

# The `xint` source code

JEAN-FRANÇOIS BURNOL

jfbu (at) free (dot) fr

Package version: 1.3e (2019/04/05); documentation date: 2019/04/05.

From source file `xint.dtx`. Time-stamp: <06-04-2019 at 18:55:58 CEST>.

## Contents

<b>1</b>	<b>Introduction to the implementation (recent changes)</b>	<b>2</b>
<b>2</b>	<b>Package <code>xintkernel</code> implementation</b>	<b>4</b>
<b>3</b>	<b>Package <code>xinttools</code> implementation</b>	<b>18</b>
<b>4</b>	<b>Package <code>xintcore</code> implementation</b>	<b>54</b>
<b>5</b>	<b>Package <code>xint</code> implementation</b>	<b>112</b>
<b>6</b>	<b>Package <code>xintbinhex</code> implementation</b>	<b>154</b>
<b>7</b>	<b>Package <code>xintgcd</code> implementation</b>	<b>166</b>
<b>8</b>	<b>Package <code>xintfrac</code> implementation</b>	<b>179</b>
<b>9</b>	<b>Package <code>xintseries</code> implementation</b>	<b>262</b>
<b>10</b>	<b>Package <code>xintcfrac</code> implementation</b>	<b>271</b>
<b>11</b>	<b>Package <code>xintexpr</code> implementation</b>	<b>294</b>
<b>12</b>	<b>Package <code>xinttrig</code> implementation</b>	<b>390</b>
<b>13</b>	<b>Package <code>xintlog</code> implementation</b>	<b>406</b>
<b>14</b>	<b>Cumulative line count</b>	<b>409</b>

## 1 Introduction to the implementation (recent changes)

This is 1.3e of 2019/04/05.

Please refer to [CHANGES.pdf](#) or [CHANGES.html](#).

Internet: <http://mirrors.ctan.org/macros/generic/xint/CHANGES.html>

We keep here only a brief timeline of the most important changes.

At 1.3e the indices which were added at 1.3c got removed: their inclusion caused extra time in the build of [sourcexint.pdf](#), larger file size, and the macros created using `\csname...\endcsname` were not indexed, of course the indexing of functions would have needed systematic extra mark-up. Besides their functionality is advantageously made available via the search function in PDF viewers. Already the local tables of contents are useful enough most of the time when one searches something.

- Release 1.3e of 2019/04/01: [xinttrig](#), [xintlog](#), `\xintdefefunc` ``non-protected'' variant of `\xintdeffunc`. Indices removed from [sourcexint.pdf](#).
- Release 1.3d of 2019/01/06: bugfix of 1.2p bug for division with a zero dividend and a one-digit divisor, `\xinteval` et al. wrappers, `gcd()` and `lcm()` work with fractions.
- Release 1.3c of 2018/06/17: documentation better hyperlinked, [sourcexint.pdf](#) with indices of macros. Colon in `:=` now optional for `\xintdefvar` and `\xintdeffunc`.
- Release 1.3b of 2018/05/18: randomness related additions (still WIP).
- Release 1.3a of 2018/03/07: efficiency fix of the mechanism for recursive functions.
- Release 1.3 of 2018/03/01: addition and subtraction use systematically least common multiple of denominators. Extensive under-the-hood refactoring of `\xintNewExpr` and `\xintdeffunc` which now allow recursive definitions. Removal of 1.2o deprecated macros.
- Release 1.2q of 2018/02/06: bugfix release (1.2l subtraction bug in special situation); tacit multiplication extended to cases such as  $10!20!30!$ .
- Release 1.2p of 2017/12/05: maps `//` and `/:` to the floored, not truncated, division. Simultaneous assignments possible with `\xintdefvar`. Efficiency improvements in [xinttools](#).
- Release 1.2o of 2017/08/29: massive deprecations of those macros from [xintcore](#) and [xint](#) which filtered their arguments via `\xintNum`.
- Release 1.2n of 2017/08/06: improvements of [xintbinhex](#).
- Release 1.2m of 2017/07/31: rewrite of [xintbinhex](#) in the style of the 1.2 techniques.
- Release 1.2l of 2017/07/26: under the hood efficiency improvements in the style of the 1.2 techniques; subtraction refactored. Compatibility of most [xintfrac](#) macros with arguments using non-delimited `\the\numexpr` or `\the\mathcode` etc...
- Release 1.2i of 2016/12/13: under the hood efficiency improvements in the style of the 1.2 techniques.
- Release 1.2 of 2015/10/10: complete refactoring of the core arithmetic macros and faster `\xintexpr` parser.
- Release 1.1 of 2014/10/28: extensive changes in [xintexpr](#). Addition and subtraction do not multiply denominators blindly but sometimes produce smaller ones. Also with that release, packages [xintkernel](#) and [xintcore](#) got extracted from [xinttools](#) and [xint](#).

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrac](#), [xintexpr](#), [xinttrig](#), [xintlog](#)

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 `\XINT\_setcatcodes`. Additional temporary configuration is used at some locations. For example `!` is of catcode letter in `xintexpr` and there are locations with funny catcodes e.g. using some letters with the math shift catcode.

## 2 Package [xintkernel](#) implementation

.1	Catcodes, $\varepsilon$ -TeX and reload detection . . .	4	.10	<code>\xint_zapspace</code> . . . . .	10
.1.1	<code>\XINT_setcatcodes</code> , <code>\XINT_storecatcodes</code> , <code>\XINT_restorecatcodes_endinput</code> . . . . .	5	.11	<code>\odef</code> , <code>\oodef</code> , <code>\fdef</code> . . . . .	10
.2	Package identification . . . . .	7	.12	<code>\xintReverseOrder</code> . . . . .	10
.3	Constants . . . . .	7	.13	<code>\xintLength</code> . . . . .	11
.4	(WIP) <code>\xint_texuniformdeviate</code> and needed counts . . . . .	7	.14	<code>\xintLastItem</code> . . . . .	11
.5	Token management utilities . . . . .	8	.15	<code>\xintLengthUpTo</code> . . . . .	12
.6	"gob til" macros and UD style fork . . . . .	8	.16	<code>\xintreplicate</code> . . . . .	13
.7	<code>\xint_afterfi</code> . . . . .	9	.17	<code>\xintgobble</code> . . . . .	14
.8	<code>\xint_bye</code> , <code>\xint_Bye</code> . . . . .	9	.18	(WIP) <code>\xintUniformDeviat</code> . . . . .	16
.9	<code>\xintdothis</code> , <code>\xintorthat</code> . . . . .	9	.19	<code>\xintMessage</code> , <code>\ifxintverbose</code> . . . . .	17
			.20	<code>\ifxintglobaldefs</code> , <code>\XINT_global</code> . . . . .	17
			.21	(WIP) Expandable error message . . . . .	17

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.

1.1. separated package.

1.2i. `\xintreplicate`, `\xintgobble`, `\xintLengthUpTo` and `\xintLastItem`, and faster `\xintLength`.

1.3b. `\xintUniformDeviat`.

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

The code for reload detection was initially copied from  $\text{\textcircled{H}}\text{\textcircled{E}}\text{\textcircled{I}}\text{\textcircled{K}}\text{\textcircled{O}}$ '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%
```

### 2.1.1 \XINT\_setcatcodes, \XINT\_storecatcodes, \XINT\_restorecatcodes\_endinput

```

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

```

75 }%
76 \def\XINT_setcatcodes
77 {%
78   \catcode61=12 % =
79   \catcode32=10 % space
80   \catcode13=5 % ^^M
81   \endlinechar=13 %
82   \catcode123=1 % {
83   \catcode125=2 % }
84   \catcode95=11 % _ LETTER
85   \catcode35=6 % #
86   \catcode44=12 % ,
87   \catcode45=12 % -
88   \catcode46=12 % .
89   \catcode58=11 % : LETTER
90   \catcode60=12 % <
91   \catcode62=12 % >
92   \catcode43=12 % +
93   \catcode42=12 % *
94   \catcode40=12 % (
95   \catcode41=12 % )
96   \catcode47=12 % /
97   \catcode96=12 % `
98   \catcode94=11 % ^ LETTER
99   \catcode36=3 % $
100  \catcode91=12 % [
101  \catcode93=12 % ]
102  \catcode33=12 % ! (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 }%

```

## 2.2 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 [2019/04/05 1.3e Paraphernalia for the xint packages (JFB)]%
```

## 2.3 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
```

## 2.4 (WIP) `\xint_texuniformdeviate` and needed counts

```

155 \ifdefined\pdfuniformdeviate \let\xint_texuniformdeviate\pdfuniformdeviate\fi
156 \ifdefined\uniformdeviate \let\xint_texuniformdeviate\uniformdeviate \fi
157 \ifx\xint_texuniformdeviate\relax\let\xint_texuniformdeviate\xint_undefined\fi
158 \ifdefined\xint_texuniformdeviate
159   \csname newcount\endcsname\xint_c_ii^xiv
160   \xint_c_ii^xiv 16384 % "4000, 2**14
161   \csname newcount\endcsname\xint_c_ii^xxi
162   \xint_c_ii^xxi 2097152 % "200000, 2**21
163 \fi
```

## 2.5 Token management utilities

**1.3b.** `\xint_gobandstop_...` macros because this is handy for `\xintRandomDigits`.

```

164 \def\xINT_tmpa { }%
165 \ifx\xINT_tmpa\space\else
166   \immediate\write-1{Package xintkernel Warning: ATTENTION!}%
167   \immediate\write-1{\string\space\xINT_tmpa macro does not have its normal
168     meaning.}%
169   \immediate\write-1{\XINT_tmpa\xINT_tmpa\xINT_tmpa\xINT_tmpa
170     All kinds of catastrophes will ensue!!!!}%
171 \fi
172 \def\xINT_tmpb {}%
173 \ifx\xINT_tmpb\empty\else
174   \immediate\write-1{Package xintkernel Warning: ATTENTION!}%
175   \immediate\write-1{\string\empty\xINT_tmpa macro does not have its normal
176     meaning.}%
177   \immediate\write-1{\XINT_tmpa\xINT_tmpa\xINT_tmpa\xINT_tmpa
178     All kinds of catastrophes will ensue!!!!}%
179 \fi
180 \let\xINT_tmpa\relax \let\xINT_tmpb\relax
181 \ifdefined\space\else\def\space { }\fi
182 \ifdefined\empty\else\def\empty {} \fi
183 \let\xint_gobble_\empty
184 \long\def\xint_gobble_i   #1{%
185 \long\def\xint_gobble_ii  #1#2{%
186 \long\def\xint_gobble_iii #1#2#3{%
187 \long\def\xint_gobble_iv  #1#2#3#4{%
188 \long\def\xint_gobble_v   #1#2#3#4#5{%
189 \long\def\xint_gobble_vi   #1#2#3#4#5#6{%
190 \long\def\xint_gobble_vii  #1#2#3#4#5#6#7{%
191 \long\def\xint_gobble_viii #1#2#3#4#5#6#7#8{%
192 \let\xint_gob_andstop_\space
193 \long\def\xint_gob_andstop_i   #1{ }%
194 \long\def\xint_gob_andstop_ii  #1#2{ }%
195 \long\def\xint_gob_andstop_iii #1#2#3{ }%
196 \long\def\xint_gob_andstop_iv  #1#2#3#4{ }%
197 \long\def\xint_gob_andstop_v   #1#2#3#4#5{ }%
198 \long\def\xint_gob_andstop_vi   #1#2#3#4#5#6{ }%
199 \long\def\xint_gob_andstop_vii #1#2#3#4#5#6#7{ }%
200 \long\def\xint_gob_andstop_viii #1#2#3#4#5#6#7#8{ }%
201 \long\def\xint_firstofone  #1{#1}%
202 \long\def\xint_firstoftwo  #1#2{#1}%
203 \long\def\xint_secondoftwo #1#2{#2}%
204 \let\xint_stop_aftergobble\xint_gob_andstop_i
205 \long\def\xint_stop_atfirstofone  #1{ #1}%
206 \long\def\xint_stop_atfirstoftwo  #1#2{ #1}%
207 \long\def\xint_stop_atsecondoftwo #1#2{ #2}%
208 \long\def\xint_exchangetwo_keepbraces  #1#2{#{2}{#1}}%

