfloatS[\XINTdigits]{#3}\XINTdigits.{#2}}%
2169 }%
2170 \def\XINT_fldiv_opt #1[\xint:#2]%#3#4%
2171 {%
2172   \expandafter\XINT_fldiv_opt_a\the\numexpr #2.#1%
2173 }%

2174 \def\XINT_fldiv_opt_a #1.#2#3#4%
2175 {%
2176   #2[#1]{\expandafter\XINT_FL_div_a\romannumeral0\XINTinfloatS[#1]{#4}#1.#3}%
2177 }%
2178 \def\XINT_FL_div_a #1[#2]#3.#4%
2179 {%
2180   \expandafter\XINT_FL_div_b\romannumeral0\XINTinfloatS[#3]{#4}/#1e#2%
2181 }%

2182 \def\XINT_FL_div_b #1[#2]{#1e#2}%

```

7.69 \xintFloatPow, \XINTinFloatPow

1.07: initial version. 1.09j has re-organized the core loop.

2015/12/07. I have hesitated to map $^$ 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).

```

2183 \def\xintFloatPow {\romannumeral0\xintfloatpow}%
2184 \def\xintfloatpow #1{\XINT_flpow_chkopt \xintfloat #1\xint:}%
2185 \def\XINTinFloatPow {\romannumeral0\XINTinfloatpow}%
2186 \def\XINTinfloatpow #1{\XINT_flpow_chkopt \XINTinfloatS #1\xint:}%
2187 \def\XINT_flpow_chkopt #1#2%

```

```

2188 {%
2189   \ifx [#2\expandafter\XINT_flpow_opt
2190     \else\expandafter\XINT_flpow_noopt
2191   \fi
2192   #1#2%
2193 }%
2194 \def\XINT_flpow_noopt #1#2\xint:#3%
2195 {%
2196   \expandafter\XINT_flpow_checkB_a
2197   \the\numexpr #3.\XINTdigits.{#2}{#1[\XINTdigits]}}%
2198 }%
2199 \def\XINT_flpow_opt #1[\xint:#2]%
2200 {%
2201   \expandafter\XINT_flpow_opt_a\the\numexpr #2.#1%
2202 }%
2203 \def\XINT_flpow_opt_a #1.#2#3#4%
2204 {%
2205   \expandafter\XINT_flpow_checkB_a\the\numexpr #4.#1.{#3}{#2[#1]}}%
2206 }%
2207 \def\XINT_flpow_checkB_a #1%
2208 {%
2209   \xint_UDzerominusfork
2210   #1-\XINT_flpow_BisZero
2211   0#1{\XINT_flpow_checkB_b -}%
2212   0-{\XINT_flpow_checkB_b }{#1}%
2213   \krof
2214 }%
2215 \def\XINT_flpow_BisZero .#1.#2#3{#3{1[0]}}%

2216 \def\XINT_flpow_checkB_b #1#2.#3.%
2217 {%
2218   \expandafter\XINT_flpow_checkB_c
2219   \the\numexpr\xintLength{#2}+\xint_c_iii.#3.#2.{#1}%
2220 }%
2221 \def\XINT_flpow_checkB_c #1.#2.%
2222 {%
2223   \expandafter\XINT_flpow_checkB_d\the\numexpr#1+#2.#1.#2.%
2224 }%

1.2f rounds input to P digits, first.

2225 \def\XINT_flpow_checkB_d #1.#2.#3.#4.#5#6%
2226 {%
2227   \expandafter \XINT_flpow_aa
2228   \romannumeral0\XINTinfloat [#3]{#6}{#2}{#1}{#4}{#5}%
2229 }%

2230 \def\XINT_flpow_aa #1[#2]#3%
2231 {%

```

7 Package *xintfrac* implementation

```

2232 \expandafter\XINT_flpow_ab\the\numexpr #2-#3\expandafter.%
2233 \romannumeral\XINT_rep #3\endcsname0.#1.%
2234 }%

2235 \def\XINT_flpow_ab #1.#2.#3.{\XINT_flpow_a #3#2[#1]}%

2236 \def\XINT_flpow_a #1%
2237 {%
2238 \xint_UDzerominusfork
2239 #1-\XINT_flpow_zero
2240 0#1{\XINT_flpow_b \iftrue}%
2241 0-{\XINT_flpow_b \iffalse#1}%
2242 \krof
2243 }%
2244 \def\XINT_flpow_zero #1[#2]#3#4#5#6%
2245 {%
2246 #6{\if 1#51\xint_dothis {0[0]}\fi
2247 \xint_orthat
2248 {\XINT_signalcondition{DivisionByZero}{0 to the power #4}{0[0]}}%
2249 }%
2250 }%

2251 \def\XINT_flpow_b #1#2[#3]#4#5%
2252 {%
2253 \XINT_flpow_loopI #5.#3.#2.#4.{#1\ifodd #5 \xint_c_i\fi\fi}%
2254 }%

2255 \def\XINT_flpow_truncate #1.#2.#3.%
2256 {%
2257 \expandafter\XINT_flpow_truncate_a
2258 \romannumeral0\XINT_split_fromleft
2259 #3.#2\xint_bye2345678\xint_bye..#1.#3.%
2260 }%

2261 \def\XINT_flpow_truncate_a #1.#2.#3.{#3+\xintLength{#2}.#1.}%
2262 \def\XINT_flpow_loopI #1.%
2263 {%
2264 \ifnum #1=\xint_c_i\expandafter\XINT_flpow_ItoIII\fi
2265 \ifodd #1
2266 \expandafter\XINT_flpow_loopI_odd
2267 \else
2268 \expandafter\XINT_flpow_loopI_even
2269 \fi
2270 #1.%
2271 }%

```

7 Package *xintfrac* implementation

```

2272 \def\XINT_flpow_ItoIII\ifodd #1\fi #2.#3.#4.#5.#6%
2273 {%
2274     \expandafter\XINT_flpow_III\the\numexpr #6+\xint_c_.#3.#4.#5.%
2275 }%

2276 \def\XINT_flpow_loopI_even #1.#2.#3.%#4.%
2277 {%
2278     \expandafter\XINT_flpow_loopI
2279     \the\numexpr #1/\xint_c_ii\expandafter.%
2280     \the\numexpr\expandafter\XINT_flpow_truncate
2281     \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.%
2282 }%
2283 \def\XINT_flpow_loopI_odd #1.#2.#3.#4.%
2284 {%
2285     \expandafter\XINT_flpow_loopII
2286     \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter.%
2287     \the\numexpr\expandafter\XINT_flpow_truncate
2288     \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#2.#3.%
2289 }%
2290 \def\XINT_flpow_loopII #1.%
2291 {%
2292     \ifnum #1 = \xint_c_i\expandafter\XINT_flpow_ItoIII\fi
2293     \ifodd #1
2294         \expandafter\XINT_flpow_loopII_odd
2295     \else
2296         \expandafter\XINT_flpow_loopII_even
2297     \fi
2298     #1.%
2299 }%
2300 \def\XINT_flpow_loopII_even #1.#2.#3.%#4.%
2301 {%
2302     \expandafter\XINT_flpow_loopII
2303     \the\numexpr #1/\xint_c_ii\expandafter.%
2304     \the\numexpr\expandafter\XINT_flpow_truncate
2305     \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.%
2306 }%
2307 \def\XINT_flpow_loopII_odd #1.#2.#3.#4.#5.#6.%
2308 {%
2309     \expandafter\XINT_flpow_loopII_odda
2310     \the\numexpr\expandafter\XINT_flpow_truncate
2311     \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2312     #1.#2.#3.%
2313 }%
2314 \def\XINT_flpow_loopII_odda #1.#2.#3.#4.#5.#6.%
2315 {%
2316     \expandafter\XINT_flpow_loopII
2317     \the\numexpr #4/\xint_c_ii-\xint_c_i\expandafter.%
2318     \the\numexpr\expandafter\XINT_flpow_truncate
2319     \the\numexpr\xint_c_ii*#5\expandafter.\romannumeral0\xintiisqr{#6}.#3.%

```

```

2320      #1.#2.%
2321 }%
```

```

2322 \def\XINT_flpow_IItoIII\ifodd #1\fi #2.#3.#4.#5.#6.#7.#8%
2323 {%
2324     \expandafter\XINT_flpow_III\the\numexpr #8+\xint_c_\expandafter.%
2325     \the\numexpr\expandafter\XINT_flpow_truncate
2326     \the\numexpr#3+#6\expandafter.\romannumeral0\xintiimul{#4}{#7}.#5.%
2327 }%
```

This ending is common with `\xintFloatPower`.

In the case of negative exponent we need to inverse the Q-digits mantissa. This requires no special attention now as 1.2k's `\xintFloat` does correct rounding of fractions hence it is easy to bound the total error. It can be checked that the algorithm after final rounding to the target precision computes a value Z whose distance to the exact theoretical will be less than $0.52 \text{ ulp}(Z)$ (and worst cases can only be slightly worse than $0.51 \text{ ulp}(Z)$).

In the case of the half-integer exponent (only via the expression interface,) the computation (which proceeds via `\XINTinFloatPowerH`) ends with a square root. This square root extraction is done with 3 guard digits (the power operations were done with more.) Then the value is rounded to the target precision. There is thus this rounding to 3 guard digits (in the case of negative exponent the reciprocal is computed before the square-root), then the square root is (computed with exact rounding for these 3 guard digits), and then there is the final rounding of this to the target precision. The total error (for positive as well as negative exponent) has been estimated to at worst possibly exceed slightly $0.5125 \text{ ulp}(Z)$, and at any rate it is less than $0.52 \text{ ulp}(Z)$.

```

2328 \def\XINT_flpow_III #1.#2.#3.#4.#5%
2329 {%
2330     \expandafter\XINT_flpow_IIIend
2331     \xint_UDsignfork
2332     #5{{1/#3[-#2]}}}%
2333     -{{#3[#2]}}}%
2334     \krof #1%
2335 }%
```

```

2336 \def\XINT_flpow_IIIend #1#2#3%
2337     {#3{\if#21\xint_afterfi{\expandafter-\romannumeral`&&@\fi#1}}}%

```

7.70 `\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 \wedge in expressions is mapped to this routine.

Same modifications as in `\xintFloatPow` for 1.2f.

1.2f adds a special private macro for allowing half-integral exponents for use with \wedge within `\xintfloatexpr`. The exponent will be first truncated to either an integer or an half-integer. The macro is not for general use.

1.2k does anew this 1.2f handling of half-integer exponents for the `\xintfloatexpr` parser: with 1.2f's code the final square-root extraction was applied to a value already rounded to the target precision, unneedlessly losing precision.

```

2338 \def\xintFloatPower {\romannumeral0\xintfloatpower}%

```

```

2339 \def\xintfloatpower #1{\XINT_flpower_chkopt \xintfloat #1\xint:}%
2340 \def\XINTinFloatPower {\romannumeral0\XINTinfloatpower }%
2341 \def\XINTinfloatpower #1{\XINT_flpower_chkopt \XINTinfloatS #1\xint:}%

```

First the special macro for use by the expression parser which checks if one raises to an half-integer exponent. This is always with `\XINTdigits` precision. Rewritten for 1.2k in order for the final square root to keep three guard digits.

We have to be careful that exponent #2 is not constrained by TeX bound. And we must allow fractions. The 1.2k variant does a rounding to nearest integer of half-integer, 1.2f did a truncation rather (this is done after truncation of #2 to fixed point with one digit after mark.) We try to recognize quickly the case of integer exponent, for speed, but there is overhead of going through `\xintiTrunc1`.

```

2342 \def\XINTinFloatPowerH {\romannumeral0\XINTinfloatpowerh }%

```

```

2343 \def\XINTinfloatpowerh #1#2%
2344 {%
2345   \expandafter\XINT_flpowerh_a\romannumeral0\xintitrunc1{#2};%
2346   \XINTdigits.{#1}\XINTinfloatS[\XINTdigits]}%
2347 }%

2348 \def\XINT_flpowerh_a #1;%
2349 {%
2350   \if0\xintLDg{#1}\expandafter\XINT_flpowerh_int
2351   \else\expandafter\XINT_flpowerh_b
2352   \fi #1.%
2353 }%
2354 \def\XINT_flpowerh_int #1%
2355 {%
2356   \if0#1\expandafter\XINT_flpower_BisZero
2357   \else\expandafter\XINT_flpowerh_i
2358   \fi #1%
2359 }%
2360 \def\XINT_flpowerh_i #10.{\expandafter\XINT_flpower_checkB_a#1.}%
2361 \def\XINT_flpowerh_b #1.%
2362 {%
2363   \expandafter\XINT_flpowerh_c\romannumeral0\xintdsrr{\xintDouble{#1}}.%
2364 }%
2365 \def\XINT_flpowerh_c #1.%
2366 {%
2367   \ifodd\xintLDg{#1} %<- intentional space
2368   \expandafter\XINT_flpowerh_d\else\expandafter\XINT_flpowerh_e
2369   \fi #1.%
2370 }%
2371 \def\XINT_flpowerh_d #1.\XINTdigits.#2#3%
2372 {%
2373   \XINT_flpower_checkB_a #1.\XINTdigits.{#2}\XINT_flpowerh_finish
2374 }%
2375 \def\XINT_flpowerh_finish #1%
2376   {\XINTinfloatS[\XINTdigits]{\XINTinFloatSqrt[\XINTdigits+\xint_c_iii]{#1}}}%

```


7 Package *xintfrac* implementation

```

2377 \def\XINT_flpowerh_e #1.%
2378   {\expandafter\XINT_flpower_checkB_a\romannumeral0\xinthal{#1}.}%

  Start of macro. Check for optional argument.

2379 \def\XINT_flpower_chkopt #1#2%
2380 {%
2381   \ifx [#2\expandafter\XINT_flpower_opt
2382     \else\expandafter\XINT_flpower_noopt
2383     \fi
2384   #1#2%
2385 }%
2386 \def\XINT_flpower_noopt #1#2\xint:#3%
2387 {%
2388   \expandafter\XINT_flpower_checkB_a
2389   \romannumeral0\xintnum{#3}.\XINTdigits.{#2}{#1[\XINTdigits]}%
2390 }%
2391 \def\XINT_flpower_opt #1[\xint:#2]%
2392 {%
2393   \expandafter\XINT_flpower_opt_a\the\numexpr #2.#1%
2394 }%
2395 \def\XINT_flpower_opt_a #1.#2#3#4%
2396 {%
2397   \expandafter\XINT_flpower_checkB_a
2398   \romannumeral0\xintnum{#4}.#1.{#3}{#2[#1]}%
2399 }%
2400 \def\XINT_flpower_checkB_a #1%
2401 {%
2402   \xint_UDzerominusfork
2403   #1-{\XINT_flpower_BisZero 0}%
2404   0#1{\XINT_flpower_checkB_b -}%
2405   0-{\XINT_flpower_checkB_b }{#1}%
2406   \krof
2407 }%
2408 \def\XINT_flpower_BisZero 0.#1.#2#3{#3{1[0]}}%
2409 \def\XINT_flpower_checkB_b #1#2.#3.%
2410 {%
2411   \expandafter\XINT_flpower_checkB_c
2412   \the\numexpr\xintLength{#2}+\xint_c_iii.#3.#2.{#1}%
2413 }%

2414 \def\XINT_flpower_checkB_c #1.#2.%
2415 {%
2416   \expandafter\XINT_flpower_checkB_d\the\numexpr#1+#2.#1.#2.%
2417 }%

2418 \def\XINT_flpower_checkB_d #1.#2.#3.#4.#5#6%
2419 {%
2420   \expandafter \XINT_flpower_aa
2421   \romannumeral0\XINTinfloat [#3]{#6}{#2}{#1}{#4}{#5}%
2422 }%

```

```

2423 \def\XINT_flpower_aa #1[#2]#3%
2424 {%
2425   \expandafter\XINT_flpower_ab\the\numexpr #2-#3\expandafter.%
2426   \romannumeral\XINT_rep #3\endcsname0.#1.%
2427 }%
2428 \def\XINT_flpower_ab #1.#2.#3.{\XINT_flpower_a #3#2[#1]}%
2429 \def\XINT_flpower_a #1%
2430 {%
2431   \xint_UDzerominusfork
2432   #1-\XINT_flpow_zero
2433   0#1{\XINT_flpower_b \iftrue}%
2434   0-{\XINT_flpower_b \iffalse#1}%
2435   \krof
2436 }%
2437 \def\XINT_flpower_b #1#2[#3]#4#5%
2438 {%
2439   \XINT_flpower_loopI #5.#3.#2.#4.{#1\xintiOdd{#5}\fi}%
2440 }%
2441 \def\XINT_flpower_loopI #1.%
2442 {%
2443   \if1\XINT_isOne {#1}\xint_dothis\XINT_flpower_ItoIII\fi
2444   \ifodd\xintLDg{#1} %<- intentional space
2445     \xint_dothis{\expandafter\XINT_flpower_loopI_odd}\fi
2446   \xint_orthat{\expandafter\XINT_flpower_loopI_even}%

2447   \romannumeral0\XINT_half
2448   #1\xint_bye\xint_Bye345678\xint_bye
2449   *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax.%
2450 }%
2451 \def\XINT_flpower_ItoIII #1.#2.#3.#4.#5%
2452 {%
2453   \expandafter\XINT_flpow_III\the\numexpr #5+\xint_c_.#2.#3.#4.%
2454 }%
2455 \def\XINT_flpower_loopI_even #1.#2.#3.#4.%
2456 {%
2457   \expandafter\XINT_flpower_toloopI
2458   \the\numexpr\expandafter\XINT_flpow_truncate
2459   \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#1.%
2460 }%
2461 \def\XINT_flpower_toloopI #1.#2.#3.#4.{\XINT_flpower_loopI #4.#1.#2.#3.}%
2462 \def\XINT_flpower_loopI_odd #1.#2.#3.#4.%
2463 {%
2464   \expandafter\XINT_flpower_toloopII
2465   \the\numexpr\expandafter\XINT_flpow_truncate
2466   \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.%
2467   #1.#2.#3.%
2468 }%
2469 \def\XINT_flpower_toloopII #1.#2.#3.#4.{\XINT_flpower_loopII #4.#1.#2.#3.}%
2470 \def\XINT_flpower_loopII #1.%

```

```

2471 {%
2472   \if1\XINT_isOne{#1}\xint_dothis\XINT_flpower_IItoIII\fi
2473   \ifodd\xintLDg{#1} %<- intentional space
2474   \xint_dothis{\expandafter\XINT_flpower_loopII_odd}\fi
2475   \xint_orthat{\expandafter\XINT_flpower_loopII_even}%

2476   \romannumeral0\XINT_half#1\xint_bye\xint_Bye345678\xint_bye
2477   *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax.%
2478 }%
2479 \def\XINT_flpower_loopII_even #1.#2.#3.#4.%
2480 {%
2481   \expandafter\XINT_flpower_toloopII
2482   \the\numexpr\expandafter\XINT_flpow_truncate
2483   \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#1.%
2484 }%
2485 \def\XINT_flpower_loopII_odd #1.#2.#3.#4.#5.#6.%
2486 {%
2487   \expandafter\XINT_flpower_loopII_odda
2488   \the\numexpr\expandafter\XINT_flpow_truncate
2489   \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2490   #1.#2.#3.%
2491 }%
2492 \def\XINT_flpower_loopII_odda #1.#2.#3.#4.#5.#6.%
2493 {%
2494   \expandafter\XINT_flpower_toloopII
2495   \the\numexpr\expandafter\XINT_flpow_truncate
2496   \the\numexpr\xint_c_ii*#5\expandafter.\romannumeral0\xintiisqr{#6}.#3.%
2497   #4.#1.#2.%
2498 }%
2499 \def\XINT_flpower_IItoIII #1.#2.#3.#4.#5.#6.#7%
2500 {%
2501   \expandafter\XINT_flpow_III\the\numexpr #7+\xint_c_\expandafter.%
2502   \the\numexpr\expandafter\XINT_flpow_truncate
2503   \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2504 }%

```

7.71 \xintFloatFac, \XINTFloatFac

```

2505 \def\xintFloatFac {\romannumeral0\xintfloatfac}%
2506 \def\xintfloatfac #1{\XINT_qlfac_chkopt \xintfloat #1\xint:}%
2507 \def\XINTinFloatFac {\romannumeral0\XINTinfloatfac}%
2508 \def\XINTinfloatfac #1{\XINT_qlfac_chkopt \XINTinfloat #1\xint:}%
2509 \def\XINT_qlfac_chkopt #1#2%
2510 {%
2511   \ifx [#2\expandafter\XINT_qlfac_opt
2512   \else\expandafter\XINT_qlfac_noopt
2513   \fi
2514   #1#2%
2515 }%
2516 \def\XINT_qlfac_noopt #1#2\xint:
2517 {%

```

7 Package *xintfrac* implementation

```

2518 \expandafter\XINT_FL_fac_fork_a
2519 \the\numexpr \xintNum{#2}.\xint_c_i \XINTdigits\XINT_FL_fac_out{#1[\XINTdigits]]}%
2520 }%
2521 \def\XINT_flfac_opt #1[\xint:#2]%
2522 {%
2523 \expandafter\XINT_flfac_opt_a\the\numexpr #2.#1%
2524 }%
2525 \def\XINT_flfac_opt_a #1.#2#3%
2526 {%
2527 \expandafter\XINT_FL_fac_fork_a\the\numexpr \xintNum{#3}.\xint_c_i {#1}\XINT_FL_fac_out{#2[#1]]}%
2528 }%
2529 \def\XINT_FL_fac_fork_a #1%
2530 {%
2531 \xint_UDzerominusfork
2532 #1-\XINT_FL_fac_iszero
2533 0#1\XINT_FL_fac_isneg
2534 0-\XINT_FL_fac_fork_b #1}%
2535 \krof
2536 }%
2537 \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.

```

2538 \def\XINT_FL_fac_isneg #1.#2#3#4#5%
2539 {%
2540 #5{\XINT_signalcondition{InvalidOperation}
2541 {Factorial of negative: (-#1)!}}{0[0]}}%
2542 }%
2543 \def\XINT_FL_fac_fork_b #1.%
2544 {%
2545 \ifnum #1>\xint_c_x^viii_mone\xint_dothis\XINT_FL_fac_toobig\fi
2546 \ifnum #1>\xint_c_x^iv\xint_dothis\XINT_FL_fac_vbig \fi
2547 \ifnum #1>465 \xint_dothis\XINT_FL_fac_big\fi
2548 \ifnum #1>101 \xint_dothis\XINT_FL_fac_med\fi
2549 \xint_orthat\XINT_FL_fac_small
2550 #1.%
2551 }%
2552 \def\XINT_FL_fac_toobig #1.#2#3#4#5%
2553 {%
2554 #5{\XINT_signalcondition{InvalidOperation}
2555 {Factorial of too big: (#1)!}}{0[0]}}%
2556 }%

```

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} 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 $Q\leftarrow(P+13+k)/8$. The routine actually computes $8(Q-1)$ for use in \XINT_FL_fac_addzeros.

With 1.2f the formula is $m=P+9+k$, $Q\leftarrow(P+12+k)/8$, 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.

```
2557 \def\XINT_FL_fac_vbig
2558   {\expandafter\XINT_FL_fac_vbigloop_a
2559     \the\numexpr \XINT_FL_fac_increaseP \xint_c_i }%
2560 \def\XINT_FL_fac_big
2561   {\expandafter\XINT_FL_fac_bigloop_a
2562     \the\numexpr \XINT_FL_fac_increaseP \xint_c_ii }%
2563 \def\XINT_FL_fac_med
2564   {\expandafter\XINT_FL_fac_medloop_a
2565     \the\numexpr \XINT_FL_fac_increaseP \xint_c_iii }%
2566 \def\XINT_FL_fac_small
2567   {\expandafter\XINT_FL_fac_smallloop_a
2568     \the\numexpr \XINT_FL_fac_increaseP \xint_c_iv }%
2569 \def\XINT_FL_fac_increaseP #1#2.#3#4%
2570 {%
2571   #2\expandafter.\the\numexpr\xint_c_viii*%
2572   ((\xint_c_iv+#4+\expandafter\XINT_FL_fac_countdigits
2573     \the\numexpr #2/(#1*#3)\relax 87654321\Z)/\xint_c_viii).%
2574 }%
2575 \def\XINT_FL_fac_countdigits #1#2#3#4#5#6#7#8{\XINT_FL_fac_countdone }%
2576 \def\XINT_FL_fac_countdone #1#2\Z {#1}%
2577 \def\XINT_FL_fac_out #1;![#2]#3%
2578   {#3{\romannumeral0\XINT_mul_out
2579     #1;!1\R!1\R!1\R!1\R!1\R!%
2580     1\R!1\R!1\R!1\R!\W [#2]}}%
2581 \def\XINT_FL_fac_vbigloop_a #1.#2.%
2582 {%
2583   \XINT_FL_fac_bigloop_a \xint_c_x^iv.#2.%
2584   {\expandafter\XINT_FL_fac_vbigloop_loop\the\numexpr 100010001\expandafter.%
2585     \the\numexpr \xint_c_x^viii+#1.}%
2586 }%
2587 \def\XINT_FL_fac_vbigloop_loop #1.#2.%
2588 {%
2589   \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2590   \expandafter\XINT_FL_fac_vbigloop_loop
2591   \the\numexpr #1+\xint_c_i\expandafter.%
```

7 Package *xintfrac* implementation

```

2592 \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_mul #1!%
2593 }%
2594 \def\XINT_FL_fac_bigloop_a #1.%
2595 {%
2596 \expandafter\XINT_FL_fac_bigloop_b \the\numexpr
2597 #1+\xint_c_i-\xint_c_ii*((#1-464)/\xint_c_ii).#1.%
2598 }%
2599 \def\XINT_FL_fac_bigloop_b #1.#2.#3.%
2600 {%
2601 \expandafter\XINT_FL_fac_medloop_a
2602 \the\numexpr #1-\xint_c_i.#3.{\XINT_FL_fac_bigloop_loop #1.#2.}%
2603 }%
2604 \def\XINT_FL_fac_bigloop_loop #1.#2.%
2605 {%
2606 \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2607 \expandafter\XINT_FL_fac_bigloop_loop
2608 \the\numexpr #1+\xint_c_ii\expandafter.%
2609 \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_bigloop_mul #1!%
2610 }%
2611 \def\XINT_FL_fac_bigloop_mul #1!%
2612 {%
2613 \expandafter\XINT_FL_fac_mul
2614 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2615 }%
2616 \def\XINT_FL_fac_medloop_a #1.%
2617 {%
2618 \expandafter\XINT_FL_fac_medloop_b
2619 \the\numexpr #1+\xint_c_i-\xint_c_iii*((#1-100)/\xint_c_iii).#1.%
2620 }%
2621 \def\XINT_FL_fac_medloop_b #1.#2.#3.%
2622 {%
2623 \expandafter\XINT_FL_fac_smallloop_a
2624 \the\numexpr #1-\xint_c_i.#3.{\XINT_FL_fac_medloop_loop #1.#2.}%
2625 }%
2626 \def\XINT_FL_fac_medloop_loop #1.#2.%
2627 {%
2628 \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2629 \expandafter\XINT_FL_fac_medloop_loop
2630 \the\numexpr #1+\xint_c_iii\expandafter.%
2631 \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_medloop_mul #1!%
2632 }%
2633 \def\XINT_FL_fac_medloop_mul #1!%
2634 {%
2635 \expandafter\XINT_FL_fac_mul
2636 \the\numexpr
2637 \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2638 }%
2639 \def\XINT_FL_fac_smallloop_a #1.%
2640 {%
2641 \csname
2642 XINT_FL_fac_smallloop_\the\numexpr #1-\xint_c_iv*(#1/\xint_c_iv)\relax
2643 \endcsname #1.%

```

7 Package *xintfrac* implementation

```

2644 }%
2645 \expandafter\def\csname XINT_FL_fac_smallloop_1\endcsname #1.#2.%
2646 {%
2647   \XINT_FL_fac_addzeros #2.1000000001!.{2.#1.}{#2}%
2648 }%
2649 \expandafter\def\csname XINT_FL_fac_smallloop_2\endcsname #1.#2.%
2650 {%
2651   \XINT_FL_fac_addzeros #2.1000000002!.{3.#1.}{#2}%
2652 }%
2653 \expandafter\def\csname XINT_FL_fac_smallloop_1\endcsname #1.#2.%
2654 {%
2655   \XINT_FL_fac_addzeros #2.1000000006!.{4.#1.}{#2}%
2656 }%
2657 \expandafter\def\csname XINT_FL_fac_smallloop_0\endcsname #1.#2.%
2658 {%
2659   \XINT_FL_fac_addzeros #2.1000000024!.{5.#1.}{#2}%
2660 }%
2661 \def\XINT_FL_fac_addzeros #1.%
2662 {%
2663   \ifnum #1=\xint_c_viii \expandafter\XINT_FL_fac_addzeros_exit\fi
2664   \expandafter\XINT_FL_fac_addzeros
2665   \the\numexpr #1-\xint_c_viii.1000000000!%
2666 }%

```

We will manipulate by successive **small** multiplications Q blocks $1<8d<!$, terminated by $1;!$. We need a custom small multiplication which tells us when it has create a new block, and the least significant one should be dropped.

```

2667 \def\XINT_FL_fac_addzeros_exit #1.#2.#3#4{\XINT_FL_fac_smallloop_loop #3#21;![ -#4]]%
2668 \def\XINT_FL_fac_smallloop_loop #1.#2.%
2669 {%
2670   \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2671   \expandafter\XINT_FL_fac_smallloop_loop
2672   \the\numexpr #1+\xint_c_iv\expandafter.%
2673   \the\numexpr #2\expandafter.\romannumeral0\XINT_FL_fac_smallloop_mul #1!%
2674 }%
2675 \def\XINT_FL_fac_smallloop_mul #1!%
2676 {%
2677   \expandafter\XINT_FL_fac_mul
2678   \the\numexpr
2679     \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2680 }%[[
2681 \def\XINT_FL_fac_loop_exit #1!#2]#3{#3#2]]%
2682 \def\XINT_FL_fac_mul 1#1!%
2683   {\expandafter\XINT_FL_fac_mul_a\the\numexpr\XINT_FL_fac_smallmul 10!{#1}}%
2684 \def\XINT_FL_fac_mul_a #1-#2%
2685 {%
2686   \if#21\xint_afterfi{\expandafter\space\xint_gob_til_exclam}\else
2687   \expandafter\space\fi #11;!%
2688 }%
2689 \def\XINT_FL_fac_minimulwc_a #1#2#3#4#5!#6#7#8#9%
2690 {%
2691   \XINT_FL_fac_minimulwc_b {#1#2#3#4}{#5}{#6#7#8#9}%
2692 }%

```

```

2693 \def\XINT_FL_fac_minimulwc_b #1#2#3#4!#5%
2694 {%
2695     \expandafter\XINT_FL_fac_minimulwc_c
2696     \the\numexpr \xint_c_x^ix+#5+#2*#4!{{#1}{#2}{#3}{#4}}%
2697 }%
2698 \def\XINT_FL_fac_minimulwc_c 1#1#2#3#4#5#6!#7%
2699 {%
2700     \expandafter\XINT_FL_fac_minimulwc_d {#1#2#3#4#5}#7{#6}%
2701 }%
2702 \def\XINT_FL_fac_minimulwc_d #1#2#3#4#5%
2703 {%
2704     \expandafter\XINT_FL_fac_minimulwc_e
2705     \the\numexpr \xint_c_x^ix+#1+#2*#5+#3*#4!{#2}{#4}%
2706 }%
2707 \def\XINT_FL_fac_minimulwc_e 1#1#2#3#4#5#6!#7#8#9%
2708 {%
2709     1#6#9\expandafter!%
2710     \the\numexpr\expandafter\XINT_FL_fac_smallmul
2711     \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#7*#8!%
2712 }%
2713 \def\XINT_FL_fac_smallmul 1#1!#21#3!%
2714 {%
2715     \xint_gob_til_sc #3\XINT_FL_fac_smallmul_end;%
2716     \XINT_FL_fac_minimulwc_a #2!#3!{#1}{#2}%
2717 }%

```

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.)

```

2718 \def\XINT_FL_fac_smallmul_end;\XINT_FL_fac_minimulwc_a #1!;!#2#3[#4]%
2719 {%
2720     \ifnum #2=\xint_c_
2721         \expandafter\xint_firstoftwo\else
2722         \expandafter\xint_secondoftwo
2723     \fi
2724     {-2\relax[#4]}%
2725     {1#2\expandafter!\expandafter-\expandafter1\expandafter
2726         [\the\numexpr #4+\xint_c_viii]}%
2727 }%

```

7.72 `\xintFloatPFactorial`, `\XINTinFloatPFactorial`

2015/11/29 for 1.2f. Partial factorial `pfactorial(a,b)=(a+1)...``b`, only for non-negative integers with `a<=b<10^8`.

1.2h (2016/11/20) now avoids raising `\xintError:OutOfRangePFac` if the condition `0<=a<=b<10^8` is violated. Same as for `\xintiiPFactorial`.

```

2728 \def\xintFloatPFactorial {\romannumeral0\xintfloatpfactorial}%
2729 \def\xintfloatpfactorial #1{\XINT_flpfac_chkopt \xintfloat #1\xint:}%
2730 \def\XINTinFloatPFactorial {\romannumeral0\XINTinfloatpfactorial}%
2731 \def\XINTinfloatpfactorial #1{\XINT_flpfac_chkopt \XINTinfloat #1\xint:}%
2732 \def\XINT_flpfac_chkopt #1#2%

```



```

2733 {%
2734   \ifx [#2\expandafter\XINT_flpfac_opt
2735     \else\expandafter\XINT_flpfac_noopt
2736   \fi
2737   #1#2%
2738 }%
2739 \def\XINT_flpfac_noopt #1#2\xint:#3%
2740 {%
2741   \expandafter\XINT_FL_pfac_fork
2742   \the\numexpr \xintNum{#2}\expandafter.%
2743   \the\numexpr \xintNum{#3}.\xint_c_i{\XINTdigits}{#1[\XINTdigits]]}%
2744 }%
2745 \def\XINT_flpfac_opt #1[\xint:#2]%
2746 {%
2747   \expandafter\XINT_flpfac_opt_b\the\numexpr #2.#1%
2748 }%
2749 \def\XINT_flpfac_opt_b #1.#2#3#4%
2750 {%
2751   \expandafter\XINT_FL_pfac_fork
2752   \the\numexpr \xintNum{#3}\expandafter.%
2753   \the\numexpr \xintNum{#4}.\xint_c_i{#1}{#2[#1]}%
2754 }%
2755 \def\XINT_FL_pfac_fork #1#2.#3#4.%
2756 {%
2757   \unless\ifnum #1#2<#3#4 \xint_dothis\XINT_FL_pfac_one\fi
2758   \if-#3\xint_dothis\XINT_FL_pfac_neg \fi
2759   \if-#1\xint_dothis\XINT_FL_pfac_zero\fi
2760   \ifnum #3#4>\xint_c_x^viii_mone\xint_dothis\XINT_FL_pfac_outofrange\fi
2761   \xint_orthat \XINT_FL_pfac_increaseP #1#2.#3#4.%
2762 }%
2763 \def\XINT_FL_pfac_outofrange #1.#2.#3#4#5%
2764 {%
2765   #5{\XINT_signalcondition{InvalidOperation}
2766     {pfactorial second arg too big: 99999999 < #2}}{0[0]}}%
2767 }%
2768 \def\XINT_FL_pfac_one #1.#2.#3#4#5{#5{1[0]}}%
2769 \def\XINT_FL_pfac_zero #1.#2.#3#4#5{#5{0[0]}}%
2770 \def\XINT_FL_pfac_neg -#1.-#2.%
2771 {%
2772   \ifnum #1>\xint_c_x^viii\xint_dothis\XINT_FL_pfac_outofrange\fi
2773   \xint_orthat {%
2774     \ifodd\numexpr#2-#1\relax\xint_afterfi{\expandafter-\romannumeral`&&@}\fi
2775     \expandafter\XINT_FL_pfac_increaseP}%
2776   \the\numexpr #2-\xint_c_i\expandafter.\the\numexpr#1-\xint_c_i.%
2777 }%

```

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.