```

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

```

209 \long\def\xint_gob_til_R #1\R {}%
210 \long\def\xint_gob_til_W #1\W {}%

```



```

211 \long\def\xint_gob_til_Z #1\Z {}%
212 \long\def\xint_gob_til_zero #10{}%
213 \long\def\xint_gob_til_one #11{}%
214 \long\def\xint_gob_til_zeros_iii #1000{}%
215 \long\def\xint_gob_til_zeros_iv #10000{}%
216 \long\def\xint_gob_til_eightzeroes #100000000{}%
217 \long\def\xint_gob_til_dot #1.{}%
218 \long\def\xint_gob_til_G #1G{}%
219 \long\def\xint_gob_til_minus #1-{}%
220 \long\def\xint_UDzerominusfork #10-#2#3\krof {#2}%
221 \long\def\xint_UDzerofork #10#2#3\krof {#2}%
222 \long\def\xint_UDsignfork #1-#2#3\krof {#2}%
223 \long\def\xint_UDwfork #1\W#2#3\krof {#2}%
224 \long\def\xint_UDXINTWfork #1\XINT_W#2#3\krof {#2}%
225 \long\def\xint_UDzerosfork #100#2#3\krof {#2}%
226 \long\def\xint_UDonezerofork #110#2#3\krof {#2}%
227 \long\def\xint_UDsignsfork #1--#2#3\krof {#2}%
228 \let\xint:\char
229 \long\def\xint_gob_til_xint:#1\xint:{}%
230 \def\xint_bracedstopper{\xint:}%
231 \long\def\xint_gob_til_exclam #1!{}%
232 \long\def\xint_gob_til_sc #1;{}%

```

## 2.7 \xint\_afterfi

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

## 2.8 \xint\_bye, \xint\_Bye

### 1.09. \xint\_bye

1.2i. [\xint\\_Bye](#) for [\xintDSRr](#) and [\xintRound](#). Also [\xint\\_stop\\_afterbye](#).

```

234 \long\def\xint_bye #1\xint_bye {}%
235 \long\def\xint_Bye #1\xint_bye {}%
236 \long\def\xint_stop_afterbye #1\xint_bye { }%

```

## 2.9 \xintdothis, \xintorthat

### 1.1.

### 1.2. names without underscores.

To be used this way:

```

\xif..\xint_dothis{..}\fi
\xif..\xint_dothis{..}\fi
\xif..\xint_dothis{..}\fi
...more such...
\xint_orthat{...}

```

Ancient testing indicated it is more efficient to list first the more improbable clauses.

```

237 \long\def\xint_dothis #1#2\xint_orthat #3{\fi #1}% 1.1
238 \let\xint_orthat \xint_firstofone
239 \long\def\xintdothis #1#2\xintorthat #3{\fi #1}%
240 \let\xintorthat \xint_firstofone

```

## 2.10 `\xint_zapspaces`

### 1.1.

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

Usage: `\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`, not to `12\relax`.

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

**1.2i.** made `\long`.

```
241 \long\def\xint_zapspaces #1 #2{#1#2\xint_zapspaces }% 1.1
242 \long\def\xint_zapspaces_o #1{\expandafter\xint_zapspaces#1 \xint_gobble_i}%
```

## 2.11 `\odef`, `\oodef`, `\fdef`

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

```
243 \def\xintodef #1{\expandafter\def\expandafter#1\expandafter }%
244 \def\xintoodef #1{\expandafter\expandafter\expandafter\def
245     \expandafter\expandafter\expandafter#1%
246     \expandafter\expandafter\expandafter }%
247 \def\xintfdef #1#2%
248     {\expandafter\def\expandafter#1\expandafter{\romannumeral`&&#2}}%
249 \ifdefined\odef\else\let\odef\xintodef\fi
250 \ifdefined\oodef\else\let\oodef\xintoodef\fi
251 \ifdefined\fdef\else\let\fdef\xintfdef\fi
```

## 2.12 `\xintReverseOrder`

**1.0.** does not expand its argument. The whole of xint codebase now contains only two calls to `\XINT_rord_main` (in `xintgcd`).

Attention: removes brace pairs (and swallows spaces).

For digit tokens a faster reverse macro is provided by (1.2) `\xintReverseDigits` in `xint`.

For comma separated items, 1.2g has `\xintCSVReverse` in `xinttools`.

```
252 \def\xintReverseOrder {\romannumeral0\xintreverseorder }%
253 \long\def\xintreverseorder #1%
254 {%
255     \XINT_rord_main }#1%
256     \xint:
257     \xint_bye\xint_bye\xint_bye\xint_bye
258     \xint_bye\xint_bye\xint_bye\xint_bye
259     \xint:
```

```

260 }%
261 \long\def\XINT_roid_main #1#2#3#4#5#6#7#8#9%
262 {%
263     \xint_bye #9\XINT_roid_cleanup\xint_bye
264     \XINT_roid_main {#9#8#7#6#5#4#3#2#1}%
265 }%
266 \def\XINT_roid_cleanup #1{%
267 \long\def\XINT_roid_cleanup\xint_bye\XINT_roid_main ##1##2\xint:
268 {%
269     \expandafter#1\xint_gob_til_xint: ##1%
270 }}\XINT_roid_cleanup { }%

```

## 2.13 \xintLength

**1.0.** does not expand its argument. See `\xintNthElt{0}` from [xinttools](#) which f-expands its argument.

**1.2g.** added `\xintCSVLength` to [xinttools](#).

**1.2i.** rewrote this venerable macro. New code about 40% faster across all lengths.

```

271 \def\xintLength {\romannumeral0\xintlength }%
272 \def\xintlength #1{\long\def\xintlength ##1%
273 {%
274     \expandafter#1\the\numexpr\XINT_length_loop
275     ##1\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
276     \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
277     \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
278     \relax
279 }}\xintlength{ }%
280 \long\def\XINT_length_loop #1#2#3#4#5#6#7#8#9%
281 {%
282     \xint_gob_til_xint: #9\XINT_length_finish_a\xint:
283     \xint_c_ix+\XINT_length_loop
284 }%
285 \def\XINT_length_finish_a\xint:\xint_c_ix+\XINT_length_loop
286     #1#2#3#4#5#6#7#8#9%
287 {%
288     #9\xint_bye
289 }%

```

## 2.14 \xintLastItem

**1.2i (2016/12/10).** Output empty if input empty. One level of braces removed in output. Does not expand its argument.

```

290 \def\xintLastItem {\romannumeral0\xintlastitem }%
291 \long\def\xintlastitem #1%
292 {%
293     \XINT_last_loop {}.#1%
294     {\xint:\XINT_last_loop_enda}{\xint:\XINT_last_loop_endb}%
295     {\xint:\XINT_last_loop_endc}{\xint:\XINT_last_loop_endd}%
296     {\xint:\XINT_last_loop_ende}{\xint:\XINT_last_loop_endf}%
297     {\xint:\XINT_last_loop_endg}{\xint:\XINT_last_loop_endh}\xint_bye
298 }%
299 \long\def\XINT_last_loop #1.#2#3#4#5#6#7#8#9%

```

```

300 {%
301   \xint_gob_til_xint: #9%
302     {#8}{#7}{#6}{#5}{#4}{#3}{#2}{#1}\xint:
303   \XINT_last_loop {#9}.%
304 }%
305 \long\def\XINT_last_loop_enda #1#2\xint_bye{ #1}%
306 \long\def\XINT_last_loop_endb #1#2#3\xint_bye{ #2}%
307 \long\def\XINT_last_loop_endc #1#2#3#4\xint_bye{ #3}%
308 \long\def\XINT_last_loop_endd #1#2#3#4#5\xint_bye{ #4}%
309 \long\def\XINT_last_loop_ende #1#2#3#4#5#6\xint_bye{ #5}%
310 \long\def\XINT_last_loop_endf #1#2#3#4#5#6#7\xint_bye{ #6}%
311 \long\def\XINT_last_loop_endg #1#2#3#4#5#6#7#8\xint_bye{ #7}%
312 \long\def\XINT_last_loop_endh #1#2#3#4#5#6#7#8#9\xint_bye{ #8}%

```

## 2.15 \xintLengthUpTo

1.2i. for use by [\xintKeep](#) and [\xintTrim](#) ([xinttools](#)). The argument N **must be non-negative**.

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

1.2j. changed ending and interface to core loop.

```

313 \def\xintLengthUpTo {\romannumeral0\xintlengthupto}%
314 \long\def\xintlengthupto #1#2%
315 {%
316   \expandafter\XINT_lengthupto_loop
317   \the\numexpr#1.#2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
318     \xint_c_vii\xint_c_vi\xint_c_v\xint_c_iv
319     \xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye.%
320 }%
321 \def\XINT_lengthupto_loop_a #1%
322 {%
323   \xint_UDsignfork
324     #1\XINT_lengthupto_gt
325     -\XINT_lengthupto_loop
326   \krof #1%
327 }%
328 \long\def\XINT_lengthupto_gt #1\xint_bye.{-0}%
329 \long\def\XINT_lengthupto_loop #1.#2#3#4#5#6#7#8#9%
330 {%
331   \xint_gob_til_xint: #9\XINT_lengthupto_finish_a\xint:%
332   \expandafter\XINT_lengthupto_loop_a\the\numexpr #1-\xint_c_viii.%
333 }%
334 \def\XINT_lengthupto_finish_a\xint:\expandafter\XINT_lengthupto_loop_a
335   \the\numexpr #1-\xint_c_viii.#2#3#4#5#6#7#8#9%
336 {%
337   \expandafter\XINT_lengthupto_finish_b\the\numexpr #1-#9\xint_bye
338 }%
339 \def\XINT_lengthupto_finish_b #1#2.%
340 {%
341   \xint_UDsignfork
342     #1{-0}%
343     -{ #1#2}%
344   \krof

```



```

383     {\xint_c_ #1#1#1}%
384 \long\expandafter\def\csname XINT_rep_f4\endcsname #1%
385     {\xint_c_ #1#1#1#1}%
386 \long\expandafter\def\csname XINT_rep_f5\endcsname #1%
387     {\xint_c_ #1#1#1#1#1}%
388 \long\expandafter\def\csname XINT_rep_f6\endcsname #1%
389     {\xint_c_ #1#1#1#1#1#1}%
390 \long\expandafter\def\csname XINT_rep_f7\endcsname #1%
391     {\xint_c_ #1#1#1#1#1#1#1}%
392 \long\expandafter\def\csname XINT_rep_f8\endcsname #1%
393     {\xint_c_ #1#1#1#1#1#1#1#1}%
394 \long\expandafter\def\csname XINT_rep_f9\endcsname #1%
395     {\xint_c_ #1#1#1#1#1#1#1#1#1}%

```

## 2.17 \xintgobble

### 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.

Usage: `\romannumeral\xintgobble{N}...`

```

396 \def\xintgobble #1%
397     {\csname xint_c_\expandafter\XINT_gobble_a\the\numexpr#1.0}%
398 \def\XINT_gobble #1.{\csname xint_c_\XINT_gobble_a #1.0}%
399 \def\XINT_gobble_a #1{\xint_gob_til_zero#1\XINT_gobble_d0\XINT_gobble_b#1}%
400 \def\XINT_gobble_b #1.#2%
401     {\expandafter\XINT_gobble_c
402         \the\numexpr (#1+\xint_c_v)/\xint_c_ix-\xint_c_i\expandafter.%
403         \the\numexpr #2+\xint_c_i.#1.}%
404 \def\XINT_gobble_c #1.#2.#3.%
405     {\csname XINT_g2\the\numexpr#3-\xint_c_ix*#1\relax\XINT_gobble_a #1.#2}%
406 \def\XINT_gobble_d0\XINT_gobble_b0.#1{\endcsname}%
407 \expandafter\let\csname XINT_g10\endcsname\endcsname
408 \long\expandafter\def\csname XINT_g11\endcsname#1{\endcsname}%
409 \long\expandafter\def\csname XINT_g12\endcsname#1#2{\endcsname}%
410 \long\expandafter\def\csname XINT_g13\endcsname#1#2#3{\endcsname}%
411 \long\expandafter\def\csname XINT_g14\endcsname#1#2#3#4{\endcsname}%
412 \long\expandafter\def\csname XINT_g15\endcsname#1#2#3#4#5{\endcsname}%
413 \long\expandafter\def\csname XINT_g16\endcsname#1#2#3#4#5#6{\endcsname}%
414 \long\expandafter\def\csname XINT_g17\endcsname#1#2#3#4#5#6#7{\endcsname}%
415 \long\expandafter\def\csname XINT_g18\endcsname#1#2#3#4#5#6#7#8{\endcsname}%
416 \expandafter\let\csname XINT_g20\endcsname\endcsname
417 \long\expandafter\def\csname XINT_g21\endcsname #1#2#3#4#5#6#7#8#9%
418     {\endcsname}%
419 \long\expandafter\edef\csname XINT_g22\endcsname #1#2#3#4#5#6#7#8#9%
420     {\expandafter\noexpand\csname XINT_g21\endcsname}%
421 \long\expandafter\edef\csname XINT_g23\endcsname #1#2#3#4#5#6#7#8#9%
422     {\expandafter\noexpand\csname XINT_g22\endcsname}%
423 \long\expandafter\edef\csname XINT_g24\endcsname #1#2#3#4#5#6#7#8#9%
424     {\expandafter\noexpand\csname XINT_g23\endcsname}%
425 \long\expandafter\edef\csname XINT_g25\endcsname #1#2#3#4#5#6#7#8#9%
426     {\expandafter\noexpand\csname XINT_g24\endcsname}%
427 \long\expandafter\edef\csname XINT_g26\endcsname #1#2#3#4#5#6#7#8#9%

```

```

428 {\expandafter\noexpand\csname XINT_g25\endcsname}%
429 \long\expandafter\edef\csname XINT_g27\endcsname #1#2#3#4#5#6#7#8#9%
430 {\expandafter\noexpand\csname XINT_g26\endcsname}%
431 \long\expandafter\edef\csname XINT_g28\endcsname #1#2#3#4#5#6#7#8#9%
432 {\expandafter\noexpand\csname XINT_g27\endcsname}%
433 \expandafter\let\csname XINT_g30\endcsname\endcsname
434 \long\expandafter\edef\csname XINT_g31\endcsname #1#2#3#4#5#6#7#8#9%
435 {\expandafter\noexpand\csname XINT_g28\endcsname}%
436 \long\expandafter\edef\csname XINT_g32\endcsname #1#2#3#4#5#6#7#8#9%
437 {\noexpand\csname XINT_g31\expandafter\noexpand\csname XINT_g28\endcsname}%
438 \long\expandafter\edef\csname XINT_g33\endcsname #1#2#3#4#5#6#7#8#9%
439 {\noexpand\csname XINT_g32\expandafter\noexpand\csname XINT_g28\endcsname}%
440 \long\expandafter\edef\csname XINT_g34\endcsname #1#2#3#4#5#6#7#8#9%
441 {\noexpand\csname XINT_g33\expandafter\noexpand\csname XINT_g28\endcsname}%
442 \long\expandafter\edef\csname XINT_g35\endcsname #1#2#3#4#5#6#7#8#9%
443 {\noexpand\csname XINT_g34\expandafter\noexpand\csname XINT_g28\endcsname}%
444 \long\expandafter\edef\csname XINT_g36\endcsname #1#2#3#4#5#6#7#8#9%
445 {\noexpand\csname XINT_g35\expandafter\noexpand\csname XINT_g28\endcsname}%
446 \long\expandafter\edef\csname XINT_g37\endcsname #1#2#3#4#5#6#7#8#9%
447 {\noexpand\csname XINT_g36\expandafter\noexpand\csname XINT_g28\endcsname}%
448 \long\expandafter\edef\csname XINT_g38\endcsname #1#2#3#4#5#6#7#8#9%
449 {\noexpand\csname XINT_g37\expandafter\noexpand\csname XINT_g28\endcsname}%
450 \expandafter\let\csname XINT_g40\endcsname\endcsname
451 \expandafter\edef\csname XINT_g41\endcsname
452 {\noexpand\csname XINT_g38\expandafter\noexpand\csname XINT_g31\endcsname}%
453 \expandafter\edef\csname XINT_g42\endcsname
454 {\noexpand\csname XINT_g41\expandafter\noexpand\csname XINT_g41\endcsname}%
455 \expandafter\edef\csname XINT_g43\endcsname
456 {\noexpand\csname XINT_g42\expandafter\noexpand\csname XINT_g41\endcsname}%
457 \expandafter\edef\csname XINT_g44\endcsname
458 {\noexpand\csname XINT_g43\expandafter\noexpand\csname XINT_g41\endcsname}%
459 \expandafter\edef\csname XINT_g45\endcsname
460 {\noexpand\csname XINT_g44\expandafter\noexpand\csname XINT_g41\endcsname}%
461 \expandafter\edef\csname XINT_g46\endcsname
462 {\noexpand\csname XINT_g45\expandafter\noexpand\csname XINT_g41\endcsname}%
463 \expandafter\edef\csname XINT_g47\endcsname
464 {\noexpand\csname XINT_g46\expandafter\noexpand\csname XINT_g41\endcsname}%
465 \expandafter\edef\csname XINT_g48\endcsname
466 {\noexpand\csname XINT_g47\expandafter\noexpand\csname XINT_g41\endcsname}%
467 \expandafter\let\csname XINT_g50\endcsname\endcsname
468 \expandafter\edef\csname XINT_g51\endcsname
469 {\noexpand\csname XINT_g48\expandafter\noexpand\csname XINT_g41\endcsname}%
470 \expandafter\edef\csname XINT_g52\endcsname
471 {\noexpand\csname XINT_g51\expandafter\noexpand\csname XINT_g51\endcsname}%
472 \expandafter\edef\csname XINT_g53\endcsname
473 {\noexpand\csname XINT_g52\expandafter\noexpand\csname XINT_g51\endcsname}%
474 \expandafter\edef\csname XINT_g54\endcsname
475 {\noexpand\csname XINT_g53\expandafter\noexpand\csname XINT_g51\endcsname}%
476 \expandafter\edef\csname XINT_g55\endcsname
477 {\noexpand\csname XINT_g54\expandafter\noexpand\csname XINT_g51\endcsname}%
478 \expandafter\edef\csname XINT_g56\endcsname
479 {\noexpand\csname XINT_g55\expandafter\noexpand\csname XINT_g51\endcsname}%

```



```

480 \expandafter\edef\csname XINT_g57\endcsname
481 {\noexpand\csname XINT_g56\expandafter\noexpand\csname XINT_g51\endcsname}%
482 \expandafter\edef\csname XINT_g58\endcsname
483 {\noexpand\csname XINT_g57\expandafter\noexpand\csname XINT_g51\endcsname}%
484 \expandafter\let\csname XINT_g60\endcsname\endcsname
485 \expandafter\edef\csname XINT_g61\endcsname
486 {\noexpand\csname XINT_g58\expandafter\noexpand\csname XINT_g51\endcsname}%
487 \expandafter\edef\csname XINT_g62\endcsname
488 {\noexpand\csname XINT_g61\expandafter\noexpand\csname XINT_g61\endcsname}%
489 \expandafter\edef\csname XINT_g63\endcsname
490 {\noexpand\csname XINT_g62\expandafter\noexpand\csname XINT_g61\endcsname}%
491 \expandafter\edef\csname XINT_g64\endcsname
492 {\noexpand\csname XINT_g63\expandafter\noexpand\csname XINT_g61\endcsname}%
493 \expandafter\edef\csname XINT_g65\endcsname
494 {\noexpand\csname XINT_g64\expandafter\noexpand\csname XINT_g61\endcsname}%
495 \expandafter\edef\csname XINT_g66\endcsname
496 {\noexpand\csname XINT_g65\expandafter\noexpand\csname XINT_g61\endcsname}%
497 \expandafter\edef\csname XINT_g67\endcsname
498 {\noexpand\csname XINT_g66\expandafter\noexpand\csname XINT_g61\endcsname}%
499 \expandafter\edef\csname XINT_g68\endcsname
500 {\noexpand\csname XINT_g67\expandafter\noexpand\csname XINT_g61\endcsname}%

```

## 2.18 (WIP) \xintUniformDeviate

**1.3b.** See user manual for related information.

```

501 \ifdefined\xint_texuniformdeviate
502   \expandafter\xint_firstoftwo
503 \else\expandafter\xint_secondoftwo
504 \fi
505 {%
506   \def\xintUniformDeviate#1%
507     {\the\numexpr\expandafter\XINT_uniformdeviate_sgnfork\the\numexpr#1\xint:}%
508   \def\XINT_uniformdeviate_sgnfork#1%
509     {%
510       \if-#1\XINT_uniformdeviate_neg\fi \XINT_uniformdeviate{ }#1%
511     }%
512   \def\XINT_uniformdeviate_neg\fi\XINT_uniformdeviate#1-%
513     {%
514       \fi-\numexpr\XINT_uniformdeviate\relax
515     }%
516   \def\XINT_uniformdeviate#1#2\xint:
517     {% (
518       \expandafter\XINT_uniformdeviate_a\the\numexpr%
519         -\xint_texuniformdeviate\xint_c_ii^vii%
520         -\xint_c_ii^vii*\xint_texuniformdeviate\xint_c_ii^vii%
521         -\xint_c_ii^xiv*\xint_texuniformdeviate\xint_c_ii^vii%
522         -\xint_c_ii^xxi*\xint_texuniformdeviate\xint_c_ii^vii%
523         +\xint_texuniformdeviate#2\xint:/#2)*#2\xint:+#2\fi\relax#1%
524     }%
525   \def\XINT_uniformdeviate_a #1\xint:
526     {%
527       \expandafter\XINT_uniformdeviate_b\the\numexpr#1-(#1%

```



```

528 }%
529 \def\XINT_uniformdeviate_b#1#2\xint:{#1#2\if-#1}%
530 }%
531 {%
532 \def\xintUniformDeviate#1%
533 {%
534     \the\numexpr
535     \XINT_expandableerror{No uniformdeviate at engine level, returning 0.}%
536     0\relax
537 }%
538 }%

```

## 2.19 \xintMessage, \ifxintverbose

1.2c. 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.