7 Package *xintfrac* implementation

```

2778 \def\XINT_FL_pfac_increaseP #1.#2.#3#4%
2779 {%
2780   \expandafter\XINT_FL_pfac_a
2781   \the\numexpr \xint_c_viii*((\xint_c_iv+#4+\expandafter
2782     \XINT_FL_fac_countdigits\the\numexpr (#2-#1-\xint_c_i)%
2783     /\ifnum #2>\xint_c_x^iv #3\else(#3*\xint_c_ii)\fi\relax
2784     87654321\Z)/\xint_c_viii).#1.#2.%
2785 }%
2786 \def\XINT_FL_pfac_a #1.#2.#3.%
2787 {%
2788   \expandafter\XINT_FL_pfac_b\the\numexpr \xint_c_i+#2\expandafter.%
2789   \the\numexpr#3\expandafter.%
2790   \romannumeral0\XINT_FL_pfac_addzeroes #1.10000000011;![ -#1]%
2791 }%
2792 \def\XINT_FL_pfac_addzeroes #1.%
2793 {%
2794   \ifnum #1=\xint_c_viii \expandafter\XINT_FL_pfac_addzeroes_exit\fi
2795   \expandafter\XINT_FL_pfac_addzeroes\the\numexpr #1-\xint_c_viii.1000000000!%
2796 }%
2797 \def\XINT_FL_pfac_addzeroes_exit #1.{ }%
2798 \def\XINT_FL_pfac_b #1.%
2799 {%
2800   \ifnum #1>9999 \xint_dothis\XINT_FL_pfac_vbigloop \fi
2801   \ifnum #1>463 \xint_dothis\XINT_FL_pfac_bigloop \fi
2802   \ifnum #1>98 \xint_dothis\XINT_FL_pfac_medloop \fi
2803   \xint_orthat\XINT_FL_pfac_smallloop #1.%
2804 }%
2805 \def\XINT_FL_pfac_smallloop #1.#2.%
2806 {%
2807   \ifcase\numexpr #2-#1\relax
2808     \expandafter\XINT_FL_pfac_end_
2809   \or \expandafter\XINT_FL_pfac_end_i
2810   \or \expandafter\XINT_FL_pfac_end_ii
2811   \or \expandafter\XINT_FL_pfac_end_iii
2812   \else\expandafter\XINT_FL_pfac_smallloop_a
2813   \fi #1.#2.%
2814 }%
2815 \def\XINT_FL_pfac_smallloop_a #1.#2.%
2816 {%
2817   \expandafter\XINT_FL_pfac_smallloop_b
2818   \the\numexpr #1+\xint_c_iv\expandafter.%
2819   \the\numexpr #2\expandafter.%
2820   \romannumeral0\expandafter\XINT_FL_fac_mul
2821   \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2822 }%
2823 \def\XINT_FL_pfac_smallloop_b #1.%
2824 {%
2825   \ifnum #1>98 \expandafter\XINT_FL_pfac_medloop \else
2826     \expandafter\XINT_FL_pfac_smallloop \fi #1.%
2827 }%
2828 \def\XINT_FL_pfac_medloop #1.#2.%
2829 {%

```

7 Package *xintfrac* implementation

```

2830 \ifcase\numexpr #2-#1\relax
2831 \expandafter\XINT_FL_pfac_end_
2832 \or \expandafter\XINT_FL_pfac_end_i
2833 \or \expandafter\XINT_FL_pfac_end_ii
2834 \else\expandafter\XINT_FL_pfac_medloop_a
2835 \fi #1.#2.%
2836 }%
2837 \def\XINT_FL_pfac_medloop_a #1.#2.%
2838 {%
2839 \expandafter\XINT_FL_pfac_medloop_b
2840 \the\numexpr #1+\xint_c_iii\expandafter.%
2841 \the\numexpr #2\expandafter.%
2842 \romannumeral0\expandafter\XINT_FL_fac_mul
2843 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2844 }%
2845 \def\XINT_FL_pfac_medloop_b #1.%
2846 {%
2847 \ifnum #1>463 \expandafter\XINT_FL_pfac_bigloop \else
2848 \expandafter\XINT_FL_pfac_medloop \fi #1.%
2849 }%
2850 \def\XINT_FL_pfac_bigloop #1.#2.%
2851 {%
2852 \ifcase\numexpr #2-#1\relax
2853 \expandafter\XINT_FL_pfac_end_
2854 \or \expandafter\XINT_FL_pfac_end_i
2855 \else\expandafter\XINT_FL_pfac_bigloop_a
2856 \fi #1.#2.%
2857 }%
2858 \def\XINT_FL_pfac_bigloop_a #1.#2.%
2859 {%
2860 \expandafter\XINT_FL_pfac_bigloop_b
2861 \the\numexpr #1+\xint_c_ii\expandafter.%
2862 \the\numexpr #2\expandafter.%
2863 \romannumeral0\expandafter\XINT_FL_fac_mul
2864 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2865 }%
2866 \def\XINT_FL_pfac_bigloop_b #1.%
2867 {%
2868 \ifnum #1>9999 \expandafter\XINT_FL_pfac_vbigloop \else
2869 \expandafter\XINT_FL_pfac_bigloop \fi #1.%
2870 }%
2871 \def\XINT_FL_pfac_vbigloop #1.#2.%
2872 {%
2873 \ifnum #2=#1
2874 \expandafter\XINT_FL_pfac_end_
2875 \else\expandafter\XINT_FL_pfac_vbigloop_a
2876 \fi #1.#2.%
2877 }%
2878 \def\XINT_FL_pfac_vbigloop_a #1.#2.%
2879 {%
2880 \expandafter\XINT_FL_pfac_vbigloop
2881 \the\numexpr #1+\xint_c_i\expandafter.%

```

```

2882 \the\numexpr #2\expandafter.%
2883 \romannumeral0\expandafter\XINT_FL_fac_mul
2884 \the\numexpr\xint_c_x^viii+#1!%
2885 }%
2886 \def\XINT_FL_pfac_end_iii #1.#2.%
2887 {%
2888 \expandafter\XINT_FL_fac_out
2889 \romannumeral0\expandafter\XINT_FL_fac_mul
2890 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2891 }%
2892 \def\XINT_FL_pfac_end_ii #1.#2.%
2893 {%
2894 \expandafter\XINT_FL_fac_out
2895 \romannumeral0\expandafter\XINT_FL_fac_mul
2896 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2897 }%
2898 \def\XINT_FL_pfac_end_i #1.#2.%
2899 {%
2900 \expandafter\XINT_FL_fac_out
2901 \romannumeral0\expandafter\XINT_FL_fac_mul
2902 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2903 }%
2904 \def\XINT_FL_pfac_end_ #1.#2.%
2905 {%
2906 \expandafter\XINT_FL_fac_out
2907 \romannumeral0\expandafter\XINT_FL_fac_mul
2908 \the\numexpr \xint_c_x^viii+#1!%
2909 }%

```

7.73 \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.

```

2910 \def\xintFloatBinomial {\romannumeral0\xintfloatbinomial}%
2911 \def\xintfloatbinomial #1{\XINT_flbinom_chkopt \xintfloat #1\xint:}%
2912 \def\XINTinFloatBinomial {\romannumeral0\XINTinfloatbinomial}%
2913 \def\XINTinfloatbinomial #1{\XINT_flbinom_chkopt \XINTinfloat #1\xint:}%
2914 \def\XINT_flbinom_chkopt #1#2%
2915 {%
2916 \ifx [#2\expandafter\XINT_flbinom_opt
2917 \else\expandafter\XINT_flbinom_noopt
2918 \fi #1#2%
2919 }%
2920 \def\XINT_flbinom_noopt #1#2\xint:#3%
2921 {%
2922 \expandafter\XINT_FL_binom_a

```

7 Package *xintfrac* implementation

```

2923 \the\numexpr\xintNum{#2}\expandafter.\the\numexpr\xintNum{#3}.\XINTdigits.#1%
2924 }%
2925 \def\XINT_flbinom_opt #1[\xint:#2]#3#4%
2926 {%
2927 \expandafter\XINT_FL_binom_a
2928 \the\numexpr\xintNum{#3}\expandafter.\the\numexpr\xintNum{#4}\expandafter.%
2929 \the\numexpr #2.#1%
2930 }%
2931 \def\XINT_FL_binom_a #1.#2.%
2932 {%
2933 \expandafter\XINT_FL_binom_fork \the\numexpr #1-#2.#2.#1.%
2934 }%
2935 \def\XINT_FL_binom_fork #1#2.#3#4.#5#6.%
2936 {%
2937 \if-#5\xint_dothis \XINT_FL_binom_neg\fi
2938 \if-#1\xint_dothis \XINT_FL_binom_zero\fi
2939 \if-#3\xint_dothis \XINT_FL_binom_zero\fi
2940 \if0#1\xint_dothis \XINT_FL_binom_one\fi
2941 \if0#3\xint_dothis \XINT_FL_binom_one\fi
2942 \ifnum #5#6>\xint_c_x^viii_mone \xint_dothis\XINT_FL_binom_toobig\fi
2943 \ifnum #1#2>#3#4 \xint_dothis\XINT_FL_binom_ab \fi
2944 \xint_orthat\XINT_FL_binom_aa
2945 #1#2.#3#4.#5#6.%
2946 }%
2947 \def\XINT_FL_binom_neg #1.#2.#3.#4.#5%
2948 {%
2949 #5[#4]{\XINT_signalcondition{InvalidOperation}
2950 {binomial with first arg negative: #3}}{0[0]}}%
2951 }%
2952 \def\XINT_FL_binom_toobig #1.#2.#3.#4.#5%
2953 {%
2954 #5[#4]{\XINT_signalcondition{InvalidOperation}
2955 {binomial with first arg too big: 99999999 < #3}}{0[0]}}%
2956 }%
2957 \def\XINT_FL_binom_one #1.#2.#3.#4.#5{#5[#4]{1[0]}}%
2958 \def\XINT_FL_binom_zero #1.#2.#3.#4.#5{#5[#4]{0[0]}}%
2959 \def\XINT_FL_binom_aa #1.#2.#3.#4.#5%
2960 {%
2961 #5[#4]{\xintDiv{\XINT_FL_pfac_increaseP
2962 #2.#3.\xint_c_iv{#4+\xint_c_i}{\XINTinfloat[#4+\xint_c_iii]}}%
2963 {\XINT_FL_fac_fork_b
2964 #1.\xint_c_iv{#4+\xint_c_i}\XINT_FL_fac_out{\XINTinfloat[#4+\xint_c_iii]}}}%
2965 }%
2966 \def\XINT_FL_binom_ab #1.#2.#3.#4.#5%
2967 {%
2968 #5[#4]{\xintDiv{\XINT_FL_pfac_increaseP
2969 #1.#3.\xint_c_iv{#4+\xint_c_i}{\XINTinfloat[#4+\xint_c_iii]}}%
2970 {\XINT_FL_fac_fork_b
2971 #2.\xint_c_iv{#4+\xint_c_i}\XINT_FL_fac_out{\XINTinfloat[#4+\xint_c_iii]}}}%
2972 }%

```

7.74 `\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.

```

2973 \def\xintFloatSqrt      {\romannumeral0\xintfloatsqr }%
2974 \def\xintfloatsqr      #1{\XINT_flsqrt_chkopt \xintfloat #1\xint:}%
2975 \def\XINTinFloatSqrt    {\romannumeral0\XINTinfloatsqr }%
2976 \def\XINTinfloatsqr    #1{\XINT_flsqrt_chkopt \XINTinfloat #1\xint:}%
2977 \def\XINT_flsqrt_chkopt #1#2%
2978 {%
2979   \ifx [#2\expandafter\XINT_flsqrt_opt
2980     \else\expandafter\XINT_flsqrt_noopt
2981     \fi #1#2%
2982 }%
2983 \def\XINT_flsqrt_noopt #1#2\xint:%
2984 {%
2985   \expandafter\XINT_FL_sqrt_a
2986     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.#1%
2987 }%
2988 \def\XINT_flsqrt_opt #1[\xint:#2]#3%
2989 {%
2990   \expandafter\XINT_flsqrt_opt_a\the\numexpr #2.#1%
2991 }%
2992 \def\XINT_flsqrt_opt_a #1.#2#3%
2993 {%
2994   \expandafter\XINT_FL_sqrt_a\romannumeral0\XINTinfloat[#1]{#3}#1.#2%
2995 }%
2996 \def\XINT_FL_sqrt_a #1%
2997 {%
2998   \xint_UDzerominusfork
2999   #1-\XINT_FL_sqrt_iszero
3000   0#1\XINT_FL_sqrt_isneg
3001   0-{\XINT_FL_sqrt_pos #1}%
3002   \krof
3003 }%[
3004 \def\XINT_FL_sqrt_iszero #1]#2.#3{#3[#2]{0[0]}}%
3005 \def\XINT_FL_sqrt_isneg #1]#2.#3%
3006 {%
3007   #3[#2]{\XINT_signalcondition{InvalidOperation}
3008     {Square root of negative: -#1}}{0[0]}}%
3009 }%

```

7 Package *xintfrac* implementation

```

3010 \def\XINT_FL_sqrt_pos #1[#2]#3.%
3011 {%
3012   \expandafter\XINT_flsqrt
3013   \the\numexpr #3\ifodd #2 \xint_dothis {+\xint_c_iii.(#2+\xint_c_i).0}\fi
3014   \xint_orthat {+\xint_c_ii.#2.{}#100.#3.%
3015 }%

3016 \def\XINT_flsqrt #1.#2.%
3017 {%
3018   \expandafter\XINT_flsqrt_a
3019   \the\numexpr #2/\xint_c_ii-(#1-\xint_c_i)/\xint_c_ii.#1.%
3020 }%

3021 \def\XINT_flsqrt_a #1.#2.#3#4.#5.%
3022 {%
3023   \expandafter\XINT_flsqrt_b
3024   \the\numexpr (#2-\xint_c_i)/\xint_c_ii\expandafter.%
3025   \romannumeral0\XINT_sqrt_start #2.#4#3.#5.#2.#4#3.#5.#1.%
3026 }%

3027 \def\XINT_flsqrt_b #1.#2#3%
3028 {%
3029   \expandafter\XINT_flsqrt_c
3030   \romannumeral0\xintiisub
3031   {\XINT_dsx_addzeros {#1}#2;}%
3032   {\xintiiDivRound{\XINT_dsx_addzeros {#1}#3;}%
3033     {\XINT_dbl#2\xint_bye2345678\xint_bye*\xint_c_ii\relax}}.%
3034 }%

3035 \def\XINT_flsqrt_c #1.#2.%
3036 {%
3037   \expandafter\XINT_flsqrt_d
3038   \romannumeral0\XINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..%
3039 }%

3040 \def\XINT_flsqrt_d #1.#2#3.%
3041 {%
3042   \ifnum #2=\xint_c_v
3043   \expandafter\XINT_flsqrt_f\else\expandafter\XINT_flsqrt_finish\fi
3044   #2#3.#1.%
3045 }%

3046 \def\XINT_flsqrt_finish #1#2.#3.#4.#5.#6.#7.#8{#8[#6]{#3#1[#7]}}%

```

```

3047 \def\XINT_flsqrt_f 5#1.%
3048   {\expandafter\XINT_flsqrt_g\romannumeral0\xintinum{#1}\relax.}%
3049 \def\XINT_flsqrt_g #1#2#3.{\if\relax#2\xint_dothis{\XINT_flsqrt_h #1}\fi
3050   \xint_orthat{\XINT_flsqrt_finish 5.}}%
3051 \def\XINT_flsqrt_h #1{\ifnum #1<\xint_c_iii\xint_dothis{\XINT_flsqrt_again}\fi
3052   \xint_orthat{\XINT_flsqrt_finish 5.}}%

```

```

3053 \def\XINT_flsqrt_again #1.#2.%
3054 {%
3055   \expandafter\XINT_flsqrt_again_a\the\numexpr #2+\xint_c_viii.%
3056 }%

```

```

3057 \def\XINT_flsqrt_again_a #1.#2.#3.%
3058 {%
3059   \expandafter\XINT_flsqrt_b
3060   \the\numexpr (#1-\xint_c_i)/\xint_c_ii\expandafter.%
3061   \romannumeral0\XINT_sqrt_start #1.#2000000000.#3.%
3062   #1.#2000000000.#3.%
3063 }%

```

7.75 \xintFloatE, \XINTinFloatE

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

1.2k had to rewrite this since there is no more a \XINT_float_a macro. Attention about \XINTinFloatE: it is for use by xintexpr.sty, contrarily to other \XINTinFloat<foo> macros it inserts itself the [\XINTdigits] thing, and with value 0 it produces on output 0[N], not 0[0].

```

3064 \def\xintFloatE   {\romannumeral0\xintfloate }%
3065 \def\xintfloate #1{\XINT_floate_chkopt #1\xint:}%
3066 \def\XINT_floate_chkopt #1%
3067 {%
3068   \ifx [#1\expandafter\XINT_floate_opt
3069     \else\expandafter\XINT_floate_noopt
3070   \fi #1%
3071 }%
3072 \def\XINT_floate_noopt #1\xint:%
3073 {%
3074   \expandafter\XINT_floate_post
3075   \romannumeral0\XINTinfloat[\XINTdigits]{#1}\XINTdigits.%
3076 }%
3077 \def\XINT_floate_opt [\xint:#1]%
3078 {%
3079   \expandafter\XINT_floate_opt_a\the\numexpr #1.%
3080 }%
3081 \def\XINT_floate_opt_a #1.#2%
3082 {%
3083   \expandafter\XINT_floate_post
3084   \romannumeral0\XINTinfloat[#1]{#2}#1.%

```



```

3085 }%
3086 \def\XINT_floate_post #1%
3087 {%
3088   \xint_UDzerominusfork
3089   #1-\XINT_floate_zero
3090   0#1\XINT_floate_neg
3091   0-\XINT_floate_pos
3092   \krof #1%
3093 }%[
3094 \def\XINT_floate_zero #1]#2.#3{ 0.e0}%
3095 \def\XINT_floate_neg-{\expandafter-\romannumeral0\XINT_floate_pos}%

3096 \def\XINT_floate_pos #1#2[#3]#4.#5%
3097 {%
3098   \expandafter\XINT_float_pos_done\the\numexpr#3+#4+#5-\xint_c_i.#1.#2;%
3099 }%
3100 \def\XINTinFloatE {\romannumeral0\XINTinfloate }%
3101 \def\XINTinfloate
3102   {\expandafter\XINT_infloate\romannumeral0\XINTinfloat[\XINTdigits]]}%
3103 \def\XINT_infloate #1[#2]#3%
3104   {\expandafter\XINT_infloate_end\the\numexpr #3+#2.{#1}}%
3105 \def\XINT_infloate_end #1.#2{ #2[#1]}%

```

7.76 \XINTinFloatMod

1.1. Pour emploi dans *xintexpr*. Code shortened at 1.2p.

```

3106 \def\XINTinFloatMod {\romannumeral0\XINTinfloatmod [\XINTdigits]]}%
3107 \def\XINTinfloatmod [#1]#2#3%
3108 {%
3109   \XINTinfloat[#1]{\xintMod
3110     {\romannumeral0\XINTinfloat[#1]{#2}}}%
3111     {\romannumeral0\XINTinfloat[#1]{#3}}}%
3112 }%

```

7.77 \XINTinFloatDivFloor

1.2p. Formerly // and /: in *\xintfloatexpr* used *\xintDivFloor* and *\xintMod*, hence did not round their operands to float precision beforehand.

```

3113 \def\XINTinFloatDivFloor {\romannumeral0\XINTinfloatdivfloor [\XINTdigits]]}%
3114 \def\XINTinfloatdivfloor [#1]#2#3%
3115 {%
3116   \xintdivfloor
3117     {\romannumeral0\XINTinfloat[#1]{#2}}}%
3118     {\romannumeral0\XINTinfloat[#1]{#3}}}%
3119 }%

```

7.78 \backslash XINTinFloatDivMod

1.2p. Pour emploi dans *xintexpr*, donc je ne prends pas la peine de faire l'expansion du modulo, qui se produira dans le \backslash csname.

Hésitation sur le quotient, faut-il l'arrondir immédiatement ? Finalement non, le produire comme un integer.

```

3120 \def\XINTinFloatDivMod {\romannumeral0\XINTinfloatdivmod [\XINTdigits]]}%
3121 \def\XINTinfloatdivmod [#1]#2#3%
3122 {%
3123   \expandafter\XINT_infloatdivmod
3124   \romannumeral0\xintdivmod
3125       {\romannumeral0\XINTinfloat[#1]{#2}}}%
3126       {\romannumeral0\XINTinfloat[#1]{#3}}}%
3127   {#1}%
3128 }%
3129 \def\XINT_infloatdivmod #1#2#3{ #1,\XINTinFloat[#3]{#2}}%
```

At End of \LaTeX Document deprecation message

1.2o

```

3130 \ifdefined\documentclass\ifdefined\AtEndDocument
3131   \AtEndDocument{%
3132 \XINT_ifFlagRaised{Deprecated-xintfrac}%
3133   {\PackageError{xintfrac}
3134 {Usage of deprecated macros!}
3135 {This deprecated macro from xintfrac.sty has been detected:\MessageBreak
3136 %\XINT_ifFlagRaised{xintNeq}}{%
3137 \string\xintNeq\space(use \string\xintNotEq\space or xint's \string\xintiiNotEq!)\MessageBreak
3138 %}{}}%
3139 It will get removed at some future release.}}%
3140   {% no deprecated macro used (at top level...)
3141   }%
3142 }\fi\fi
3143 \XINT_restorecatcodes_endinput%
```

8 Package *xintseries* implementation

.1	Catcodes, ε -TeX and reload detection . . .	259	.7	<code>\xintRationalSeries</code>	262
.2	Package identification	260	.8	<code>\xintRationalSeriesX</code>	263
.3	<code>\xintSeries</code>	260	.9	<code>\xintFxpPtPowerSeries</code>	264
.4	<code>\xintiSeries</code>	260	.10	<code>\xintFxpPtPowerSeriesX</code>	265
.5	<code>\xintPowerSeries</code>	261	.11	<code>\xintFloatPowerSeries</code>	265
.6	<code>\xintPowerSeriesX</code>	262	.12	<code>\xintFloatPowerSeriesX</code>	267

The commenting is currently (2018/02/06) 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 [2018/02/06 1.2q 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,rationfunction,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:}%
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:#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:#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:}%
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:#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:#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 . . .	268	.16	<code>\xintiGctoF</code>	280
.2	Package identification	269	.17	<code>\xintCtoCv</code> , <code>\xintCstoCv</code>	281
.3	<code>\xintCFrac</code>	269	.18	<code>\xintiCstoCv</code>	282
.4	<code>\xintGCFrac</code>	270	.19	<code>\xintGctoCv</code>	282
.5	<code>\xintGGCFrac</code>	272	.20	<code>\xintiGctoCv</code>	284
.6	<code>\xintGctoGCx</code>	273	.21	<code>\xintFtoCv</code>	285
.7	<code>\xintFtoCs</code>	273	.22	<code>\xintFtoCCv</code>	285
.8	<code>\xintFtoCx</code>	274	.23	<code>\xintCntoF</code>	285
.9	<code>\xintFtoC</code>	274	.24	<code>\xintGcntoF</code>	286
.10	<code>\xintFtoGC</code>	275	.25	<code>\xintCntoCs</code>	287
.11	<code>\xintFGtoC</code>	275	.26	<code>\xintCntoGC</code>	287
.12	<code>\xintFtoCC</code>	276	.27	<code>\xintGcntoGC</code>	288
.13	<code>\xintCtoF</code> , <code>\xintCstoF</code>	277	.28	<code>\xintCstoGC</code>	289
.14	<code>\xintiCstoF</code>	278	.29	<code>\xintGctoGC</code>	289
.15	<code>\xintGctoF</code>	278			

The commenting is currently (2018/02/06) 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`.

There is partial dependency on *xinttools* due to `\xintCstoF` and `\xintCsToCv`.

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 [2018/02/06 1.2q 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:
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:
57 {%
58   \expandafter\XINT_cfrac_A\romannumeral0\xintrawwithzeros {#1}\Z
59   \relax\relax
60 }%
61 \def\XINT_cfrac_opt_b\fi\XINT_cfrac_noopt [\xint:#1]%
62 {%
63   \fi\csname XINT_cfrac_opt#1\endcsname
64 }%
65 \def\XINT_cfrac_optl #1%
66 {%
67   \expandafter\XINT_cfrac_A\romannumeral0\xintrawwithzeros {#1}\Z
68   \relax\hfill
69 }%
70 \def\XINT_cfrac_optc #1%
71 {%
72   \expandafter\XINT_cfrac_A\romannumeral0\xintrawwithzeros {#1}\Z

```

```

73   \relax\relax
74 }%
75 \def\XINT_cfrac_optr #1%
76 {%
77   \expandafter\XINT_cfrac_A\romannumeral0\xintrawithzeros {#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:}%
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:%
128 {%
129   \XINT_gcfrac #1+!\relax\relax
130 }%
131 \def\XINT_gcfrac_opt_b\fi\XINT_gcfrac_noopt [\xint:#1]%
132 {%
133   \fi\csname XINT_gcfrac_opt#1\endcsname
134 }%
135 \def\XINT_gcfrac_optl #1%
136 {%
137   \XINT_gcfrac #1+!\relax\hfill
138 }%
139 \def\XINT_gcfrac_optc #1%
140 {%
141   \XINT_gcfrac #1+!\relax\relax
142 }%
143 \def\XINT_gcfrac_optr #1%
144 {%
145   \XINT_gcfrac #1+!\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_exclam #3\XINT_gcfrac_endloop!%
155   \XINT_gcfrac_loop {{#3}{#2}{#1}}%
156 }%
157 \def\XINT_gcfrac_endloop!\XINT_gcfrac_loop #1#2#3%
158 {%
159   \XINT_gcfrac_T #2#3#1!!%
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_exclam #5\XINT_gcfrac_end!\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_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:}%
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:
182 {%
183   \XINT_ggcfrac #1+!\relax\relax
184 }%
185 \def\XINT_ggcfrac_opt_b\fi\XINT_ggcfrac_noopt [\xint:#1]%
186 {%
187   \fi\csname XINT_ggcfrac_opt#1\endcsname
188 }%
189 \def\XINT_ggcfrac_optl #1%
190 {%
191   \XINT_ggcfrac #1+!\relax\hfill
192 }%
193 \def\XINT_ggcfrac_optc #1%
194 {%
195   \XINT_ggcfrac #1+!\relax\relax
196 }%
197 \def\XINT_ggcfrac_optr #1%
198 {%
199   \XINT_ggcfrac #1+!\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_exclam #3\XINT_ggcfrac_endloop!%
209   \XINT_ggcfrac_loop {{#3}{#2}{#1}}%
210 }%
211 \def\XINT_ggcfrac_endloop!\XINT_ggcfrac_loop #1#2#3%
212 {%
213   \XINT_ggcfrac_T #2#3#1!!%
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_exclam #5\XINT_ggcfrac_end!\XINT_ggcfrac_U
219     #1#2{#5+\cfrac{#1#4#2}{#3}}%
220 }%
221 \def\XINT_ggcfrac_end!\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+!/}%
232 \def\XINT_gctgcx_loop_a #1#2#3#4+#5/%
233 {%
234   \xint_gob_til_exclam #5\XINT_gctgcx_end!%
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_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\xintraawithzeros {#1}\Z
246 }%
247 \def\XINT_ftc_A #1/#2\Z
248 {%
249   \expandafter\XINT_ftc_B\romannumeral0\xintiidivision {#1}{#2}{#2}%
250 }%
251 \def\XINT_ftc_B #1#2%
252 {%
253   \XINT_ftc_C #2.{#1}%
254 }%
255 \def\XINT_ftc_C #1%
256 {%
257   \xint_gob_til_zero #1\XINT_ftc_integer 0\XINT_ftc_D #1%
258 }%
259 \def\XINT_ftc_integer 0\XINT_ftc_D 0#1.#2#3{ #2}%
260 \def\XINT_ftc_D #1.#2#3{\XINT_ftc_loop_a {#1}{#3}{#1}{#2, }}% 1.09m adds a space
261 \def\XINT_ftc_loop_a
262 {%
263   \expandafter\XINT_ftc_loop_d\romannumeral0\XINT_div_prepare
264 }%
265 \def\XINT_ftc_loop_d #1#2%
266 {%
267   \XINT_ftc_loop_e #2.{#1}%
268 }%
269 \def\XINT_ftc_loop_e #1%

```

```

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

```

9.8 \xintFtoCx

```

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

```

9.9 \xintFtoC

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

```

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

```

9.10 `\xintFtoGC`

```

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

```

9.11 `\xintFGtoC`

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

```

318 \def\xintFGtoC {\romannumeral0\xintfgtoc}%
319 \def\xintfgtoc#1%
320 {%
321   \expandafter\XINT_fgtc_a\romannumeral0\xintraewithzeros {#1}\Z
322 }%
323 \def\XINT_fgtc_a #1/#2\Z #3%
324 {%
325   \expandafter\XINT_fgtc_b\romannumeral0\xintraewithzeros {#3}\Z #1/#2\Z { }%
326 }%
327 \def\XINT_fgtc_b #1/#2\Z
328 {%
329   \expandafter\XINT_fgtc_c\romannumeral0\xintiidivision {#1}{#2}{#2}%
330 }%
331 \def\XINT_fgtc_c #1#2#3#4/#5\Z
332 {%
333   \expandafter\XINT_fgtc_d\romannumeral0\xintiidivision
334     {#4}{#5}{#5}{#1}{#2}{#3}%
335 }%
336 \def\XINT_fgtc_d #1#2#3#4#5#6#7%
337 {%
338   \xintifEq {#1}{#4}{\XINT_fgtc_da {#1}{#2}{#3}{#4}}%
339     {\xint_thirdofthree}%
340 }%
341 \def\XINT_fgtc_da #1#2#3#4#5#6#7%
342 {%
343   \XINT_fgtc_e {#2}{#5}{#3}{#6}{#7}{#1}}%
344 }%
345 \def\XINT_fgtc_e #1%
346 {%
347   \xintiiifZero {#1}{\expandafter\xint_firstofone\xint_gobble_iii}%
348     {\XINT_fgtc_f {#1}}%
349 }%
350 \def\XINT_fgtc_f #1#2%
351 {%
352   \xintiiifZero {#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}!%
438 }%
439 \def\xintCtoF {\romannumeral0\xintctof }%
440 \def\xintctof #1%
441 {%
442   \expandafter\XINT_ctf_prep \romannumeral`&&@#1!%
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_exclam #5\XINT_ctf_end!%
451   \expandafter\XINT_ctf_loop_b
452   \romannumeral0\xintrawwithzeros {#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\xint:#4\xint:}%
458 {\romannumeral0\XINT_mul_fork #2\xint:#3\xint:}%
459 {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#6\xint:}%
460 {\XINT_mul_fork #1\xint:#4\xint:}}%
461 {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#5\xint:}%
462 {\XINT_mul_fork #1\xint:#3\xint:}}%
463 }%
464 \def\XINT_ctf_loop_c #1#2%
465 {%
466 \expandafter\XINT_ctf_loop_d\expandafter {\expandafter{#2}{#1}}%
467 }%
468 \def\XINT_ctf_loop_d #1#2%
469 {%
470 \expandafter\XINT_ctf_loop_e\expandafter {\expandafter{#2}#1}%
471 }%
472 \def\XINT_ctf_loop_e #1#2%
473 {%
474 \expandafter\XINT_ctf_loop_a\expandafter{#2}#1%
475 }%
476 \def\XINT_ctf_end #1.#2#3#4#5{\xintrawithzeros {#2/#3}}% 1.09b removes [0]

```

9.14 \xintiCstof

```

477 \def\xintiCstof {\romannumeral0\xinticstof }%
478 \def\xinticstof #1%
479 {%
480 \expandafter\XINT_icstf_prep \romannumeral`&&@#1,!,%
481 }%
482 \def\XINT_icstf_prep
483 {%
484 \XINT_icstf_loop_a 1001%
485 }%
486 \def\XINT_icstf_loop_a #1#2#3#4#5,%
487 {%
488 \xint_gob_til_exclam #5\XINT_icstf_end!%
489 \expandafter
490 \XINT_icstf_loop_b \romannumeral`&&@#5.{#1}{#2}{#3}{#4}%
491 }%
492 \def\XINT_icstf_loop_b #1.#2#3#4#5%
493 {%
494 \expandafter\XINT_icstf_loop_c\expandafter
495 {\romannumeral0\xintiiadd {#5}{\XINT_mul_fork #1\xint:#3\xint:}}%
496 {\romannumeral0\xintiiadd {#4}{\XINT_mul_fork #1\xint:#2\xint:}}%
497 {#2}{#3}%
498 }%
499 \def\XINT_icstf_loop_c #1#2%
500 {%
501 \expandafter\XINT_icstf_loop_a\expandafter {#2}{#1}%
502 }%
503 \def\XINT_icstf_end#1.#2#3#4#5{\xintrawithzeros {#2/#3}}% 1.09b removes [0]

```

9.15 \xintGctof

```

504 \def\xintGctoF {\romannumeral0\xintgctof }%
505 \def\xintgctof #1%
506 {%
507   \expandafter\xINT_gctf_prep \romannumeral`&&@#1+!/ %
508 }%
509 \def\xINT_gctf_prep
510 {%
511   \XINT_gctf_loop_a 1001%
512 }%
513 \def\xINT_gctf_loop_a #1#2#3#4#5+%
514 {%
515   \expandafter\xINT_gctf_loop_b
516   \romannumeral0\xintraewithzeros {#5}.{#1}{#2}{#3}{#4}%
517 }%
518 \def\xINT_gctf_loop_b #1/#2.#3#4#5#6%
519 {%
520   \expandafter\xINT_gctf_loop_c\expandafter
521   {\romannumeral0\xINT_mul_fork #2\xint:#4\xint:}%
522   {\romannumeral0\xINT_mul_fork #2\xint:#3\xint:}%
523   {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#6\xint:}%
524     {\XINT_mul_fork #1\xint:#4\xint:}}%
525   {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#5\xint:}%
526     {\XINT_mul_fork #1\xint:#3\xint:}}%
527 }%
528 \def\xINT_gctf_loop_c #1#2%
529 {%
530   \expandafter\xINT_gctf_loop_d\expandafter {\expandafter{#2}{#1}}%
531 }%
532 \def\xINT_gctf_loop_d #1#2%
533 {%
534   \expandafter\xINT_gctf_loop_e\expandafter {\expandafter{#2}#1}%
535 }%
536 \def\xINT_gctf_loop_e #1#2%
537 {%
538   \expandafter\xINT_gctf_loop_f\expandafter {\expandafter{#2}#1}%
539 }%
540 \def\xINT_gctf_loop_f #1#2/%
541 {%
542   \xint_gob_til_exclam #2\xINT_gctf_end!%
543   \expandafter\xINT_gctf_loop_g
544   \romannumeral0\xintraewithzeros {#2}.#1%
545 }%
546 \def\xINT_gctf_loop_g #1/#2.#3#4#5#6%
547 {%
548   \expandafter\xINT_gctf_loop_h\expandafter
549   {\romannumeral0\xINT_mul_fork #1\xint:#6\xint:}%
550   {\romannumeral0\xINT_mul_fork #1\xint:#5\xint:}%
551   {\romannumeral0\xINT_mul_fork #2\xint:#4\xint:}%
552   {\romannumeral0\xINT_mul_fork #2\xint:#3\xint:}%
553 }%
554 \def\xINT_gctf_loop_h #1#2%
555 {%

```

```

556 \expandafter\XINT_gctf_loop_i\expandafter {\expandafter{#2}{#1}}%
557 }%
558 \def\XINT_gctf_loop_i #1#2%
559 {%
560 \expandafter\XINT_gctf_loop_j\expandafter {\expandafter{#2}{#1}}%
561 }%
562 \def\XINT_gctf_loop_j #1#2%
563 {%
564 \expandafter\XINT_gctf_loop_a\expandafter {#2}{#1}%
565 }%
566 \def\XINT_gctf_end #1.#2#3#4#5{\xintrowwithzeros {#2/#3}}% 1.09b removes [0]

```

9.16 \xintiGctoF

```

567 \def\xintiGctoF {\romannumeral0\xintigctoF}%
568 \def\xintigctoF #1%
569 {%
570 \expandafter\XINT_igctf_prep \romannumeral`&&@#1+!/ %
571 }%
572 \def\XINT_igctf_prep
573 {%
574 \XINT_igctf_loop_a 1001%
575 }%
576 \def\XINT_igctf_loop_a #1#2#3#4#5+%
577 {%
578 \expandafter\XINT_igctf_loop_b
579 \romannumeral`&&@#5.{#1}{#2}{#3}{#4}%
580 }%
581 \def\XINT_igctf_loop_b #1.#2#3#4#5%
582 {%
583 \expandafter\XINT_igctf_loop_c\expandafter
584 {\romannumeral0\xintiiadd {#5}{\XINT_mul_fork #1\xint:#3\xint:}}%
585 {\romannumeral0\xintiiadd {#4}{\XINT_mul_fork #1\xint:#2\xint:}}%
586 {#2}{#3}%
587 }%
588 \def\XINT_igctf_loop_c #1#2%
589 {%
590 \expandafter\XINT_igctf_loop_f\expandafter {\expandafter{#2}{#1}}%
591 }%
592 \def\XINT_igctf_loop_f #1#2#3#4/%
593 {%
594 \xint_gob_til_exclam #4\XINT_igctf_end!%
595 \expandafter\XINT_igctf_loop_g
596 \romannumeral`&&@#4.{#2}{#3}{#1}%
597 }%
598 \def\XINT_igctf_loop_g #1.#2#3%
599 {%
600 \expandafter\XINT_igctf_loop_h\expandafter
601 {\romannumeral0\XINT_mul_fork #1\xint:#3\xint:}%
602 {\romannumeral0\XINT_mul_fork #1\xint:#2\xint:}%
603 }%
604 \def\XINT_igctf_loop_h #1#2%
605 {%
606 \expandafter\XINT_igctf_loop_i\expandafter {#2}{#1}%

```



```

607 }%
608 \def\XINT_igctf_loop_i #1#2#3#4%
609 {%
610   \XINT_igctf_loop_a {#3}{#4}{#1}{#2}%
611 }%
612 \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.

```

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