1.3e. set the `\newlinechar`.

```

539 \def\xintMessage #1#2#3{%
540     \edef\XINT_newlinechar{\the\n newlinechar}%
541     \newlinechar10
542     \immediate\write128{Package #1 #2: (on line \the\inputlineno)}%
543     \immediate\write128{\space\space\space\space#3}%
544     \newlinechar\XINT_newlinechar\space
545 }%
546 \newif\ifxintverbose

```

## 2.20 \ifxintglobaldefs, \XINT\_global

1.3c.

```

547 \newif\ifxintglobaldefs
548 \def\XINT_global{\ifxintglobaldefs\global\fi}%

```

## 2.21 (WIP) Expandable error message

1.21. but really belongs to next major release beyond 1.3.

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.

```

549 \def\XINT_expandableerror #1#2{%
550     \def\XINT_expandableerror ##1{%
551         \expandafter\expandafter\expandafter
552         \XINT_expandableerror_continue\xint_firstofone{#2#1##1#1}}%
553     \def\XINT_expandableerror_continue ##1#1##2#1{##1}%
554 }%
555 \begingroup\lccode`$ 32 \catcode`/ 11 \catcode`! 11 \catcode32 11 % $
556 \lowercase{\endgroup\XINT_expandableerror$ \ ! /\let\ ! /\xint_undefined}% $
557 \XINT_restorecatcodes_endinput%

```

### 3 Package [xinttools](#) implementation

.1	Catcodes, $\varepsilon$ -TeX and reload detection . . .	18			
.2	Package identification . . . . .	19	.21	<code>\XINT_xflet</code> . . . . .	35
.3	<code>\xintgodef</code> , <code>\xintgoodef</code> , <code>\xintgdef</code> . . . . .	19	.22	<code>\xintApplyInline</code> . . . . .	36
.4	<code>\xintRevWithBraces</code> . . . . .	19	.23	<code>\xintFor</code> , <code>\xintFor*</code> , <code>\xintBreakFor</code> , <code>\xintBreakForAndDo</code> . . . . .	36
.5	<code>\xintZapFirstSpaces</code> . . . . .	20	.24	<code>\XINT_forever</code> , <code>\xintintegers</code> , <code>\xintdi-</code> <code>mensions</code> , <code>\xinrationals</code> . . . . .	39
.6	<code>\xintZapLastSpaces</code> . . . . .	21	.25	<code>\xintForpair</code> , <code>\xintForthree</code> , <code>\xintFor-</code> <code>four</code> . . . . .	41
.7	<code>\xintZapSpaces</code> . . . . .	22	.26	<code>\xintAssign</code> , <code>\xintAssignArray</code> , <code>\xint-</code> <code>DigitsOf</code> . . . . .	42
.8	<code>\xintZapSpacesB</code> . . . . .	22	.27	<code>\xintExpandArgs</code> . . . . .	45
.9	<code>\xintCSVtoList</code> , <code>\xintCSVtoListNon-</code> <code>Stripped</code> . . . . .	22	.28	CSV (non user documented) variants of Length, Keep, Trim, NthElt, Reverse . . . . .	45
.10	<code>\xintListWithSep</code> . . . . .	24	.28.1	<code>\xintLength:f:csv</code> . . . . .	46
.11	<code>\xintNthElt</code> . . . . .	25	.28.2	<code>\xintLengthUpTo:f:csv</code> . . . . .	46
.12	<code>\xintKeep</code> . . . . .	26	.28.3	<code>\xintKeep:f:csv</code> . . . . .	47
.13	<code>\xintKeepUnbraced</code> . . . . .	27	.28.4	<code>\xintTrim:f:csv</code> . . . . .	49
.14	<code>\xintTrim</code> . . . . .	29	.28.5	<code>\xintNthEltPy:f:csv</code> . . . . .	51
.15	<code>\xintTrimUnbraced</code> . . . . .	30	.28.6	<code>\xintReverse:f:csv</code> . . . . .	52
.16	<code>\xintApply</code> . . . . .	31	.28.7	<code>\xintFirstItem:f:csv</code> . . . . .	52
.17	<code>\xintApplyUnbraced</code> . . . . .	31	.28.8	<code>\xintLastItem:f:csv</code> . . . . .	52
.18	<code>\xintSeq</code> . . . . .	32	.28.9	Public names for the undocumented csv macros: <code>\xintCSVLength</code> , <code>\xintCSVKeep</code> , <code>\xintCSVTrim</code> , <code>\xintCSVNthEltPy</code> , <code>\xintCSVReverse</code> , <code>\xintCSV-</code> <code>FirstItem</code> , <code>\xintCSVLastItem</code> . . . . .	53
.19	<code>\xintloop</code> , <code>\xintbreakloop</code> , <code>\xintbreak-</code> <code>loopanddo</code> , <code>\xintloopskiptonext</code> . . . . .	34			
.20	<code>\xintiloop</code> , <code>\xintiloopindex</code> , <code>\xint-</code> <code>bracediloopindex</code> , <code>\xintouteriloopindex</code> , <code>\xintbracedouteriloopindex</code> , <code>\xintbreak-</code> <code>iloop</code> , <code>\xintbreakiloopanddo</code> , <code>\xintiloop-</code>				

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`.

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

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

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2 \catcode13=5 % ^^M
3 \endlinechar=13 %
4 \catcode123=1 % {
5 \catcode125=2 % }
6 \catcode64=11 % @
7 \catcode35=6 % #
8 \catcode44=12 % ,
9 \catcode45=12 % -
10 \catcode46=12 % .
11 \catcode58=12 % :
12 \let\z\endgroup
13 \expandafter\let\expandafter\x\csname ver@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

```

## 3.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xinttools}%
46 [2019/04/05 1.3e 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!

```

## 3.3 `\xintgodef`, `\xintgoodef`, `\xintgfdef`

1.09i. For use in `\xintAssign`.

```

49 \def\xintgodef {\global\xintodef }%
50 \def\xintgoodef {\global\xintoodef }%
51 \def\xintgfdef {\global\xintfdef }%

```

## 3.4 `\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\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{ }%

```

### 3.5 \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}%
```

### 3.6 \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 nor 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{ }%
```

### 3.7 `\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\xintzapspace }%
114 \def\xintzapspace#1{%
115 \long\def\xintzapspace ##1% like \xintZapFirstSpaces.
116     {\XINT_zapsp_a #1##1\xint:#1#1\xint:}%
117 }\xintzapspace{ }%
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_a\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{ }%

```

### 3.8 `\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\xintzapspaceb }%
130 \long\def\xintzapspaceb #1{\XINT_zapspb_one? #1\xint:\xint:%
131     \xint_bye\xintzapspace {#1}}%
132 \long\def\XINT_zapspb_one? #1#2%
133     {\xint_gob_til_xint: #1\XINT_zapspb_onlyspace\xint:%
134     \xint_gob_til_xint: #2\XINT_zapspb_bracedorone\xint:%
135     \xint_bye {#1}}%
136 \def\XINT_zapspb_onlyspace\xint:%
137     \xint_gob_til_xint:\xint:\XINT_zapspb_bracedorone\xint:%
138     \xint_bye #1\xint_bye\xintzapspace #2{ }%
139 \long\def\XINT_zapspb_bracedorone\xint:%
140     \xint_bye #1\xint:\xint_bye\xintzapspace #2{ #1}%

```

### 3.9 `\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.

ATTENTION: if the input is empty the output contains one item (empty, of course). This means an `\xintFor` loop always executes at least once the iteration, contrarily to `\xintFor*`.

```

141 \def\xintCSVtoList {\romannumeral0\xintcsvtolist }%
142 \long\def\xintcsvtolist #1{\expandafter\xintApply

```

```

143     \expandafter\xintzapspacesb
144     \expandafter{\romannumeral0\xintcsvtolistnonstripped{#1}}}%
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_end_b`).

```

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}}%

```

### 3.10 \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}%

```

### 3.11 \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_stop_afterbye
266     0#1\xint_stop_afterbye
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{ }%

```

### 3.12 \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_keepnone
294     0#1\XINT_keep_neg
295     0-{\XINT_keep_pos #1}%
296     \krof
297 }%
298 \long\def\XINT_keep_keepnone .#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:\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}%

```

### 3.13 \xintKeepUnbraced

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

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

Modified for 1.2i like `\xintKeep`.

```

351 \def\xintKeepUnbraced      {\romannumeral0\xintkeepunbraced }%

```

```

352 \def\xintKeepUnbracedNoExpand {\romannumeral0\xintkeepunbracednoexpand }%
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_keeptime
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}%

```

### 3.14 `\xintTrim`

First included in release 1.09m.

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

`\xintTrimNoExpand` does not expand the L argument.

Speed improvements 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:\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}%

```

### 3.15 \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 }%
```

### 3.16 \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}%
```

### 3.17 \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}%

```

### 3.18 \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_stop_atfirstoftwo
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
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_seqo_nb
626   \fi
627   {#1}{#2}{#3}{{#1}}%
628 }%
629 \def\XINT_seqo_nb #1#2#3%
630 {%
631   \expandafter\XINT_seqo_nc\expandafter{\the\numexpr #1+#3}{#2}{#3}%
632 }%
633 \def\XINT_seqo_nc #1#2%
634 {%
635   \ifnum #1<#2
636     \expandafter\XINT_seqo_o
637   \else
638     \expandafter\XINT_seqo_nd
639   \fi
640   {#1}{#2}%
641 }%
642 \def\XINT_seqo_nd #1#2#3#4{\XINT_seqo_nb {#1}{#2}{#3}{#4{#1}}}%

```

### 3.19 `\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}}%

```

### 3.20 `\xintilooop`, `\xintilooopindex`, `\xintbracedilooopindex`, `\xintouterilooopindex`, `\xintbracedouterilooopindex`, `\xintbreakilooop`, `\xintbreakilooopaddo`, `\xintilooopskiptonext`, `\xintilooopskipandredo`

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

«braced» variants added (2018/04/24) for 1.3b.

```

650 \def\xintilooop [#1+#2]{%
651   \expandafter\xintilooop_a\the\numexpr #1\expandafter.\the\numexpr #2.%
652 \long\def\xintilooop_a #1.#2.#3#4\repeat{%
653   #3#4\xintilooop_again\fi\xint_gobble_iii {#1}{#2}{#3#4}}%
654 \def\xintilooop_again\fi\xint_gobble_iii #1#2{%
655   \fi\expandafter\xintilooop_again_b\the\numexpr#1+#2.#2.%
656 \long\def\xintilooop_again_b #1.#2.#3{%
657   #3\xintilooop_again\fi\xint_gobble_iii {#1}{#2}{#3}}%
658 \long\def\xintbreakilooop #1\xintilooop_again\fi\xint_gobble_iii #2#3#4{%
659 \long\def\xintbreakilooopaddo
660   #1.#2\xintilooop_again\fi\xint_gobble_iii #3#4#5{#1}%
661 \long\def\xintilooopindex #1\xintilooop_again\fi\xint_gobble_iii #2%

```

```

662             {#2#1\xintiloop_again\fi\xint_gobble_iii {#2}}%
663 \long\def\xintbracediloopindex #1\xintiloop_again\fi\xint_gobble_iii #2%
664     {{{#2}#1\xintiloop_again\fi\xint_gobble_iii {#2}}}%
665 \long\def\xintouteriloopindex #1\xintiloop_again
666             #2\xintiloop_again\fi\xint_gobble_iii #3%
667     {#3#1\xintiloop_again #2\xintiloop_again\fi\xint_gobble_iii {#3}}%
668 \long\def\xintbracedouteriloopindex #1\xintiloop_again
669             #2\xintiloop_again\fi\xint_gobble_iii #3%
670     {{{#3}#1\xintiloop_again #2\xintiloop_again\fi\xint_gobble_iii {#3}}}%
671 \long\def\xintiloopskipnext #1\xintiloop_again\fi\xint_gobble_iii #2#3{%
672     \expandafter\xintiloop_again_b \the\numexpr#2+#3.#3.}%
673 \long\def\xintiloopskipandredo #1\xintiloop_again\fi\xint_gobble_iii #2#3#4{%
674     #4\xintiloop_again\fi\xint_gobble_iii {#2}{#3}{#4}}%

```

### 3.21 \XINT\_xflet

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

```

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

```

709 }%

### 3.22 `\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.

```

710 \catcode`Z 3
711 \long\def\xintApplyInline #1#2%
712 {%
713   \long\expandafter\def\expandafter\XINT_inline_macro
714   \expandafter ##\expandafter 1\expandafter {#1{##1}}%
715   \XINT_xflet\XINT_inline_b #2Z% this Z has catcode 3
716 }%
717 \def\XINT_inline_b
718 {%
719   \ifx\XINT_token Z\expandafter\xint_gobble_i
720   \else\expandafter\XINT_inline_d\fi
721 }%
722 \long\def\XINT_inline_d #1%
723 {%
724   \long\def\XINT_item{#1}\XINT_xflet\XINT_inline_e
725 }%
726 \def\XINT_inline_e
727 {%
728   \ifx\XINT_token Z\expandafter\XINT_inline_w
729   \else\expandafter\XINT_inline_f\fi
730 }%
731 \def\XINT_inline_f
732 {%
733   \expandafter\XINT_inline_g\expandafter{\XINT_inline_macro {##1}}%
734 }%
735 \long\def\XINT_inline_g #1%
736 {%
737   \expandafter\XINT_inline_macro\XINT_item
738   \long\def\XINT_inline_macro ##1{#1}\XINT_inline_d
739 }%
740 \def\XINT_inline_w #1%
741 {%
742   \expandafter\XINT_inline_macro\XINT_item
743 }%

```

### 3.23 `\xintFor`, `\xintFor*`, `\xintBreakFor`, `\xintBreakForAndDo`

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

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

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

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

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

```

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

```

```

786 \ifx\XINT_token D\XINT_to_forever\fi
787 \expandafter\the\expandafter\XINT_toks\romannumeral0\xintcsvtolist {#1}Z%
788 }%
789 \def\XINT_to_forever\fi #1\xintcsvtolist #2{\fi \XINT_forever #2}%
790 \long\def\XINT_forx *#1#2in#3#4#5%
791 {%
792 \expandafter\XINT_toks\expandafter
793 {\expandafter\XINT_forx_d\the\numexpr #2\relax {#5}}%
794 \XINT_xflet\XINT_forx_forever? #3Z%
795 }%
796 \def\XINT_forx_forever?
797 {%
798 \ifx\XINT_token U\XINT_to_forxever\fi
799 \ifx\XINT_token V\XINT_to_forxever\fi
800 \ifx\XINT_token D\XINT_to_forxever\fi
801 \XINT_forx_empty?
802 }%
803 \def\XINT_to_forxever\fi #1\XINT_forx_empty? {\fi \XINT_forever }%
804 \catcode`U 11
805 \catcode`D 11
806 \catcode`V 11
807 \def\XINT_forx_empty?
808 {%
809 \ifx\XINT_token Z\expandafter\xintBreakFor\fi
810 \the\XINT_toks
811 }%
812 \long\def\XINT_for_d #1#2#3%
813 {%
814 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
815 \XINT_toks {{#3}}%
816 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
817 \the\XINT_toks \csname XINT_for_right#1\endcsname }%
818 \XINT_toks {\XINT_x\let\xintifForFirst\xint_secondoftwo
819 \let\xintifForLast\xint_secondoftwo\XINT_for_d #1{#2}}%
820 \futurelet\XINT_token\XINT_for_last?
821 }%
822 \long\def\XINT_forx_d #1#2#3%
823 {%
824 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
825 \XINT_toks {{#3}}%
826 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
827 \the\XINT_toks \csname XINT_for_right#1\endcsname }%
828 \XINT_toks {\XINT_x\let\xintifForFirst\xint_secondoftwo
829 \let\xintifForLast\xint_secondoftwo\XINT_forx_d #1{#2}}%
830 \XINT_xflet\XINT_for_last?
831 }%
832 \def\XINT_for_last?
833 {%
834 \ifx\XINT_token Z\expandafter\XINT_for_last?yes\fi
835 \the\XINT_toks
836 }%
837 \def\XINT_for_last?yes

```

```

838 {%
839 \let\xintifForLast\xint_firstoftwo
840 \xintBreakForAndDo{\XINT_x\xint_gobble_i Z}%
841 }%

```

### 3.24 \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).

```

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

```

```

881     {\romannumeral0\xintiimul {#2}{#4}}%
882     {\romannumeral0\xintiimul {#1}{#4}}%
883     {\romannumeral0\xintiimul {#2}{#3}}%
884     }%
885 }%
886 \def\xINT_?expr_Vd #1#2#3{\expandafter\xINT_?expr_Ve\expandafter {#2}{#3}{#1}}%
887 \def\xINT_?expr_Ve #1#2{\expandafter\xINT_?expr_Vf\expandafter {#2}{#1}}%
888 \def\xINT_?expr_Vf #1#2#3{{#2/#3}{0}{#1}{#2}{#3}}%
889 \def\xINT_?expr_Ui {{\numexpr 1\relax}{1}}%
890 \def\xINT_?expr_Di {{\dimexpr 0pt\relax}{65536}}%
891 \def\xINT_?expr_Vi {{1/1}{0111}}%
892 \def\xINT_?expr_U #1#2%
893     {\expandafter{\expandafter\numexpr\the\numexpr #1+#2\relax\relax}{#2}}%
894 \def\xINT_?expr_D #1#2%
895     {\expandafter{\expandafter\dimexpr\the\numexpr #1+#2\relax sp\relax}{#2}}%
896 \def\xINT_?expr_V #1#2{\XINT_?expr_Vx #2}%
897 \def\xINT_?expr_Vx #1#2%
898 {%
899     \expandafter\xINT_?expr_Vy\expandafter
900     {\romannumeral0\xintiiaadd {#1}{#2}{#2}}%
901 }%
902 \def\xINT_?expr_Vy #1#2#3#4%
903 {%
904     \expandafter{\romannumeral0\xintiiaadd {#3}{#1}/#4}{#1}{#2}{#3}{#4}}%
905 }%
906 \def\xINT_forever_a #1#2#3#4%
907 {%
908     \ifx #4[\expandafter\xINT_forever_opt_a
909     \else\expandafter\xINT_forever_b
910     \fi #1#2#3#4%
911 }%
912 \def\xINT_forever_b #1#2#3Z{\expandafter\xINT_forever_c\the\xINT_toks #2#3}%
913 \long\def\xINT_forever_c #1#2#3#4#5%
914     {\expandafter\xINT_forever_d\expandafter #2#4#5{#3}Z}%
915 \def\xINT_forever_opt_a #1#2#3[#4+#5]#6Z%
916 {%
917     \expandafter\expandafter\expandafter
918     \XINT_forever_opt_c\expandafter\the\expandafter\xINT_toks
919     \romannumeral`&&@#1{#4}{#5}#3%
920 }%
921 \long\def\xINT_forever_opt_c #1#2#3#4#5#6{\XINT_forever_d #2{#4}{#5}#6{#3}Z}%
922 \long\def\xINT_forever_d #1#2#3#4#5%
923 {%
924     \long\def\xINT_y ##1##2##3##4##5##6##7##8##9{#5}%
925     \XINT_toks {{#2}}%
926     \long\edef\xINT_x {\noexpand\xINT_y \csname XINT_for_left#1\endcsname
927         \the\xINT_toks \csname XINT_for_right#1\endcsname }%
928     \XINT_x
929     \let\xintifForFirst\xint_secondoftwo
930     \let\xintifForLast\xint_secondoftwo
931     \expandafter\xINT_forever_d\expandafter #1\romannumeral`&&@#4{#2}{#3}#4{#5}%
932 }%

```



### 3.25 \xintForpair, \xintForthree, \xintForfour

1.09c.

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

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

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

```

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

```

```

977 \expandafter\the\expandafter\XINT_toks #4jZ%
978 }%
979 \long\def\XINT_forfour_d #1#2#3(#4)#5%
980 {%
981 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
982 \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
983 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
984 \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_iii\endcsname}%
985 \ifx #5j\expandafter\XINT_for_last?yes\fi
986 \XINT_x
987 \let\xintifForFirst\xint_secondoftwo
988 \let\xintifForLast\xint_secondoftwo
989 \XINT_forfour_d #1{#2}%
990 }%
991 \catcode`Z 11
992 \catcode`j 11

```

### 3.26 \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.