```

```

654 }%
655 \def\XINT_ctcv_loop_f #1#2#3#4#5%
656 {%
657   \expandafter\XINT_ctcv_loop_g\expandafter
658   {\romannumeral0\xintraawithzeros {#1/#2}}{#5}{#1}{#2}{#3}{#4}%
659 }%
660 \def\XINT_ctcv_loop_g #1#2{\XINT_ctcv_loop_a {#2{#1}}}% 1.09b removes [0]
661 \def\XINT_ctcv_end #1.#2#3#4#5#6{ #6}%

```

9.18 *\xintiCstoCv*

```

662 \def\xintiCstoCv {\romannumeral0\xinticstocv }%
663 \def\xinticstocv #1%
664 {%
665   \expandafter\XINT_icstcv_prep \romannumeral`&&@#1,!,%
666 }%
667 \def\XINT_icstcv_prep
668 {%
669   \XINT_icstcv_loop_a {}1001%
670 }%
671 \def\XINT_icstcv_loop_a #1#2#3#4#5#6,%
672 {%
673   \xint_gob_til_exclam #6\XINT_icstcv_end!%
674   \expandafter
675   \XINT_icstcv_loop_b \romannumeral`&&@#6.{#2}{#3}{#4}{#5}{#1}%
676 }%
677 \def\XINT_icstcv_loop_b #1.#2#3#4#5%
678 {%
679   \expandafter\XINT_icstcv_loop_c\expandafter
680   {\romannumeral0\xintiiadd {#5}{\XINT_mul_fork #1\xint:#3\xint:}}%
681   {\romannumeral0\xintiiadd {#4}{\XINT_mul_fork #1\xint:#2\xint:}}%
682   {{#2}{#3}}%
683 }%
684 \def\XINT_icstcv_loop_c #1#2%
685 {%
686   \expandafter\XINT_icstcv_loop_d\expandafter {#2}{#1}%
687 }%
688 \def\XINT_icstcv_loop_d #1#2%
689 {%
690   \expandafter\XINT_icstcv_loop_e\expandafter
691   {\romannumeral0\xintraawithzeros {#1/#2}}{#1}{#2}}%
692 }%
693 \def\XINT_icstcv_loop_e #1#2#3#4{\XINT_icstcv_loop_a {#4{#1}}#2#3}%
694 \def\XINT_icstcv_end #1.#2#3#4#5#6{ #6}% 1.09b removes [0]

```

9.19 *\xintGctoCv*

```

695 \def\xintGctoCv {\romannumeral0\xintgctocv }%
696 \def\xintgctocv #1%
697 {%
698   \expandafter\XINT_gctcv_prep \romannumeral`&&@#1+!/,%
699 }%
700 \def\XINT_gctcv_prep
701 {%

```

```

702 \XINT_gctcv_loop_a {}1001%
703 }%
704 \def\XINT_gctcv_loop_a #1#2#3#4#5#6+%
705 {%
706 \expandafter\XINT_gctcv_loop_b
707 \romannumeral0\xintraewithzeros {#6}. {#2}{#3}{#4}{#5}{#1}%
708 }%
709 \def\XINT_gctcv_loop_b #1/#2.#3#4#5#6%
710 {%
711 \expandafter\XINT_gctcv_loop_c\expandafter
712 {\romannumeral0\XINT_mul_fork #2\xint:#4\xint:}%
713 {\romannumeral0\XINT_mul_fork #2\xint:#3\xint:}%
714 {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#6\xint:}%
715 {\XINT_mul_fork #1\xint:#4\xint:}}%
716 {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#5\xint:}%
717 {\XINT_mul_fork #1\xint:#3\xint:}}%
718 }%
719 \def\XINT_gctcv_loop_c #1#2%
720 {%
721 \expandafter\XINT_gctcv_loop_d\expandafter {\expandafter{#2}{#1}}%
722 }%
723 \def\XINT_gctcv_loop_d #1#2%
724 {%
725 \expandafter\XINT_gctcv_loop_e\expandafter {\expandafter{#2}{#1}}%
726 }%
727 \def\XINT_gctcv_loop_e #1#2%
728 {%
729 \expandafter\XINT_gctcv_loop_f\expandafter {#2}#1%
730 }%
731 \def\XINT_gctcv_loop_f #1#2%
732 {%
733 \expandafter\XINT_gctcv_loop_g\expandafter
734 {\romannumeral0\xintraewithzeros {#1/#2}}{#1}{#2}}%
735 }%
736 \def\XINT_gctcv_loop_g #1#2#3#4%
737 {%
738 \XINT_gctcv_loop_h {#4{#1}}{#2#3}% 1.09b removes [0]
739 }%
740 \def\XINT_gctcv_loop_h #1#2#3/%
741 {%
742 \xint_gob_til_exclam #3\XINT_gctcv_end!%
743 \expandafter\XINT_gctcv_loop_i
744 \romannumeral0\xintraewithzeros {#3}.#2{#1}%
745 }%
746 \def\XINT_gctcv_loop_i #1/#2.#3#4#5#6%
747 {%
748 \expandafter\XINT_gctcv_loop_j\expandafter
749 {\romannumeral0\XINT_mul_fork #1\xint:#6\xint:}%
750 {\romannumeral0\XINT_mul_fork #1\xint:#5\xint:}%
751 {\romannumeral0\XINT_mul_fork #2\xint:#4\xint:}%
752 {\romannumeral0\XINT_mul_fork #2\xint:#3\xint:}%
753 }%

```

```

754 \def\XINT_gctcv_loop_j #1#2%
755 {%
756   \expandafter\XINT_gctcv_loop_k\expandafter {\expandafter{#2}{#1}}%
757 }%
758 \def\XINT_gctcv_loop_k #1#2%
759 {%
760   \expandafter\XINT_gctcv_loop_l\expandafter {\expandafter{#2}{#1}}%
761 }%
762 \def\XINT_gctcv_loop_l #1#2%
763 {%
764   \expandafter\XINT_gctcv_loop_m\expandafter {\expandafter{#2}{#1}}%
765 }%
766 \def\XINT_gctcv_loop_m #1#2{\XINT_gctcv_loop_a {#2}{#1}}%
767 \def\XINT_gctcv_end #1.#2#3#4#5#6{ #6}%

```

9.20 \xintiGctoCv

```

768 \def\xintiGctoCv {\romannumeral0\xintigctocv}%
769 \def\xintigctocv #1%
770 {%
771   \expandafter\XINT_igctcv_prep \romannumeral`&&@#1+!/;%
772 }%
773 \def\XINT_igctcv_prep
774 {%
775   \XINT_igctcv_loop_a {}1001%
776 }%
777 \def\XINT_igctcv_loop_a #1#2#3#4#5#6+%
778 {%
779   \expandafter\XINT_igctcv_loop_b
780   \romannumeral`&&@#6.{#2}{#3}{#4}{#5}{#1}%
781 }%
782 \def\XINT_igctcv_loop_b #1.#2#3#4#5%
783 {%
784   \expandafter\XINT_igctcv_loop_c\expandafter
785   {\romannumeral0\xintiiadd {#5}{\XINT_mul_fork #1\xint:#3\xint:}}%
786   {\romannumeral0\xintiiadd {#4}{\XINT_mul_fork #1\xint:#2\xint:}}%
787   {{#2}{#3}}%
788 }%
789 \def\XINT_igctcv_loop_c #1#2%
790 {%
791   \expandafter\XINT_igctcv_loop_f\expandafter {\expandafter{#2}{#1}}%
792 }%
793 \def\XINT_igctcv_loop_f #1#2#3#4/%
794 {%
795   \xint_gob_til_exclam #4\XINT_igctcv_end_a!%
796   \expandafter\XINT_igctcv_loop_g
797   \romannumeral`&&@#4.#1#2{#3}%
798 }%
799 \def\XINT_igctcv_loop_g #1.#2#3#4#5%
800 {%
801   \expandafter\XINT_igctcv_loop_h\expandafter
802   {\romannumeral0\XINT_mul_fork #1\xint:#5\xint:}%
803   {\romannumeral0\XINT_mul_fork #1\xint:#4\xint:}%
804   {{#2}{#3}}%

```

```

805 }%
806 \def\XINT_igctcv_loop_h #1#2%
807 {%
808   \expandafter\XINT_igctcv_loop_i\expandafter {\expandafter{#2}{#1}}%
809 }%
810 \def\XINT_igctcv_loop_i #1#2{\XINT_igctcv_loop_k #2{#2#1}}%
811 \def\XINT_igctcv_loop_k #1#2%
812 {%
813   \expandafter\XINT_igctcv_loop_l\expandafter
814   {\romannumeral0\xintraawithzeros {#1/#2}}%
815 }%
816 \def\XINT_igctcv_loop_l #1#2#3{\XINT_igctcv_loop_a {#3{#1}}#2}%1.09i removes [0]
817 \def\XINT_igctcv_end_a #1.#2#3#4#5%
818 {%
819   \expandafter\XINT_igctcv_end_b\expandafter
820   {\romannumeral0\xintraawithzeros {#2/#3}}%
821 }%
822 \def\XINT_igctcv_end_b #1#2{ #2{#1}}% 1.09b removes [0]

```

9.21 `\xintFtoCv`

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

```

823 \def\xintFtoCv {\romannumeral0\xintftocv }%
824 \def\xintftocv #1%
825 {%
826   \xinticstocv {\xintFtoCs {#1}}%
827 }%

```

9.22 `\xintFtoCCv`

```

828 \def\xintFtoCCv {\romannumeral0\xintftoccv }%
829 \def\xintftoccv #1%
830 {%
831   \xintigctocv {\xintFtoCC {#1}}%
832 }%

```

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.

```

833 \def\xintCntoF {\romannumeral0\xintcntof }%
834 \def\xintcntof #1%
835 {%
836   \expandafter\XINT_cntf\expandafter {\the\numexpr #1}%
837 }%
838 \def\XINT_cntf #1#2%
839 {%
840   \ifnum #1>\xint_c_
841     \xint_afterfi {\expandafter\XINT_cntf_loop\expandafter
842                   {\the\numexpr #1-1\expandafter}\expandafter
843                   {\romannumeral`&&@#2{#1}}{#2}}%
844   \else

```

```

845     \xint_afterfi
846     {\ifnum #1=\xint_c_
847       \xint_afterfi {\expandafter\space \romannumeral`&&@#2{0}}}%
848       \else \xint_afterfi { }% 1.09m now returns nothing.
849     \fi}%
850   \fi
851 }%
852 \def\xINT_cntf_loop #1#2#3%
853 {%
854   \ifnum #1>\xint_c_ \else \XINT_cntf_exit \fi
855   \expandafter\xINT_cntf_loop\expandafter
856   {\the\numexpr #1-1\expandafter }\expandafter
857   {\romannumeral0\xintadd {\xintDiv {1[0]}\{#2}\{#3{#1}}}%
858   {#3}%
859 }%
860 \def\xINT_cntf_exit \fi
861   \expandafter\xINT_cntf_loop\expandafter
862   #1\expandafter #2#3%
863 {%
864   \fi\xint_gobble_ii #2%
865 }%

```

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.

```

866 \def\xintGCntoF {\romannumeral0\xintgcntof }%
867 \def\xintgcntof #1%
868 {%
869   \expandafter\xINT_gcntf\expandafter {\the\numexpr #1}%
870 }%
871 \def\xINT_gcntf #1#2#3%
872 {%
873   \ifnum #1>\xint_c_
874     \xint_afterfi {\expandafter\xINT_gcntf_loop\expandafter
875       {\the\numexpr #1-1\expandafter}\expandafter
876       {\romannumeral`&&@#2{#1}\{#2}\{#3}}}%
877   \else
878     \xint_afterfi
879     {\ifnum #1=\xint_c_
880       \xint_afterfi {\expandafter\space\romannumeral`&&@#2{0}}}%
881       \else \xint_afterfi { }% 1.09m now returns nothing rather than 0/1[0]
882     \fi}%
883   \fi
884 }%
885 \def\xINT_gcntf_loop #1#2#3#4%
886 {%
887   \ifnum #1>\xint_c_ \else \XINT_gcntf_exit \fi
888   \expandafter\xINT_gcntf_loop\expandafter
889   {\the\numexpr #1-1\expandafter }\expandafter
890   {\romannumeral0\xintadd {\xintDiv {#4{#1}\{#2}\{#3{#1}}}%
891   {#3}\{#4}%

```

```

892 }%
893 \def\XINT_gcntf_exit \fi
894   \expandafter\XINT_gcntf_loop\expandafter
895   #1\expandafter #2#3#4%
896 {%
897   \fi\xint_gobble_ii #2%
898 }%

```

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.

```

899 \def\xintCntoCs {\romannumeral0\xintcntocs }%
900 \def\xintcntocs #1%
901 {%
902   \expandafter\XINT_cntcs\expandafter {\the\numexpr #1}%
903 }%
904 \def\XINT_cntcs #1#2%
905 {%
906   \ifnum #1<0
907     \xint_afterfi { }% 1.09i: a 0/1[0] was here, now the macro returns nothing
908   \else
909     \xint_afterfi {\expandafter\XINT_cntcs_loop\expandafter
910                   {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
911                   {\romannumeral`&&@#2{#1}}{#2}}% produced coeff not braced
912   \fi
913 }%
914 \def\XINT_cntcs_loop #1#2#3%
915 {%
916   \ifnum #1>-\xint_c_i \else \XINT_cntcs_exit \fi
917   \expandafter\XINT_cntcs_loop\expandafter
918   {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
919   {\romannumeral`&&@#3{#1}, #2}{#3}% space added, 1.09m
920 }%
921 \def\XINT_cntcs_exit \fi
922   \expandafter\XINT_cntcs_loop\expandafter
923   #1\expandafter #2#3%
924 {%
925   \fi\XINT_cntcs_exit_b #2%
926 }%
927 \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.

```

928 \def\xintCntoGC {\romannumeral0\xintcntogc }%
929 \def\xintcntogc #1%
930 {%
931   \expandafter\xINT_cntgc\expandafter {\the\numexpr #1}%
932 }%
933 \def\xINT_cntgc #1#2%
934 {%
935   \ifnum #1<0
936     \xint_afterfi { }% 1.09i there was as strange 0/1[0] here, removed
937   \else
938     \xint_afterfi {\expandafter\xINT_cntgc_loop\expandafter
939                   {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
940                   {\expandafter{\romannumeral`&&@#2{#1}}{#2}}}%
941   \fi
942 }%
943 \def\xINT_cntgc_loop #1#2#3%
944 {%
945   \ifnum #1>-\xint_c_i \else \XINT_cntgc_exit \fi
946   \expandafter\xINT_cntgc_loop\expandafter
947   {\the\numexpr #1-\xint_c_i\expandafter }\expandafter
948   {\expandafter{\romannumeral`&&@#3{#1}}+1/#2}{#3}%
949 }%
950 \def\xINT_cntgc_exit \fi
951   \expandafter\xINT_cntgc_loop\expandafter
952   #1\expandafter #2#3%
953 {%
954   \fi\xINT_cntgc_exit_b #2%
955 }%
956 \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.

```

957 \def\xintGCntoGC {\romannumeral0\xintgcntogc }%
958 \def\xintgcntogc #1%
959 {%
960   \expandafter\xINT_gcntgc\expandafter {\the\numexpr #1}%
961 }%
962 \def\xINT_gcntgc #1#2#3%
963 {%
964   \ifnum #1<0
965     \xint_afterfi { }% 1.09i now returns nothing
966   \else
967     \xint_afterfi {\expandafter\xINT_gcntgc_loop\expandafter
968                   {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
969                   {\expandafter{\romannumeral`&&@#2{#1}}{#2}{#3}}}%
970   \fi
971 }%
972 \def\xINT_gcntgc_loop #1#2#3#4%
973 {%
974   \ifnum #1>-\xint_c_i \else \XINT_gcntgc_exit \fi

```



```

975 \expandafter\XINT_gcntgc_loop_b\expandafter
976 {\expandafter{\romannumeral`&&@#4{#1}}/#2}{#3{#1}}{#1}{#3}{#4}%
977 }%
978 \def\XINT_gcntgc_loop_b #1#2#3%
979 {%
980 \expandafter\XINT_gcntgc_loop\expandafter
981 {\the\numexpr #3-\xint_c_i \expandafter}\expandafter
982 {\expandafter{\romannumeral`&&@#2}+#1}%
983 }%
984 \def\XINT_gcntgc_exit \fi
985 \expandafter\XINT_gcntgc_loop_b\expandafter #1#2#3#4#5%
986 {%
987 \fi\XINT_gcntgc_exit_b #1%
988 }%
989 \def\XINT_gcntgc_exit_b #1/{ }%

```

9.28 \xintCstoGC

```

990 \def\xintCstoGC {\romannumeral0\xintcstogc }%
991 \def\xintcstogc #1%
992 {%
993 \expandafter\XINT_cstc_prep \romannumeral`&&@#1,!,%
994 }%
995 \def\XINT_cstc_prep #1,{\XINT_cstc_loop_a {{#1}}}%
996 \def\XINT_cstc_loop_a #1#2,%
997 {%
998 \xint_gob_til_exclam #2\XINT_cstc_end!%
999 \XINT_cstc_loop_b {#1}{#2}%
1000 }%
1001 \def\XINT_cstc_loop_b #1#2{\XINT_cstc_loop_a {#1+1/{#2}}}%
1002 \def\XINT_cstc_end!\XINT_cstc_loop_b #1#2{ #1}%

```

9.29 \xintGctoGC

```

1003 \def\xintGctoGC {\romannumeral0\xintgctogc }%
1004 \def\xintgctogc #1%
1005 {%
1006 \expandafter\XINT_gctgc_start \romannumeral`&&@#1+!/,%
1007 }%
1008 \def\XINT_gctgc_start {\XINT_gctgc_loop_a {}}%
1009 \def\XINT_gctgc_loop_a #1#2+#3/%
1010 {%
1011 \xint_gob_til_exclam #3\XINT_gctgc_end!%
1012 \expandafter\XINT_gctgc_loop_b\expandafter
1013 {\romannumeral`&&@#2}{#3}{#1}%
1014 }%
1015 \def\XINT_gctgc_loop_b #1#2%
1016 {%
1017 \expandafter\XINT_gctgc_loop_c\expandafter
1018 {\romannumeral`&&@#2}{#1}%
1019 }%
1020 \def\XINT_gctgc_loop_c #1#2#3%
1021 {%
1022 \XINT_gctgc_loop_a {#3{#2}+#1}/}%

```

9 Package *xintcfrac* implementation

```
1023 }%
1024 \def\XINT_gctgc_end!\expandafter\XINT_gctgc_loop_b
1025 {%
1026     \expandafter\XINT_gctgc_end_b
1027 }%
1028 \def\XINT_gctgc_end_b #1#2#3{ #3{#1}}%
1029 \XINT_restorecatcodes_endinput%
```

10 Package `xintexpr` implementation

Contents

10.1	Old comments	292
10.2	Catcodes, ε -TeX and reload detection	292
10.3	Package identification	294
10.4	Locking and unlocking	294
10.5	<code>\XINT_expr_wrap</code> , <code>\XINT_iexpr_wrap</code>	295
10.6	<code>\XINT_protectii</code> , <code>\XINT_expr_usethe</code>	295
10.7	<code>\XINT_expr_print</code> , <code>\XINT_iexpr_print</code> , <code>\XINT_boolexpr_print</code>	295
10.8	<code>\xintexpr</code> , <code>\xintiexpr</code> , <code>\xintfloatexpr</code> , <code>\xintiexpr</code>	295
10.9	<code>\xinttheexpr</code> , <code>\xinttheiexpr</code> , <code>\xintthefloatexpr</code> , <code>\xinttheiexpr</code>	295
10.10	<code>\xintthe</code>	295
10.11	<code>\thexintexpr</code> , <code>\thexintiexpr</code> , <code>\thexintfloatexpr</code> , <code>\thexintiexpr</code>	295
10.12	<code>\xintthecoords</code>	295
10.13	<code>\xintbareeval</code> , <code>\xintbarefloateval</code> , <code>\xintbareiieval</code>	296
10.14	<code>\xintthebareeval</code> , <code>\xintthebarefloateval</code> , <code>\xintthebareiieval</code>	296
10.15	<code>\xinteval</code> , <code>\xintiieval</code>	296
10.16	<code>\xintieval</code> , <code>\XINT_iexpr_wrap</code>	296
10.17	<code>\xintfloateval</code> , <code>\XINT_flexpr_wrap</code> , <code>\XINT_flexpr_print</code>	296
10.18	<code>\xintboolexpr</code> , <code>\xinttheboolexpr</code> , <code>\thexintboolexpr</code>	297
10.19	<code>\xintifboolexpr</code> , <code>\xintifboolfloatexpr</code> , <code>\xintifbooliexpr</code>	297
10.20	Macros handling csv lists on output (for <code>\XINT_expr_print</code> et al. routines)	297
10.21	<code>\XINT_expr_getnext</code> : fetching some number then an operator	299
10.22	The integer or decimal number or hexa-decimal number or function name or variable name or special hacky things big parser	300
10.23	<code>\XINT_expr_getop</code> : finding the next operator or closing parenthesis or end of expression	307
10.24	Expansion spanning; opening and closing parentheses	309
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	310
10.26	Macros for list selectors: <code>[list][N]</code> , <code>[list][:b]</code> , <code>[list][a:]</code> , <code>[list][a:b]</code>	316
10.27	Macros for a..b list generation	322
10.28	Macros for a..[d]..b list generation	323
10.29	The comma as binary operator	325
10.30	The minus as prefix operator of variable precedence level	326
10.31	<code>?</code> as two-way and <code>??</code> as three-way conditionals with braced branches	327
10.32	<code>!</code> as postfix factorial operator	327
10.33	The A/B[N] mechanism	327
10.34	<code>\XINT_expr_op_`</code> for recognizing functions	328
10.35	The bool, togl, protect pseudo “functions”	329
10.36	The break function	329
10.37	The qint, qfrac, qfloat “functions”	329
10.38	<code>\XINT_expr_op__</code> for recognizing variables	329
10.39	User defined variables: <code>\xintdefvar</code> , <code>\xintdefiivar</code> , <code>\xintdeffloatvar</code>	330
10.40	<code>\xintunassignvar</code>	331
10.41	seq and the implementation of dummy variables	332
10.42	add, mul	338
10.43	subs	339
10.44	rseq	339
10.45	iter	341

10.46	<code>rrseq</code>	342
10.47	<code>iterr</code>	344
10.48	Macros handling csv lists for functions with multiple comma separated arguments in expressions	346
10.49	The num, reduce, abs, sgn, frac, floor, ceil, sqr, sqrt, sqrtr, float, round, trunc, mod, quo, rem, divmod, 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	349
10.50	f-expandable versions of the <code>\xintSeqB::csv</code> and alike routines, for <code>\xintNewExpr</code>	357
10.51	User defined functions: <code>\xintdeffunc</code> , <code>\xintdefiifunc</code> , <code>\xintdeffloatfunc</code>	359
10.52	<code>\xintNewFunction</code>	360
10.53	<code>\xintNewExpr</code> , <code>\xintNewIExpr</code> , <code>\xintNewFloatExpr</code> , <code>\xintNewIIEExpr</code>	361

This is release 1.2q of [2018/02/06].

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 [2018/02/06 1.2q Expandable expression parser (JFB)]%
58 \catcode\! 11
59 \let\XINT_Cmp \xintiiCmp
```

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...).

```
60 \def\xint_gob_til_! #1!{% ! with catcode 11
61 \def\XINT_expr_lockscan#1{% not used for decimal numbers in xintexpr 1.2
62 \def\XINT_expr_lockscan##1!{\expandafter#1\csname .##1\endcsname}%
63 }\XINT_expr_lockscan{ }%
64 \def\XINT_expr_lockit#1{%
65 \def\XINT_expr_lockit##1{\expandafter#1\csname .##1\endcsname}%
66 }\XINT_expr_lockit{ }%
67 \def\XINT_expr_unlock_hex_in #1% expanded inside \csname...\endcsname
68 {\expandafter\XINT_expr_inhex\romannumeral`&&\XINT_expr_unlock#1;}%
69 \def\XINT_expr_inhex #1.#2#3;% expanded inside \csname...\endcsname
70 {%
71 \if#2>%
72 \xintHexToDec{#1}%
73 \else
74 \xintiiMul{\xintiiPow{625}{\xintLength{#3}}}{\xintHexToDec{#1#3}}%
75 [\the\numexpr-4*\xintLength{#3}]%
76 \fi
77 }%
78 \def\XINT_expr_unlock {\expandafter\XINT_expr_unlock_a\string }%
79 \def\XINT_expr_unlock_a #1.={}%
80 \def\XINT_expr_unexpectedtoken {\xintError:ignored }%
81 \let\XINT_expr_done\space
```

10.5 `\XINT_expr_wrap`, `\XINT_iiexpr_wrap`

```
82 \def\XINT_expr_wrap { !\XINT_expr_usethe\XINT_protectii\XINT_expr_print }%
83 \def\XINT_iiexpr_wrap { !\XINT_expr_usethe\XINT_protectii\XINT_iiexpr_print }%
```

10.6 `\XINT_protectii`, `\XINT_expr_usethe`

```
84 \def\XINT_protectii #1{\noexpand\XINT_protectii\noexpand #1\noexpand }%
85 \protected\def\XINT_expr_usethe\XINT_protectii {\xintError:missing_xintthe!}%
```

10.7 `\XINT_expr_print`, `\XINT_iiexpr_print`, `\XINT_boolexpr_print`

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

```
86 \def\XINT_expr_print #1{\xintSPRaw::csv {\XINT_expr_unlock #1}}%
87 \def\XINT_iiexpr_print #1{\xintCSV::csv {\XINT_expr_unlock #1}}%
88 \def\XINT_boolexpr_print #1{\xintIsTrue::csv {\XINT_expr_unlock #1}}%
```

10.8 `\xintexpr`, `\xintiexpr`, `\xintfloatexpr`, `\xintiexpr`

```
89 \def\xintexpr {\romannumeral0\xinteval }%
90 \def\xintiexpr {\romannumeral0\xintieval }%
91 \def\xintfloatexpr {\romannumeral0\xintfloateval }%
92 \def\xintiexpr {\romannumeral0\xintiieval }%
```

10.9 `\xinttheexpr`, `\xinttheiexpr`, `\xintthefloatexpr`, `\xinttheiexpr`

```
93 \def\xinttheexpr
94 {\romannumeral`&&\expandafter\XINT_expr_print\romannumeral0\xintbareeval }%
95 \def\xinttheiexpr {\romannumeral`&&\xintthe\xintiexpr }%
96 \def\xintthefloatexpr {\romannumeral`&&\xintthe\xintfloatexpr }%
97 \def\xinttheiexpr
98 {\romannumeral`&&\expandafter\XINT_iiexpr_print\romannumeral0\xintbareiieval }%
```

10.10 `\xintthe`

```
99 \def\xintthe #1{\romannumeral`&&\expandafter\xint_gobble_iii\romannumeral`&&#1}%
```

10.11 `\thexintexpr`, `\thexintiexpr`, `\thexintfloatexpr`, `\thexintiexpr`

New with 1.2h. I have been three years long very strict in terms of prefixing macros, but well.

```
100 \let\thexintexpr \xinttheexpr
101 \let\thexintiexpr \xinttheiexpr
102 \let\thexintfloatexpr\xintthefloatexpr
103 \let\thexintiexpr \xinttheiexpr
```

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!

```

104 \def\xintthecoords #1{\romannumeral`&&\expandafter\expandafter\expandafter
105     \XINT_thecoords_a
106     \expandafter\xint_gobble_iii\romannumeral0#1}%
107 \def\XINT_thecoords_a #1#2% #1=print macro, indispensable for scientific notation
108     {\expandafter\XINT_expr_unlock\csname.=\expandafter\XINT_thecoords_b
109         \romannumeral`&&@#1#2,!,!,^%\endcsname }%
110 \def\XINT_thecoords_b #1#2,#3#4,%
111     {\xint_gob_til_! #3\XINT_thecoords_c ! (#1#2, #3#4)\XINT_thecoords_b }%
112 \def\XINT_thecoords_c #1^{}%

```

10.13 \xintbareeval, \xintbarefloateval, \xintbareieval

```

113 \def\xintbareeval
114     {\expandafter\XINT_expr_until_end_a\romannumeral`&&\XINT_expr_getnext }%
115 \def\xintbarefloateval
116     {\expandafter\XINT_flexpr_until_end_a\romannumeral`&&\XINT_expr_getnext }%
117 \def\xintbareieval
118     {\expandafter\XINT_iexpr_until_end_a\romannumeral`&&\XINT_expr_getnext }%

```

10.14 \xintthebareeval, \xintthebarefloateval, \xintthebareieval

```

119 \def\xintthebareeval      {\expandafter\XINT_expr_unlock\romannumeral0\xintbareeval}%
120 \def\xintthebarefloateval {\expandafter\XINT_expr_unlock\romannumeral0\xintbarefloateval}%
121 \def\xintthebareieval    {\expandafter\XINT_expr_unlock\romannumeral0\xintbareieval}%

```

10.15 \xinteval, \xintiieval

```

122 \def\xinteval    {\expandafter\XINT_expr_wrap\romannumeral0\xintbareeval }%
123 \def\xintiieval {\expandafter\XINT_iexpr_wrap\romannumeral0\xintbareieval }%

```

10.16 \xintieval, \XINT_iexpr_wrap

Optional argument since 1.1.

```

124 \def\xintieval #1%
125     {\ifx [#1\expandafter\XINT_iexpr_withopt\else\expandafter\XINT_iexpr_noopt \fi #1}%
126 \def\XINT_iexpr_noopt
127     {\expandafter\XINT_iexpr_wrap \expandafter 0\romannumeral0\xintbareeval }%
128 \def\XINT_iexpr_withopt [#1]%
129     {%
130         \expandafter\XINT_iexpr_wrap\expandafter
131         {\the\numexpr \xint_zapspaces #1 \xint_gobble_i\expandafter}%
132         \romannumeral0\xintbareeval
133     }%
134 \def\XINT_iexpr_wrap #1#2%
135     {%
136         \expandafter\XINT_expr_wrap
137         \csname .=\xintRound::csv {#1}{\XINT_expr_unlock #2}\endcsname
138     }%

```

10.17 \xintfloateval, \XINT_flexpr_wrap, \XINT_flexpr_print

Optional argument since 1.1

```

139 \def\xintfloateval #1%

```



```

140 {%
141   \ifx [#1\expandafter\XINT_flexpr_withopt_a\else\expandafter\XINT_flexpr_noopt
142   \fi #1%
143 }%
144 \def\XINT_flexpr_noopt
145 {%
146   \expandafter\XINT_flexpr_withopt_b\expandafter\xinttheDigits
147   \romannumeral0\xintbarefloateval
148 }%
149 \def\XINT_flexpr_withopt_a [#1]%
150 {%
151   \expandafter\XINT_flexpr_withopt_b\expandafter
152   {\the\numexpr\xint_zapspace #1 \xint_gobble_i\expandafter}%
153   \romannumeral0\xintbarefloateval
154 }%
155 \def\XINT_flexpr_withopt_b #1#2%
156 {%
157   \expandafter\XINT_flexpr_wrap\csname .;#1.=% ; and not : as before b'cause NewExpr
158   \XINTinFloat::csv {#1}{\XINT_expr_unlock #2}\endcsname
159 }%
160 \def\XINT_flexpr_wrap { !\XINT_expr_usethe\XINT_protectii\XINT_flexpr_print }%
161 \def\XINT_flexpr_print #1%
162 {%
163   \expandafter\xintPFloat::csv
164   \romannumeral`&&\expandafter\XINT_expr_unlock_sp\string #1!%
165 }%
166 \catcode`: 12
167   \def\XINT_expr_unlock_sp #1.;#2.=#3!{{#2}{#3}}%
168 \catcode`: 11

```

10.18 \xintboolexpr, \xinttheboolexpr, \thexintboolexpr

```

169 \def\xintboolexpr      {\romannumeral0\expandafter\expandafter\expandafter
170   \XINT_boolexpr_done \expandafter\xint_gobble_iv\romannumeral0\xinteval }%
171 \def\xinttheboolexpr   {\romannumeral`&&\expandafter\expandafter\expandafter
172   \XINT_boolexpr_print\expandafter\xint_gobble_iv\romannumeral0\xinteval }%
173 \let\thexintboolexpr\xinttheboolexpr
174 \def\XINT_boolexpr_done { !\XINT_expr_usethe\XINT_protectii\XINT_boolexpr_print }%

```

10.19 \xintifboolexpr, \xintifboolfloatexpr, \xintifboolliexpr

Do not work with comma separated expressions.

```

175 \def\xintifboolexpr      #1{\romannumeral0\xintifnotzero {\xinttheexpr #1\relax}}%
176 \def\xintifboolfloatexpr #1{\romannumeral0\xintifnotzero {\xintthefloatexpr #1\relax}}%
177 \def\xintifboolliexpr    #1{\romannumeral0\xintifnotzero {\xinttheiiexpr #1\relax}}%

```

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

10.20.1	\XINT:::end	298
10.20.2	\xintCSV:::csv	298
10.20.3	\xintSPRaw, \xintSPRaw:::csv	298
10.20.4	\xintIsTrue:::csv	298
10.20.5	\xintRound:::csv	298

10.20.6	<code>\XINTinFloat::csv</code>	299
10.20.7	<code>\xintPFloat::csv</code>	299

Changed completely for 1.1, which adds the optional arguments to `\xintiexpr` 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`

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

10.20.2 `\xintCSV::csv`

```
179 \def\xintCSV::csv #1{\expandafter\XINT_csv::_a\romannumeral`&&@#1,^}%
180 \def\XINT_csv::_a {\XINT_csv::_b {}}%
181 \def\XINT_csv::_b #1#2,{\expandafter\XINT_csv::_c \romannumeral`&&@#2,{#1}}%
182 \def\XINT_csv::_c #1{\if ^#1\expandafter\XINT::_end\fi\XINT_csv::_d #1}%
183 \def\XINT_csv::_d #1,#2{\XINT_csv::_b {#2, #1}}% possibly, item #1 is empty.
```

10.20.3 `\xintSPRaw, \xintSPRaw::csv`

```
184 \def\xintSPRaw {\romannumeral0\xintspraw }%
185 \def\xintspraw #1{\expandafter\XINT_spraw\romannumeral`&&@#1[\W]}%
186 \def\XINT_spraw #1[#2#3]{\xint_gob_til_W #2\XINT_spraw_a\W\XINT_spraw_p #1[#2#3]}%
187 \def\XINT_spraw_a\W\XINT_spraw_p #1[\W]{ #1}%
188 \def\XINT_spraw_p #1[\W]{\xintpraw {#1}}%
189 \def\xintSPRaw::csv #1{\romannumeral0\expandafter\XINT_spraw::_a\romannumeral`&&@#1,^}%
190 \def\XINT_spraw::_a {\XINT_spraw::_b {}}%
191 \def\XINT_spraw::_b #1#2,{\expandafter\XINT_spraw::_c \romannumeral`&&@#2,{#1}}%
192 \def\XINT_spraw::_c #1{\if ,#1\xint_dothis\XINT_spraw::_e\fi
193     \if ^#1\xint_dothis\XINT::_end\fi
194     \xint_orthat\XINT_spraw::_d #1}%
195 \def\XINT_spraw::_d #1,{\expandafter\XINT_spraw::_e\romannumeral0\XINT_spraw #1[\W],}%
196 \def\XINT_spraw::_e #1,#2{\XINT_spraw::_b {#2, #1}}%
```

10.20.4 `\xintIsTrue::csv`

```
197 \def\xintIsTrue::csv #1{\romannumeral0\expandafter\XINT_istrue::_a\romannumeral`&&@#1,^}%
198 \def\XINT_istrue::_a {\XINT_istrue::_b {}}%
199 \def\XINT_istrue::_b #1#2,{\expandafter\XINT_istrue::_c \romannumeral`&&@#2,{#1}}%
200 \def\XINT_istrue::_c #1{\if ,#1\xint_dothis\XINT_istrue::_e\fi
201     \if ^#1\xint_dothis\XINT::_end\fi
202     \xint_orthat\XINT_istrue::_d #1}%
203 \def\XINT_istrue::_d #1,{\expandafter\XINT_istrue::_e\romannumeral0\xintisnotzero {#1},}%
204 \def\XINT_istrue::_e #1,#2{\XINT_istrue::_b {#2, #1}}%
```

10.20.5 `\xintRound::csv`

```
205 \def\XINT::_end #1,#2#3{\xint_gobble_i #3}%
206 \def\xintRound::csv #1#2{\romannumeral0\expandafter\XINT_round::_b\expandafter
207     {\the\numexpr#1\expandafter}\expandafter{\expandafter}\romannumeral`&&@#2,^}%
208 \def\XINT_round::_b #1#2#3,{\expandafter\XINT_round::_c \romannumeral`&&@#3,{#1}{#2}}%
209 \def\XINT_round::_c #1{\if ,#1\xint_dothis\XINT_round::_e\fi
210     \if ^#1\xint_dothis\XINT::_end\fi
211     \xint_orthat\XINT_round::_d #1}%

```

```

212 \def\XINT_round::_d #1,#2{%
213     \expandafter\XINT_round::_e\romannumeral0\ifnum#2>\xint_c_
214     \expandafter\xintround\else\expandafter\xintiround\fi {#2}{#1},{#2}}%
215 \def\XINT_round::_e #1,#2#3{\XINT_round::_b {#2}{#3, #1}}%

```

10.20.6 \XINTinFloat::csv

```

216 \def\XINTinFloat::csv #1#2{\romannumeral0\expandafter\XINT_infloat::_b\expandafter
217     {\the\numexpr #1\expandafter}\expandafter{\expandafter}\romannumeral`&&@#2,^,%}
218 \def\XINT_infloat::_b #1#2#3,{\XINT_infloat::_c #3,{#1}{#2}}%
219 \def\XINT_infloat::_c #1{\if ,#1\xint_dothis\XINT_infloat::_e\fi
220     \if ^#1\xint_dothis\XINT_:::_end\fi
221     \xint_orthat\XINT_infloat::_d #1}%
222 \def\XINT_infloat::_d #1,#2%
223     {\expandafter\XINT_infloat::_e\romannumeral0\XINTinfloat [#2]{#1},{#2}}%
224 \def\XINT_infloat::_e #1,#2#3{\XINT_infloat::_b {#2}{#3, #1}}%

```

10.20.7 \xintPFloat::csv

```

225 \def\xintPFloat::csv #1#2{\romannumeral0\expandafter\XINT_pfloat::_b\expandafter
226     {\the\numexpr #1\expandafter}\expandafter{\expandafter}\romannumeral`&&@#2,^,%}
227 \def\XINT_pfloat::_b #1#2#3,{\expandafter\XINT_pfloat::_c \romannumeral`&&@#3,{#1}{#2}}%
228 \def\XINT_pfloat::_c #1{\if ,#1\xint_dothis\XINT_pfloat::_e\fi
229     \if ^#1\xint_dothis\XINT_:::_end\fi
230     \xint_orthat\XINT_pfloat::_d #1}%
231 \def\XINT_pfloat::_d #1,#2%
232     {\expandafter\XINT_pfloat::_e\romannumeral0\XINT_pfloat_opt [\xint:#2]{#1},{#2}}%
233 \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.

```

234 \def\XINT_expr_getnext #1%
235 {%
236     \expandafter\XINT_expr_getnext_a\romannumeral`&&@#1%
237 }%
238 \def\XINT_expr_getnext_a #1%
239 {% screens out sub-expressions and \count or \dimen registers/variables
240     \xint_gob_til_! #1\XINT_expr_subexpr !% recall this ! has catcode 11
241     \ifcat\relax#1% \count or \numexpr etc... token or count, dimen, skip cs
242         \expandafter\XINT_expr_countetc
243     \else
244         \expandafter\expandafter\expandafter\XINT_expr_getnextfork\expandafter\string
245         \fi
246     #1%
247 }%
248 \def\XINT_expr_subexpr !#1\fi !{\expandafter\XINT_expr_getop\xint_gobble_iii }%

```

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

```

249 \def\XINT_expr_countetc #1%
250 {%
251     \ifx\count#1\else\ifx\dimen#1\else\ifx\numexpr#1\else\ifx\dimexpr#1\else

```

```

252 \ifx\skip#1\else\ifx\glueexpr#1\else\ifx\fontdimen#1\else\ifx\ht#1\else
253 \ifx\dp#1\else\ifx\wd#1\else\ifx\fontcharht#1\else\ifx\fontcharwd#1\else
254 \ifx\fontchardp#1\else\ifx\fontcharic#1\else
255 \XINT_expr_unpackvar
256 \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
257 \expandafter\XINT_expr_getnext\number #1%
258 }%
259 \def\XINT_expr_unpackvar\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
260 \expandafter\XINT_expr_getnext\number #1%
261 {\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
262 \expandafter\XINT_expr_getop\csname .=\number#1\endcsname }%
263 \begingroup
264 \lccode`*=`#
265 \lowercase{\endgroup
266 \def\XINT_expr_getnextfork #1{%
267 \if#1*\xint_dothis {\XINT_expr_scan_macropar *}\fi
268 \if#1[\xint_dothis {\xint_c_xviii ({} }\fi
269 \if#1+\xint_dothis \XINT_expr_getnext \fi
270 \if#1.\xint_dothis {\XINT_expr_startdec}\fi
271 \if#1-\xint_dothis -\fi
272 \if#1(\xint_dothis {\xint_c_xviii ({} }\fi
273 \xint_orthat {\XINT_expr_scan_nbr_or_func #1}%
274 }}%
275 \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)	301
10.22.2	Fractional part	303
10.22.3	Scientific notation	303
10.22.4	Hexadecimal numbers	304
10.22.5	Parsing names of functions and variables	306

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|flexpr}_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.

```

276 \catcode96 11 % `
277 \def\XINT_expr_scan_nbr_or_func #1% this #1 has necessarily here catcode 12
278 {%
279   \if "#1\xint_dothis \XINT_expr_scanhex_I\fi
280   \if `#1\xint_dothis {\XINT_expr_onliteral_`}\fi
281   \ifnum \xint_c_ix<1#1 \xint_dothis \XINT_expr_startint\fi
282   \xint_orthat \XINT_expr_scanfunc #1%
283 }%
284 \def\XINT_expr_onliteral_` #1#2#3({\xint_c_xviii `{#2}}%
285 \catcode96 12 % `
286 \def\XINT_expr_startint #1%
287 {%
288   \if #10\expandafter\XINT_expr_gobz_a\else\XINT_expr_scanint_a\fi #1%
289 }%
290 \def\XINT_expr_scanint_a #1#2%
291   {\expandafter\XINT_expr_getop\curname.=#1%
292   \expandafter\XINT_expr_scanint_b\romannumeral`&&@#2}%
293 \def\XINT_expr_gobz_a #1%
294   {\expandafter\XINT_expr_getop\curname.=
295   \expandafter\XINT_expr_gobz_scanint_b\romannumeral`&&@#1}%
296 \def\XINT_expr_startdec #1%
297   {\expandafter\XINT_expr_getop\curname.=
298   \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.

```

299 \def\XINT_expr_scanint_b #1%
300 {%
301   \ifcat \relax #1\expandafter\XINT_expr_scanint_endbycs\expandafter #1\fi
302   \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanint_c\fi
303   \string#1\XINT_expr_scanint_d
304 }%
305 \def\XINT_expr_scanint_d #1%
306 {%
307   \expandafter\XINT_expr_scanint_b\romannumeral`&&@#1%
308 }%
309 \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.

Extended for 1.2l to ignore underscore character `_` if encountered within digits; so it can serve as separator for better readability.

```

310 \def\XINT_expr_scanint_c\string #1\XINT_expr_scanint_d
311 {%
312   \if _#1\xint_dothis\XINT_expr_scanint_d\fi
313   \if e#1\xint_dothis{[\the\numexpr0\XINT_expr_scanexp_a +}\fi
314   \if E#1\xint_dothis{[\the\numexpr0\XINT_expr_scanexp_a +}\fi
315   \if .#1\xint_dothis{\XINT_expr_startdec_a .}\fi
316   \xint_orthat {\endcsname #1}%
317 }%
318 \def\XINT_expr_startdec_a .#1%
319 {%
320   \expandafter\XINT_expr_scandec_a\romannumeral`&&@#1%
321 }%
322 \def\XINT_expr_scandec_a #1%
323 {%
324   \if .#1\xint_dothis{\endcsname.}\fi
325   \xint_orthat {\XINT_expr_scandec_b 0.#1}%
326 }%
327 \def\XINT_expr_gobz_scanint_b #1%
328 {%
329   \ifcat \relax #1\expandafter\XINT_expr_gobz_scanint_endbycs\expandafter #1\fi
330   \ifnum\xint_c_x<1\string#1 \else\expandafter\XINT_expr_gobz_scanint_c\fi
331   \string#1\XINT_expr_scanint_d
332 }%
333 \def\XINT_expr_gobz_scanint_endbycs#1#2\XINT_expr_scanint_d{0\endcsname #1}%
334 \def\XINT_expr_gobz_scanint_c\string #1\XINT_expr_scanint_d
335 {%
336   \if _#1\xint_dothis\XINT_expr_gobz_scanint_d\fi
337   \if e#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
338   \if E#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
339   \if .#1\xint_dothis{\XINT_expr_gobz_startdec_a .}\fi
340   \if 0#1\xint_dothis\XINT_expr_gobz_scanint_d\fi
341   \xint_orthat {0\endcsname #1}%
342 }%
343 \def\XINT_expr_gobz_scanint_d #1%
344 {%
345   \expandafter\XINT_expr_gobz_scanint_b\romannumeral`&&@#1%
346 }%
347 \def\XINT_expr_gobz_startdec_a .#1%
348 {%
349   \expandafter\XINT_expr_gobz_scandec_a\romannumeral`&&@#1%
350 }%
351 \def\XINT_expr_gobz_scandec_a #1%
352 {%
353   \if .#1\xint_dothis{0\endcsname.}\fi

```

```
354 \xint_orthat {\XINT_expr_gobz_scandec_b 0.#1}%
355 }%
```

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 ?

```
356 \def\XINT_expr_scandec_b #1.#2%
357 {%
358   \ifcat \relax #2\expandafter\XINT_expr_scandec_endbycs\expandafter#2\fi
359   \ifnum\xint_c_ix<1\string#2 \else\expandafter\XINT_expr_scandec_c\fi
360   \string#2\expandafter\XINT_expr_scandec_d\the\numexpr #1-\xint_c_i.%
361 }%
362 \def\XINT_expr_scandec_endbycs #1#2\XINT_expr_scandec_d
363   \the\numexpr#3-\xint_c_i.{[#3]\endcsname #1}%
364 \def\XINT_expr_scandec_d #1.#2%
365 {%
366   \expandafter\XINT_expr_scandec_b
367   \the\numexpr #1\expandafter.\romannumeral`&&@#2%
368 }%
369 \def\XINT_expr_scandec_c\string #1#2\the\numexpr#3-\xint_c_i.%
370 {%
371   \if _#1\xint_dothis{\XINT_expr_scandec_d#3.}\fi
372   \if e#1\xint_dothis{[\the\numexpr#3\XINT_expr_scanexp_a +}\fi
373   \if E#1\xint_dothis{[\the\numexpr#3\XINT_expr_scanexp_a +}\fi
374   \xint_orthat {[#3]\endcsname #1}%
375 }%
376 \def\XINT_expr_gobz_scandec_b #1.#2%
377 {%
378   \ifcat \relax #2\expandafter\XINT_expr_gobz_scandec_endbycs\expandafter#2\fi
379   \ifnum\xint_c_ix<1\string#2 \else\expandafter\XINT_expr_gobz_scandec_c\fi
380   \if0#2\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
381   {\expandafter\XINT_expr_gobz_scandec_b}%
382   {\string#2\expandafter\XINT_expr_scandec_d}\the\numexpr#1-\xint_c_i.%
383 }%
384 \def\XINT_expr_gobz_scandec_endbycs #1#2\xint_c_i.{0[0]\endcsname #1}%
385 \def\XINT_expr_gobz_scandec_c\if0#1#2\fi #3\numexpr#4-\xint_c_i.%
386 {%
387   \if _#1\xint_dothis{\XINT_expr_gobz_scandec_b #4.}\fi
388   \if e#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
389   \if E#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
390   \xint_orthat {0[0]\endcsname #1}%
391 }%
```

10.22.3 Scientific notation

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

```

392 \def\XINT_expr_scanexp_a #1#2%
393 {%
394   #1\expandafter\XINT_expr_scanexp_b\romannumeral`&&@#2%
395 }%
396 \def\XINT_expr_scanexp_b #1%
397 {%
398   \ifcat \relax #1\expandafter\XINT_expr_scanexp_endbycs\expandafter #1\fi
399   \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanexp_c\fi
400   \string#1\XINT_expr_scanexp_d
401 }%
402 \def\XINT_expr_scanexp_endbycs#1#2\XINT_expr_scanexp_d {}\endcsname #1}%
403 \def\XINT_expr_scanexp_d #1%
404 {%
405   \expandafter\XINT_expr_scanexp_bb\romannumeral`&&@#1%
406 }%
407 \def\XINT_expr_scanexp_c\string #1\XINT_expr_scanexp_d
408 {%
409   \if _#1\xint_dothis \XINT_expr_scanexp_d \fi
410   \if +#1\xint_dothis {\XINT_expr_scanexp_a +}\fi
411   \if -#1\xint_dothis {\XINT_expr_scanexp_a -}\fi
412   \xint_orthat {}\endcsname #1}%
413 }%
414 \def\XINT_expr_scanexp_bb #1%
415 {%
416   \ifcat \relax #1\expandafter\XINT_expr_scanexp_endbycs_b\expandafter #1\fi
417   \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanexp_cb\fi
418   \string#1\XINT_expr_scanexp_db
419 }%
420 \def\XINT_expr_scanexp_endbycs_b#1#2\XINT_expr_scanexp_db {}\endcsname #1}%
421 \def\XINT_expr_scanexp_db #1%
422 {%
423   \expandafter\XINT_expr_scanexp_bb\romannumeral`&&@#1%
424 }%
425 \def\XINT_expr_scanexp_cb\string #1\XINT_expr_scanexp_db
426 {%
427   \if _#1\xint_dothis\XINT_expr_scanexp_d\fi
428   \xint_orthat{}\endcsname #1}%
429 }%

```

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`).

Extended for 1.2l to ignore underscore character `_` if encountered within digits.

```

430 \def\XINT_expr_scanhex_I #1% #1="
431 {%
432   \expandafter\XINT_expr_getop\csname.=\expandafter
433   \XINT_expr_unlock_hex_in\csname.=\XINT_expr_scanhexI_a
434 }%

```



```

435 \def\XINT_expr_scanhexI_a #1%
436 {%
437   \ifcat #1\relax\xint_dothis{.>\endcsname\endcsname #1}\fi
438   \ifx !#1\xint_dothis{.>\endcsname\endcsname !}\fi
439   \xint_orthat {\expandafter\XINT_expr_scanhexI_aa\string #1}%
440 }%
441 \def\XINT_expr_scanhexI_aa #1%
442 {%
443   \if\ifnum`#1>`/
444     \ifnum`#1>`9
445     \ifnum`#1>`@
446     \ifnum`#1>`F
447     0\else1\fi\else0\fi\else1\fi\else0\fi 1%
448     \expandafter\XINT_expr_scanhexI_b
449   \else
450     \if_#1\xint_dothis{\expandafter\XINT_expr_scanhexI_bgob}\fi
451     \if .#1\xint_dothis{\expandafter\XINT_expr_scanhex_transition}\fi
452     \xint_orthat % gather what we got so far, leave catcode 12 #1 in stream
453     {\xint_afterfi {.>\endcsname\endcsname}}%
454   \fi
455   #1%
456 }%
457 \def\XINT_expr_scanhexI_b #1#2%
458 {%
459   #1\expandafter\XINT_expr_scanhexI_a\romannumeral`&&@#2%
460 }%
461 \def\XINT_expr_scanhexI_bgob #1#2%
462 {%
463   \expandafter\XINT_expr_scanhexI_a\romannumeral`&&@#2%
464 }%
465 \def\XINT_expr_scanhex_transition .#1%
466 {%
467   \expandafter.\expandafter.\expandafter
468   \XINT_expr_scanhexII_a\romannumeral`&&@#1%
469 }%
470 \def\XINT_expr_scanhexII_a #1%
471 {%
472   \ifcat #1\relax\xint_dothis{\endcsname\endcsname#1}\fi
473   \ifx !#1\xint_dothis{\endcsname\endcsname !}\fi
474   \xint_orthat {\expandafter\XINT_expr_scanhexII_aa\string #1}%
475 }%
476 \def\XINT_expr_scanhexII_aa #1%
477 {%
478   \if\ifnum`#1>`/
479     \ifnum`#1>`9
480     \ifnum`#1>`@
481     \ifnum`#1>`F
482     0\else1\fi\else0\fi\else1\fi\else0\fi 1%
483     \expandafter\XINT_expr_scanhexII_b
484   \else
485     \if_#1\xint_dothis{\expandafter\XINT_expr_scanhexII_bgob}\fi
486     \xint_orthat{\xint_afterfi {\endcsname\endcsname}}%

```

```

487 \fi
488 #1%
489 }%
490 \def\XINT_expr_scanhexII_b #1#2%
491 {%
492   #1\expandafter\XINT_expr_scanhexII_a\romannumeral`&&@#2%
493 }%
494 \def\XINT_expr_scanhexII_bgob #1#2%
495 {%
496   \expandafter\XINT_expr_scanhexII_a\romannumeral`&&@#2%
497 }%

```

10.22.5 Parsing names of functions and variables

```

498 \def\XINT_expr_scanfunc
499 {%
500   \expandafter\XINT_expr_func\romannumeral`&&\XINT_expr_scanfunc_a
501 }%
502 \def\XINT_expr_scanfunc_a #1#2%
503 {%
504   \expandafter #1\romannumeral`&&\expandafter\XINT_expr_scanfunc_b\romannumeral`&&@#2%
505 }%

```

This handles: 1) (indirectly) tacit multiplication by a variable in front of a 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.

```

506 \def\XINT_expr_scanfunc_b #1%
507 {%
508   \ifx !#1\xint_dothis{(_)\fi
509   \ifcat \relax#1\xint_dothis{(_)\fi
510   \if (#1\xint_dothis{\xint_firstoftwo{(`)}\fi
511   \if @#1\xint_dothis \XINT_expr_scanfunc_a \fi
512   \if _#1\xint_dothis \XINT_expr_scanfunc_a \fi
513   \ifnum \xint_c_ix<1\string#1 \xint_dothis \XINT_expr_scanfunc_a \fi
514   \ifcat a#1\xint_dothis \XINT_expr_scanfunc_a \fi
515   \xint_orthat {(_}%
516   #1%
517 }%

```

Comments written 2015/11/12: earlier there was an `\ifcsname` test for checking if we had a variable in front of a (, for tacit multiplication for example in $x(y+z(x+w))$ to work. But after I had implemented functions (that was yesterday...), I had the problem if was impossible to re-declare a variable name such as "f" as a function name. The problem is that here we can not test if the function is available because we don't know if we are in `expr`, `iiexpr` or `floatexpr`. The `\xint_c_xviii` causes all fetching operations to stop and control is handed over to the routines which will be `expr`, `iiexpr` ou `floatexpr` specific, i.e. the `\XINT_{expr|iiexpr|flexpr}_op_{`|_}` which are invoked by the `until_{op}>_b` macros earlier in the stream. Functions may exist for one but not the two other parsers. Variables are declared via one parser and usable in the others, but naturally `\xintiexpr` has its restrictions.

Thinking about this again I decided to treat a priori cases such as $x(...)$ as functions, after having assigned to each variable a low-weight macro which will convert this into `_getop\.=<value of x>*(...)`. To activate that macro at the right time I could for this exploit the "onliteral" 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.

```
518 \def\XINT_expr_func #1(#2{\if _#2\xint_dothis\XINT_expr_op__\fi
519 \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 mutiplication 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 $x/2\max(3,5)$, 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.

1.2q adds tacit multiplication in cases such as $(1+1)3$ or $5!7!$

```
520 \def\XINT_expr_getop #1#2% this #1 is the current locked computed value
521 {%
522   \expandafter\XINT_expr_getop_a\expandafter #1\romannumeral`&&@#2%
523 }%
524 \catcode`* 11
525 \def\XINT_expr_getop_a #1#2%
526 {%
527   \ifx \relax #2\xint_dothis\xint_firstofthree\fi
528   \ifcat \relax #2\xint_dothis\xint_secondofthree\fi
529   \ifnum\xint_c_ix<1\string#2 \xint_dothis\xint_secondofthree\fi
```

```

530 \if _#2\xint_dothis \xint_secondofthree\fi
531 \if @#2\xint_dothis \xint_secondofthree\fi
532 \if (#2\xint_dothis \xint_secondofthree\fi
533 \ifcat a#2\xint_dothis \xint_secondofthree\fi
534 \xint_orthat \xint_thirdofthree
535 {\XINT_expr_founded #1}%
536 {\XINT_expr_precedence_*** *#1#2}% tacit multiplication with higher precedence
537 {\expandafter\XINT_expr_getop_b \string#2#1}%
538 }%
539 \catcode`* 12
540 \def\XINT_expr_founded {\xint_c_ \relax }% \relax is a place holder here.

```

? is a very special operator with top precedence which will check if the next token is another ?, while avoiding removing a brace pair from token stream due to its syntax. Pre 1.1 releases used : rather than ??, but we need : for Python like slices of lists.

```

541 \def\XINT_expr_getop_b #1%
542 {%
543 \if '#1\xint_dothis{\XINT_expr_binopwrdrd } \fi
544 \if ?#1\xint_dothis{\XINT_expr_precedence_? ?} \fi
545 \xint_orthat {\XINT_expr_scanop_a #1}%
546 }%
547 \def\XINT_expr_binopwrdrd #1#2'\{\expandafter\XINT_expr_foundedop_a
548 \csname XINT_expr_itself_\xint_zapspace #2 \xint_gobble_i\endcsname #1}%
549 \def\XINT_expr_scanop_a #1#2#3%
550 {\expandafter\XINT_expr_scanop_b\expandafter #1\expandafter #2\romannumeral`&&@#3}%
551 \def\XINT_expr_scanop_b #1#2#3%
552 {%
553 \ifcat#3\relax\xint_dothis{\XINT_expr_foundedop_a #1#2#3}\fi
554 \ifcsname XINT_expr_itself_#1#3\endcsname
555 \xint_dothis
556 {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#3\endcsname #2}\fi
557 \xint_orthat {\XINT_expr_foundedop_a #1#2#3}%
558 }%
559 \def\XINT_expr_scanop_c #1#2#3%
560 {%
561 \expandafter\XINT_expr_scanop_d\expandafter #1\expandafter #2\romannumeral`&&@#3%
562 }%
563 \def\XINT_expr_scanop_d #1#2#3%
564 {%
565 \ifcat#3\relax \xint_dothis{\XINT_expr_foundedop #1#2#3}\fi
566 \ifcsname XINT_expr_itself_#1#3\endcsname
567 \xint_dothis
568 {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#3\endcsname #2}\fi
569 \xint_orthat {\csname XINT_expr_precedence_#1\endcsname #1#2#3}%
570 }%
571 \def\XINT_expr_foundedop_a #1%
572 {%
573 \ifcsname XINT_expr_precedence_#1\endcsname
574 \csname XINT_expr_precedence_#1\endcsname\expandafter\endcsname
575 \expandafter #1%
576 \else
577 \xint_afterfi{\XINT_expr_unknown_operator {#1}\XINT_expr_getop}%

```

```

578   \fi
579 }%
580 \def\XINT_expr_unknown_operator #1{\xintError:removed \xint_gobble_i {#1}}%
581 \def\XINT_expr_foundedop #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.

```

582 \catcode`) 11
583 \def\XINT_tmpa #1#2#3#4%
584 {%
585   \def#1##1%
586   {%
587     \xint_UDsignfork
588       ##1{\expandafter#1\romannumeral`&&@#3}%
589       -{#2##1}%
590     \krof
591   }%
592   \def#2##1##2%
593   {%
594     \ifcase ##1\expandafter\XINT_expr_done
595     \or\xint_afterfi{\XINT_expr_extra_)
596       \expandafter #1\romannumeral`&&\XINT_expr_getop }%
597     \else
598     \xint_afterfi{\expandafter#1\romannumeral`&&\csname XINT_#4_op_##2\endcsname }%
599     \fi
600   }%
601 }%
602 \def\XINT_expr_extra_) {\xintError:removed }%
603 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
604   \expandafter\XINT_tmpa
605   \csname XINT_#1_until_end_a\expandafter\endcsname
606   \csname XINT_#1_until_end_b\expandafter\endcsname
607   \csname XINT_#1_op_-vi\endcsname
608   {#1}%
609 }%
610 \def\XINT_tmpa #1#2#3#4#5#6%
611 {%
612   \def #1##1{\expandafter #3\romannumeral`&&\XINT_expr_getnext }%
613   \def #2{\expandafter #3\romannumeral`&&\XINT_expr_getnext }%
614   \def #3##1{\xint_UDsignfork
615     ##1{\expandafter #3\romannumeral`&&@#5}%
616     -{#4##1}%
617     \krof }%
618   \def #4##1##2{\ifcase ##1\expandafter\XINT_expr_missing_)
619   \or \csname XINT_#6_op_##2\expandafter\endcsname
620   \else
621   \xint_afterfi{\expandafter #3\romannumeral`&&\csname XINT_#6_op_##2\endcsname }%
622   \fi
623 }%

```

```

624 }%
625 \def\XINT_expr_missing_) {\xintError:inserted \xint_c_ \XINT_expr_done }%

```

We should be using `until_` notation to stay synchronous with `until_+`, `until_*` etc..., but I found that `until_` was more telling.

```

626 \catcode` ) 12
627 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
628   \expandafter\XINT_tmpa
629   \csname XINT_#1_op_(\expandafter\endcsname
630   \csname XINT_#1_oparen\expandafter\endcsname
631   \csname XINT_#1_until_)_a\expandafter\endcsname
632   \csname XINT_#1_until_)_b\expandafter\endcsname
633   \csname XINT_#1_op_-vi\endcsname
634   {#1}%
635 }%
636 \expandafter\let\csname XINT_expr_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	310
10.25.2	The , &, xor, <, >, =, <=, >=, !=, //, /:, .., +, -, *, /, ^, ..[, and].. operators for expr, floatexpr and iiexpr operators	311
10.25.3	The]+,]-,]*,]/,]^, +[, -[, *[, /[, and ^[list operators	313
10.25.4	The 'and', 'or', 'xor', and 'mod' as infix operator words	315
10.25.5	The , &&, **, **[,]** operators as synonyms	316

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.

```

637 \expandafter\let\csname XINT_expr_precedence_]\endcsname\xint_c_i
638 \expandafter\let\csname XINT_expr_precedence_;\endcsname\xint_c_i
639 \let\XINT_expr_precedence_a \xint_c_xviii
640 \let\XINT_expr_precedence_!? \xint_c_ii
641 \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.)

```

642 \catcode`. 11 \catcode`= 11 \catcode`+ 11
643 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
644     \expandafter\let\csname XINT_#1_op_)\endcsname \XINT_expr_getop
645     \expandafter\let\csname XINT_#1_op_;\endcsname \space
646     \expandafter\def\csname XINT_#1_op_]\endcsname ##1{\XINT_expr_scanop_a a##1}%
647     \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 dispendious, but I need the starting value in a `\numexpr` compatible form in the iteration loops.

```

648 \expandafter\def\csname XINT_#1_op_++\endcsname ##1##2\relax
649 {\expandafter\XINT_expr_foundedend \expandafter
650   {\expandafter\.=\csname .=\xintiCeil{\XINT_expr_unlock ##1}\endcsname }}%
651 }%
652 \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.

```

653 \catcode\& 12
654 \xintFor* #1 in {{==}{<=}{>=}{!=}{&&}{||}{**}{//}{/:}{..}{..[]}{.}{.}%
655      {+}{-}{-[]}{*}{/}{/[]}{**}[^[]}{a+}{a-}{a*}{a/}{a**}{a^}%
656      [][]{:}{:[]}{!}?{++}{++}}}%{***}}
657 \do {\expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}}%
658 \catcode\& 7
659 \expandafter\let\csname XINT_expr_precedence_***\endcsname \xint_c_viii

```

10.25.2 The |, &, xor, <, >, =, <=, >=, !=, //, /:, .., +, -, *, /, ^, ..[, and].. operators for expr, floatexpr and iixpr operators

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

```

660 \def\XINT_expr_defbin_c #1#2#3#4#5#6#7#8%
661 {%
662   \def #1##1% \XINT_expr_op_<op> ou flexpr ou iiexpr
663   {% keep value, get next number and operator, then do until
664     \expandafter #2\expandafter ##1%
665     \romannumeral`&&@\expandafter\XINT_expr_getnext }%
666   \def #2##1##2% \XINT_expr_until_<op>_a ou flexpr ou iiexpr
667   {\xint_UDsignfork ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
668     -{#3##1##2}%
669     \krof }%
670   \def #3##1##2##3##4% \XINT_expr_until_<op>_b ou flexpr ou iiexpr

```

```

671  {% either execute next operation now, or first do next (possibly unary)
672    \ifnum ##2>#7%
673      \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
674        \csname XINT_#8_op_##3\endcsname {##4}}%
675      \else \xint_afterfi {\expandafter ##2\expandafter ##3%
676        \csname .=#6{\XINT_expr_unlock ##1}{\XINT_expr_unlock ##4}\endcsname }%
677      \fi }%
678  \let #7#5%
679 }%
680 \def\XINT_expr_defbin_b #1#2#3#4#5%
681 {%
682   \expandafter\XINT_expr_defbin_c
683   \csname XINT_#1_op_#2\expandafter\endcsname
684   \csname XINT_#1_until_#2_a\expandafter\endcsname
685   \csname XINT_#1_until_#2_b\expandafter\endcsname
686   \csname XINT_#1_op_#4\expandafter\endcsname
687   \csname xint_c_#3\expandafter\endcsname
688   \csname #5\expandafter\endcsname
689   \csname XINT_expr_precedence_#2\endcsname {#1}%
690 }%
691 \XINT_expr_defbin_b {expr}    |   {iii}{vi} {xintOR}%
692 \XINT_expr_defbin_b {flexpr} |   {iii}{vi} {xintOR}%
693 \XINT_expr_defbin_b {iiexpr} |   {iii}{vi} {xintOR}%
694 \XINT_expr_defbin_b {expr}    &   {iv}{vi}  {xintAND}%
695 \XINT_expr_defbin_b {flexpr} &   {iv}{vi}  {xintAND}%
696 \XINT_expr_defbin_b {iiexpr} &   {iv}{vi}  {xintAND}%
697 \XINT_expr_defbin_b {expr}    {xor}{iii}{vi} {xintXOR}%
698 \XINT_expr_defbin_b {flexpr} {xor}{iii}{vi} {xintXOR}%
699 \XINT_expr_defbin_b {iiexpr} {xor}{iii}{vi} {xintXOR}%
700 \XINT_expr_defbin_b {expr}    <   {v}{vi}   {xintLt}%
701 \XINT_expr_defbin_b {flexpr} <   {v}{vi}   {xintLt}%
702 \XINT_expr_defbin_b {iiexpr} <   {v}{vi}   {xintiiLt}%
703 \XINT_expr_defbin_b {expr}    >   {v}{vi}   {xintGt}%
704 \XINT_expr_defbin_b {flexpr} >   {v}{vi}   {xintGt}%
705 \XINT_expr_defbin_b {iiexpr} >   {v}{vi}   {xintiiGt}%
706 \XINT_expr_defbin_b {expr}    =   {v}{vi}   {xintEq}%
707 \XINT_expr_defbin_b {flexpr} =   {v}{vi}   {xintEq}%
708 \XINT_expr_defbin_b {iiexpr} =   {v}{vi}   {xintiiEq}%
709 \XINT_expr_defbin_b {expr}    {<=} {v}{vi}   {xintLtorEq}%
710 \XINT_expr_defbin_b {flexpr} {<=} {v}{vi}   {xintLtorEq}%
711 \XINT_expr_defbin_b {iiexpr} {<=} {v}{vi}   {xintiiLtorEq}%
712 \XINT_expr_defbin_b {expr}    {>=} {v}{vi}   {xintGtorEq}%
713 \XINT_expr_defbin_b {flexpr} {>=} {v}{vi}   {xintGtorEq}%
714 \XINT_expr_defbin_b {iiexpr} {>=} {v}{vi}   {xintiiGtorEq}%
715 \XINT_expr_defbin_b {expr}    {!=} {v}{vi}   {xintNotEq}%
716 \XINT_expr_defbin_b {flexpr} {!=} {v}{vi}   {xintNotEq}%
717 \XINT_expr_defbin_b {iiexpr} {!=} {v}{vi}   {xintiiNotEq}%
718 \XINT_expr_defbin_b {expr}    {...} {iii}{vi} {xintSeq::csv}%
719 \XINT_expr_defbin_b {flexpr} {...} {iii}{vi} {xintSeq::csv}%
720 \XINT_expr_defbin_b {iiexpr} {...} {iii}{vi} {xintiiSeq::csv}%
721 \XINT_expr_defbin_b {expr}    {//} {vii}{vii}{xintDivFloor}% CHANGED IN 1.2p!
722 \XINT_expr_defbin_b {flexpr} {//} {vii}{vii}{XINTinFloatDivFloor}% "
```



```

723 \XINT_expr_defbin_b {iiexpr}{//} {vii}{vii}{xintiiDivFloor}% "
724 \XINT_expr_defbin_b {expr} {/:} {vii}{vii}{xintMod}% "
725 \XINT_expr_defbin_b {flexpr}{/:} {vii}{vii}{XINTinFloatMod}% "
726 \XINT_expr_defbin_b {iiexpr}{/:} {vii}{vii}{xintiiMod}% "
727 \XINT_expr_defbin_b {expr} + {vi}{vi} {xintAdd}%
728 \XINT_expr_defbin_b {flexpr} + {vi}{vi} {XINTinFloatAdd}%
729 \XINT_expr_defbin_b {iiexpr} + {vi}{vi} {xintiiAdd}%
730 \XINT_expr_defbin_b {expr} - {vi}{vi} {xintSub}%
731 \XINT_expr_defbin_b {flexpr} - {vi}{vi} {XINTinFloatSub}%
732 \XINT_expr_defbin_b {iiexpr} - {vi}{vi} {xintiiSub}%
733 \XINT_expr_defbin_b {expr} * {vii}{vii}{xintMul}%
734 \XINT_expr_defbin_b {flexpr} * {vii}{vii}{XINTinFloatMul}%
735 \XINT_expr_defbin_b {iiexpr} * {vii}{vii}{xintiiMul}%
736 \XINT_expr_defbin_b {expr} / {vii}{vii}{xintDiv}%
737 \XINT_expr_defbin_b {flexpr} / {vii}{vii}{XINTinFloatDiv}%
738 \XINT_expr_defbin_b {iiexpr} / {vii}{vii}{xintiiDivRound}% CHANGED IN 1.1!
739 \XINT_expr_defbin_b {expr} ^ {ix}{ix} {xintPow}%
740 \XINT_expr_defbin_b {flexpr} ^ {ix}{ix} {XINTinFloatPowerH}%
741 \XINT_expr_defbin_b {iiexpr} ^ {ix}{ix} {xintiiPow}%
742 \XINT_expr_defbin_b {expr} {..[]}{iii}{vi} {xintSeqA::csv}%
743 \XINT_expr_defbin_b {flexpr}{..[]}{iii}{vi} {XINTinFloatSeqA::csv}%
744 \XINT_expr_defbin_b {iiexpr}{..[]}{iii}{vi} {xintiiSeqA::csv}%
745 \XINT_expr_defbin_b {expr} {[]}{}{iii}{vi} {xintSeqB::csv}%
746 \XINT_expr_defbin_b {flexpr}{[]}{}{iii}{vi} {XINTinFloatSeqB::csv}%
747 \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`).

```

748 \def\XINT_expr_binop_inline_a
749   {\expandafter\xint_gobble_i\romannumeral`&&\XINT_expr_binop_inline_b}%
750 \def\XINT_expr_binop_inline_b #1#2,{\XINT_expr_binop_inline_c #2,{#1}}%
751 \def\XINT_expr_binop_inline_c #1{%
752   \if ,#1\xint_dothis\XINT_expr_binop_inline_e\fi
753   \if ^#1\xint_dothis\XINT_expr_binop_inline_end\fi
754   \xint_orthat\XINT_expr_binop_inline_d #1}%
755 \def\XINT_expr_binop_inline_d #1,#2{,#2{#1}\XINT_expr_binop_inline_b {#2}}%
756 \def\XINT_expr_binop_inline_e #1,#2{,\XINT_expr_binop_inline_b {#2}}%
757 \def\XINT_expr_binop_inline_end #1,#2{}%
758 \def\XINT_expr_deflistopr_c #1#2#3#4#5#6#7#8%
759 {%
760   \def #1#1% \XINT_expr_op_<op> ou flexpr ou iiexpr
761   {% keep value, get next number and operator, then do until
762     \expandafter #2\expandafter ##1%
763     \romannumeral`&&\expandafter\XINT_expr_getnext}%
764   \def #2##1##2% \XINT_expr_until_<op>_a ou flexpr ou iiexpr
765   {\xint_UDsignfork ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
766     -{##1##2}}%
767   \krof}%
768   \def #3##1##2##3##4% \XINT_expr_until_<op>_b ou flexpr ou iiexpr
769   {% either execute next operation now, or first do next (possibly unary)

```

```

770 \ifnum ##2>#7%
771 \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
772 \csname XINT_#8_op_##3\endcsname {##4}}%
773 \else \xint_afterfi {\expandafter ##2\expandafter ##3%
774 \csname .=\expandafter\XINT_expr_binop_inline_a\expandafter
775 {\expandafter\expandafter\expandafter#6\expandafter
776 \xint_exchangetwo_keepbraces\expandafter
777 {\expandafter\XINT_expr_unlock\expandafter ##4\expandafter}\expandafter}%
778 \romannumeral`&&\XINT_expr_unlock ##1,^,\endcsname }%
779 \fi }%
780 \let #7#5%
781 }%
782 \def\XINT_expr_deflistopr_b #1#2#3#4%
783 {%
784 \expandafter\XINT_expr_deflistopr_c
785 \csname XINT_#1_op_#2\expandafter\endcsname
786 \csname XINT_#1_until_#2_a\expandafter\endcsname
787 \csname XINT_#1_until_#2_b\expandafter\endcsname
788 \csname XINT_#1_op_#3\expandafter\endcsname
789 \csname xint_c_#3\expandafter\endcsname
790 \csname #4\expandafter\endcsname
791 \csname XINT_expr_precedence_#2\endcsname {#1}%
792 }%