```

993 \def\xintAssign{\def\XINT_flet_macro {\XINT_assign_fork}\XINT_flet_zapsp }%
994 \def\XINT_assign_fork
995 {%
996 \let\XINT_assign_def\def
997 \ifx\XINT_token[\expandafter\XINT_assign_opt
998 \else\expandafter\XINT_assign_a
999 \fi
1000 }%
1001 \def\XINT_assign_opt [#1]%
1002 {%
1003 \ifcsname #1def\endcsname
1004 \expandafter\let\expandafter\XINT_assign_def \csname #1def\endcsname
1005 \else
1006 \expandafter\let\expandafter\XINT_assign_def \csname xint#1def\endcsname
1007 \fi
1008 \XINT_assign_a
1009 }%
1010 \long\def\XINT_assign_a #1\to
1011 {%
1012 \def\XINT_flet_macro{\XINT_assign_b}%
1013 \expandafter\XINT_flet_zapsp\romannumeral`&&@#1\xint:\to
1014 }%
1015 \long\def\XINT_assign_b
1016 {%
1017 \ifx\XINT_token\bgroup
1018 \expandafter\XINT_assign_c
1019 \else\expandafter\XINT_assign_f
1020 \fi

```

```

1021 }%
1022 \long\def\XINT_assign_f #1\xint:\to #2%
1023 {%
1024   \XINT_assign_def #2{#1}%
1025 }%
1026 \long\def\XINT_assign_c #1%
1027 {%
1028   \def\xint_temp {#1}%
1029   \ifx\xint_temp\xint_bracedstopper
1030     \expandafter\XINT_assign_e
1031   \else
1032     \expandafter\XINT_assign_d
1033   \fi
1034 }%
1035 \long\def\XINT_assign_d #1\to #2%
1036 {%
1037   \expandafter\XINT_assign_def\expandafter #2\expandafter{\xint_temp}%
1038   \XINT_assign_c #1\to
1039 }%
1040 \def\XINT_assign_e #1\to {}%
1041 \def\xintRelaxArray #1%
1042 {%
1043   \edef\XINT_restoreescapechar {\escapechar\the\escapechar\relax}%
1044   \escapechar -1
1045   \expandafter\def\expandafter\xint_arrayname\expandafter {\string #1}%
1046   \XINT_restoreescapechar
1047   \xintilop [\csname\xint_arrayname 0\endcsname+-1]
1048   \global
1049   \expandafter\let\csname\xint_arrayname\xintilopindex\endcsname\relax
1050   \ifnum \xintilopindex > \xint_c_
1051     \repeat
1052     \global\expandafter\let\csname\xint_arrayname 00\endcsname\relax
1053     \global\let #1\relax
1054 }%
1055 \def\xintAssignArray{\def\XINT_flet_macro {\XINT_assignarray_fork}%
1056   \XINT_flet_zapsp }%
1057 \def\XINT_assignarray_fork
1058 {%
1059   \let\XINT_assignarray_def\def
1060   \ifx\XINT_token[\expandafter\XINT_assignarray_opt
1061     \else\expandafter\XINT_assignarray
1062   \fi
1063 }%
1064 \def\XINT_assignarray_opt [#1]%
1065 {%
1066   \ifcsname #1def\endcsname
1067     \expandafter\let\expandafter\XINT_assignarray_def \csname #1def\endcsname
1068   \else
1069     \expandafter\let\expandafter\XINT_assignarray_def
1070       \csname xint#1def\endcsname
1071   \fi
1072   \XINT_assignarray

```

```

1073 }%
1074 \long\def\XINT_assignarray #1\to #2%
1075 {%
1076   \edef\XINT_restoreescapechar {\escapechar\the\escapechar\relax }%
1077   \escapechar -1
1078   \expandafter\def\expandafter\xint_arrayname\expandafter {\string #2}%
1079   \XINT_restoreescapechar
1080   \def\xint_itemcount {0}%
1081   \expandafter\XINT_assignarray_loop \romannumeral`&&@#1\xint:
1082   \csname\xint_arrayname 00\expandafter\endcsname
1083   \csname\xint_arrayname 0\expandafter\endcsname
1084   \expandafter {\xint_arrayname}#2%
1085 }%
1086 \long\def\XINT_assignarray_loop #1%
1087 {%
1088   \def\xint_temp {#1}%
1089   \ifx\xint_temp\xint_bracedstopper
1090     \expandafter\def\csname\xint_arrayname 0\expandafter\endcsname
1091       \expandafter{\the\numexpr\xint_itemcount}%
1092     \expandafter\expandafter\expandafter\XINT_assignarray_end
1093   \else
1094     \expandafter\def\expandafter\xint_itemcount\expandafter
1095       {\the\numexpr\xint_itemcount+\xint_c_i}%
1096     \expandafter\XINT_assignarray_def
1097       \csname\xint_arrayname\xint_itemcount\expandafter\endcsname
1098     \expandafter{\xint_temp }%
1099     \expandafter\XINT_assignarray_loop
1100   \fi
1101 }%
1102 \def\XINT_assignarray_end #1#2#3#4%
1103 {%
1104   \def #4##1%
1105   {%
1106     \romannumeral0\expandafter #1\expandafter{\the\numexpr ##1}%
1107   }%
1108   \def #1##1%
1109   {%
1110     \ifnum ##1<\xint_c_
1111       \xint_afterfi{\XINT_expandableerror{Array index negative: 0 > ##1} }%
1112     \else
1113       \xint_afterfi {%
1114         \ifnum ##1>#2
1115           \xint_afterfi
1116             {\XINT_expandableerror{Array index beyond range: ##1 > #2} }%
1117         \else\xint_afterfi
1118       {\expandafter\expandafter\expandafter\space\csname #3##1\endcsname}%
1119       \fi}%
1120   \fi
1121 }%
1122 }%
1123 \let\xintDigitsOf\xintAssignArray

```

### 3.27 `\xintExpandArgs`

1.3a. Added for the needs of user defined functions for the expression parsers. Should I re-code it to gain a bit in argument grabbing? Must be f-expandable.

```
1124 \def\xintExpandArgs#1#2{\csname #1\expandafter\endcsname
1125   \romannumeral0\xintapply\xint_firstofone{#2}}%
```

### 3.28 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.

### 3.28.1 `\xintLength:f:csv`

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

```
1126 \def\xintLength:f:csv {\romannumeral0\xintlength:f:csv}%
1127 \def\xintlength:f:csv #1%
1128 {\long\def\xintlength:f:csv ##1{%
1129   \expandafter#1\the\numexpr\expandafter\XINT_length:f:csv_a
1130   \romannumeral`&&##1\xint:,\xint:,\xint:,\xint:,%
1131   \xint:,\xint:,\xint:,\xint:,\xint:,%
1132   \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
1133   \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
1134   \relax
1135 }}\xintlength:f:csv { }%
```

Must first check if empty list.

```
1136 \long\def\XINT_length:f:csv_a #1%
1137 {%
1138   \xint_gob_til_xint: #1\xint_c_\xint_bye\xint:%
1139   \XINT_length:f:csv_loop #1%
1140 }%
1141 \long\def\XINT_length:f:csv_loop #1,#2,#3,#4,#5,#6,#7,#8,#9,%
1142 {%
1143   \xint_gob_til_xint: #9\XINT_length:f:csv_finish\xint:%
1144   \xint_c_ix+\XINT_length:f:csv_loop
1145 }%
1146 \def\XINT_length:f:csv_finish\xint:\xint_c_ix+\XINT_length:f:csv_loop
1147   #1,#2,#3,#4,#5,#6,#7,#8,#9,{#9\xint_bye}%
```

### 3.28.2 `\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.

```
1148 \def\xintLengthUpTo:f:csv {\romannumeral0\xintlengthupto:f:csv}%
1149 \long\def\xintlengthupto:f:csv #1#2%
1150 {%
1151   \expandafter\XINT_lengthupto:f:csv_a
1152   \the\numexpr#1\expandafter.%
1153   \romannumeral`&&##2\xint:,\xint:,\xint:,\xint:,%
1154   \xint:,\xint:,\xint:,\xint:,%
1155   \xint_c_viii,\xint_c_vii,\xint_c_vi,\xint_c_v,%
1156   \xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye.%
1157 }%
```

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

```

1158 \long\def\XINT_lengthupto:f:csv_a #1.#2%
1159 {%
1160     \xint_gob_til_xint: #2\XINT_lengthupto:f:csv_empty\xint:%
1161     \XINT_lengthupto:f:csv_loop_b #1.#2%
1162 }%
1163 \def\XINT_lengthupto:f:csv_empty\xint:%
1164     \XINT_lengthupto:f:csv_loop_b #1.#2\xint_bye.{ #1}%
1165 \def\XINT_lengthupto:f:csv_loop_a #1%
1166 {%
1167     \xint_UDsignfork
1168     #1\XINT_lengthupto:f:csv_gt
1169     -\XINT_lengthupto:f:csv_loop_b
1170     \krof #1%
1171 }%
1172 \long\def\XINT_lengthupto:f:csv_gt #1\xint_bye.{-0}%
1173 \long\def\XINT_lengthupto:f:csv_loop_b #1.#2,#3,#4,#5,#6,#7,#8,#9,%
1174 {%
1175     \xint_gob_til_xint: #9\XINT_lengthupto:f:csv_finish_a\xint:%
1176     \expandafter\XINT_lengthupto:f:csv_loop_a\the\numexpr #1-\xint_c_viii.%
1177 }%
1178 \def\XINT_lengthupto:f:csv_finish_a\xint:
1179     \expandafter\XINT_lengthupto:f:csv_loop_a
1180     \the\numexpr #1-\xint_c_viii.#2,#3,#4,#5,#6,#7,#8,#9,%
1181 {%
1182     \expandafter\XINT_lengthupto:f:csv_finish_b\the\numexpr #1-#9\xint_bye
1183 }%
1184 \def\XINT_lengthupto:f:csv_finish_b #1#2.%
1185 {%
1186     \xint_UDsignfork
1187     #1{-0}%
1188     -{ #1#2}%
1189     \krof
1190 }%

```

### 3.28.3 \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.

```

1191 \def\xintKeep:f:csv {\romannumeral0\xintkeep:f:csv }%
1192 \long\def\xintkeep:f:csv #1#2%
1193 {%
1194     \expandafter\xint_stop_aftergobble
1195     \romannumeral0\expandafter\XINT_keep:f:csv_a
1196     \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
1197 }%
1198 \def\XINT_keep:f:csv_a #1%
1199 {%
1200     \xint_UDzerominusfork
1201     #1-\XINT_keep:f:csv_keepnone

```