```

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

```

793 \XINT_expr_deflistopr_b {expr} {a+}{vi} {xintAdd}%
794 \XINT_expr_deflistopr_b {expr} {a-}{vi} {xintSub}%
795 \XINT_expr_deflistopr_b {expr} {a*}{vii} {xintMul}%
796 \XINT_expr_deflistopr_b {expr} {a/}{vii} {xintDiv}%
797 \XINT_expr_deflistopr_b {expr} {a^}{ix} {xintPow}%
798 \XINT_expr_deflistopr_b {iiexpr}{a+}{vi} {xintiiAdd}%
799 \XINT_expr_deflistopr_b {iiexpr}{a-}{vi} {xintiiSub}%
800 \XINT_expr_deflistopr_b {iiexpr}{a*}{vii} {xintiiMul}%
801 \XINT_expr_deflistopr_b {iiexpr}{a/}{vii} {xintiiDivRound}%
802 \XINT_expr_deflistopr_b {iiexpr}{a^}{ix} {xintiiPow}%
803 \XINT_expr_deflistopr_b {flexpr}{a+}{vi} {XINTinFloatAdd}%
804 \XINT_expr_deflistopr_b {flexpr}{a-}{vi} {XINTinFloatSub}%
805 \XINT_expr_deflistopr_b {flexpr}{a*}{vii} {XINTinFloatMul}%
806 \XINT_expr_deflistopr_b {flexpr}{a/}{vii} {XINTinFloatDiv}%
807 \XINT_expr_deflistopr_b {flexpr}{a^}{ix} {XINTinFloatPowerH}%
808 \def\XINT_expr_deflistopl_c #1#2#3#4#5#6#7%
809 {%
810 \def #1#1{\expandafter#2\expandafter##1\romannumeral`&&@%
811 \expandafter #3\romannumeral`&&\XINT_expr_getnext }%
812 \def #2##1##2##3##4%
813 {% either execute next operation now, or first do next (possibly unary)
814 \ifnum ##2>#6%
815 \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
816 \csname XINT_#7_op_##3\endcsname {##4}}%
817 \else \xint_afterfi {\expandafter ##2\expandafter ##3%
818 \csname .=\expandafter\XINT_expr_binop_inline_a\expandafter

```

```

819      {\expandafter#5\expandafter
820      {\expandafter\XINT_expr_unlock\expandafter ##1\expandafter}\expandafter}%
821      \romannumeral`&&\XINT_expr_unlock ##4,^,\endcsname }%
822      \fi }%
823      \let #6#4%
824 }%
825 \def\XINT_expr_deflistopl_b #1#2#3#4%
826 {%
827   \expandafter\XINT_expr_deflistopl_c
828   \csname XINT_#1_op_#2\expandafter\endcsname
829   \csname XINT_#1_until_#2\expandafter\endcsname
830   \csname XINT_#1_until_)_a\expandafter\endcsname
831   \csname xint_c_#3\expandafter\endcsname
832   \csname #4\expandafter\endcsname
833   \csname XINT_expr_precedence_#2\endcsname {#1}%
834 }%

```

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

```

835 \XINT_expr_deflistopl_b {expr} {+[]\{vi\} {xintAdd}%
836 \XINT_expr_deflistopl_b {expr} {-[]\{vi\} {xintSub}%
837 \XINT_expr_deflistopl_b {expr} {*[]\{vii\} {xintMul}%
838 \XINT_expr_deflistopl_b {expr} {/[]\{vii\} {xintDiv}%
839 \XINT_expr_deflistopl_b {expr} {^[]\{ix\} {xintPow}%
840 \XINT_expr_deflistopl_b {iiexpr}{+[]\{vi\} {xintiiAdd}%
841 \XINT_expr_deflistopl_b {iiexpr}{-[]\{vi\} {xintiiSub}%
842 \XINT_expr_deflistopl_b {iiexpr}{*[]\{vii\} {xintiiMul}%
843 \XINT_expr_deflistopl_b {iiexpr}{/[]\{vii\} {xintiiDivRound}%
844 \XINT_expr_deflistopl_b {iiexpr}{^[]\{ix\} {xintiiPow}%
845 \XINT_expr_deflistopl_b {flexpr}{+[]\{vi\} {XINTinFloatAdd}%
846 \XINT_expr_deflistopl_b {flexpr}{-[]\{vi\} {XINTinFloatSub}%
847 \XINT_expr_deflistopl_b {flexpr}{*[]\{vii\} {XINTinFloatMul}%
848 \XINT_expr_deflistopl_b {flexpr}{/[]\{vii\} {XINTinFloatDiv}%
849 \XINT_expr_deflistopl_b {flexpr}{^[]\{ix\} {XINTinFloatPowerH}%

```

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

```

850 \xintFor #1 in {and,or,xor,mod} \do {%
851   \expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}}%
852 \expandafter\let\csname XINT_expr_precedence_and\expandafter\endcsname
853   \csname XINT_expr_precedence_&\endcsname
854 \expandafter\let\csname XINT_expr_precedence_or\expandafter\endcsname
855   \csname XINT_expr_precedence_|\endcsname
856 \expandafter\let\csname XINT_expr_precedence_mod\expandafter\endcsname
857   \csname XINT_expr_precedence_/\endcsname
858 \xintFor #1 in {expr, flexpr, iiexpr} \do {%
859   \expandafter\let\csname XINT_#1_op_and\expandafter\endcsname
860     \csname XINT_#1_op_&\endcsname
861   \expandafter\let\csname XINT_#1_op_or\expandafter\endcsname
862     \csname XINT_#1_op_|\endcsname
863   \expandafter\let\csname XINT_#1_op_mod\expandafter\endcsname
864     \csname XINT_#1_op_/\endcsname
865 }%

```

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

```

866 \expandafter\let\csname XINT_expr_precedence_==\expandafter\endcsname
867       \csname XINT_expr_precedence_=\endcsname
868 \expandafter\let\csname XINT_expr_precedence_&\string&\expandafter\endcsname
869       \csname XINT_expr_precedence_&\endcsname
870 \expandafter\let\csname XINT_expr_precedence_||\expandafter\endcsname
871       \csname XINT_expr_precedence_||\endcsname
872 \expandafter\let\csname XINT_expr_precedence_**\expandafter\endcsname
873       \csname XINT_expr_precedence_^\endcsname
874 \expandafter\let\csname XINT_expr_precedence_a**\expandafter\endcsname
875       \csname XINT_expr_precedence_a^\endcsname
876 \expandafter\let\csname XINT_expr_precedence_**[\expandafter\endcsname
877       \csname XINT_expr_precedence_^\endcsname
878 \xintFor #1 in {expr, fexpr, iexpr} \do {%
879   \expandafter\let\csname XINT_#1_op_==\expandafter\endcsname
880       \csname XINT_#1_op_=\endcsname
881   \expandafter\let\csname XINT_#1_op_&\string&\expandafter\endcsname
882       \csname XINT_#1_op_&\endcsname
883   \expandafter\let\csname XINT_#1_op_||\expandafter\endcsname
884       \csname XINT_#1_op_||\endcsname
885   \expandafter\let\csname XINT_#1_op_**\expandafter\endcsname
886       \csname XINT_#1_op_^\endcsname
887   \expandafter\let\csname XINT_#1_op_a**\expandafter\endcsname
888       \csname XINT_#1_op_a^\endcsname
889   \expandafter\let\csname XINT_#1_op_**[\expandafter\endcsname
890       \csname XINT_#1_op_^\endcsname
891 }%
```

10.26 Macros for list selectors: [list][N], [list][:b], [list][a:], [list][a:b]

10.26.1	<code>\xintListSel:x:csv</code>	318
10.26.2	<code>\xintListSel:f:csv</code>	319
10.26.3	<code>\xintKeep:x:csv</code>	320
10.26.4	<code>\xintKeep:f:csv</code>	321
10.26.5	<code>\xintTrim:f:csv</code>	321
10.26.6	<code>\xintNthEltPy:f:csv</code>	321
10.26.7	<code>\xintLength:f:csv</code>	321
10.26.8	<code>\xintReverse:f:csv</code>	321

Python slicing was first implemented for 1.1 (27 octobre 2014). But it used `\xintCSVtoList` and `\xintListWithSep{,}` to convert back and forth to token lists for use of `\xintKeep`, `\xintTrim`, `\xintNthElt`. Not very efficient! Also `[list][a:b]` was Python like but not `[list][N]` which counted items starting at one, and returned the length for `N=0`.

Release 1.2g changed this so `[list][N]` now counts starting at zero and `len(list)` computes the number of items. Also 1.2g had its own f-expandable macros handling directly the comma separated lists. They are located into `xinttools.sty`.

1.2j improved the `xinttools.sty` macros and furthermore it made the Python slicing in expressions a bit more efficient still by exploiting in some cases that expansion happens in `\csname...\endcsname` and does not have to be f-expandable. But the f-expandable variants must be kept for use by `\xintNewExpr` and `\xintdeffunc`.

```

892 \def\xINT_tmpa #1#2#3#4#5#6%
893 {%
```

```

894 \def #1##1% \XINT_expr_op_][
895 {%
896   \expandafter #2\expandafter ##1\romannumeral`&&\XINT_expr_getnext
897 }%
898 \def #2##1##2% \XINT_expr_until_][_a
899 {\xint_UDsignfork
900   ##2{\expandafter #2\expandafter ##1\romannumeral`&&#4}%
901   -{#3##1##2}%
902   \krof }%
903 \def #3##1##2##3##4% \XINT_expr_until_][_b
904 {%
905   \ifnum ##2>#5%
906     \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&%
907                   \csname XINT_#6_op_##3\endcsname {##4}}%
908   \else
909     \xint_afterfi
910     {\expandafter ##2\expandafter ##3\csname
911      .=\expandafter\xintListSel:x:csv % will be \xintListSel:f:csv in \xintNewExpr output
912      \romannumeral`&&\XINT_expr_unlock ##4;% selector
913      \XINT_expr_unlock ##1;\endcsname % unlock already pre-positioned for \xintNewExpr
914      }%
915     \fi
916   }%
917   \let #5\xint_c_ii
918 }%
919 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
920 \expandafter\XINT_tmpa
921   \csname XINT_#1_op_][\expandafter\endcsname
922   \csname XINT_#1_until_][_a\expandafter\endcsname
923   \csname XINT_#1_until_][_b\expandafter\endcsname
924   \csname XINT_#1_op_-vi\expandafter\endcsname
925   \csname XINT_expr_precedence_][\endcsname {#1}%
926 }%
927 \def\XINT_tmpa #1#2#3#4#5#6%
928 {%
929   \def #1##1% \XINT_expr_op_:
930   {%
931     \expandafter #2\expandafter ##1\romannumeral`&&\XINT_expr_getnext
932   }%
933   \def #2##1##2% \XINT_expr_until_:_a
934   {\xint_UDsignfork
935     ##2{\expandafter #2\expandafter ##1\romannumeral`&&#4}%
936     -{#3##1##2}%
937     \krof }%
938   \def #3##1##2##3##4% \XINT_expr_until_:_b
939   {%
940     \ifnum ##2>#5%
941       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&%
942                     \csname XINT_#6_op_##3\endcsname {##4}}%
943     \else
944       \xint_afterfi
945       {\expandafter ##2\expandafter ##3\csname

```

```

946      .:=\xintNum{\XINT_expr_unlock ##1};\xintNum{\XINT_expr_unlock ##4}%
947      \endcsname
948    }%
949    \fi
950  }%
951  \let #5\xint_c_iii
952}%
953\xintFor #1 in {expr,flexpr,iiexpr} \do {%
954\expandafter\XINT_tmpa
955  \csname XINT_#1_op_:\expandafter\endcsname
956  \csname XINT_#1_until:_a\expandafter\endcsname
957  \csname XINT_#1_until:_b\expandafter\endcsname
958  \csname XINT_#1_op_-vi\expandafter\endcsname
959  \csname XINT_expr_precedence_:\endcsname {#1}%
960}%
961\catcode`[ 11 \catcode`] 11
962\let\XINT_expr_precedence_:\xint_c_iii
963\def\XINT_expr_op_:] #1%
964{%
965  \expandafter\xint_c_i\expandafter )%
966  \csname .:=]\xintNum{\XINT_expr_unlock #1}\endcsname
967}%
968\let\XINT_flexpr_op_:] \XINT_expr_op_:]
969\let\XINT_iiexpr_op_:] \XINT_expr_op_:]
970\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).

```

971\edef\XINT_expr_op_][: #1{\xint_c_ii\noexpand\XINT_expr_itself_][#10\string :}%
972\let\XINT_flexpr_op_][: \XINT_expr_op_][:
973\let\XINT_iiexpr_op_][: \XINT_expr_op_][:
974\catcode`[ 12 \catcode`] 12

```

10.26.1 `\xintListSel:x:csv`

1.2j. Because there is `\xintKeep:x:csv` which is faster than `\xintKeep:f:csv`.

```

975\def\xintListSel:x:csv #1%
976{%
977  \if ]\noexpand#1\xint_dothis\XINT_listsel:_s\fi
978  \if : \noexpand#1\xint_dothis\XINT_listxs sel:_:\fi
979  \xint_orthat {\XINT_listsel:_nth #1}%
980}%
981\def\XINT_listsel:_s #1#2;#3;%
982{%
983  \if-#1\expandafter\xintKeep:f:csv\else\expandafter\xintTrim:f:csv\fi
984  {#1#2}{#3}%
985}%
986\def\XINT_listsel:_nth #1;#2;{\xintNthEltPy:f:csv {\xintNum{#1}}{#2}}%

```

`\XINT_listsel:_nth` and `\XINT_listsel:_s` located in `\xintListSel:f:csv`.

```

987 \def\XINT_listxselsel:_: #1#2;#3#4;%
988 {%
989   \xint_UDsignsfork
990     #1#3\XINT_listxselsel:_N:N
991     #1-\XINT_listxselsel:_N:P
992     -#3\XINT_listxselsel:_P:N
993     --\XINT_listxselsel:_P:P
994   \krof #1#2;#3#4;%
995 }%
996 \def\XINT_listxselsel:_P:P #1;#2;#3;%
997 {%
998   \unless\ifnum #1<#2 \expandafter\xint_gobble_iii\fi
999   \xintKeep:x:csv{#2-#1}{\xintTrim:f:csv{#1}{#3}}%
1000 }%
1001 \def\XINT_listxselsel:_N:N #1;#2;#3;%
1002 {%
1003   \expandafter\XINT_listxselsel:_N:N_a
1004   \the\numexpr #2-#1\expandafter;\the\numexpr#1+\xintLength:f:csv{#3};#3;%
1005 }%
1006 \def\XINT_listxselsel:_N:N_a #1;#2;#3;%
1007 {%
1008   \unless\ifnum #1>\xint_c_ \expandafter\xint_gobble_iii\fi
1009   \xintKeep:x:csv{#1}{\xintTrim:f:csv{\ifnum#2<\xint_c_\xint_c_\else#2\fi}{#3}}%
1010 }%
1011 \def\XINT_listxselsel:_N:P #1;#2;#3;{\expandafter\XINT_listxselsel:_N:P_a
1012   \the\numexpr #1+\xintLength:f:csv{#3};#2;#3;%}
1013 \def\XINT_listxselsel:_N:P_a #1#2;%
1014   {\if -#1\expandafter\XINT_listxselsel:_O:P\fi\XINT_listxselsel:_P:P #1#2;%}
1015 \def\XINT_listxselsel:_O:P\XINT_listxselsel:_P:P #1;{\XINT_listxselsel:_P:P 0;%}
1016 \def\XINT_listxselsel:_P:N #1;#2;#3;{\expandafter\XINT_listxselsel:_P:N_a
1017   \the\numexpr #2+\xintLength:f:csv{#3};#1;#3;%}
1018 \def\XINT_listxselsel:_P:N_a #1#2;#3;%
1019   {\if -#1\expandafter\XINT_listxselsel:_P:O\fi\XINT_listxselsel:_P:P #3;#1#2;%}
1020 \def\XINT_listxselsel:_P:O\XINT_listxselsel:_P:P #1;#2;{\XINT_listxselsel:_P:P #1;0;%}

```

10.26.2 \xintListSel:f:csv

1.2g. Since 1.2j this is needed only for \xintNewExpr and user defined functions. Some extras compared to \xintListSel:x:csv because things may not yet have been expanded in the \xintNewExpr context.

```

1021 \def\xintListSel:f:csv #1%
1022 {%
1023   \if ]\noexpand#1\xint_dothis{\expandafter\XINT_listsel:_s\romannumeral`&&@}\fi
1024   \if :\noexpand#1\xint_dothis{\XINT_listsel:_:}\fi
1025   \xint_orthat {\XINT_listsel:_nth #1}%
1026 }%
1027 \def\XINT_listsel:_: #1;#2;%
1028 {%
1029   \expandafter\XINT_listsel:_:a
1030   \the\numexpr #1\expandafter;\the\numexpr #2\expandafter;\romannumeral`&&@%
1031 }%
1032 \def\XINT_listsel:_:a #1#2;#3#4;%

```

```

1033 {%
1034   \xint_UDsignsfork
1035     #1#3\XINT_listsel:_N:N
1036     #1-\XINT_listsel:_N:P
1037     -#3\XINT_listsel:_P:N
1038     --\XINT_listsel:_P:P
1039   \krof #1#2;#3#4;%
1040 }%
1041 \def\XINT_listsel:_P:P #1;#2;#3;%
1042 {%
1043   \unless\ifnum #1<#2 \xint_afterfi{\expandafter\space\xint_gobble_iii}\fi
1044   \xintKeep:f:csv{#2-#1}{\xintTrim:f:csv{#1}{#3}}%
1045 }%
1046 \def\XINT_listsel:_N:N #1;#2;#3;%
1047 {%
1048   \unless\ifnum #1<#2 \expandafter\XINT_listsel:_N:N_abort\fi
1049   \expandafter\XINT_listsel:_N:N_a
1050   \the\numexpr#1+\xintLength:f:csv{#3}\expandafter;\the\numexpr#2-#1;#3;%
1051 }%
1052 \def\XINT_listsel:_N:N_abort #1;#2;#3;{ }%
1053 \def\XINT_listsel:_N:N_a #1;#2;#3;%
1054 {%
1055   \xintKeep:f:csv{#2}{\xintTrim:f:csv{\ifnum#1<\xint_c_\xint_c_\else#1\fi}{#3}}%
1056 }%
1057 \def\XINT_listsel:_N:P #1;#2;#3;{\expandafter\XINT_listsel:_N:P_a
1058   \the\numexpr #1+\xintLength:f:csv{#3};#2;#3;%}
1059 \def\XINT_listsel:_N:P_a #1#2;%
1060   {\if -#1\expandafter\XINT_listsel:_O:P\fi\XINT_listsel:_P:P #1#2;%}
1061 \def\XINT_listsel:_O:P\XINT_listsel:_P:P #1;{\XINT_listsel:_P:P 0;}%
1062 \def\XINT_listsel:_P:N #1;#2;#3;{\expandafter\XINT_listsel:_P:N_a
1063   \the\numexpr #2+\xintLength:f:csv{#3};#1;#3;%}
1064 \def\XINT_listsel:_P:N_a #1#2;#3;%
1065   {\if -#1\expandafter\XINT_listsel:_P:O\fi\XINT_listsel:_P:P #3;#1#2;%}
1066 \def\XINT_listsel:_P:O\XINT_listsel:_P:P #1;#2;{\XINT_listsel:_P:P #1;0;}%

```

10.26.3 \xintKeep:x:csv

1.2j. This macro is used only with positive first argument.

```

1067 \def\xintKeep:x:csv #1#2%
1068 {%
1069   \expandafter\xint_gobble_i
1070   \romannumeral0\expandafter\XINT_keep:x:csv_pos
1071   \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
1072 }%
1073 \def\XINT_keep:x:csv_pos #1.#2%
1074 {%
1075   \expandafter\XINT_keep:x:csv_loop\the\numexpr#1-\xint_c_viii.%
1076   #2\xint_Bye,\xint_Bye,\xint_Bye,\xint_Bye,%
1077   \xint_Bye,\xint_Bye,\xint_Bye,\xint_Bye,\xint_bye
1078 }%
1079 \def\XINT_keep:x:csv_loop #1%
1080 {%

```



```

1081 \xint_gob_til_minus#1\XINT_keep:x:csv_finish-%
1082 \XINT_keep:x:csv_loop_pickeight #1%
1083 }%
1084 \def\XINT_keep:x:csv_loop_pickeight #1.#2,#3,#4,#5,#6,#7,#8,#9,%
1085 {%
1086     ,#2,#3,#4,#5,#6,#7,#8,#9%
1087 \expandafter\XINT_keep:x:csv_loop\the\numexpr#1-\xint_c_viii.%
1088 }%
1089 \def\XINT_keep:x:csv_finish-\XINT_keep:x:csv_loop_pickeight -#1.%
1090 {%
1091     \csname XINT_keep:x:csv_finish#1\endcsname
1092 }%
1093 \expandafter\def\csname XINT_keep:x:csv_finish1\endcsname
1094     #1,#2,#3,#4,#5,#6,#7,{,#1,#2,#3,#4,#5,#6,#7\xint_Bye}%
1095 \expandafter\def\csname XINT_keep:x:csv_finish2\endcsname
1096     #1,#2,#3,#4,#5,#6,{,#1,#2,#3,#4,#5,#6\xint_Bye}%
1097 \expandafter\def\csname XINT_keep:x:csv_finish3\endcsname
1098     #1,#2,#3,#4,#5,{,#1,#2,#3,#4,#5\xint_Bye}%
1099 \expandafter\def\csname XINT_keep:x:csv_finish4\endcsname
1100     #1,#2,#3,#4,{,#1,#2,#3,#4\xint_Bye}%
1101 \expandafter\def\csname XINT_keep:x:csv_finish5\endcsname
1102     #1,#2,#3,{,#1,#2,#3\xint_Bye}%
1103 \expandafter\def\csname XINT_keep:x:csv_finish6\endcsname
1104     #1,#2,{,#1,#2\xint_Bye}%
1105 \expandafter\def\csname XINT_keep:x:csv_finish7\endcsname
1106     #1,{,#1\xint_Bye}%
1107 \expandafter\let\csname XINT_keep:x:csv_finish8\endcsname\xint_Bye

```

10.26.4 `\xintKeep:f:csv`

1.2g, code in `xinttools.sty`. Refactored in 1.2j.

10.26.5 `\xintTrim:f:csv`

1.2g, code in `xinttools.sty`. Refactored in 1.2j.

10.26.6 `\xintNthEltPy:f:csv`

1.2g, code in `xinttools.sty`. Refactored in 1.2j.

10.26.7 `\xintLength:f:csv`

1.2g, code in `xinttools.sty`. Refactored in 1.2j.

10.26.8 `\xintReverse:f:csv`

1.2g, code in `xinttools.sty`.

10.27 Macros for a..b list generation

10.27.1	<code>\xintSeq::csv</code>	322
10.27.2	<code>\xintiiSeq::csv</code>	323

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

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

```

1108 \def\xintSeq::csv {\romannumeral0\xintseq::csv}%
1109 \def\xintseq::csv #1#2%
1110 {%
1111   \expandafter\XINT_seq::csv\expandafter
1112     {\the\numexpr \xintiCeil{#1}\expandafter}\expandafter
1113     {\the\numexpr \xintiFloor{#2}}%
1114}%
1115 \def\XINT_seq::csv #1#2%
1116 {%
1117   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
1118     \expandafter\XINT_seq::csv_z
1119   \or
1120     \expandafter\XINT_seq::csv_p
1121   \else
1122     \expandafter\XINT_seq::csv_n
1123   \fi
1124   {#2}{#1}%
1125}%
1126 \def\XINT_seq::csv_z #1#2{ #1/1[0]}%
1127 \def\XINT_seq::csv_p #1#2%
1128 {%
1129   \ifnum #1>#2
1130     \expandafter\expandafter\expandafter\XINT_seq::csv_p
1131   \else
1132     \expandafter\XINT_seq::csv_e
1133   \fi
1134   \expandafter{\the\numexpr #1-\xint_c_i}{#2},#1/1[0]%
1135}%
1136 \def\XINT_seq::csv_n #1#2%
1137 {%
1138   \ifnum #1<#2
1139     \expandafter\expandafter\expandafter\XINT_seq::csv_n
1140   \else
1141     \expandafter\XINT_seq::csv_e
1142   \fi
1143   \expandafter{\the\numexpr #1+\xint_c_i}{#2},#1/1[0]%
1144}%
1145 \def\XINT_seq::csv_e #1,{ }%
```

10.27.2 \xintiiSeq::csv

```

1146 \def\xintiiSeq::csv {\romannumeral0\xintiiseq::csv}%
1147 \def\xintiiseq::csv #1#2%
1148 {%
1149   \expandafter\XINT_iiseq::csv\expandafter
1150     {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
1151 }%
1152 \def\XINT_iiseq::csv #1#2%
1153 {%
1154   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
1155   \expandafter\XINT_iiseq::csv_z
1156   \or
1157   \expandafter\XINT_iiseq::csv_p
1158   \else
1159   \expandafter\XINT_iiseq::csv_n
1160   \fi
1161   {#2}{#1}%
1162 }%
1163 \def\XINT_iiseq::csv_z #1#2{ #1}%
1164 \def\XINT_iiseq::csv_p #1#2%
1165 {%
1166   \ifnum #1>#2
1167     \expandafter\expandafter\expandafter\XINT_iiseq::csv_p
1168   \else
1169     \expandafter\XINT_seq::csv_e
1170   \fi
1171   \expandafter{\the\numexpr #1-\xint_c_i}{#2},#1%
1172 }%
1173 \def\XINT_iiseq::csv_n #1#2%
1174 {%
1175   \ifnum #1<#2
1176     \expandafter\expandafter\expandafter\XINT_iiseq::csv_n
1177   \else
1178     \expandafter\XINT_seq::csv_e
1179   \fi
1180   \expandafter{\the\numexpr #1+\xint_c_i}{#2},#1%
1181 }%
1182 \def\XINT_seq::csv_e #1,{ }%

```

10.28 Macros for a..[d].b list generation

10.28.1	\xintSeqA::csv, \xintiiSeqA::csv, \XINTinFloatSeqA::csv	323
10.28.2	\xintSeqB::csv	324
10.28.3	\xintiiSeqB::csv	324
10.28.4	\XINTinFloatSeqB::csv	325

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

```

1183 \def\xintSeqA::csv #1%
1184   {\expandafter\XINT_seqa::csv\expandafter{\romannumeral0\xintra {#1}}}%

```

```

1185 \def\XINT_seqa::csv #1#2{\expandafter\XINT_seqa::csv_a \romannumeral0\xintra {#2};#1;}%
1186 \def\xintiiSeqA::csv #1{\expandafter\XINT_iiseqa::csv\expandafter{\romannumeral`&&@#1}}%
1187 \def\XINT_iiseqa::csv #1#2{\expandafter\XINT_seqa::csv_a\romannumeral`&&@#2;#1;}%
1188 \def\XINTinFloatSeqA::csv #1{\expandafter\XINT_flseqa::csv\expandafter
1189   {\romannumeral0\XINTinfloat [\XINTdigits]{#1}}}%
1190 \def\XINT_flseqa::csv #1#2%
1191   {\expandafter\XINT_seqa::csv_a\romannumeral0\XINTinfloat [\XINTdigits]{#2};#1;}%
1192 \def\XINT_seqa::csv_a #1{\xint_UDzerominusfork
1193   #1-{z}%
1194   0#1{n}%
1195   0-{p}%
1196   \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.

```

1197 \def\xintSeqB::csv #1#2%
1198   {\expandafter\XINT_seqb::csv \expandafter{\romannumeral0\xintra{#2}}{#1}}%
1199 \def\XINT_seqb::csv #1#2{\expandafter\XINT_seqb::csv_a\romannumeral`&&@#2#1!}%
1200 \def\XINT_seqb::csv_a #1#2;#3;#4!\expandafter\XINT_expr_seq_empty?
1201   \romannumeral0\csname XINT_seqb::csv_#1\endcsname {#3}{#4}{#2}}%
1202 \def\XINT_seqb::csv_p #1#2#3%
1203 {%
1204   \xintifCmp {#1}{#2}{, #1\expandafter\XINT_seqb::csv_p\expandafter}%
1205   {, #1\xint_gobble_iii}{\xint_gobble_iii}%

```

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

```

1206   {\romannumeral0\xintadd {#3}{#1}}{#2}{#3}%
1207 }%
1208 \def\XINT_seqb::csv_n #1#2#3%
1209 {%
1210   \xintifCmp {#1}{#2}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1211   {, #1\expandafter\XINT_seqb::csv_n\expandafter}%
1212   {\romannumeral0\xintadd {#3}{#1}}{#2}{#3}%
1213 }%
1214 \def\XINT_seqb::csv_z #1#2#3{, #1}%

```

10.28.3 \xintiiSeqB::csv

```

1215 \def\xintiiSeqB::csv #1#2{\XINT_iiseqb::csv #1#2}%
1216 \def\XINT_iiseqb::csv #1#2#3#4%
1217   {\expandafter\XINT_iiseqb::csv_a
1218     \romannumeral`&&\expandafter \XINT_expr_unlock\expandafter#2%
1219     \romannumeral`&&\XINT_expr_unlock #4!}%
1220 \def\XINT_iiseqb::csv_a #1#2;#3;#4!\expandafter\XINT_expr_seq_empty?
1221   \romannumeral`&&\csname XINT_iiseqb::csv_#1\endcsname {#3}{#4}{#2}}%
1222 \def\XINT_iiseqb::csv_p #1#2#3%
1223 {%
1224   \xintSgnFork{\XINT_Cmp {#1}{#2}}{, #1\expandafter\XINT_iiseqb::csv_p\expandafter}%
1225   {, #1\xint_gobble_iii}{\xint_gobble_iii}%
1226   {\romannumeral0\xintiiadd {#3}{#1}}{#2}{#3}%

```

```

1227 }%
1228 \def\XINT_iiseqb::csv_n #1#2#3%
1229 {%
1230   \xintSgnFork{\XINT_Cmp {#1}{#2}}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1231   {, #1\expandafter\XINT_iiseqb::csv_n\expandafter}%
1232   {\romannumeral0\xintiadd {#3}{#1}}{#2}{#3}%
1233 }%
1234 \def\XINT_iiseqb::csv_z #1#2#3{, #1}%

```

10.28.4 \XINTinFloatSeqB::csv

```

1235 \def\XINTinFloatSeqB::csv #1#2{\expandafter\XINT_flseqb::csv \expandafter
1236   {\romannumeral0\XINTinfloat [\XINTdigits]{#2}}{#1}}%
1237 \def\XINT_flseqb::csv #1#2{\expandafter\XINT_flseqb::csv_a\romannumeral`&&@#2#1!}%
1238 \def\XINT_flseqb::csv_a #1#2;#3;#4!{\expandafter\XINT_expr_seq_empty?
1239   \romannumeral`&&@\csname XINT_flseqb::csv_#1\endcsname {#3}{#4}{#2}}%
1240 \def\XINT_flseqb::csv_p #1#2#3%
1241 {%
1242   \xintifCmp {#1}{#2}{, #1\expandafter\XINT_flseqb::csv_p\expandafter}%
1243   {, #1\xint_gobble_iii}{\xint_gobble_iii}%
1244   {\romannumeral0\XINTinfloatadd {#3}{#1}}{#2}{#3}%
1245 }%
1246 \def\XINT_flseqb::csv_n #1#2#3%
1247 {%
1248   \xintifCmp {#1}{#2}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1249   {, #1\expandafter\XINT_flseqb::csv_n\expandafter}%
1250   {\romannumeral0\XINTinfloatadd {#3}{#1}}{#2}{#3}%
1251 }%
1252 \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.

```

1253 \def\XINT_tmpa #1#2#3#4#5#6%
1254 {%
1255   \def #1##1% \XINT_expr_op_,
1256   {%
1257     \expandafter #2\expandafter ##1\romannumeral`&&@\XINT_expr_getnext
1258   }%
1259   \def #2##1##2% \XINT_expr_until_,_a
1260   {\xint_UDsignfork
1261     ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
1262     -{#3##1##2}%
1263   \krof }%
1264   \def #3##1##2##3##4% \XINT_expr_until_,_b
1265   {%
1266     \ifnum ##2>\xint_c_ii
1267       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
1268         \csname XINT_#6_op_##3\endcsname {##4}}%
1269     \else
1270       \xint_afterfi
1271       {\expandafter ##2\expandafter ##3%
1272       \csname .=\XINT_expr_unlock ##1,\XINT_expr_unlock ##4\endcsname }%
1273     \fi

```

```

1274 }%
1275 \let #5\xint_c_ii
1276 }%
1277 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1278 \expandafter\XINT_tmpa
1279 \csname XINT_#1_op_,\expandafter\endcsname
1280 \csname XINT_#1_until_,_a\expandafter\endcsname
1281 \csname XINT_#1_until_,_b\expandafter\endcsname
1282 \csname XINT_#1_op_-vi\expandafter\endcsname
1283 \csname XINT_expr_precedence_,\endcsname {#1}%
1284 }%

```

10.30 The minus as prefix operator of variable precedence level

Inherits the precedence level of the previous infix operator.

```

1285 \def\XINT_tmpa #1#2#3%
1286 {%
1287 \expandafter\XINT_tmph
1288 \csname XINT_#1_op_-#3\expandafter\endcsname
1289 \csname XINT_#1_until_-#3_a\expandafter\endcsname
1290 \csname XINT_#1_until_-#3_b\expandafter\endcsname
1291 \csname xint_c_#3\endcsname {#1}#2%
1292 }%
1293 \def\XINT_tmph #1#2#3#4#5#6%
1294 {%
1295 \def #1% \XINT_expr_op_-<level>
1296 {% get next number+operator then switch to _until macro
1297 \expandafter #2\romannumeral`&&\XINT_expr_getnext
1298 }%
1299 \def #2##1% \XINT_expr_until_-<l>_a
1300 {\xint_UDsignfork
1301 ##1{\expandafter #2\romannumeral`&&@#1}%
1302 -{#3##1}%
1303 \krof }%
1304 \def #3##1##2##3% \XINT_expr_until_-<l>_b
1305 {% _until tests precedence level with next op, executes now or postpones
1306 \ifnum ##1>#4%
1307 \xint_afterfi {\expandafter #2\romannumeral`&&@%
1308 \csname XINT_#5_op_##2\endcsname {##3}}%
1309 \else
1310 \xint_afterfi {\expandafter ##1\expandafter ##2%
1311 \csname .=#6{\XINT_expr_unlock ##3}\endcsname }%
1312 \fi
1313 }%
1314 }%

```

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.

```

1315 \xintApplyInline{\XINT_tmpa {expr}\xintOpp}{vi}{vii}{viii}{ix}}%
1316 \xintApplyInline{\XINT_tmpa {flexpr}\xintOpp}{vi}{vii}{viii}{ix}}%
1317 \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)}.)

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.

```

1318 \let\XINT_expr_precedence_? \xint_c_x
1319 \def\XINT_expr_op_? #1#2%
1320   {\XINT_expr_op_?checka #2!\xint_bye\XINT_expr_op_?a #1{#2}}%
1321 \def\XINT_expr_op_?checka #1{\expandafter\XINT_expr_op_?checkb\detokenize{#1}}%
1322 \def\XINT_expr_op_?checkb #1{\if ?#1\expandafter\XINT_expr_op_?checkc
1323   \else\expandafter\xint_bye\fi }%
1324 \def\XINT_expr_op_?checkc #1{\xint_gob_til_! #1\XINT_expr_op_?? !\xint_bye}%
1325 \def\XINT_expr_op_?a #1#2#3%
1326 {%
1327   \xintiiifNotZero{\XINT_expr_unlock #1}{\XINT_expr_getnext #2}{\XINT_expr_getnext #3}%
1328 }%
1329 \let\XINT_flexpr_op_?\XINT_expr_op_?
1330 \let\XINT_iiexpr_op_?\XINT_expr_op_?
1331 \def\XINT_expr_op_?? !\xint_bye\xint_bye\XINT_expr_op_?a #1#2#3#4#5%
1332 {%
1333   \xintiiifSgn {\XINT_expr_unlock #1}%
1334   {\XINT_expr_getnext #3}{\XINT_expr_getnext #4}{\XINT_expr_getnext #5}%
1335 }%

```

10.32 ! as postfix factorial operator

```

1336 \let\XINT_expr_precedence_! \xint_c_x
1337 \def\XINT_expr_op_! #1{\expandafter\XINT_expr_getop
1338   \csname .=\xintFac{\XINT_expr_unlock #1}\endcsname }%
1339 \def\XINT_flexpr_op_! #1{\expandafter\XINT_expr_getop
1340   \csname .=\XINTinFloatFac{\XINT_expr_unlock #1}\endcsname }%
1341 \def\XINT_iiexpr_op_! #1{\expandafter\XINT_expr_getop
1342   \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.

```

1343 \catcode`[ 11
1344 \catcode`* 11
1345 \let\XINT_expr_precedence_ \xint_c_vii
1346 \def\XINT_expr_op_[ #1#2]{\expandafter\XINT_expr_getop
1347     \csname .=\xintE{\XINT_expr_unlock #1}%
1348     {\xint_zapspaces #2 \xint_gobble_i}\endcsname}%
1349 \def\XINT_iiexpr_op_[ #1#2]{\expandafter\XINT_expr_getop
1350     \csname .=\xintiiE{\XINT_expr_unlock #1}%
1351     {\xint_zapspaces #2 \xint_gobble_i}\endcsname}%
1352 \def\XINT_flexpr_op_[ #1#2]{\expandafter\XINT_expr_getop
1353     \csname .=\XINTinFloatE{\XINT_expr_unlock #1}%
1354     {\xint_zapspaces #2 \xint_gobble_i}\endcsname}%
1355 \catcode`\ 12
1356 \catcode`* 12

```

10.34 `\XINT_expr_op_`` for recognizing functions

The "onliteral" 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 "onliteral" 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 `onliteral_<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.

```

1357 \def\XINT_tmpa #1#2#3{%
1358     \def #1##1%
1359     {%
1360         \ifcsname XINT_#3_func_##1\endcsname
1361         \xint_dothis{\expandafter\expandafter
1362             \csname XINT_#3_func_##1\endcsname\romannumeral`&&@#2}\fi
1363         \ifcsname XINT_expr_onliteral_##1\endcsname
1364         \xint_dothis{\csname XINT_expr_onliteral_##1\endcsname}\fi
1365         \xint_orthat{\XINT_expr_unknown_function {##1}%
1366             \expandafter\XINT_expr_func_unknown\romannumeral`&&@#2}%
1367     }%
1368 }%
1369 \def\XINT_expr_unknown_function #1{\xintError:removed \xint_gobble_i {#1}}%
1370 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1371     \expandafter\XINT_tmpa
1372     \csname XINT_#1_op_`\expandafter\endcsname
1373     \csname XINT_#1_oparen\endcsname
1374     {#1}%
1375 }%
1376 \def\XINT_expr_func_unknown #1#2#3%
1377     {\expandafter #1\expandafter #2\csname .=\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”.

```

1378 \def\XINT_expr_onliteral_bool #1)%
1379     {\expandafter\XINT_expr_getop\csname .=\xintBool{#1}\endcsname }%
1380 \def\XINT_expr_onliteral_togl #1)%
1381     {\expandafter\XINT_expr_getop\csname .=\xintToggle{#1}\endcsname }%
1382 \def\XINT_expr_onliteral_protect #1)%
1383     {\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.

```

1384 \def\XINT_expr_func_break #1#2#3%
1385     {\expandafter #1\expandafter #2\csname .=?\romannumeral`&&\XINT_expr_unlock #3\endcsname }%
1386 \let\XINT_flexpr_func_break \XINT_expr_func_break
1387 \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.

```

1388 \def\XINT_expr_onliteral_qint #1)%
1389     {\expandafter\XINT_expr_getop\csname .=\xintiNum{#1}\endcsname }%
1390 \def\XINT_expr_onliteral_qfrac #1)%
1391     {\expandafter\XINT_expr_getop\csname .=\xintRaw{#1}\endcsname }%
1392 \def\XINT_expr_onliteral_qfloat #1)%
1393     {\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`.

```

1394 \def\XINT_expr_op__ #1% op__ with two _'s
1395     {%
1396         \ifcsname XINT_expr_var_#1\endcsname
1397             \expandafter\xint_firstoftwo
1398         \else
1399             \expandafter\xint_secondoftwo
1400         \fi
1401         {\expandafter\expandafter\expandafter
1402          \XINT_expr_getop\csname XINT_expr_var_#1\endcsname}%
1403         {\XINT_expr_unknown_variable {#1}%
1404          \expandafter\XINT_expr_getop\csname .:=0\endcsname}%
1405     }%
1406 \def\XINT_expr_unknown_variable #1{\xintError:removed \xint_gobble_i {#1}}%
1407 \let\XINT_flexpr_op__ \XINT_expr_op__
1408 \let\XINT_iiexpr_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 an underscore are reserved.

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

1.2c adds the "onliteral" 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)).

1.2p (2017/12/01) extends \xintdefvar for simultaneous assignments to multiple variables.

```

1409 \catcode`* 11
1410 \def\xINT_expr_defvar_one #1#2%
1411 {%
1412   \expandafter\edef\csname XINT_expr_var_#1\endcsname
1413     {\expandafter\noexpand#2}%
1414   \expandafter\edef\csname XINT_expr_onliteral_#1\endcsname
1415     {\XINT_expr_precedence_*** *\expandafter\noexpand#2}%
1416   \ifxintverbose\xintMessage{xintexpr}{Info}
1417     {Variable "#1" defined with value \expandafter\xINT_expr_unlock#2.}%
1418   \fi
1419 }%
1420 \catcode`* 12
1421 \def\xINT_expr_defvar #1#2#3;%
1422 {%
1423   \edef\xINT_expr_tmpa{#2}%
1424   \edef\xINT_expr_tmpa{\xint_zapspaces_o\xINT_expr_tmpa}%
1425   \edef\xINT_expr_tmpc{\xintCSVLength{\XINT_expr_tmpa}}%
1426   \ifcase\xINT_expr_tmpc
1427     \xintMessage {xintexpr}{Warning}
1428     {Aborting: impossible to declare variable with empty name.}%
1429   \or
1430     \edef\xINT_expr_tmpb{\romannumeral0#1#3\relax}%
1431     \XINT_expr_defvar_one\xINT_expr_tmpa\xINT_expr_tmpb
1432   \else

```

```

1433 \edef\XINT_expr_tmpb
1434   {\expandafter\XINT_expr_unlock\romannumeral0#1#3\relax}%
1435 \edef\XINT_expr_tmpd{\xintCSVLength{\XINT_expr_tmpb}}%
1436 \ifnum\XINT_expr_tmpc=\XINT_expr_tmpd\space
1437   \xintAssignArray\xintCSVtoList\XINT_expr_tmpa\to\XINT_expr_tmpvar
1438   \xintAssignArray
1439     \xintApply\XINT_expr_lockit{\xintCSVtoList\XINT_expr_tmpb}%
1440   \to\XINT_expr_tmpval
1441   \def\XINT_expr_tmpd{1}%
1442   \xintloop
1443     \expandafter\XINT_expr_defvar_one
1444     \csname XINT_expr_tmpvar\XINT_expr_tmpd\expandafter\endcsname
1445     \csname XINT_expr_tmpval\XINT_expr_tmpd\endcsname
1446   \ifnum\XINT_expr_tmpd<\XINT_expr_tmpc\space
1447     \edef\XINT_expr_tmpd{\the\numexpr\XINT_expr_tmpd+1}%
1448   \repeat
1449   \xintRelaxArray\XINT_expr_tmpvar
1450   \xintRelaxArray\XINT_expr_tmpval
1451 \else
1452   \xintMessage {xintexpr}{Warning}
1453   {Aborting: mismatch between number of variables (\XINT_expr_tmpc)
1454    and number of values (\XINT_expr_tmpd).}%
1455 \fi
1456 \fi
1457 }%
1458 \catcode`: 12
1459 \def\xintdefvar      #1:={\XINT_expr_defvar\xintbareeval      {#1}}%
1460 \def\xintdefiivar    #1:={\XINT_expr_defvar\xintbareiieval    {#1}}%
1461 \def\xintdeffloatvar #1:={\XINT_expr_defvar\xintbarefloateval {#1}}%
1462 \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`.

```

1463 \def\xintunassignvar #1{%
1464   \edef\XINT_expr_tmpa{#1}%
1465   \edef\XINT_expr_tmpa {\xint_zapspaces_o\XINT_expr_tmpa}%
1466   \ifcsname XINT_expr_var_\XINT_expr_tmpa\endcsname
1467     \ifnum\expandafter\xintLength\expandafter{\XINT_expr_tmpa}=\@ne
1468       \expandafter\XINT_expr_makedummy\XINT_expr_tmpa
1469       \ifxintverbose\xintMessage {xintexpr}{Info}%
1470       {Character \XINT_expr_tmpa\space usable as dummy variable (if with catcode letter).}%
1471     \fi
1472   \else
1473     \expandafter\edef\csname XINT_expr_var_\XINT_expr_tmpa\endcsname
1474     {\csname .=\0\endcsname\noexpand\XINT_expr_undefined {\XINT_expr_tmpa}}%
1475     \expandafter\edef\csname XINT_expr_onliteral_\XINT_expr_tmpa\endcsname
1476     {\csname .=\0\endcsname\noexpand\XINT_expr_undefined {\XINT_expr_tmpa}*}%
1477     \ifxintverbose\xintMessage {xintexpr}{Info}
1478     {Variable \XINT_expr_tmpa\space has been ``unassigned''.}%

```

```

1479     \fi
1480   \fi
1481   \else
1482     \xintMessage {xintexpr}{Warning}
1483     {Error: there was no such variable \XINT_expr_tmpa\space to unassign.}%
1484   \fi
1485 }%
1486 \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	332
10.41.2	omit and abort	333
10.41.3	The special variables @, @1, @2, @3, @4, @@, @@(1), ..., @@@, @@@(1), ... for recursion	334
10.41.4	\XINT_expr_onliteral_seq	335
10.41.5	\XINT_expr_onliteral_seq_a	335
10.41.6	\XINT_isbalanced_a for \XINT_expr_onliteral_seq_a	335
10.41.7	\XINT_allexpr_func_seqx	336
10.41.8	Evaluation over list, \XINT_expr_seq:_a with break, abort, omit	336
10.41.9	Evaluation over ++ generated lists with \XINT_expr_seq:_A	337

All of seq, add, mul, rseq, etc... (actually all of the extensive changes from xintexpr 1.09n to 1.1) was done around June 15-25th 2014, but the problem is that I did not document the code enough, and I had a hard time understanding in October what I had done in June. Despite the lesson, again being short on time, I do not document enough my current understanding of the innards of the beast...

I added subs, and iter in October (also the [:n], [n:] list extractors), proving I did at least understand a bit (or rather could imitate) my earlier code (but don't ask me to explain \xintNew-Expr !)

The \XINT_expr_onliteral_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_onliteral_<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` was adjoined `\xintnewdummy` by 1.2k for a public interface. It 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.

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.

```

1487 \catcode`* 11
1488 \def\XINT_expr_makedummy #1%
1489 {%
1490   \expandafter\def\csname XINT_expr_var_#1\endcsname ##1\relax !#1##2%
1491     {##2##1\relax !#1##2}%
1492   \expandafter\def\csname XINT_expr_onliteral_#1\endcsname ##1\relax !#1##2%
1493     {\XINT_expr_precedence_*** *##2(##1\relax !#1##2}%
1494 }%
1495 \xintApplyUnbraced \XINT_expr_makedummy {abcdefghijklmnopqrstuvwxyz}%
1496 \xintApplyUnbraced \XINT_expr_makedummy {ABCDEFGHIJKLMNOPQRSTUVWXYZ}%
1497 \def\xintnewdummy #1{%
1498   \XINT_expr_makedummy{#1}%
1499   \ifxintverbose\xintMessage {xintexpr}{Info}%
1500     {Character #1 now usable as dummy variable (if with catcode letter).}%
1501   \fi
1502 }%
1503 \edef\XINT_expr_var_nil {\expandafter\noexpand\csname . = \endcsname}%
1504 \edef\XINT_expr_onliteral_nil
1505   {\XINT_expr_precedence_*** *\expandafter\noexpand\csname . = \endcsname {}%
1506 \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 `!{}}{}}\.=!` et ensuite il y aura `\relax`, et le résultat sera `\.=!` pour omit ou `\.=^` pour abort. Les routines des boucles `seq`, `iter`, etc... peuvent alors repérer le `!` ou `^` et agir en conséquence (un long paragraphe pour ne décrire que partiellement une ou deux lignes de codes...).

Mais `^C` a été fait alors que je n'avais pas encore les variables muettes. Je dois trouver autre chose, car `seq(2^C, C=1..5)` est alors impossible. De toute façon ce `^C` était à usage interne uniquement.

Il me faut un symbole d'opérateur qui ne rentre pas en conflit. Bon je vais prendre `!?`. Ensuite au lieu de hacker `until_end`, il vaut mieux lui donner précedence 2 (mais ça ne pourra pas marcher à l'intérieur de parenthèses il faut d'abord les fermer manuellement) et lui associer un simplement

un op spécial. Je n'avais pas fait cela peut-être pour éviter d'avoir à définir plusieurs macros. Le #1 dans la définition de `\XINT_expr_op_!` est le résultat de l'évaluation forcée précédente.

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

```
1507 \edef\XINT_expr_var_omit #1\relax !{\string !?!\relax !}%
1508 \edef\XINT_expr_var_abort #1\relax !{\string !?^\relax !}%
1509 \def\XINT_expr_op_! #1#2\relax {\expandafter\XINT_expr_foundend\csname .=#2\endcsname}%
1510 \let\XINT_iiexpr_op_! \XINT_expr_op_!
1511 \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 "onliteral" 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.

```
1512 \catcode`? 3 \catcode`* 11
1513 \def\XINT_expr_var_@ #1~#2{#2#1~#2}%
1514 \expandafter\let\csname XINT_expr_var_@1\endcsname \XINT_expr_var_@
1515 \expandafter\def\csname XINT_expr_var_@2\endcsname #1~#2#3{#3#1~#2#3}%
1516 \expandafter\def\csname XINT_expr_var_@3\endcsname #1~#2#3#4{#4#1~#2#3#4}%
1517 \expandafter\def\csname XINT_expr_var_@4\endcsname #1~#2#3#4#5{#5#1~#2#3#4#5}%
1518 \def\XINT_expr_onliteral_@ #1~#2{\XINT_expr_precedence_*** *#2(#1~#2}%
1519 \expandafter\let\csname XINT_expr_onliteral_@1\endcsname \XINT_expr_onliteral_@
1520 \expandafter\def\csname XINT_expr_onliteral_@2\endcsname #1~#2#3%
1521     {\XINT_expr_precedence_*** *#3(#1~#2#3}%
1522 \expandafter\def\csname XINT_expr_onliteral_@3\endcsname #1~#2#3#4%
1523     {\XINT_expr_precedence_*** *#4(#1~#2#3#4}%
1524 \expandafter\def\csname XINT_expr_onliteral_@4\endcsname #1~#2#3#4#5%
1525     {\XINT_expr_precedence_*** *#5(#1~#2#3#4#5}%
1526 \catcode`* 12
1527 \def\XINT_expr_func_@@ #1#2#3#4~#5?%
1528 {%
1529     \expandafter#1\expandafter#2\romannumeral0\XINTntheltnoexpand
1530     {\XINTNum{\XINT_expr_unlock#3}}{#5}#4~#5?%
1531 }%
1532 \def\XINT_expr_func_@@@ #1#2#3#4~#5~#6?%
1533 {%
1534     \expandafter#1\expandafter#2\romannumeral0\XINTntheltnoexpand
1535     {\XINTNum{\XINT_expr_unlock#3}}{#6}#4~#5~#6?%
1536 }%
1537 \def\XINT_expr_func_@@@@ #1#2#3#4~#5~#6~#7?%
1538 {%
1539     \expandafter#1\expandafter#2\romannumeral0\XINTntheltnoexpand
1540     {\XINTNum{\XINT_expr_unlock#3}}{#7}#4~#5~#6~#7?%
1541 }%
1542 \let\XINT_flexpr_func_@@\XINT_expr_func_@@
1543 \let\XINT_flexpr_func_@@@\XINT_expr_func_@@@
1544 \let\XINT_flexpr_func_@@@@\XINT_expr_func_@@@@
```

```

1545 \def\XINT_iiexpr_func_@@ #1#2#3#4~#5?%
1546 {%
1547   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1548   {\XINT_expr_unlock#3}{#5}#4~#5?%
1549 }%
1550 \def\XINT_iiexpr_func_@@@ #1#2#3#4~#5~#6?%
1551 {%
1552   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1553   {\XINT_expr_unlock#3}{#6}#4~#5~#6?%
1554 }%
1555 \def\XINT_iiexpr_func_@@@@ #1#2#3#4~#5~#6~#7?%
1556 {%
1557   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1558   {\XINT_expr_unlock#3}{#7}#4~#5~#6~#7?%
1559 }%
1560 \catcode`? 11

```

10.41.4 \XINT_expr_onliteral_seq

```

1561 \def\XINT_expr_onliteral_seq
1562 {\expandafter\XINT_expr_onliteral_seq_f\romannumeral`&&\XINT_expr_onliteral_seq_a {}}%
1563 \def\XINT_expr_onliteral_seq_f #1#2{\xint_c_xviii `{seqx}#2)\relax #1}%

```

10.41.5 \XINT_expr_onliteral_seq_a

```

1564 \def\XINT_expr_onliteral_seq_a #1#2,%
1565 {%
1566   \ifcase\XINT_isbalanced_a \relax #1#2(\xint_bye)\xint_bye
1567   \expandafter\XINT_expr_onliteral_seq_c
1568   \or\expandafter\XINT_expr_onliteral_seq_b
1569   \else\expandafter\xintError:we_are_doomed
1570   \fi {#1#2},%
1571 }%
1572 \def\XINT_expr_onliteral_seq_b #1,{\XINT_expr_onliteral_seq_a {#1,}}%
1573 \def\XINT_expr_onliteral_seq_c #1,#2#3% #3 pour absorber le =
1574 {%
1575   \XINT_expr_onliteral_seq_d {#2{#1}}}%
1576 }%
1577 \def\XINT_expr_onliteral_seq_d #1#2#3)%
1578 {%
1579   \ifcase\XINT_isbalanced_a \relax #2#3(\xint_bye)\xint_bye
1580   \or\expandafter\XINT_expr_onliteral_seq_e
1581   \else\expandafter\xintError:we_are_doomed
1582   \fi
1583   {#1}{#2#3}%
1584 }%
1585 \def\XINT_expr_onliteral_seq_e #1#2{\XINT_expr_onliteral_seq_d {#1}{#2}}%

```

10.41.6 \XINT_isbalanced_a for \XINT_expr_onliteral_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.

```

1586 % use as \XINT_isbalanced_a \relax #1(\xint_bye)\xint_bye
1587 \def\XINT_isbalanced_a #1({\XINT_isbalanced_b #1)\xint_bye }%

```



```

1588 \def\XINT_isbalanced_b #1)#2%
1589   {\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

1590 \def\XINT_isbalanced_error #1)\xint_bye {\xint_c_mone}%

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

1591 \def\XINT_isbalanced_c\xint_bye\XINT_isbalanced_error #1%
1592   {\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.

1593 \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.

1594 \def\XINT_isbalanced_d #1)#2%
1595   {\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.

1596 \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>}.

```

1597 \def\XINT_expr_func_seqx #1#2{\XINT_allexpr_seqx \xintthebareval }%
1598 \def\XINT_flexpr_func_seqx #1#2{\XINT_allexpr_seqx \xintthebarefloateval}%
1599 \def\XINT_iexpr_func_seqx #1#2{\XINT_allexpr_seqx \xintthebareiieval }%
1600 \def\XINT_allexpr_seqx #1#2#3#4%
1601 {%
1602   \expandafter \XINT_expr_getop
1603   \csname .=\expandafter\XINT_expr_seq:_aa
1604     \romannumeral`&&\XINT_expr_unlock #2!{#1#4\relax !#3}\endcsname
1605 }%
1606 \def\XINT_expr_seq:_aa #1{\if +#1\expandafter\XINT_expr_seq:_A\else
1607   \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.)

```

1608 \def\XINT_expr_seq:_a #1!#2{\expandafter\XINT_expr_seq_empty?
1609         \romannumeral0\XINT_expr_seq:_b {#2}#1,^,}%
1610 \def\XINT_expr_seq:_b #1#2#3,{%
1611     \if ,#2\xint_dothis\XINT_expr_seq:_noop\fi
1612     \if ^#2\xint_dothis\XINT_expr_seq:_end\fi
1613     \xint_orthat{\expandafter\XINT_expr_seq:_c}\csname.=#2#3\endcsname {#1}%
1614 }%
1615 \def\XINT_expr_seq:_noop\csname.=,#1\endcsname #2{\XINT_expr_seq:_b {#2}#1,}%
1616 \def\XINT_expr_seq:_end \csname.=^\endcsname #1{%
1617 \def\XINT_expr_seq:_c #1#2{\expandafter\XINT_expr_seq:_d\romannumeral`&&@#2#1{#2}}%
1618 \def\XINT_expr_seq:_d #1{\if #1^\xint_dothis\XINT_expr_seq:_abort\fi
1619     \if #1?\xint_dothis\XINT_expr_seq:_break\fi
1620     \if #1!\xint_dothis\XINT_expr_seq:_omit\fi
1621     \xint_orthat{\XINT_expr_seq:_goon #1}}%
1622 \def\XINT_expr_seq:_abort #1!#2#3#4#5^,{}%
1623 \def\XINT_expr_seq:_break #1!#2#3#4#5^,{,#1}%
1624 \def\XINT_expr_seq:_omit #1!#2#3#4{\XINT_expr_seq:_b {#4}}%
1625 \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`.

```

1626 \def\XINT_expr_seq_empty? #1{%
1627 \def\XINT_expr_seq_empty? ##1{\if ,##1\expandafter\xint_gobble_i\fi #1\endcsname }}%
1628 \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.

```

1629 \def\XINT_expr_seq:_A +#1!%
1630     {\expandafter\XINT_expr_seq_empty?\romannumeral0\XINT_expr_seq:_D #1}%
1631 \def\XINT_expr_seq:_D #1#2{\expandafter\XINT_expr_seq:_E\romannumeral`&&@#2#1{#2}}%
1632 \def\XINT_expr_seq:_E #1{\if #1^\xint_dothis\XINT_expr_seq:_Abort\fi
1633     \if #1?\xint_dothis\XINT_expr_seq:_Break\fi
1634     \if #1!\xint_dothis\XINT_expr_seq:_Omit\fi
1635     \xint_orthat{\XINT_expr_seq:_Goon #1}}%
1636 \def\XINT_expr_seq:_Abort #1!#2#3#4{%
1637 \def\XINT_expr_seq:_Break #1!#2#3#4{,#1}%
1638 \def\XINT_expr_seq:_Omit #1!#2#3%
1639     {\expandafter\XINT_expr_seq:_D
1640         \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
1641 \def\XINT_expr_seq:_Goon #1!#2#3%
1642     {,#1\expandafter\XINT_expr_seq:_D
1643         \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.

```
1644 \def\XINT_expr_onliteral_add
1645 {\expandafter\XINT_expr_onliteral_add_f\romannumeral`&&\XINT_expr_onliteral_seq_a {}}%
1646 \def\XINT_expr_onliteral_add_f #1#2{\xint_c_xviii `{opxadd}#2)\relax #1}%
1647 \def\XINT_expr_onliteral_mul
1648 {\expandafter\XINT_expr_onliteral_mul_f\romannumeral`&&\XINT_expr_onliteral_seq_a {}}%
1649 \def\XINT_expr_onliteral_mul_f #1#2{\xint_c_xviii `{opxmul}#2)\relax #1}%

```

10.42.1 `\XINT_expr_func_opxadd`, `\XINT_flexpr_func_opxadd`, `\XINT_iiexpr_func_opxadd` and same for mul

modified 1.2c.

```
1650 \def\XINT_expr_func_opxadd #1#2{\XINT_allexpr_opx \xintbareeval {\xintAdd 0}}%
1651 \def\XINT_flexpr_func_opxadd #1#2{\XINT_allexpr_opx \xintbarefloateval {\XINTinFloatAdd 0}}%
1652 \def\XINT_iiexpr_func_opxadd #1#2{\XINT_allexpr_opx \xintbareiieval {\xintiiAdd 0}}%
1653 \def\XINT_expr_func_opxmul #1#2{\XINT_allexpr_opx \xintbareeval {\xintMul 1}}%
1654 \def\XINT_flexpr_func_opxmul #1#2{\XINT_allexpr_opx \xintbarefloateval {\XINTinFloatMul 1}}%
1655 \def\XINT_iiexpr_func_opxmul #1#2{\XINT_allexpr_opx \xintbareiieval {\xintiiMul 1}}%

```

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

```
1656 \def\XINT_allexpr_opx #1#2#3#4#5%
1657 {%
1658   \expandafter\XINT_expr_getop
1659   \csname.=\romannumeral`&&\expandafter\XINT_expr_op:_a
1660   \romannumeral`&&\XINT_expr_unlock #3!{#1#5\relax !#4}{#2}\endcsname
1661 }%
1662 \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.

```
1663 \def\XINT_expr_op:_b #1#2#3#4#5,%
1664   \if ,#4\xint_dothis\XINT_expr_op:_noop\fi
1665   \if ^#4\xint_dothis\XINT_expr_op:_end\fi
1666   \xint_orthat{\expandafter\XINT_expr_op:_c}\csname.=#4#5\endcsname {#3}#1{#2}%
1667 }%
1668 \def\XINT_expr_op:_c #1#2#3#4{\expandafter\XINT_expr_op:_d\romannumeral0#2#1#3{#4}{#2}}%
1669 \def\XINT_expr_op:_d #1!#2#3#4#5%
1670   {\expandafter\XINT_expr_op:_b\expandafter #4\expandafter
1671    {\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.

```
1672 \def\XINT_expr_op:_noop\csname.=,#1\endcsname #2#3#4{\XINT_expr_op:_b #3{#4}{#2}#1,%}
1673 \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.

```

1674 \def\XINT_expr_onliteral_sub
1675 {\expandafter\XINT_expr_onliteral_sub_f\romannumeral`&&\XINT_expr_onliteral_seq_a {}}%
1676 \def\XINT_expr_onliteral_sub_f #1#2{\xint_c_xviii `{subx}#2)\relax #1}%
1677 \def\XINT_expr_func_subx #1#2{\XINT_allexpr_subx \xintbareeval }%
1678 \def\XINT_flexpr_func_subx #1#2{\XINT_allexpr_subx \xintbarefloateval}%
1679 \def\XINT_iexpr_func_subx #1#2{\XINT_allexpr_subx \xintbareiieval}%
1680 \def\XINT_allexpr_subx #1#2#3#4% #2 is the value to assign to the dummy variable
1681 {% #3 is the dummy variable, #4 is the expression to evaluate
1682   \expandafter\expandafter\expandafter\XINT_expr_getop
1683   \expandafter\XINT_expr_subx:_end\romannumeral0#1#4\relax !#3#2%
1684 }%
1685 \def\XINT_expr_subx:_end #1!#2#3{#1}%

```

10.44 rseq

10.44.1	<code>\XINT_expr_rseqx</code>	...	339
10.44.2	<code>\XINT_expr_rseqy</code>	...	340
10.44.3	<code>\XINT_expr_rseq:_a</code> etc...	...	340
10.44.4	<code>\XINT_expr_rseq:_A</code> etc...	...	340

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...

```

1686 \def\XINT_expr_func_rseq {\XINT_allexpr_rseq \xintbareeval \xintthebareeval }%
1687 \def\XINT_flexpr_func_rseq {\XINT_allexpr_rseq \xintbarefloateval \xintthebarefloateval }%
1688 \def\XINT_iexpr_func_rseq {\XINT_allexpr_rseq \xintbareiieval \xintthebareiieval }%
1689 \def\XINT_allexpr_rseq #1#2#3%
1690 {%
1691   \expandafter\XINT_expr_rseqx\expandafter #1\expandafter#2\expandafter
1692   #3\romannumeral`&&\XINT_expr_onliteral_seq_a {}}%
1693 }%

```

10.44.1 `\XINT_expr_rseqx`

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

```

1694 \def\XINT_expr_rseqx #1#2#3#4#5%
1695 {%
1696   \expandafter\XINT_expr_rseqy\romannumeral0#1(#5)\relax #3#4#2%
1697 }%

```

10.44.2 \XINT_expr_rseq

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

```

1698 \def\XINT_expr_rseq #1#2#3#4#5%
1699 {%
1700   \expandafter \XINT_expr_getop
1701   \csname .=\XINT_expr_unlock #2%
1702   \expandafter\XINT_expr_rseq:_aa
1703       \romannumeral`&&@\XINT_expr_unlock #1!{#5#4\relax !#3}#2\endcsname
1704 }%
1705 \def\XINT_expr_rseq:_aa #1{\if +#1\expandafter\XINT_expr_rseq:_A\else
1706       \expandafter\XINT_expr_rseq:_a\fi #1}%

```

10.44.3 \XINT_expr_rseq:_a etc...

```

1707 \def\XINT_expr_rseq:_a #1!#2#3{\XINT_expr_rseq:_b {#3}{#2}#1,^,%}
1708 \def\XINT_expr_rseq:_b #1#2#3#4,%{
1709   \if ,#3\xint_dothis\XINT_expr_rseq:_noop\fi
1710   \if ^#3\xint_dothis\XINT_expr_rseq:_end\fi
1711   \xint_orthat{\expandafter\XINT_expr_rseq:_c}\csname .=#3#4\endcsname
1712   {#1}{#2}%
1713 }%
1714 \def\XINT_expr_rseq:_noop\csname .=#1\endcsname #2#3{\XINT_expr_rseq:_b {#2}{#3}#1,%}
1715 \def\XINT_expr_rseq:_end \csname .=#1\endcsname #1#2{%}
1716 \def\XINT_expr_rseq:_c #1#2#3%
1717   {\expandafter\XINT_expr_rseq:_d\romannumeral`&&@#3#1~#2{#3}}%
1718 \def\XINT_expr_rseq:_d #1{%
1719   \if ^#1\xint_dothis\XINT_expr_rseq:_abort\fi
1720   \if ?#1\xint_dothis\XINT_expr_rseq:_break\fi
1721   \if !#1\xint_dothis\XINT_expr_rseq:_omit\fi
1722   \xint_orthat{\XINT_expr_rseq:_goon #1}%}
1723 \def\XINT_expr_rseq:_goon #1!#2#3~#4#5{,#1\expandafter\XINT_expr_rseq:_b
1724   \romannumeral0\XINT_expr_lockit {#1}{#5}}%
1725 \def\XINT_expr_rseq:_omit #1!#2#3~{\XINT_expr_rseq:_b }%
1726 \def\XINT_expr_rseq:_abort #1!#2#3~#4#5#6^,{,%}
1727 \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.

```

1728 \def\XINT_expr_rseq:_A +#1!#2#3{\XINT_expr_rseq:_D #1#3{#2}}%
1729 \def\XINT_expr_rseq:_D #1#2#3%
1730   {\expandafter\XINT_expr_rseq:_E\romannumeral`&&@#3#1~#2{#3}}%
1731 \def\XINT_expr_rseq:_E #1{\if #1^\xint_dothis\XINT_expr_rseq:_Abort\fi
1732   \if #1?\xint_dothis\XINT_expr_rseq:_Break\fi
1733   \if #1!\xint_dothis\XINT_expr_rseq:_Omit\fi
1734   \xint_orthat{\XINT_expr_rseq:_Goon #1}%}
1735 \def\XINT_expr_rseq:_Goon #1!#2#3~#4#5%
1736   {,#1\expandafter\XINT_expr_rseq:_D
1737   \csname .=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1738   \romannumeral0\XINT_expr_lockit{#1}{#5}}%

```

```

1739 \def\XINT_expr_rseq:_Omit #1!#2#3~%#4#5%
1740     {\expandafter\XINT_expr_rseq:_D
1741         \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname }%
1742 \def\XINT_expr_rseq:_Abort #1!#2#3~#4#5{ }%
1743 \def\XINT_expr_rseq:_Break #1!#2#3~#4#5{,#1}%

```

10.45 iter

10.45.1	<code>\XINT_expr_iterx</code>	...	341
10.45.2	<code>\XINT_expr_itory</code>	...	341
10.45.3	<code>\XINT_expr_iter:_a</code> etc...	...	342
10.45.4	<code>\XINT_expr_iter:_A</code> etc...	...	342

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`.

```

1744 \def\XINT_expr_func_iter {\XINT_allexpr_iter \xintbareeval \xintthebareeval }%
1745 \def\XINT_flexpr_func_iter {\XINT_allexpr_iter \xintbarefloateval \xintthebarefloateval }%
1746 \def\XINT_iexpr_func_iter {\XINT_allexpr_iter \xintbareiieval \xintthebareiieval }%
1747 \def\XINT_allexpr_iter #1#2#3%
1748 {%
1749     \expandafter\XINT_expr_iterx\expandafter #1\expandafter#2\expandafter
1750     #3\romannumeral`&&\XINT_expr_onliteral_seq_a { }%
1751 }%

```

10.45.1 `\XINT_expr_iterx`

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

```

1752 \def\XINT_expr_iterx #1#2#3#4#5%
1753 {%
1754     \expandafter\XINT_expr_itory\romannumeral0#1(#5)\relax #3#4#2%
1755 }%

```

10.45.2 `\XINT_expr_itory`

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

```

1756 \def\XINT_expr_itory #1#2#3#4#5%
1757 {%
1758     \expandafter \XINT_expr_getop
1759     \csname .=%
1760     \expandafter\XINT_expr_iter:_aa
1761     \romannumeral`&&\XINT_expr_unlock #1!{#5#4\relax !#3}#2\endcsname
1762 }%
1763 \def\XINT_expr_iter:_aa #1{\if +#1\expandafter\XINT_expr_iter:_A\else
1764     \expandafter\XINT_expr_iter:_a\fi #1}%

```

10.45.3 `\XINT_expr_iter:_a` etc...

```

1765 \def\XINT_expr_iter:_a #1!#2#3{\XINT_expr_iter:_b {#3}{#2}#1,^,%}
1766 \def\XINT_expr_iter:_b #1#2#3#4,{%
1767     \if ,#3\xint_dothis\XINT_expr_iter:_noop\fi
1768     \if ^#3\xint_dothis\XINT_expr_iter:_end\fi
1769     \xint_orthat{\expandafter\XINT_expr_iter:_c}%
1770     \csname.=#3#4\endcsname {#1}{#2}%
1771 }%
1772 \def\XINT_expr_iter:_noop\csname.=,#1\endcsname #2#3{\XINT_expr_iter:_b {#2}{#3}#1,%}
1773 \def\XINT_expr_iter:_end \csname.=^\endcsname #1#2{\XINT_expr:_unlock #1}%
1774 \def\XINT_expr_iter:_c #1#2#3%
1775     {\expandafter\XINT_expr_iter:_d\romannumeral`&&@#3#1~#2{#3}}%
1776 \def\XINT_expr_iter:_d #1{%
1777     \if ^#1\xint_dothis\XINT_expr_iter:_abort\fi
1778     \if ?#1\xint_dothis\XINT_expr_iter:_break\fi
1779     \if !#1\xint_dothis\XINT_expr_iter:_omit\fi
1780     \xint_orthat{\XINT_expr_iter:_goon #1}%
1781 \def\XINT_expr_iter:_goon #1!#2#3~#4#5%
1782     {\expandafter\XINT_expr_iter:_b\romannumeral0\XINT_expr_lockit {#1}{#5}}%
1783 \def\XINT_expr_iter:_omit #1!#2#3~{\XINT_expr_iter:_b }%
1784 \def\XINT_expr_iter:_abort #1!#2#3~#4#5#6^,{\XINT_expr_unlock #4}%
1785 \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.

```

1786 \def\XINT_expr_iter:_A +#1!#2#3{\XINT_expr_iter:_D #1#3{#2}}%
1787 \def\XINT_expr_iter:_D #1#2#3%
1788     {\expandafter\XINT_expr_iter:_E\romannumeral`&&@#3#1~#2{#3}}%
1789 \def\XINT_expr_iter:_E #1{\if #1^\xint_dothis\XINT_expr_iter:_Abort\fi
1790     \if #1?\xint_dothis\XINT_expr_iter:_Break\fi
1791     \if #1!\xint_dothis\XINT_expr_iter:_Omit\fi
1792     \xint_orthat{\XINT_expr_iter:_Goon #1}%
1793 \def\XINT_expr_iter:_Goon #1!#2#3~#4#5%
1794     {\expandafter\XINT_expr_iter:_D
1795     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1796     \romannumeral0\XINT_expr_lockit{#1}{#5}}%
1797 \def\XINT_expr_iter:_Omit #1!#2#3~%#4#5%
1798     {\expandafter\XINT_expr_iter:_D
1799     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname }%
1800 \def\XINT_expr_iter:_Abort #1!#2#3~#4#5{\XINT_expr:_unlock #4}%
1801 \def\XINT_expr_iter:_Break #1!#2#3~#4#5{#1}%

```

10.46 rrseq

10.46.1	<code>\XINT_expr_rrseqx</code>	343
10.46.2	<code>\XINT_expr_rrseqy</code>	343
10.46.3	<code>\XINT_expr_rrseq:_a</code> etc...	343
10.46.4	<code>\XINT_expr_rrseq:_A</code> etc...	344

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.

```

1802 \def\XINT_expr_func_rrseq  {\XINT_allexpr_rrseq \xintbareeval \xintthebareeval }%
1803 \def\XINT_flexpr_func_rrseq {\XINT_allexpr_rrseq \xintbarefloateval \xintthebarefloateval }%
1804 \def\XINT_iiexpr_func_rrseq {\XINT_allexpr_rrseq \xintbareiieval \xintthebareiieval }%
1805 \def\XINT_allexpr_rrseq #1#2#3%
1806 {%
1807   \expandafter\XINT_expr_rrseq\expandafter #1\expandafter#2\expandafter
1808   #3\romannumeral`&&\XINT_expr_onliteral_seq_a }%
1809 }%

```

10.46.1 \XINT_expr_rrseqx

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

```

1810 \def\XINT_expr_rrseqx #1#2#3#4#5%
1811 {%
1812   \expandafter\XINT_expr_rrseq\romannumeral0#1(#5)\expandafter\relax
1813   \expandafter{\romannumeral0\xintapply \XINT_expr_lockit
1814     {\xintRevWithBraces{\xintCSVtoListNonStripped{\XINT_expr_unlock #3}}}%
1815   #3#4#2%
1816 }%

```

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

```

1817 \def\XINT_expr_rrseqy #1#2#3#4#5#6%
1818 {%
1819   \expandafter \XINT_expr_getop
1820   \csname .=\XINT_expr_unlock #3%
1821   \expandafter\XINT_expr_rrseq:_aa
1822     \romannumeral`&&\XINT_expr_unlock #1!{#6#5\relax !#4}{#2}\endcsname
1823 }%
1824 \def\XINT_expr_rrseq:_aa #1{\if +#1\expandafter\XINT_expr_rrseq:_A\else
1825   \expandafter\XINT_expr_rrseq:_a\fi #1}%

```

10.46.3 \XINT_expr_rrseq:_a etc...