```

1202      0#1\XINT_keep:f:csv_neg
1203      0-{\XINT_keep:f:csv_pos #1}%
1204      \krof
1205 }%
1206 \long\def\XINT_keep:f:csv_keeptime .#1{,%
1207 \long\def\XINT_keep:f:csv_neg #1.#2%
1208 {%
1209   \expandafter\XINT_keep:f:csv_neg_done\expandafter,%
1210   \romannumeral0%
1211   \expandafter\XINT_keep:f:csv_neg_a\the\numexpr
1212   #1-\numexpr\XINT_length:f:csv_a
1213   #2\xint:,\xint:,\xint:,\xint:,%
1214   \xint:,\xint:,\xint:,\xint:,\xint:,%
1215   \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
1216   \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
1217   .#2\xint_bye
1218 }%
1219 \def\XINT_keep:f:csv_neg_a #1%
1220 {%
1221   \xint_UDsignfork
1222   #1{\expandafter\XINT_keep:f:csv_trimloop\the\numexpr-\xint_c_ix+}%
1223   -\XINT_keep:f:csv_keeptime
1224   \krof
1225 }%
1226 \def\XINT_keep:f:csv_keeptime #1.{ }%
1227 \long\def\XINT_keep:f:csv_neg_done #1\xint_bye{#1}%
1228 \def\XINT_keep:f:csv_trimloop #1#2.%
1229 {%
1230   \xint_gob_til_minus#1\XINT_keep:f:csv_trimloop_finish-%
1231   \expandafter\XINT_keep:f:csv_trimloop
1232   \the\numexpr#1#2-\xint_c_ix\expandafter.\XINT_keep:f:csv_trimloop_trimtime
1233 }%
1234 \long\def\XINT_keep:f:csv_trimloop_trimtime #1,#2,#3,#4,#5,#6,#7,#8,#9,{%
1235 \def\XINT_keep:f:csv_trimloop_finish-%
1236   \expandafter\XINT_keep:f:csv_trimloop
1237   \the\numexpr-#1-\xint_c_ix\expandafter.\XINT_keep:f:csv_trimloop_trimtime
1238   {\csname XINT_trim:f:csv_finish#1\endcsname}%
1239 \long\def\XINT_keep:f:csv_pos #1.#2%
1240 {%
1241   \expandafter\XINT_keep:f:csv_pos_fork
1242   \romannumeral0\XINT_lengthupto:f:csv_a
1243   #1.#2\xint:,\xint:,\xint:,\xint:,%
1244   \xint:,\xint:,\xint:,\xint:,%
1245   \xint_c_viii,\xint_c_vii,\xint_c_vi,\xint_c_v,%
1246   \xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye.%
1247   .#1.{ }#2\xint_bye%
1248 }%
1249 \def\XINT_keep:f:csv_pos_fork #1#2.%
1250 {%
1251   \xint_UDsignfork
1252   #1{\expandafter\XINT_keep:f:csv_loop\the\numexpr-\xint_c_viii+}%
1253   -\XINT_keep:f:csv_keeptime

```



```

1254 \krof
1255 }%
1256 \long\def\XINT_keep:f:csv_pos_keepall #1.#2#3\xint_bye{,#3}%
1257 \def\XINT_keep:f:csv_loop #1#2.%
1258 {%
1259 \xint_gob_til_minus#1\XINT_keep:f:csv_loop_end-%
1260 \expandafter\XINT_keep:f:csv_loop
1261 \the\numexpr#1#2-\xint_c_viii\expandafter.\XINT_keep:f:csv_loop_pickeight
1262 }%
1263 \long\def\XINT_keep:f:csv_loop_pickeight
1264 #1#2,#3,#4,#5,#6,#7,#8,#9,{{#1,#2,#3,#4,#5,#6,#7,#8,#9}}%
1265 \def\XINT_keep:f:csv_loop_end-\expandafter\XINT_keep:f:csv_loop
1266 \the\numexpr-#1-\xint_c_viii\expandafter.\XINT_keep:f:csv_loop_pickeight
1267 {\csname XINT_keep:f:csv_end#1\endcsname}%
1268 \long\expandafter\def\csname XINT_keep:f:csv_end1\endcsname
1269 #1#2,#3,#4,#5,#6,#7,#8,#9\xint_bye {#1,#2,#3,#4,#5,#6,#7,#8}%
1270 \long\expandafter\def\csname XINT_keep:f:csv_end2\endcsname
1271 #1#2,#3,#4,#5,#6,#7,#8\xint_bye {#1,#2,#3,#4,#5,#6,#7}%
1272 \long\expandafter\def\csname XINT_keep:f:csv_end3\endcsname
1273 #1#2,#3,#4,#5,#6,#7\xint_bye {#1,#2,#3,#4,#5,#6}%
1274 \long\expandafter\def\csname XINT_keep:f:csv_end4\endcsname
1275 #1#2,#3,#4,#5,#6\xint_bye {#1,#2,#3,#4,#5}%
1276 \long\expandafter\def\csname XINT_keep:f:csv_end5\endcsname
1277 #1#2,#3,#4,#5\xint_bye {#1,#2,#3,#4}%
1278 \long\expandafter\def\csname XINT_keep:f:csv_end6\endcsname
1279 #1#2,#3,#4\xint_bye {#1,#2,#3}%
1280 \long\expandafter\def\csname XINT_keep:f:csv_end7\endcsname
1281 #1#2,#3\xint_bye {#1,#2}%
1282 \long\expandafter\def\csname XINT_keep:f:csv_end8\endcsname
1283 #1#2\xint_bye {#1}%

```

### 3.28.4 \xintTrim:f:csv

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

```

1284 \def\xintTrim:f:csv {\romannumeral0\xinttrim:f:csv }%
1285 \long\def\xinttrim:f:csv #1#2%
1286 {%
1287 \expandafter\xint_stop_aftergobble
1288 \romannumeral0\expandafter\XINT_trim:f:csv_a
1289 \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
1290 }%
1291 \def\XINT_trim:f:csv_a #1%
1292 {%
1293 \xint_UDzerominusfork
1294 #1-\XINT_trim:f:csv_trimnone
1295 0#1\XINT_trim:f:csv_neg
1296 0-{\XINT_trim:f:csv_pos #1}%
1297 \krof
1298 }%
1299 \long\def\XINT_trim:f:csv_trimnone .#1{,#1}%
1300 \long\def\XINT_trim:f:csv_neg #1.#2%
1301 {%

```

```

1302 \expandafter\XINT_trim:f:csv_neg_a\the\numexpr
1303 #1-\numexpr\XINT_length:f:csv_a
1304 #2\xint:,\xint:,\xint:,\xint:,%
1305 \xint:,\xint:,\xint:,\xint:,\xint:,%
1306 \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
1307 \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
1308 .{ }#2\xint_bye
1309 }%
1310 \def\XINT_trim:f:csv_neg_a #1%
1311 {%
1312 \xint_UDsignfork
1313 #1{\expandafter\XINT_keep:f:csv_loop\the\numexpr-\xint_c_viii+}%
1314 -\XINT_trim:f:csv_trimall
1315 \krof
1316 }%
1317 \def\XINT_trim:f:csv_trimall {\expandafter,\xint_bye}%
1318 \long\def\XINT_trim:f:csv_pos #1.#2%
1319 {%
1320 \expandafter\XINT_trim:f:csv_pos_done\expandafter,%
1321 \romannumeral0%
1322 \expandafter\XINT_trim:f:csv_loop\the\numexpr#1-\xint_c_ix.%
1323 #2\xint:,\xint:,\xint:,\xint:,\xint:,%
1324 \xint:,\xint:,\xint:,\xint:,\xint:\xint_bye
1325 }%
1326 \def\XINT_trim:f:csv_loop #1#2.%
1327 {%
1328 \xint_gob_til_minus#1\XINT_trim:f:csv_finish-%
1329 \expandafter\XINT_trim:f:csv_loop\the\numexpr#1#2\XINT_trim:f:csv_loop_trimnine
1330 }%
1331 \long\def\XINT_trim:f:csv_loop_trimnine #1,#2,#3,#4,#5,#6,#7,#8,#9,%
1332 {%
1333 \xint_gob_til_xint: #9\XINT_trim:f:csv_toofew\xint:-\xint_c_ix.%
1334 }%
1335 \def\XINT_trim:f:csv_toofew\xint:{*\xint_c_}%
1336 \def\XINT_trim:f:csv_finish-%
1337 \expandafter\XINT_trim:f:csv_loop\the\numexpr-#1\XINT_trim:f:csv_loop_trimnine
1338 {%
1339 \csname XINT_trim:f:csv_finish#1\endcsname
1340 }%
1341 \long\expandafter\def\csname XINT_trim:f:csv_finish1\endcsname
1342 #1,#2,#3,#4,#5,#6,#7,#8,{ }%
1343 \long\expandafter\def\csname XINT_trim:f:csv_finish2\endcsname
1344 #1,#2,#3,#4,#5,#6,#7,{ }%
1345 \long\expandafter\def\csname XINT_trim:f:csv_finish3\endcsname
1346 #1,#2,#3,#4,#5,#6,{ }%
1347 \long\expandafter\def\csname XINT_trim:f:csv_finish4\endcsname
1348 #1,#2,#3,#4,#5,{ }%
1349 \long\expandafter\def\csname XINT_trim:f:csv_finish5\endcsname
1350 #1,#2,#3,#4,{ }%
1351 \long\expandafter\def\csname XINT_trim:f:csv_finish6\endcsname
1352 #1,#2,#3,{ }%
1353 \long\expandafter\def\csname XINT_trim:f:csv_finish7\endcsname

```

```

1354 #1,#2,{ }%
1355 \long\expandafter\def\csname XINT_trim:f:csv_finish8\endcsname
1356 #1,{ }%
1357 \expandafter\let\csname XINT_trim:f:csv_finish9\endcsname\space
1358 \long\def\XINT_trim:f:csv_pos_done #1\xint:#2\xint_bye{#1}%

```

### 3.28.5 \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.

```

1359 \def\xintNthEltPy:f:csv {\romannumeral0\xintntheltpy:f:csv }%
1360 \long\def\xintntheltpy:f:csv #1#2%
1361 {%
1362   \expandafter\XINT_nthelt:f:csv_a
1363   \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
1364 }%
1365 \def\XINT_nthelt:f:csv_a #1%
1366 {%
1367   \xint_UDsignfork
1368     #1\XINT_nthelt:f:csv_neg
1369     -\XINT_nthelt:f:csv_pos
1370   \krof #1%
1371 }%
1372 \long\def\XINT_nthelt:f:csv_neg -#1.#2%
1373 {%
1374   \expandafter\XINT_nthelt:f:csv_neg_fork
1375   \the\numexpr\XINT_length:f:csv_a
1376   #2\xint:,\xint:,\xint:,\xint:,%
1377   \xint:,\xint:,\xint:,\xint:,\xint:,%
1378   \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
1379   \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
1380   -#1.#2,\xint_bye
1381 }%
1382 \def\XINT_nthelt:f:csv_neg_fork #1%
1383 {%
1384   \if#1-\expandafter\xint_stop_afterbye\fi
1385   \expandafter\XINT_nthelt:f:csv_neg_done
1386   \romannumeral0%
1387   \expandafter\XINT_keep:f:csv_trimloop\the\numexpr-\xint_c_ix+#1%
1388 }%
1389 \long\def\XINT_nthelt:f:csv_neg_done#1,#2\xint_bye{ #1}%
1390 \long\def\XINT_nthelt:f:csv_pos #1.#2%
1391 {%
1392   \expandafter\XINT_nthelt:f:csv_pos_done
1393   \romannumeral0%
1394   \expandafter\XINT_trim:f:csv_loop\the\numexpr#1-\xint_c_ix.%
1395   #2\xint:,\xint:,\xint:,\xint:,\xint:,%
1396   \xint:,\xint:,\xint:,\xint:,\xint:,\xint_bye
1397 }%
1398 \def\XINT_nthelt:f:csv_pos_done #1{%
1399 \long\def\XINT_nthelt:f:csv_pos_done ##1,##2\xint_bye{%
1400   \xint_gob_til_xint:##1\XINT_nthelt:f:csv_pos_cleanup\xint:#1##1}%

```

```
1401 }\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...

```
1402 \xint_firstofone{\long\def\XINT_nthelt:f:csv_pos_cleanup\xint:} %
1403 #1\xint:{ #1}%
```