Attention que ? a catcode 3 ici et dans iter.

```

1826 \catcode`? 3
1827 \def\XINT_expr_rrseq:_a #1!#2#3{\XINT_expr_rrseq:_b {#3}{#2}#1,^,%}
1828 \def\XINT_expr_rrseq:_b #1#2#3#4,{%
1829   \if ,#3\xint_dothis\XINT_expr_rrseq:_noop\fi
1830   \if ^#3\xint_dothis\XINT_expr_rrseq:_end\fi
1831   \xint_orthat{\expandafter\XINT_expr_rrseq:_c}\csname .=#3#4\endcsname
1832   {#1}{#2}%
1833 }%
1834 \def\XINT_expr_rrseq:_noop\csname .=#1\endcsname #2#3{\XINT_expr_rrseq:_b {#2}{#3}#1,%}
1835 \def\XINT_expr_rrseq:_end \csname .=#1\endcsname #1#2{%}
1836 \def\XINT_expr_rrseq:_c #1#2#3%
1837   {\expandafter\XINT_expr_rrseq:_d\romannumeral`&&#3#1~#2?{#3}}%

```

```

1838 \def\XINT_expr_rrseq:_d #1{%
1839   \if ^#1\xint_dothis\XINT_expr_rrseq:_abort\fi
1840   \if ?#1\xint_dothis\XINT_expr_rrseq:_break\fi
1841   \if !#1\xint_dothis\XINT_expr_rrseq:_omit\fi
1842   \xint_orthat{\XINT_expr_rrseq:_goon #1}%
1843 }%
1844 \def\XINT_expr_rrseq:_goon #1!#2#3~#4?#5{,#1\expandafter\XINT_expr_rrseq:_b\expandafter
1845   {\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
1846 \def\XINT_expr_rrseq:_omit #1!#2#3~{\XINT_expr_rrseq:_b }%
1847 \def\XINT_expr_rrseq:_abort #1!#2#3~#4?#5#6^,{ }%
1848 \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.

```

1849 \def\XINT_expr_rrseq:_A +#1!#2#3{\XINT_expr_rrseq:_D #1{#3}{#2}}%
1850 \def\XINT_expr_rrseq:_D #1#2#3%
1851   {\expandafter\XINT_expr_rrseq:_E\romannumeral`&&@#3#1~#2?{#3}}%
1852 \def\XINT_expr_rrseq:_Goon #1!#2#3~#4?#5%
1853   {,#1\expandafter\XINT_expr_rrseq:_D
1854     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1855     \expandafter{\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
1856 \def\XINT_expr_rrseq:_Omit #1!#2#3~%#4?#5%
1857   {\expandafter\XINT_expr_rrseq:_D
1858     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
1859 \def\XINT_expr_rrseq:_Abort #1!#2#3~#4?#5}%
1860 \def\XINT_expr_rrseq:_Break #1!#2#3~#4?#5{,#1}%
1861 \def\XINT_expr_rrseq:_E #1{\if #1^\xint_dothis\XINT_expr_rrseq:_Abort\fi
1862   \if #1?\xint_dothis\XINT_expr_rrseq:_Break\fi
1863   \if #1!\xint_dothis\XINT_expr_rrseq:_Omit\fi
1864   \xint_orthat{\XINT_expr_rrseq:_Goon #1}}%

```

10.47 iterr

10.47.1	\XINT_expr_iterrx	344
10.47.2	\XINT_expr_ittery	345
10.47.3	\XINT_expr_itter:_a etc.	345
10.47.4	\XINT_expr_itter:_A etc.	346

```

1865 \def\XINT_expr_func_itter   {\XINT_allexpr_itter \xintbareeval      \xintthebareeval      }%
1866 \def\XINT_flexpr_func_itter {\XINT_allexpr_itter \xintbarefloateval \xintthebarefloateval }%
1867 \def\XINT_iexpr_func_itter  {\XINT_allexpr_itter \xintbareiieval   \xintthebareiieval   }%
1868 \def\XINT_allexpr_itter #1#2#3%
1869 {%
1870   \expandafter\XINT_expr_iterrx\expandafter #1\expandafter #2\expandafter
1871   #3\romannumeral`&&@\XINT_expr_onliteral_seq_a }%
1872 }%

```

10.47.1 \XINT_expr_iterrx

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


```

1873 \def\XINT_expr_iterrx #1#2#3#4#5%
1874 {%
1875   \expandafter\XINT_expr_itterry\romannumeral0#1(#5)\expandafter\relax
1876   \expandafter{\romannumeral0\xintapply \XINT_expr_lockit
1877     {\xintRevWithBraces{\xintCSVtoListNonStripped{\XINT_expr_unlock #3}}}}%
1878   #3#4#2%
1879 }%

```

10.47.2 \XINT_expr_itterry

#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 \xintbareieval

```

1880 \def\XINT_expr_itterry #1#2#3#4#5#6%
1881 {%
1882   \expandafter \XINT_expr_getop
1883   \csname .=%
1884   \expandafter\XINT_expr_iterr:_aa
1885   \romannumeral`&&\XINT_expr_unlock #1!{#6#5\relax !#4}{#2}\endcsname
1886 }%
1887 \def\XINT_expr_iterr:_aa #1{\if +#1\expandafter\XINT_expr_iterr:_A\else
1888   \expandafter\XINT_expr_iterr:_a\fi #1}%

```

10.47.3 \XINT_expr_iterr:_a etc...

```

1889 \def\XINT_expr_iterr:_a #1!#2#3{\XINT_expr_iterr:_b {#3}{#2}#1,^,%}
1890 \def\XINT_expr_iterr:_b #1#2#3#4,{%
1891   \if ,#3\xint_dothis\XINT_expr_iterr:_noop\fi
1892   \if ^#3\xint_dothis\XINT_expr_iterr:_end\fi
1893   \xint_orthat{\expandafter\XINT_expr_iterr:_c}%
1894   \csname .=#3#4\endcsname {#1}{#2}%
1895 }%
1896 \def\XINT_expr_iterr:_noop\csname .=#1\endcsname #2#3{\XINT_expr_iterr:_b {#2}{#3}#1,%}
1897 \def\XINT_expr_iterr:_end \csname .=#1\endcsname #1#2%
1898   {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1899     {\XINT_expr:_unlock}{\xintReverseOrder{#1\space}}}%
1900 \def\XINT_expr_iterr:_c #1#2#3%
1901   {\expandafter\XINT_expr_iterr:_d\romannumeral`&&#3#1~#2?{#3}}%
1902 \def\XINT_expr_iterr:_d #1{%
1903   \if ^#1\xint_dothis\XINT_expr_iterr:_abort\fi
1904   \if ?#1\xint_dothis\XINT_expr_iterr:_break\fi
1905   \if !#1\xint_dothis\XINT_expr_iterr:_omit\fi
1906   \xint_orthat{\XINT_expr_iterr:_goon #1}%
1907 }%
1908 \def\XINT_expr_iterr:_goon #1!#2#3~#4?#5{\expandafter\XINT_expr_iterr:_b\expandafter
1909   {\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}{#4}}{#5}}}%
1910 \def\XINT_expr_iterr:_omit #1!#2#3~{\XINT_expr_iterr:_b }%
1911 \def\XINT_expr_iterr:_abort #1!#2#3~#4?#5#6^,%
1912   {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1913     {\XINT_expr:_unlock}{\xintReverseOrder{#4\space}}}%
1914 \def\XINT_expr_iterr:_break #1!#2#3~#4?#5#6^,%
1915   {\expandafter\xint_gobble_iv\romannumeral0\xintapplyunbraced

```

```

1916      {\XINT_expr:_unlock}{\xintReverseOrder{#4\space}},#1}%
1917 \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.

1918 \def\XINT_expr_iterr:_A +#1!#2#3{\XINT_expr_iterr:_D #1{#3}{#2}}%
1919 \def\XINT_expr_iterr:_D #1#2#3%
1920   {\expandafter\XINT_expr_iterr:_E\romannumeral`&&@#3#1~#2?{#3}}%
1921 \def\XINT_expr_iterr:_Goon #1!#2#3~#4?#5%
1922   {\expandafter\XINT_expr_iterr:_D
1923     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1924     \expandafter{\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}{#4}}{#5}}%
1925 \def\XINT_expr_iterr:_Omit #1!#2#3~%#4?#5%
1926   {\expandafter\XINT_expr_iterr:_D
1927     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
1928 \def\XINT_expr_iterr:_Abort #1!#2#3~#4?#5%
1929   {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1930     {\XINT_expr:_unlock}{\xintReverseOrder{#4\space}}}%
1931 \def\XINT_expr_iterr:_Break #1!#2#3~#4?#5%
1932   {\expandafter\xint_gobble_iv\romannumeral0\xintapplyunbraced
1933     {\XINT_expr:_unlock}{\xintReverseOrder{#4\space}},#1}%
1934 \def\XINT_expr_iterr:_E #1{\if #1^\xint_dothis\XINT_expr_iterr:_Abort\fi
1935     \if #1?\xint_dothis\XINT_expr_iterr:_Break\fi
1936     \if #1!\xint_dothis\XINT_expr_iterr:_Omit\fi
1937     \xint_orthat{\XINT_expr_iterr:_Goon #1}}%
1938 \catcode`? 11

```

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

10.48.1	<code>\xintANDof:csv</code>	347
10.48.2	<code>\xintORof:csv</code>	347
10.48.3	<code>\xintXORof:csv</code>	347
10.48.4	Generic csv routine	347
10.48.5	<code>\xintMaxof:csv, \xintiiMaxof:csv</code>	348
10.48.6	<code>\xintMinof:csv, \xintiiMinof:csv</code>	348
10.48.7	<code>\xintSum:csv, \xintiiSum:csv</code>	348
10.48.8	<code>\xintPrd:csv, \xintiiPrd:csv</code>	348
10.48.9	<code>\xintGCDof:csv, \xintLCMof:csv</code>	348
10.48.10	<code>\xintiiGCDof:csv, \xintiiLCMof:csv</code>	349
10.48.11	<code>\XINTinFloatdigits, \XINTinFloatSqrtdigits, \XINTinFloatFacdigits</code>	349
10.48.12	<code>\XINTinFloatMaxof:csv, \XINTinFloatMinof:csv</code>	349
10.48.13	<code>\XINTinFloatSum:csv, \XINTinFloatPrd:csv</code>	349

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.

```
1939 \def\xintANDof:csv #1{\expandafter\xint_andof:_a\romannumeral`&&@#1,,^}%
1940 \def\xint_andof:_a #1{\if ,#1\expandafter\xint_andof:_e
1941         \else\expandafter\xint_andof:_c\fi #1}%
1942 \def\xint_andof:_c #1,{\xintiifNotZero {#1}{\XINT_andof:_a}{\XINT_andof:_no}}%
1943 \def\xint_andof:_no #1^{0}%
1944 \def\xint_andof:_e #1^{1}% works with empty list
```

10.48.2 `\xintORof:csv`

1.09a. For use by `\xintexpr`.

```
1945 \def\xintORof:csv #1{\expandafter\xint_orof:_a\romannumeral`&&@#1,,^}%
1946 \def\xint_orof:_a #1{\if ,#1\expandafter\xint_orof:_e
1947         \else\expandafter\xint_orof:_c\fi #1}%
1948 \def\xint_orof:_c #1,{\xintiifNotZero{#1}{\XINT_orof:_yes}{\XINT_orof:_a}}%
1949 \def\xint_orof:_yes #1^{1}%
1950 \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`).

```
1951 \def\xintXORof:csv #1{\expandafter\xint_xorof:_a\expandafter 0\romannumeral`&&@#1,,^}%
1952 \def\xint_xorof:_a #1#2,{\XINT_xorof:_b #2,#1}%
1953 \def\xint_xorof:_b #1{\if ,#1\expandafter\xint_xorof:_e
1954         \else\expandafter\xint_xorof:_c\fi #1}%
1955 \def\xint_xorof:_c #1,#2%
1956         {\xintiifNotZero {#1}{\if #20\xint_afterfi{\XINT_xorof:_a 1}%
1957                 \else\xint_afterfi{\XINT_xorof:_a 0}\fi}%
1958                 {\XINT_xorof:_a #2}%
1959         }%
1960 \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.

```
1961 \def\xint_oncsv:_empty #1,^,#2{#2}%
1962 \def\xint_oncsv:_end ^,#1#2#3#4{#1}%
1963 \def\xint_oncsv:_a #1#2#3%
1964     {\if ,#3\expandafter\xint_oncsv:_empty\else\expandafter\xint_oncsv:_b\fi #1#2#3}%
1965 \def\xint_oncsv:_b #1#2#3,%
1966     {\expandafter\xint_oncsv:_c \expandafter{\romannumeral`&&@#2{#3}}#1#2}%
1967 \def\xint_oncsv:_c #1#2#3#4,{\expandafter\xint_oncsv:_d \romannumeral`&&@#4,{#1}#2#3}%
1968 \def\xint_oncsv:_d #1%
1969     {\if ^#1\expandafter\xint_oncsv:_end\else\expandafter\xint_oncsv:_e\fi #1}%
1970 \def\xint_oncsv:_e #1,#2#3#4%
1971     {\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`.

```
1972 \def\xintMaxof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmax
1973           \expandafter\xint_firstofone\romannumeral`&&@#1,^{0/1[0]}}%
1974 \def\xintiiMaxof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiimax
1975           \expandafter\xint_firstofone\romannumeral`&&@#1,^{0}%
```

10.48.6 `\xintMinof:csv`, `\xintiiMinof:csv`

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

```
1976 \def\xintMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmin
1977           \expandafter\xint_firstofone\romannumeral`&&@#1,^{0/1[0]}}%
1978 \def\xintiiMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiimin
1979           \expandafter\xint_firstofone\romannumeral`&&@#1,^{0}%
```

10.48.7 `\xintSum:csv`, `\xintiiSum:csv`

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

```
1980 \def\xintSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintadd
1981           \expandafter\xint_firstofone\romannumeral`&&@#1,^{0/1[0]}}%
1982 \def\xintiiSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiadd
1983           \expandafter\xint_firstofone\romannumeral`&&@#1,^{0}%
```

10.48.8 `\xintPrd:csv`, `\xintiiPrd:csv`

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

```
1984 \def\xintPrd:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmul
1985           \expandafter\xint_firstofone\romannumeral`&&@#1,^{1/1[0]}}%
1986 \def\xintiiPrd:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiimul
1987           \expandafter\xint_firstofone\romannumeral`&&@#1,^{1}%
```

10.48.9 `\xintGCDof:csv`, `\xintLCMof:csv`

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

```
1988 \def\xintGCDof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintgcd
1989           \expandafter\xint_firstofone\romannumeral`&&@#1,^{1}%
1990 \def\xintLCMof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintlcm
1991           \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.

```
1992 \def\xintiiGCDof:csv #1{\expandafter\XINT_oncsvg:_a\expandafter\xintiigcd
1993           \expandafter\xint_firstofone\romannumeral`&&@#1,^,1}%
1994 \def\xintiilCMof:csv #1{\expandafter\XINT_oncsvg:_a\expandafter\xintiilcm
1995           \expandafter\xint_firstofone\romannumeral`&&@#1,^,0}%
```

10.48.11 `\XINTinFloatdigits`, `\XINTinFloatSqrtdigits`, `\XINTinFloatFacdigits`

for `\xintNewExpr` matters, mainly.

```
1996 \def\XINTinFloatdigits {\XINTinFloat [\XINTdigits]]}%
1997 \def\XINTinFloatSqrtdigits {\XINTinFloatSqrt[\XINTdigits]]}%
1998 \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

```
1999 \def\XINTinFloatMaxof:csv #1{\expandafter\XINT_oncsvg:_a\expandafter\xintmax
2000           \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^,{0[0]}}}%
2001 \def\XINTinFloatMinof:csv #1{\expandafter\XINT_oncsvg:_a\expandafter\xintmin
2002           \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^,{0[0]}}}%
```

10.48.13 `\XINTinFloatSum:csv`, `\XINTinFloatPrd:csv`

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

```
2003 \def\XINTinFloatSum:csv #1{\expandafter\XINT_oncsvg:_a\expandafter\XINTinfloatadd
2004           \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^,{0[0]}}}%
2005 \def\XINTinFloatPrd:csv #1{\expandafter\XINT_oncsvg:_a\expandafter\XINTinfloatmul
2006           \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`, `divmod`, `gcd`, `lcm`, `max`, `min`, ``+``, ``*``, `?`, `!`, `not`, `all`, `any`, `xor`, `if`, `ifsgn`, `even`, `odd`, `first`, `last`, `len`, `reversed`, `factorial` and `binomial` functions

```
2007 \def\XINT_expr_twoargs #1,#2,{{#1}{#2}}}%
2008 \def\XINT_expr_totwoargs #1#2{#1,#2}%
2009 \def\XINT_expr_argadopt #1,#2,#3.#4#5%
2010 {%
2011   \if\relax#3\relax\expandafter\xint_firstoftwo\else
2012     \expandafter\xint_secondoftwo\fi
2013   {#4}{#5[\xintNum {#2}]]{#1}}%
2014 }%
2015 \def\XINT_expr_oneortwo #1#2#3,#4,#5.%
2016 {%
2017   \if\relax#5\relax\expandafter\xint_firstoftwo\else
2018     \expandafter\xint_secondoftwo\fi
```

```

2019      {#1{0}}{#2{\xintNum {#4}}{#3}%
2020 }%
2021 \def\xINT_iiexpr_oneortwo #1#2,#3,#4.%
2022 {%
2023     \if\relax#4\relax\expandafter\xint_firstoftwo\else
2024         \expandafter\xint_secondoftwo\fi
2025     {#1{0}}{#1{#3}}{#2}%
2026 }%
2027 \def\xINT_expr_func_num #1#2#3%
2028     {\expandafter #1\expandafter #2\csname.=\xintNum {\xINT_expr_unlock #3}\endcsname }%
2029 \let\xINT_flexpr_func_num\xINT_expr_func_num
2030 \let\xINT_iiexpr_func_num\xINT_expr_func_num
2031 % [0] added Oct 25. For interaction with SPRaw::csv
2032 \def\xINT_expr_func_reduce #1#2#3%
2033     {\expandafter #1\expandafter #2\csname.=\xintIrr {\xINT_expr_unlock #3}[0]\endcsname }%
2034 \let\xINT_flexpr_func_reduce\xINT_expr_func_reduce
2035 % no \XINT_iiexpr_func_reduce
2036 \def\xINT_expr_func_abs #1#2#3%
2037     {\expandafter #1\expandafter #2\csname.=\xintAbs {\xINT_expr_unlock #3}\endcsname }%
2038 \let\xINT_flexpr_func_abs\xINT_expr_func_abs
2039 \def\xINT_iiexpr_func_abs #1#2#3%
2040     {\expandafter #1\expandafter #2\csname.=\xintiAbs {\xINT_expr_unlock #3}\endcsname }%
2041 \def\xINT_expr_func_sgn #1#2#3%
2042     {\expandafter #1\expandafter #2\csname.=\xintSgn {\xINT_expr_unlock #3}\endcsname }%
2043 \let\xINT_flexpr_func_sgn\xINT_expr_func_sgn
2044 \def\xINT_iiexpr_func_sgn #1#2#3%
2045     {\expandafter #1\expandafter #2\csname.=\xintiSgn {\xINT_expr_unlock #3}\endcsname }%
2046 \def\xINT_expr_func_frac #1#2#3%
2047     {\expandafter #1\expandafter #2\csname.=\xintTFrac {\xINT_expr_unlock #3}\endcsname }%
2048 \def\xINT_flexpr_func_frac #1#2#3{\expandafter #1\expandafter #2\csname
2049     .=\XINTinFloatFracdigits {\xINT_expr_unlock #3}\endcsname }%

no \XINT_iiexpr_func_frac

2050 \def\xINT_expr_func_floor #1#2#3%
2051     {\expandafter #1\expandafter #2\csname .=\xintFloor {\xINT_expr_unlock #3}\endcsname }%
2052 \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.

2053 \def\xINT_iiexpr_func_floor #1#2#3%
2054 {%
2055     \expandafter #1\expandafter #2\csname.=\xintiFloor {\xINT_expr_unlock #3}\endcsname }%
2056 \def\xINT_expr_func_ceil #1#2#3%
2057     {\expandafter #1\expandafter #2\csname .=\xintCeil {\xINT_expr_unlock #3}\endcsname }%
2058 \let\xINT_flexpr_func_ceil\xINT_expr_func_ceil
2059 \def\xINT_iiexpr_func_ceil #1#2#3%
2060 {%
2061     \expandafter #1\expandafter #2\csname.=\xintiCeil {\xINT_expr_unlock #3}\endcsname }%
2062 \def\xINT_expr_func_sqr #1#2#3%
2063     {\expandafter #1\expandafter #2\csname.=\xintSqr {\xINT_expr_unlock #3}\endcsname }%
2064 \def\xINT_flexpr_func_sqr #1#2#3%
2065 {%

```

```

2066 \expandafter #1\expandafter #2\csname
2067 .=\XINTinFloatMul{\XINT_expr_unlock #3}{\XINT_expr_unlock #3}\endcsname
2068 }%
2069 \def\XINT_expr_func_factorial #1#2#3%
2070 {%
2071 \expandafter #1\expandafter #2\csname .=%
2072 \expandafter\XINT_expr_argandopt
2073 \romannumeral`&&\XINT_expr_unlock#3,,.\xintFac\XINTinFloatFac
2074 \endcsname
2075 }%
2076 \def\XINT_flexpr_func_factorial #1#2#3%
2077 {%
2078 \expandafter #1\expandafter #2\csname .=%
2079 \expandafter\XINT_expr_argandopt
2080 \romannumeral`&&\XINT_expr_unlock#3,,.\XINTinFloatFacdigits\XINTinFloatFac
2081 \endcsname
2082 }%
2083 \def\XINT_iiexpr_func_factorial #1#2#3%
2084 {%
2085 \expandafter #1\expandafter #2\csname.=\xintiiFac{\XINT_expr_unlock #3}\endcsname
2086 }%
2087 \def\XINT_iiexpr_func_sqr #1#2#3%
2088 {\expandafter #1\expandafter #2\csname.=\xintiiSqr {\XINT_expr_unlock #3}\endcsname }%
2089 \def\XINT_expr_func_sqrt #1#2#3%
2090 {%
2091 \expandafter #1\expandafter #2\csname .=%
2092 \expandafter\XINT_expr_argandopt
2093 \romannumeral`&&\XINT_expr_unlock#3,,.\XINTinFloatSqrtdigits\XINTinFloatSqrt
2094 \endcsname
2095 }%
2096 \let\XINT_flexpr_func_sqrt\XINT_expr_func_sqrt
2097 \def\XINT_iiexpr_func_sqrt #1#2#3%
2098 {\expandafter #1\expandafter #2\csname.=\xintiiSqrt {\XINT_expr_unlock #3}\endcsname }%
2099 \def\XINT_iiexpr_func_sqrtr #1#2#3%
2100 {\expandafter #1\expandafter #2\csname.=\xintiiSqrR {\XINT_expr_unlock #3}\endcsname }%
2101 \def\XINT_expr_func_round #1#2#3%
2102 {%
2103 \expandafter #1\expandafter #2\csname .=%
2104 \expandafter\XINT_expr_oneortwo
2105 \expandafter\xintiRound\expandafter\xintRound
2106 \romannumeral`&&\XINT_expr_unlock #3,,.\endcsname
2107 }%
2108 \let\XINT_flexpr_func_round\XINT_expr_func_round
2109 \def\XINT_iiexpr_func_round #1#2#3%
2110 {%
2111 \expandafter #1\expandafter #2\csname .=%
2112 \expandafter\XINT_iiexpr_oneortwo\expandafter\xintiRound
2113 \romannumeral`&&\XINT_expr_unlock #3,,.\endcsname
2114 }%
2115 \def\XINT_expr_func_trunc #1#2#3%
2116 {%
2117 \expandafter #1\expandafter #2\csname .=%

```

```

2118 \expandafter\XINT_expr_oneortwo
2119 \expandafter\xintiTrunc\expandafter\xintTrunc
2120 \romannumeral`&&\XINT_expr_unlock #3,,.\endcsname
2121 }%
2122 \let\XINT_flexpr_func_trunc\XINT_expr_func_trunc
2123 \def\XINT_iiexpr_func_trunc #1#2#3%
2124 {%
2125 \expandafter #1\expandafter #2\csname .=%
2126 \expandafter\XINT_iiexpr_oneortwo\expandafter\xintiTrunc
2127 \romannumeral`&&\XINT_expr_unlock #3,,.\endcsname
2128 }%
2129 \def\XINT_expr_func_float #1#2#3%
2130 {%
2131 \expandafter #1\expandafter #2\csname .=%
2132 \expandafter\XINT_expr_argandopt
2133 \romannumeral`&&\XINT_expr_unlock #3,,.\XINTinFloatdigits\XINTinFloat
2134 \endcsname
2135 }%
2136 \let\XINT_flexpr_func_float\XINT_expr_func_float
2137 % \XINT_iiexpr_func_float not defined
2138 \def\XINT_expr_func_divmod #1#2#3%
2139 {%
2140 \expandafter #1\expandafter #2\csname .=%
2141 \expandafter\XINT_expr_totwoargs
2142 \romannumeral0\expandafter\xintdivmod
2143 \romannumeral`&&\expandafter\XINT_expr_twoargs
2144 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2145 }%
2146 \def\XINT_flexpr_func_divmod #1#2#3%
2147 {%
2148 \expandafter #1\expandafter #2\csname .=%
2149 \expandafter\XINTinFloatDivMod
2150 \romannumeral`&&\expandafter\XINT_expr_twoargs
2151 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2152 }%
2153 \def\XINT_iiexpr_func_divmod #1#2#3%
2154 {%
2155 \expandafter #1\expandafter #2\csname .=%
2156 \expandafter\XINT_expr_totwoargs
2157 \romannumeral0\expandafter\xintiidivmod
2158 \romannumeral`&&\expandafter\XINT_expr_twoargs
2159 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2160 }%
2161 \def\XINT_expr_func_mod #1#2#3%
2162 {%
2163 \expandafter #1\expandafter #2\csname .=%
2164 \expandafter\expandafter\expandafter\xintMod
2165 \expandafter\XINT_expr_twoargs
2166 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2167 }%
2168 \def\XINT_flexpr_func_mod #1#2#3%
2169 {%

```



```

2170 \expandafter #1\expandafter #2\csname .=%
2171 \expandafter\XINTinFloatMod
2172 \romannumeral`&&\expandafter\XINT_expr_twoargs
2173 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2174 }%
2175 \def\XINT_iiexpr_func_mod #1#2#3%
2176 {%
2177 \expandafter #1\expandafter #2\csname .=%
2178 \expandafter\expandafter\expandafter\xintiMod
2179 \expandafter\XINT_expr_twoargs
2180 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2181 }%
2182 \def\XINT_expr_func_binomial #1#2#3%
2183 {%
2184 \expandafter #1\expandafter #2\csname .=%
2185 \expandafter\expandafter\expandafter\xintBinomial
2186 \expandafter\XINT_expr_twoargs
2187 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2188 }%
2189 \def\XINT_flexpr_func_binomial #1#2#3%
2190 {%
2191 \expandafter #1\expandafter #2\csname .=%
2192 \expandafter\expandafter\expandafter\XINTinFloatBinomial
2193 \expandafter\XINT_expr_twoargs
2194 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2195 }%
2196 \def\XINT_iiexpr_func_binomial #1#2#3%
2197 {%
2198 \expandafter #1\expandafter #2\csname .=%
2199 \expandafter\expandafter\expandafter\xintiBinomial
2200 \expandafter\XINT_expr_twoargs
2201 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2202 }%
2203 \def\XINT_expr_func_pfactorial #1#2#3%
2204 {%
2205 \expandafter #1\expandafter #2\csname .=%
2206 \expandafter\expandafter\expandafter\xintPFactorial
2207 \expandafter\XINT_expr_twoargs
2208 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2209 }%
2210 \def\XINT_flexpr_func_pfactorial #1#2#3%
2211 {%
2212 \expandafter #1\expandafter #2\csname .=%
2213 \expandafter\expandafter\expandafter\XINTinFloatPFactorial
2214 \expandafter\XINT_expr_twoargs
2215 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2216 }%
2217 \def\XINT_iiexpr_func_pfactorial #1#2#3%
2218 {%
2219 \expandafter #1\expandafter #2\csname .=%
2220 \expandafter\expandafter\expandafter\xintiiPFactorial
2221 \expandafter\XINT_expr_twoargs

```

```

2222 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2223 }%
2224 \def\XINT_expr_func_quo #1#2#3%
2225 {%
2226 \expandafter #1\expandafter #2\csname .=%
2227 \expandafter\expandafter\expandafter\xintiQuo
2228 \expandafter\XINT_expr_twoargs
2229 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2230 }%
2231 \let\XINT_flexpr_func_quo\XINT_expr_func_quo
2232 \def\XINT_iiexpr_func_quo #1#2#3%
2233 {%
2234 \expandafter #1\expandafter #2\csname .=%
2235 \expandafter\expandafter\expandafter\xintiiQuo
2236 \expandafter\XINT_expr_twoargs
2237 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2238 }%
2239 \def\XINT_expr_func_rem #1#2#3%
2240 {%
2241 \expandafter #1\expandafter #2\csname .=%
2242 \expandafter\expandafter\expandafter\xintiRem
2243 \expandafter\XINT_expr_twoargs
2244 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2245 }%
2246 \let\XINT_flexpr_func_rem\XINT_expr_func_rem
2247 \def\XINT_iiexpr_func_rem #1#2#3%
2248 {%
2249 \expandafter #1\expandafter #2\csname .=%
2250 \expandafter\expandafter\expandafter\xintiiRem
2251 \expandafter\XINT_expr_twoargs
2252 \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2253 }%
2254 \def\XINT_expr_func_gcd #1#2#3%
2255 {\expandafter #1\expandafter #2\csname
2256 .=\xintGCDof:csv{\XINT_expr_unlock #3}\endcsname }%
2257 \let\XINT_flexpr_func_gcd\XINT_expr_func_gcd
2258 \def\XINT_iiexpr_func_gcd #1#2#3%
2259 {\expandafter #1\expandafter #2\csname
2260 .=\xintiiGCDof:csv{\XINT_expr_unlock #3}\endcsname }%
2261 \def\XINT_expr_func_lcm #1#2#3%
2262 {\expandafter #1\expandafter #2\csname
2263 .=\xintLCMof:csv{\XINT_expr_unlock #3}\endcsname }%
2264 \let\XINT_flexpr_func_lcm\XINT_expr_func_lcm
2265 \def\XINT_iiexpr_func_lcm #1#2#3%
2266 {\expandafter #1\expandafter #2\csname
2267 .=\xintiiLCMof:csv{\XINT_expr_unlock #3}\endcsname }%
2268 \def\XINT_expr_func_max #1#2#3%
2269 {\expandafter #1\expandafter #2\csname
2270 .=\xintMaxof:csv{\XINT_expr_unlock #3}\endcsname }%
2271 \def\XINT_iiexpr_func_max #1#2#3%
2272 {\expandafter #1\expandafter #2\csname
2273 .=\xintiiMaxof:csv{\XINT_expr_unlock #3}\endcsname }%

```

```

2274 \def\XINT_flexpr_func_max #1#2#3%
2275     {\expandafter #1\expandafter #2\csname
2276         .=\XINTinFloatMaxof:csv{\XINT_expr_unlock #3}\endcsname }%
2277 \def\XINT_expr_func_min #1#2#3%
2278     {\expandafter #1\expandafter #2\csname
2279         .=\xintMinof:csv{\XINT_expr_unlock #3}\endcsname }%
2280 \def\XINT_iiexpr_func_min #1#2#3%
2281     {\expandafter #1\expandafter #2\csname
2282         .=\xintiiMinof:csv{\XINT_expr_unlock #3}\endcsname }%
2283 \def\XINT_flexpr_func_min #1#2#3%
2284     {\expandafter #1\expandafter #2\csname
2285         .=\XINTinFloatMinof:csv{\XINT_expr_unlock #3}\endcsname }%
2286 \expandafter\def\csname XINT_expr_func_+\endcsname #1#2#3%
2287     {\expandafter #1\expandafter #2\csname
2288         .=\xintSum:csv{\XINT_expr_unlock #3}\endcsname }%
2289 \expandafter\def\csname XINT_flexpr_func_+\endcsname #1#2#3%
2290     {\expandafter #1\expandafter #2\csname
2291         .=\XINTinFloatSum:csv{\XINT_expr_unlock #3}\endcsname }%
2292 \expandafter\def\csname XINT_iiexpr_func_+\endcsname #1#2#3%
2293     {\expandafter #1\expandafter #2\csname
2294         .=\xintiiSum:csv{\XINT_expr_unlock #3}\endcsname }%
2295 \expandafter\def\csname XINT_expr_func_*\endcsname #1#2#3%
2296     {\expandafter #1\expandafter #2\csname
2297         .=\xintPrd:csv{\XINT_expr_unlock #3}\endcsname }%
2298 \expandafter\def\csname XINT_flexpr_func_*\endcsname #1#2#3%
2299     {\expandafter #1\expandafter #2\csname
2300         .=\XINTinFloatPrd:csv{\XINT_expr_unlock #3}\endcsname }%
2301 \expandafter\def\csname XINT_iiexpr_func_*\endcsname #1#2#3%
2302     {\expandafter #1\expandafter #2\csname
2303         .=\xintiiPrd:csv{\XINT_expr_unlock #3}\endcsname }%
2304 \def\XINT_expr_func_? #1#2#3%
2305     {\expandafter #1\expandafter #2\csname
2306         .=\xintiiIsNotZero {\XINT_expr_unlock #3}\endcsname }%
2307 \let\XINT_flexpr_func_? \XINT_expr_func_?
2308 \let\XINT_iiexpr_func_? \XINT_expr_func_?
2309 \def\XINT_expr_func_! #1#2#3%
2310     {\expandafter #1\expandafter #2\csname .=\xintiiIsZero {\XINT_expr_unlock #3}\endcsname }%
2311 \let\XINT_flexpr_func_! \XINT_expr_func_!
2312 \let\XINT_iiexpr_func_! \XINT_expr_func_!
2313 \def\XINT_expr_func_not #1#2#3%
2314     {\expandafter #1\expandafter #2\csname .=\xintiiIsZero {\XINT_expr_unlock #3}\endcsname }%
2315 \let\XINT_flexpr_func_not \XINT_expr_func_not
2316 \let\XINT_iiexpr_func_not \XINT_expr_func_not
2317 \def\XINT_expr_func_all #1#2#3%
2318     {\expandafter #1\expandafter #2\csname
2319         .=\xintANDof:csv{\XINT_expr_unlock #3}\endcsname }%
2320 \let\XINT_flexpr_func_all\XINT_expr_func_all
2321 \let\XINT_iiexpr_func_all\XINT_expr_func_all
2322 \def\XINT_expr_func_any #1#2#3%
2323     {\expandafter #1\expandafter #2\csname
2324         .=\xintORof:csv{\XINT_expr_unlock #3}\endcsname }%
2325 \let\XINT_flexpr_func_any\XINT_expr_func_any

```

```

2326 \let\XINT_iiexpr_func_any\XINT_expr_func_any
2327 \def\XINT_expr_func_xor #1#2#3%
2328     {\expandafter #1\expandafter #2\csname
2329         .=\xintXORof:csv{\XINT_expr_unlock #3}\endcsname }%
2330 \let\XINT_flexpr_func_xor\XINT_expr_func_xor
2331 \let\XINT_iiexpr_func_xor\XINT_expr_func_xor
2332 \def\xintifNotZero: #1,#2,#3,{\xintiifNotZero{#1}{#2}{#3}}%
2333 \def\XINT_expr_func_if #1#2#3%
2334     {\expandafter #1\expandafter #2\csname
2335         .=\expandafter\xintifNotZero:\romannumeral`&&\XINT_expr_unlock #3,\endcsname }%
2336 \let\XINT_flexpr_func_if\XINT_expr_func_if
2337 \let\XINT_iiexpr_func_if\XINT_expr_func_if
2338 \def\xintifSgn: #1,#2,#3,#4,{\xintiifSgn{#1}{#2}{#3}{#4}}%
2339 \def\XINT_expr_func_ifsgn #1#2#3%
2340 {%
2341     \expandafter #1\expandafter #2\csname
2342         .=\expandafter\xintifSgn:\romannumeral`&&\XINT_expr_unlock #3,\endcsname
2343 }%
2344 \let\XINT_flexpr_func_ifsgn\XINT_expr_func_ifsgn
2345 \let\XINT_iiexpr_func_ifsgn\XINT_expr_func_ifsgn
2346 \def\XINT_expr_func_len #1#2#3%
2347     {\expandafter#1\expandafter#2%
2348         \csname.=\xintLength:f:csv {\XINT_expr_unlock#3}\endcsname }%
2349 \let\XINT_flexpr_func_len \XINT_expr_func_len
2350 \let\XINT_iiexpr_func_len \XINT_expr_func_len

```

1.2k has \xintFirstItem:f:csv for improved \xintNewExpr compatibility.

```

2351 \def\XINT_expr_func_first #1#2#3%
2352     {\expandafter #1\expandafter #2\csname.=%
2353         \xintFirstItem:f:csv{\XINT_expr_unlock #3}\endcsname}%
2354 \let\XINT_flexpr_func_first\XINT_expr_func_first
2355 \let\XINT_iiexpr_func_first\XINT_expr_func_first

```

1.2k has \xintLastItem:f:csv for efficiency and improved \xintNewExpr compatibility.

```

2356 \def\XINT_expr_func_last #1#2#3%
2357     {\expandafter #1\expandafter #2\csname.=%
2358         \xintLastItem:f:csv{\XINT_expr_unlock #3}\endcsname}%
2359 \let\XINT_flexpr_func_last\XINT_expr_func_last
2360 \let\XINT_iiexpr_func_last\XINT_expr_func_last
2361 \def\XINT_expr_func_odd #1#2#3%
2362     {\expandafter #1\expandafter #2\csname.=\xintOdd{\XINT_expr_unlock #3}\endcsname}%
2363 \let\XINT_flexpr_func_odd\XINT_expr_func_odd
2364 \def\XINT_iiexpr_func_odd #1#2#3%
2365     {\expandafter #1\expandafter #2\csname.=\xintiiOdd{\XINT_expr_unlock #3}\endcsname}%
2366 \def\XINT_expr_func_even #1#2#3%
2367     {\expandafter #1\expandafter #2\csname.=\xintEven{\XINT_expr_unlock #3}\endcsname}%
2368 \let\XINT_flexpr_func_even\XINT_expr_func_even
2369 \def\XINT_iiexpr_func_even #1#2#3%
2370     {\expandafter #1\expandafter #2\csname.=\xintiiEven{\XINT_expr_unlock #3}\endcsname}%
2371 \def\XINT_expr_func_nuple #1#2#3%
2372     {\expandafter #1\expandafter #2\csname .=\XINT_expr_unlock #3\endcsname }%
2373 \let\XINT_flexpr_func_nuple\XINT_expr_func_nuple

```

2374 \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.

2375 \def\XINT_expr_func_reversed #1#2#3%

2376 {\expandafter #1\expandafter #2\csname .=%

2377 \xintReverse:f:csv {\XINT_expr_unlock #3}\endcsname }%

2378 \let\XINT_flexpr_func_reversed\XINT_expr_func_reversed

2379 \let\XINT_iiexpr_func_reversed\XINT_expr_func_reversed

10.50 f-expandable versions of the \xintSeqB::csv and alike routines, for \xintNewExpr

10.50.1	\xintSeqB:f:csv	357
10.50.2	\xintiiSeqB:f:csv	358
10.50.3	\XINTinFloatSeqB:f:csv	358

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.

2380 \def\xintSeqB:f:csv #1#2%

2381 {\expandafter\XINT_seqb:f:csv \expandafter{\romannumeral0\xintraw{#2}}{#1}}%

2382 \def\XINT_seqb:f:csv #1#2{\expandafter\XINT_seqb:f:csv_a\romannumeral`&&@#2#1!}%

2383 \def\XINT_seqb:f:csv_a #1#2;#3;#4!{%

2384 \expandafter\xint_gobble_i\romannumeral`&&@%

2385 \xintifCmp {#3}{#4}\XINT_seqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_seqb:f:csv_bg

2386 #1{#3}{#4}{#2}}%

2387 \def\XINT_seqb:f:csv_be #1#2#3#4#5{,#2}%

2388 \def\XINT_seqb:f:csv_bl #1{\if #1p\expandafter\XINT_seqb:f:csv_pa\else

2389 \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%

2390 \def\XINT_seqb:f:csv_pa #1#2#3#4{\expandafter\XINT_seqb:f:csv_p\expandafter

2391 {\romannumeral0\xintadd{#4}{#1}}{#2}{#3,#1}{#4}}%

2392 \def\XINT_seqb:f:csv_p #1#2%

2393 {%

2394 \xintifCmp {#1}{#2}\XINT_seqb:f:csv_pa\XINT_seqb:f:csv_pb\XINT_seqb:f:csv_pc

2395 {#1}{#2}}%

2396 }%

2397 \def\XINT_seqb:f:csv_pb #1#2#3#4{#3,#1}%

2398 \def\XINT_seqb:f:csv_pc #1#2#3#4{#3}%

2399 \def\XINT_seqb:f:csv_bg #1{\if #1n\expandafter\XINT_seqb:f:csv_na\else

2400 \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%

2401 \def\XINT_seqb:f:csv_na #1#2#3#4{\expandafter\XINT_seqb:f:csv_n\expandafter

2402 {\romannumeral0\xintadd{#4}{#1}}{#2}{#3,#1}{#4}}%

2403 \def\XINT_seqb:f:csv_n #1#2%

2404 {%

2405 \xintifCmp {#1}{#2}\XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_seqb:f:csv_na

2406 {#1}{#2}}%

2407 }%

2408 \def\XINT_seqb:f:csv_nb #1#2#3#4{#3,#1}%

2409 \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).

```

2410 \def\xintiiSeqB:f:csv #1#2%
2411   {\expandafter\XINT_iiseqb:f:csv \expandafter\romannumeral`&&@#2}{#1}}%
2412 \def\XINT_iiseqb:f:csv #1#2{\expandafter\XINT_iiseqb:f:csv_a\romannumeral`&&@#2#1!}%
2413 \def\XINT_iiseqb:f:csv_a #1#2;#3;#4!{%
2414   \expandafter\xint_gobble_i\romannumeral`&&@%
2415   \xintSgnFork{\XINT_Cmp {#3}{#4}}%
2416   \XINT_iiseqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_iiseqb:f:csv_bg
2417   #1{#3}{#4}{}{#2}}%
2418 \def\XINT_iiseqb:f:csv_bl #1{\if #1p\expandafter\XINT_iiseqb:f:csv_pa\else
2419   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2420 \def\XINT_iiseqb:f:csv_pa #1#2#3#4{\expandafter\XINT_iiseqb:f:csv_p\expandafter
2421   {\romannumeral0\xintiiadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2422 \def\XINT_iiseqb:f:csv_p #1#2%
2423 {%
2424   \xintSgnFork{\XINT_Cmp {#1}{#2}}%
2425   \XINT_iiseqb:f:csv_pa\XINT_iiseqb:f:csv_pb\XINT_iiseqb:f:csv_pc {#1}{#2}%
2426 }%
2427 \def\XINT_iiseqb:f:csv_pb #1#2#3#4{#3,#1}%
2428 \def\XINT_iiseqb:f:csv_pc #1#2#3#4{#3}%
2429 \def\XINT_iiseqb:f:csv_bg #1{\if #1n\expandafter\XINT_iiseqb:f:csv_na\else
2430   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2431 \def\XINT_iiseqb:f:csv_na #1#2#3#4{\expandafter\XINT_iiseqb:f:csv_n\expandafter
2432   {\romannumeral0\xintiiadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2433 \def\XINT_iiseqb:f:csv_n #1#2%
2434 {%
2435   \xintSgnFork{\XINT_Cmp {#1}{#2}}%
2436   \XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_iiseqb:f:csv_na {#1}{#2}%
2437 }%

```

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.

```

2438 \def\XINTinFloatSeqB:f:csv #1#2{\expandafter\XINT_flseqb:f:csv \expandafter
2439   {\romannumeral0\XINTinfloat [\XINTdigits]{#2}{#1}}%
2440 \def\XINT_flseqb:f:csv #1#2{\expandafter\XINT_flseqb:f:csv_a\romannumeral`&&@#2#1!}%
2441 \def\XINT_flseqb:f:csv_a #1#2;#3;#4!{%
2442   \expandafter\xint_gobble_i\romannumeral`&&@%
2443   \xintifCmp {#3}{#4}\XINT_flseqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_flseqb:f:csv_bg
2444   #1{#3}{#4}{}{#2}}%
2445 \def\XINT_flseqb:f:csv_bl #1{\if #1p\expandafter\XINT_flseqb:f:csv_pa\else
2446   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2447 \def\XINT_flseqb:f:csv_pa #1#2#3#4{\expandafter\XINT_flseqb:f:csv_p\expandafter
2448   {\romannumeral0\XINTinfloatadd{#4}{#1}}{#2}{#3,#1}{#4}}%

```

```

2449 \def\XINT_flseqb:f:csv_p #1#2%
2450 {%
2451   \xintifCmp {#1}{#2}%
2452   \XINT_flseqb:f:csv_pa\XINT_flseqb:f:csv_pb\XINT_flseqb:f:csv_pc {#1}{#2}%
2453 }%
2454 \def\XINT_flseqb:f:csv_pb #1#2#3#4{#3,#1}%
2455 \def\XINT_flseqb:f:csv_pc #1#2#3#4{#3}%
2456 \def\XINT_flseqb:f:csv_bg #1{\if #1n\expandafter\XINT_flseqb:f:csv_na\else
2457   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2458 \def\XINT_flseqb:f:csv_na #1#2#3#4{\expandafter\XINT_flseqb:f:csv_n\expandafter
2459   {\romannumeral0\XINTinfloatadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2460 \def\XINT_flseqb:f:csv_n #1#2%
2461 {%
2462   \xintifCmp {#1}{#2}%
2463   \XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_flseqb:f:csv_na {#1}{#2}%
2464 }%

```

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 $\text{add}(f(f), f=1..10)$ 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: 1.2f allows comma separated expressions; until then the user had to use explicit parentheses `\xintdeffunc foo(x,...):=(..., ..., ...)\relax`.

```

2465 \catcode`: 12
2466 \def\XINT_tmpa #1#2#3#4%
2467 {%
2468   \def #1##1(##2)##3==#4;{%
2469     \edef\XINT_expr_tmpa {##1}%
2470     \edef\XINT_expr_tmpa {\xint_zapspace_o \XINT_expr_tmpa}%
2471     \def\XINT_expr_tmppb {0}%
2472     \def\XINT_expr_tmppc {(##4)}%
2473     \xintFor #####1 in {##2} \do
2474       {\edef\XINT_expr_tmppb {\the\numexpr\XINT_expr_tmppb+\xint_c_i}%
2475        \edef\XINT_expr_tmppc {subs(\unexpanded\expandafter{\XINT_expr_tmppc},%
2476          #####1=#####\XINT_expr_tmppb)}}%
2477     }%
2478     \expandafter#3\csname XINT_#2_userfunc_\XINT_expr_tmpa\endcsname
2479       [\XINT_expr_tmppb]{\XINT_expr_tmppc}%
2480     \expandafter\XINT_expr_defuserfunc
2481       \csname XINT_#2_func_\XINT_expr_tmpa\expandafter\endcsname
2482       \csname XINT_#2_userfunc_\XINT_expr_tmpa\endcsname
2483     \ifxintverbose\xintMessage {\xintexpr}{Info}
2484     {Function \XINT_expr_tmpa\space for \string\xint #4 parser

```



```

2485      associated to \string\XINT_#2_userfunc_\XINT_expr_tmpa\space
2486      with meaning \expandafter\meaning
2487      \csname XINT_#2_userfunc_\XINT_expr_tmpa\endcsname}%
2488  \fi
2489 }%
2490 }%
2491 \catcode` : 11
2492 \XINT_tmpa\xintdeffunc      {expr}  \XINT_NewFunc      {expr}%
2493 \XINT_tmpa\xintdefiifunc    {iiexpr}\XINT_NewIIFunc    {iiexpr}%
2494 \XINT_tmpa\xintdeffloatfunc{fexpr}\XINT_NewFloatFunc{floatexpr}%
2495 \def\XINT_expr_defuserfunc #1#2{%
2496   \def #1##1##2##3{\expandafter ##1\expandafter ##2%
2497     \csname .=\expandafter #2\romannumeral`&&\XINT_expr_unlock ##3,\endcsname
2498   }%
2499 }%

```

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`.

```

2500 \def\XINT_expr_wrapit #1{\expandafter\XINT_expr_wrap\csname.=#1\endcsname}%
2501 \def\xintNewFunction #1#2[#3]#4%
2502 {%
2503   \edef\XINT_expr_tmpa {#1}%
2504   \edef\XINT_expr_tmpa {\xint_zapspaces_o \XINT_expr_tmpa}%
2505   \def\XINT_expr_tmppb ##1##2##3##4##5##6##7##8##9{#4}%
2506   \begingroup
2507     \ifcase #3\relax
2508       \toks0{,}%
2509     \or \toks0{##1,}%
2510     \or \toks0{##1,##2,}%
2511     \or \toks0{##1,##2,##3,}%
2512     \or \toks0{##1,##2,##3,##4,}%
2513     \or \toks0{##1,##2,##3,##4,##5,}%
2514     \or \toks0{##1,##2,##3,##4,##5,##6,}%
2515     \or \toks0{##1,##2,##3,##4,##5,##6,##7,}%
2516     \or \toks0{##1,##2,##3,##4,##5,##6,##7,##8,}%
2517     \else \toks0{##1,##2,##3,##4,##5,##6,##7,##8,##9,}%
2518   \fi
2519   \expandafter
2520 \endgroup
2521 \expandafter
2522 \def\csname XINT_expr_macrofunc_\XINT_expr_tmpa\expandafter\endcsname
2523 \the\toks0\expandafter{\XINT_expr_tmppb
2524   {\XINT_expr_wrapit{##1}}{\XINT_expr_wrapit{##2}}{\XINT_expr_wrapit{##3}}%
2525   {\XINT_expr_wrapit{##4}}{\XINT_expr_wrapit{##5}}{\XINT_expr_wrapit{##6}}%
2526   {\XINT_expr_wrapit{##7}}{\XINT_expr_wrapit{##8}}{\XINT_expr_wrapit{##9}}}%
2527 \expandafter\XINT_expr_newfunction
2528   \csname XINT_expr_func_\XINT_expr_tmpa\expandafter\endcsname
2529   \csname XINT_expr_macrofunc_\XINT_expr_tmpa\endcsname\xintbareeval

```



```

2530 \expandafter\XINT_expr_newfunction
2531   \csname XINT_iiexpr_func_\XINT_expr_tmpa\expandafter\endcsname
2532   \csname XINT_expr_macrofunc_\XINT_expr_tmpa\endcsname\xintbareiieval
2533 \expandafter\XINT_expr_newfunction
2534   \csname XINT_flexpr_func_\XINT_expr_tmpa\expandafter\endcsname
2535   \csname XINT_expr_macrofunc_\XINT_expr_tmpa\endcsname\xintbarefloateval
2536 \ifxintverbose
2537   \xintMessage {xintexpr}{Info}
2538     {Function \XINT_expr_tmpa\space for the expression parsers is
2539     associated to \string\XINT_expr_macrofunc_\XINT_expr_tmpa\space
2540     with meaning \expandafter\meaning
2541     \csname XINT_expr_macrofunc_\XINT_expr_tmpa\endcsname}%
2542 \fi
2543 }%
2544 \def\XINT_expr_newfunction #1#2#3%
2545 {%
2546   \def#1##1##2##3{\expandafter ##1\expandafter ##2\romannumeral`&&@%
2547     #3\expandafter#2\romannumeral`&&@\XINT_expr_unlock##3,\relax}%
2548 }%

```

10.53 \xintNewExpr, \xintNewIExpr, \xintNewFloatExpr, \xintNewIIExpr

10.53.1	\xintApply::csv	361
10.53.2	\xintApply:::csv	362
10.53.3	\XINT_expr_RApply::csv, \XINT_expr_LApply::csv, \XINT_expr_RLApply:::csv . .	362
10.53.4	Mysterious stuff	362
10.53.5	\xintNewExpr, ..., at last.	365

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

```

2549 \def\xintApply::csv #1#2%
2550   {\expandafter\XINT_applyon::_a\expandafter {\romannumeral`&&@#2}{#1}}%
2551 \def\XINT_applyon::_a #1#2{\XINT_applyon::_b {#2}{#1},,%}
2552 \def\XINT_applyon::_b #1#2#3,{\expandafter\XINT_applyon::_c \romannumeral`&&@#3,{#1}{#2}}%
2553 \def\XINT_applyon::_c #1{\if #1,\expandafter\XINT_applyon::_end
2554   \else\expandafter\XINT_applyon::_d\fi #1}%
2555 \def\XINT_applyon::_d #1,#2{\expandafter\XINT_applyon::_e\romannumeral`&&@#2{#1},{#2}}%
2556 \def\XINT_applyon::_e #1,#2#3{\XINT_applyon::_b {#2}{#3}, #1}%
2557 \def\XINT_applyon::_end #1,#2#3{\xint_secondoftwo #3}%

```

10.53.2 \xintApply::csv

```

2558 \def\xintApply::csv #1#2#3%
2559   {\expandafter\XINT_applyon::_a\expandafter{\romannumeral`&&@#2}{#1}{#3}}%
2560 \def\XINT_applyon::_a #1#2#3{\XINT_applyon::_b {#2}{#3}{#1},,%}
2561 \def\XINT_applyon::_b #1#2#3#4,%
2562   {\expandafter\XINT_applyon::_c \romannumeral`&&@#4,{#1}{#2}{#3}}%
2563 \def\XINT_applyon::_c #1{\if #1,\expandafter\XINT_applyon::_end
2564   \else\expandafter\XINT_applyon::_d\fi #1}%
2565 \def\XINT_applyon::_d #1,#2#3%
2566   {\expandafter\XINT_applyon::_e\expandafter
2567     {\romannumeral`&&\xintApply::csv {#2}{#1}}{#3}}{#2}{#3}}%
2568 \def\XINT_applyon::_e #1,#2#3#4{\XINT_applyon::_b {#2}{#3}{#4}, #1}%
2569 \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*.

```

2570 \catcode`~ 12
2571 \def\XINT_expr_RApply::csv #1#2#3#4%
2572   {\xintApply::csv{~expandafter#1~xint_exchangetwo_keepbraces{#4}}{#3}}%
2573 \def\XINT_expr_LApply::csv #1#2#3#4{\xintApply::csv{#1}{#3}}{#4}}%
2574 \def\XINT_expr_RLApply::csv #1#2{\xintApply::csv{#1}}%

```

10.53.4 Mysterious stuff

`~` and `$` of catcode 12 in what follows.

```

2575 \catcode`$ 12 % $
2576 \def\XINT_xptwo_getab_b #1#2!#3%
2577   {\expandafter\XINT_xptwo_getab_c\romannumeral`&&@#3!#1{#1#2}}%
2578 \def\XINT_xptwo_getab_c #1#2!#3#4#5#6{#1#3{#5}{#6}{#1#2}{#4}}%
2579 \def\xint_ddfork #1$$$#2#3\krof {#2}% $$
2580 \def\XINT_NEfork #1#2{\xint_ddfork
2581   #1#2\XINT_expr_RLApply::csv
2582   #1$\XINT_expr_RApply::csv% $
2583   $#2\XINT_expr_LApply::csv% $
2584   $${\XINT_NEfork_nn #1#2}% $$
2585   \krof }%
2586 \def\XINT_NEfork_nn #1#2#3#4{%
2587   \if #1#\xint_dothis{#3}\fi
2588   \if #1~\xint_dothis{#3}\fi

```

```

2589     \if #2##\xint_dothis{#3}\fi
2590     \if #2~\xint_dothis{#3}\fi
2591     \xint_orthat {\csname #4NE\endcsname }%
2592     }%
2593 \def\xint_NEfork_one #1#2!#3#4#5#6{%
2594     \if ###1\xint_dothis {#3}\fi
2595     \if ~#1\xint_dothis {#3}\fi
2596     \if $#1\xint_dothis {\xintApply::csv{#3#5}}\fi %$
2597     \xint_orthat {\csname #4NE\endcsname #6}{#1#2}%
2598 }%
2599 \toks0 {}%
2600 \xintFor #1 in
2601     {DivFloor,Mod,Round,Trunc,iRound,iTrunc,iQuo,iRem,
2602     iiDivFloor,iiDivRound,iiMod,iiQuo,iiRem,%
2603     Lt,Gt,Eq,LtorEq,GtorEq,NotEq,%
2604     iiLt,iiGt,iiEq,iiLtorEq,iiGtorEq,iiNotEq,%
2605     Add,Sub,Mul,Div,Pow,E,%
2606     iiAdd,iiSub,iiMul,iiPow,iiE,%
2607     AND,OR,XOR,%
2608     SeqA::csv,iiSeqA::csv}\do
2609 {\toks0
2610  \expandafter{\the\toks0% no space!
2611  \expandafter\let\csname xint#1NE\expandafter\endcsname\csname xint#1\expandafter
2612  \endcsname\expandafter\def\csname xint#1\endcsname #####1####2{%
2613      \expandafter\xint_NEfork
2614      \romannumeral`&&\expandafter\xint_xptwo_getab_b
2615      \romannumeral`&&@####2!{####1}{~xint#1}{xint#1}}}%
2616  }%
2617 }%
2618 % cela aurait-il un sens d'ajouter Raw et iNum (à cause de qint, qfrac,
2619 % qfloat?). Pas le temps d'y réfléchir. Je ne fais rien.
2620 \xintFor #1 in {Num,Irr,Abs,iiAbs,Sgn,iiSgn,TFrac,Floor,iFloor,Ceil,iCeil,%
2621     Sqr,iiSqr,iiSqrt,iiSqrtR,iiIsZero,iiIsNotZero,iiifNotZero,iiifSgn,%
2622     Odd,Even,iiOdd,iiEven,Opp,iiOpp,iiifZero,Fac,Binomial,%
2623     PFactorial,iiFac,iiBinomial,iiPFactorial,Bool,Toggle}\do
2624 {\toks0 \expandafter{\the\toks0%
2625     \expandafter\let\csname xint#1NE\expandafter\endcsname\csname
2626     xint#1\expandafter
2627     \endcsname\expandafter\def\csname xint#1\endcsname #####1{%
2628         \expandafter\xint_NEfork_one\romannumeral`&&@####1!{~xint#1}{xint#1}{}}}%
2629     }%
2630 }%
2631 \toks0
2632 \expandafter{\the\toks0
2633 \let\xintinFloatFacNE\xintinFloatFac
2634 \def\xintinFloatFac ##1{%
2635     \expandafter\xint_NEfork_one
2636     \romannumeral`&&@##1!{~xintinFloatFac}{xintinFloatFac}{}}}%
2637 }%
2638 \xintFor #1 in {Add,Sub,Mul,Div,Binomial,PFactorial,PowerH,E,%
2639     Mod,DivFloor,DivMod,SeqA::csv}\do
2640 {\toks0

```

```

2641 \expandafter{\the\toks0%
2642 \expandafter\let\csname XINTinFloat#1NE\expandafter\endcsname
2643         \csname XINTinFloat#1\expandafter\endcsname
2644 \expandafter\def\csname XINTinFloat#1\endcsname ####1####2{%
2645     \expandafter\XINT_NEfork
2646     \romannumeral`&&@\expandafter\XINT_xptwo_getab_b
2647     \romannumeral`&&@####2!{####1}{~XINTinFloat#1}{XINTinFloat#1}}%
2648 }%
2649 }%
2650 \xintFor #1 in {XINTinFloatdigits,XINTinFloatFracdigits,XINTinFloatSqrtdigits,XINTinFloatFacdigits}\do
2651 {\toks0
2652 \expandafter{\the\toks0%
2653 \expandafter\let\csname #1NE\expandafter\endcsname\csname #1\expandafter
2654 \endcsname\expandafter\def\csname #1\endcsname ####1{\expandafter
2655     \XINT_NEfork_one\romannumeral`&&@####1!{~#1}{#1}{}}}%
2656 }%
2657 }%
2658 \xintFor #1 in {xintSeq::csv,xintiiSeq::csv,XINTinFloatSeq::csv}\do
2659 {\toks0
2660 \expandafter{\the\toks0% no space
2661 \expandafter\let\csname #1NE\expandafter\endcsname\csname #1\expandafter
2662 \endcsname\expandafter\def\csname #1\endcsname ####1####2{%
2663     \expandafter\XINT_NEfork
2664     \romannumeral`&&@\expandafter\XINT_xptwo_getab_b
2665     \romannumeral`&&@####2!{####1}{$noexpand$#1}{#1}}%
2666 }%
2667 }%
2668 \xintFor #1 in {xintSeqB,xintiiSeqB,XINTinFloatSeqB}\do
2669 {\toks0
2670 \expandafter{\the\toks0% no space
2671 \expandafter\let\csname #1::csvNE\expandafter\endcsname\csname #1::csv\expandafter
2672 \endcsname\expandafter\def\csname #1::csv\endcsname ####1####2{%
2673     \expandafter\XINT_NEfork
2674     \romannumeral`&&@\expandafter\XINT_xptwo_getab_b
2675     \romannumeral`&&@####2!{####1}{$noexpand$#1:f:csv}{#1::csv}}%
2676 }%
2677 }%
2678 \toks0
2679 \expandafter{\the\toks0
2680 \let\XINTinFloatNE\XINTinFloat
2681 \def\XINTinFloat [#1]##2{% not ultimately general, but got tired
2682     \expandafter\XINT_NEfork_one
2683     \romannumeral`&&@####2!{~XINTinFloat[#1]}{XINTinFloat}{}{[#1]}}%
2684 \let\XINTinFloatSqrtNE\XINTinFloatSqrt
2685 \def\XINTinFloatSqrt [#1]##2{%
2686     \expandafter\XINT_NEfork_one
2687     \romannumeral`&&@####2!{~XINTinFloatSqrt[#1]}{XINTinFloatSqrt}{}{[#1]}}%
2688 \let\XINTinFloatFacNE\XINTinFloatFac
2689 \def\XINTinFloatFac [#1]##2{%
2690     \expandafter\XINT_NEfork_one
2691     \romannumeral`&&@####2!{~XINTinFloatFac[#1]}{XINTinFloatFac}{}{[#1]}}%
2692 }%

```

`\xintNewExpr` has difficulties with handling lists, hence all these macros are set-up to simply not attempt to do anything immediately, they will not try to check if the list is completely explicit.

```

2693 \xintFor #1 in {ANDof,ORof,XORof,iiMaxof,iiMinof,iiSum,iiPrd,
2694               GCDof,LCMof,Sum,Prd,Maxof,Minof,
2695               Reverse:f,FirstItem:f,LastItem:f}\do
2696 {\toks0
2697  \expandafter{\the\toks0\expandafter\def\csname xint#1:csv\endcsname {\~xint#1:csv}}%
2698 }%
2699 \xintFor #1 in
2700   {XINTinFloatMaxof,XINTinFloatMinof,XINTinFloatSum,XINTinFloatPrd}\do
2701 {\toks0
2702  \expandafter{\the\toks0\expandafter\def\csname #1:csv\endcsname {\~#1:csv}}%
2703 }%

```

`\~xintListSel:f:csv` must have space after it, the reason being in that `\XINT_expr_until:_b` inserts a `:` to signal Python slice type or argument hence this `:` would end up following immediately `\~xintListSel:f:csv` and scantokens will get confused after that. Since 1.2j there is initially `\xintListSel:x:csv`.

```

2704 \toks0 \expandafter{\the\toks0
2705         \def\xintListSel:x:csv {\~xintListSel:f:csv }%
2706 }%
2707 \odef\XINT_expr_redefinemacros {\the\toks0}%
2708 \def\XINT_expr_redefineprints
2709 {%
2710   \def\XINT_flexpr_noopt
2711     {\expandafter\XINT_flexpr_withopt_b\expandafter-\romannumeral0\xintbarefloateval }%
2712   \def\XINT_flexpr_withopt_b ##1##2%
2713     {\expandafter\XINT_flexpr_wrap\csname .;##1.=\XINT_expr_unlock ##2\endcsname }%
2714   \def\XINT_expr_unlock_sp ##1.;##2##3.=##4!%
2715     {\if -##2\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
2716     \XINTdigits{\##2##3}{\##4}}%
2717   \def\XINT_expr_print ##1{\expandafter\xintSPRaw::csv\expandafter
2718     {\romannumeral`&&\XINT_expr_unlock ##1}}%
2719   \def\XINT_iiexpr_print ##1{\expandafter\xintCSV::csv\expandafter
2720     {\romannumeral`&&\XINT_expr_unlock ##1}}%
2721   \def\XINT_boolexpr_print ##1{\expandafter\xintIsTrue::csv\expandafter
2722     {\romannumeral`&&\XINT_expr_unlock ##1}}%
2723   \def\xintCSV::csv {\~xintCSV::csv }%
2724   \def\xintSPRaw::csv {\~xintSPRaw::csv }%
2725   \def\xintPFloat::csv {\~xintPFloat::csv }%
2726   \def\xintIsTrue::csv {\~xintIsTrue::csv }%
2727   \def\xintRound::csv {\~xintRound::csv }%
2728 }%
2729 \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`

```

2730 \def\xintNewExpr {\XINT_NewExpr{\XINT_expr_redefineprints\xint_firstofone

```

```

2731 \xinttheexpr\xINT_newexpr_clean}%
2732 \def\xintNewFloatExpr{\XINT_NewExpr{\XINT_expr_redefineprints\xint_firstofone
2733 \xintthefloatexpr\xINT_newexpr_clean}%
2734 \def\xintNewIExpr {\XINT_NewExpr{\XINT_expr_redefineprints\xint_firstofone
2735 \xinttheiexpr\xINT_newexpr_clean}%
2736 \def\xintNewIIFExpr {\XINT_NewExpr{\XINT_expr_redefineprints\xint_firstofone
2737 \xinttheiexpr\xINT_newexpr_clean}%
2738 \def\xintNewBoolExpr {\XINT_NewExpr{\XINT_expr_redefineprints\xint_firstofone
2739 \xinttheboolexpr\xINT_newexpr_clean}%
2740 \def\xINT_newexpr_clean #1>{\noexpand\romannumeral`&&@}%

```

1.2c for `\xintdeffunc`, `\xintdefiifunc`, `\xintdeffloatfunc`.

```

2741 \def\xINT_NewFunc
2742 {\XINT_NewExpr,{\xint_gobble_i\xintthebareeval \XINT_newfunc_clean}%
2743 \def\xINT_NewFloatFunc
2744 {\XINT_NewExpr,{\xint_gobble_i\xintthebarefloateval\xINT_newfunc_clean}%
2745 \def\xINT_NewIIFunc
2746 {\XINT_NewExpr,{\xint_gobble_i\xintthebareiieval \XINT_newfunc_clean}%
2747 \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.

```

2748 \def\xINT_NewExpr #1#2#3#4#5#6#7[#8]%
2749 {%
2750 \begingroup
2751 \ifcase #8\relax
2752 \toks0 {\endgroup\def#6}%
2753 \or \toks0 {\endgroup\def#6##1#1}%
2754 \or \toks0 {\endgroup\def#6##1#1##2#1}%
2755 \or \toks0 {\endgroup\def#6##1#1##2#1##3#1}%
2756 \or \toks0 {\endgroup\def#6##1#1##2#1##3#1##4#1}%
2757 \or \toks0 {\endgroup\def#6##1#1##2#1##3#1##4#1##5#1}%
2758 \or \toks0 {\endgroup\def#6##1#1##2#1##3#1##4#1##5#1##6#1}%
2759 \or \toks0 {\endgroup\def#6##1#1##2#1##3#1##4#1##5#1##6#1##7#1}%
2760 \or \toks0 {\endgroup\def#6##1#1##2#1##3#1##4#1##5#1##6#1##7#1##8#1}%
2761 \or \toks0 {\endgroup\def#6##1#1##2#1##3#1##4#1##5#1##6#1##7#1##8#1##9#1}%
2762 \fi
2763 \xintexprSafeCatcodes
2764 \XINT_expr_redefinemacros
2765 #2%
2766 \XINT_NewExpr_a #3#4#5#6%
2767 }%

```

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.

```

2768 \catcode`~ 13 \catcode`@ 14 \catcode`\% 6 \catcode`# 12 \catcode`$ 11 @ $

```

```

2769 \def\XINT_NewExpr_a %1%2%3%4%5@
2770 {@
2771     \def\XINT_tmpa %1%2%3%4%5%6%7%8%9{%5}@
2772     \def~{\$noexpand$}@
2773     \catcode`: 11 \catcode`_ 11
2774     \catcode`# 12 \catcode`~ 13 \escapechar 126
2775     \endlinechar -1 \everyeof {\noexpand }@
2776     \edef\XINT_tmpb
2777     {\scantokens\expandafter
2778     {\romannumeral`&&\expandafter%2\XINT_tmpa {#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}\relax}@
2779     }@
2780     \escapechar 92 \catcode`# 6 \catcode`$ 0 @ $
2781     \edef\XINT_tmpa %1%2%3%4%5%6%7%8%9@
2782     {\scantokens\expandafter{\expandafter%3\meaning\XINT_tmpb}}@
2783     \the\toks0\expandafter{\XINT_tmpa{%1}{%2}{%3}{%4}{%5}{%6}{%7}{%8}{%9}}@
2784     %1{\ifxintverbose
2785         \xintMessage{xintexpr}{Info}@
2786         {\string%4\space now with meaning \meaning%4}@
2787     \fi}@
2788 }@
2789 \catcode`% 14
2790 \let\xintexprRestoreCatcodes\empty
2791 \def\xintexprSafeCatcodes
2792 {%
2793     \edef\xintexprRestoreCatcodes {%
2794         \catcode59=\the\catcode59 % ;
2795         \catcode34=\the\catcode34 % "
2796         \catcode63=\the\catcode63 % ?
2797         \catcode124=\the\catcode124 % |
2798         \catcode38=\the\catcode38 % &
2799         \catcode33=\the\catcode33 % !
2800         \catcode93=\the\catcode93 % ]
2801         \catcode91=\the\catcode91 % [
2802         \catcode94=\the\catcode94 % ^
2803         \catcode95=\the\catcode95 % _
2804         \catcode47=\the\catcode47 % /
2805         \catcode41=\the\catcode41 % )
2806         \catcode40=\the\catcode40 % (
2807         \catcode42=\the\catcode42 % *
2808         \catcode43=\the\catcode43 % +
2809         \catcode62=\the\catcode62 % >
2810         \catcode60=\the\catcode60 % <
2811         \catcode58=\the\catcode58 % :
2812         \catcode46=\the\catcode46 % .
2813         \catcode45=\the\catcode45 % -
2814         \catcode44=\the\catcode44 % ,
2815         \catcode61=\the\catcode61 % =
2816         \catcode32=\the\catcode32\relax % space
2817     }%
2818     \catcode59=12 % ;
2819     \catcode34=12 % "
2820     \catcode63=12 % ?

```

```

2821      \catcode124=12 % |
2822      \catcode38=4  % &
2823      \catcode33=12 % !
2824      \catcode93=12 % ]
2825      \catcode91=12 % [
2826      \catcode94=7  % ^
2827      \catcode95=8  % _
2828      \catcode47=12 % /
2829      \catcode41=12 % )
2830      \catcode40=12 % (
2831      \catcode42=12 % *
2832      \catcode43=12 % +
2833      \catcode62=12 % >
2834      \catcode60=12 % <
2835      \catcode58=12 % :
2836      \catcode46=12 % .
2837      \catcode45=12 % -
2838      \catcode44=12 % ,
2839      \catcode61=12 % =
2840      \catcode32=10 % space
2841 }%
2842 \let\XINT_tmpa\relax \let\XINT_tmpb\relax \let\XINT_tmppc\relax
2843 \XINT_restorecatcodes_endinput%

xintkernel: 496. Total number of code lines: 14108. (but 3169 lines among them
xinttools:1447. start either with {% or with }%.)
xintcore:2281. Each package starts with circa 50 lines dealing with cat-
xint:1577. codes, package identification and reloading management,
xintbinhex: 472. also for Plain  $\TeX$ . Version 1.2q of 2018/02/06.
xintgcd: 434.
xintfrac:3143.
xintseries: 386.
xintcfraction:1029.
xintexpr:2843.

```