### 3.28.6 `\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.

```
1404 \def\xintReverse:f:csv {\romannumeral0\xintreverse:f:csv }%
1405 \long\def\xintreverse:f:csv #1%
1406 {%
1407   \expandafter\XINT_reverse:f:csv_loop
1408   \expandafter{\expandafter}\romannumeral`&&#1,%
1409   \xint:,%
1410   \xint_bye,\xint_bye,\xint_bye,\xint_bye,%
1411   \xint_bye,\xint_bye,\xint_bye,\xint_bye,%
1412   \xint:
1413 }%
1414 \long\def\XINT_reverse:f:csv_loop #1#2,#3,#4,#5,#6,#7,#8,#9,%
1415 {%
1416   \xint_bye #9\XINT_reverse:f:csv_cleanup\xint_bye
1417   \XINT_reverse:f:csv_loop {,#9,#8,#7,#6,#5,#4,#3,#2#1}%
1418 }%
1419 \long\def\XINT_reverse:f:csv_cleanup\xint_bye\XINT_reverse:f:csv_loop #1#2\xint:
1420 {%
1421   \XINT_reverse:f:csv_finish #1%
1422 }%
1423 \long\def\XINT_reverse:f:csv_finish #1\xint:,{ }%
```

### 3.28.7 `\xintFirstItem:f:csv`

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

```
1424 \def\xintFirstItem:f:csv {\romannumeral0\xintfirstitem:f:csv}%
1425 \long\def\xintfirstitem:f:csv #1%
1426 {%
1427   \expandafter\XINT_first:f:csv_a\romannumeral`&&#1,\xint_bye
1428 }%
1429 \long\def\XINT_first:f:csv_a #1,#2\xint_bye{ #1}%
```

### 3.28.8 `\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.

```

1430 \def\xintLastItem:f:csv {\romannumeral0\xintlastitem:f:csv}%
1431 \long\def\xintlastitem:f:csv #1%
1432 {%
1433   \expandafter\XINT_last:f:csv_loop\expandafter{\expandafter}\expandafter.%
1434   \romannumeral`&&#1,%
1435   \xint:\XINT_last_loop_enda,\xint:\XINT_last_loop_endb,%
1436   \xint:\XINT_last_loop_endc,\xint:\XINT_last_loop_endd,%
1437   \xint:\XINT_last_loop_ende,\xint:\XINT_last_loop_endf,%
1438   \xint:\XINT_last_loop_endg,\xint:\XINT_last_loop_endh,\xint_bye
1439 }%
1440 \long\def\XINT_last:f:csv_loop #1.#2,#3,#4,#5,#6,#7,#8,#9,%
1441 {%
1442   \xint_gob_til_xint: #9%
1443   {#8}{#7}{#6}{#5}{#4}{#3}{#2}{#1}\xint:
1444   \XINT_last:f:csv_loop {#9}.%
1445 }%

```

### 3.28.9 Public names for the undocumented csv macros: `\xintCSVLength`, `\xintCSVKeep`, `\xintCSVTrim`, `\xintCSVNthEltPy`, `\xintCSVReverse`, `\xintCSVFirstItem`, `\xintCSVLastItem`

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.

```

1446 \let\xintCSVLength   \xintLength:f:csv
1447 \let\xintCSVKeep    \xintKeep:f:csv
1448 \let\xintCSVTrim    \xintTrim:f:csv
1449 \let\xintCSVNthEltPy \xintNthEltPy:f:csv
1450 \let\xintCSVReverse  \xintReverse:f:csv
1451 \let\xintCSVFirstItem\xintFirstItem:f:csv
1452 \let\xintCSVLastItem \xintLastItem:f:csv
1453 \let\XINT_tmpa\relax \let\XINT_tmpb\relax \let\XINT_tmpc\relax
1454 \XINT_restorecatcodes_endinput%

```

## 4 Package [xintcore](#) implementation

.1	Catcodes, $\varepsilon$ -TeX and reload detection . . .	54	.25	<code>\XINT_zeroes_forviii</code> . . . . .	67	
.2	Package identification . . . . .	55	.26	<code>\XINT_sepbyviii_Z</code> . . . . .	67	
.3	(WIP!) Error conditions and exceptions . . .	55	.27	<code>\XINT_sepbyviii_andcount</code> . . . . .	68	
.4	Counts for holding needed constants . . .	57	.28	<code>\XINT_rsepbyviii</code> . . . . .	68	
Routines handling integers as lists of token digits			.29	<code>\XINT_sepandrev</code> . . . . .	69	
.5	<code>\XINT_cuz_small</code> . . . . .	58	.30	<code>\XINT_sepandrev_andcount</code> . . . . .	69	
.6	<code>\xintNum</code> , <code>\xintiNum</code> . . . . .	58	.31	<code>\XINT_rev_nounsep</code> . . . . .	70	
.7	<code>\xintiiSgn</code> . . . . .	59	.32	<code>\XINT_unrevbyviii</code> . . . . .	70	
.8	<code>\xintiiOpp</code> . . . . .	60	Core arithmetic . . . . .		70	
.9	<code>\xintiiAbs</code> . . . . .	60	.33	<code>\xintiiAdd</code> . . . . .	71	
.10	<code>\xintFDg</code> . . . . .	60	.34	<code>\xintiiCmp</code> . . . . .	74	
.11	<code>\xintLDg</code> . . . . .	61	.35	<code>\xintiiSub</code> . . . . .	76	
.12	<code>\xintDouble</code> . . . . .	61	.36	<code>\xintiiMul</code> . . . . .	82	
.13	<code>\xintHalf</code> . . . . .	62	.37	<code>\xintiiDivision</code> . . . . .	86	
.14	<code>\xintInc</code> . . . . .	62	Derived arithmetic . . . . .		101	
.15	<code>\xintDec</code> . . . . .	63	.38	<code>\xintiiQuo</code> , <code>\xintiiRem</code> . . . . .	101	
.16	<code>\xintDSL</code> . . . . .	63	.39	<code>\xintiiDivRound</code> . . . . .	101	
.17	<code>\xintDSR</code> . . . . .	63	.40	<code>\xintiiDivTrunc</code> . . . . .	102	
.18	<code>\xintDSRr</code> . . . . .	64	.41	<code>\xintiiModTrunc</code> . . . . .	102	
Blocks of eight digits . . . . .			64	.42	<code>\xintiiDivMod</code> . . . . .	103
.19	<code>\XINT_cuz</code> . . . . .	64	.43	<code>\xintiiDivFloor</code> . . . . .	104	
.20	<code>\XINT_cuz_byviii</code> . . . . .	65	.44	<code>\xintiiMod</code> . . . . .	104	
.21	<code>\XINT_unsep_loop</code> . . . . .	65	.45	<code>\xintiiSqr</code> . . . . .	104	
.22	<code>\XINT_unsep_cuzsmall</code> . . . . .	66	.46	<code>\xintiiPow</code> . . . . .	105	
.23	<code>\XINT_div_unsepQ</code> . . . . .	66	.47	<code>\xintiiFac</code> . . . . .	108	
.24	<code>\XINT_div_unsepR</code> . . . . .	67	.48	<code>\XINT_useiimessage</code> . . . . .	111	

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 1.2j brought again some improvements.

The commenting continues (2019/04/05) 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.

1.3 removes all macros which were deprecated at 1.2o.

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

The code for reload detection was initially copied from  $\text{\texttt{HEIKO OBERDIEK}}$ 's packages, then modified.

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

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2 \catcode13=5 % ^^M
3 \endlinechar=13 %
4 \catcode123=1 % {
5 \catcode125=2 % }
6 \catcode64=11 % @
7 \catcode35=6 % #
8 \catcode44=12 % ,
9 \catcode45=12 % -
10 \catcode46=12 % .
11 \catcode58=12 % :
```

```

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

```

## 4.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintcore}%
46 [2019/04/05 1.3e Expandable arithmetic on big integers (JFB)]%

```

## 4.3 (WIP!) Error conditions and exceptions

As per the Mike Cowlshaw/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.

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_stop_atfirstofone{#5}%
83 }%
84 }%
85%% \let\xintUse\xint_stop_atfirstofthree % 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 \catcode`. 11
103 \let\XINT_Clamped.handler\xint_firstofone % WIP
104 \def\XINT_InvalidOperation.handler#1{NaN}% WIP
105 \def\XINT_ConversionSyntax.handler#1{NaN}% WIP
106 \def\XINT_DivisionByZero.handler#1{SignedInfinity(#1)}% WIP
107 \def\XINT_DivisionImpossible.handler#1{NaN}% WIP
108 \def\XINT_DivisionUndefined.handler#1{NaN}% WIP
109 \let\XINT_Inexact.handler\xint_firstofone % WIP
110 \def\XINT_InvalidContext.handler#1{NaN}% WIP
111 \let\XINT_Rounded.handler\xint_firstofone % WIP
112 \let\XINT_Subnormal.handler\xint_firstofone% WIP
113 \def\XINT_Overflow.handler#1{NaN}% WIP
114 \def\XINT_Underflow.handler#1{NaN}% WIP
115 \catcode`. 12

```

#### 4.4 Counts for holding needed constants

```

116 \ifdefined\m@ne\let\xint_c_mone\m@ne
117 \else\csname newcount\endcsname\xint_c_mone \xint_c_mone -1 \fi
118 \ifdefined\xint_c_x^viii\else
119 \csname newcount\endcsname\xint_c_x^viii \xint_c_x^viii 100000000
120 \fi
121 \ifdefined\xint_c_x^ix\else

```

```

122 \csname newcount\endcsname\xint_c_x^ix \xint_c_x^ix 1000000000
123 \fi
124 \newcount\xint_c_x^viii_mone \xint_c_x^viii_mone 99999999
125 \newcount\xint_c_xii_e_viii \xint_c_xii_e_viii 1200000000
126 \newcount\xint_c_xi_e_viii_mone \xint_c_xi_e_viii_mone 1099999999

```

## 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.

### 4.5 \XINT\_cuz\_small

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

```

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

```

### 4.6 \xintNum, \xintiNum

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.

\xintNum gets redefined by [xintfrac](#).

```

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

```

```

147 \xint_gob_til_xint: #9\XINT_num_end\xint:
148 #1#2#3#4#5#6#7#8#9%
149 \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.

```

150 \expandafter\XINT_num_loop
151 \else

```

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

```

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

```

## 4.7 `\xintiiSgn`

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

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

```

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

```

## 4.8 `\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".

```

186 \def\xintiiOpp {\romannumeral0\xintiiopp }%
187 \def\xintiiopp #1%
188 {%
189   \expandafter\XINT_opp \romannumeral`&&@#1%
190 }%
191 \def\XINT_opp #1{\romannumeral0\XINT_opp #1}%
192 \def\XINT_opp #1%
193 {%
194   \xint_UDzerominusfork
195   #1- { 0}%      zero
196   0#1 { }%      negative
197   0- { -#1}%    positive
198   \krof
199 }%

```

## 4.9 `\xintiiAbs`

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

```

200 \def\xintiiAbs {\romannumeral0\xintiiabs }%
201 \def\xintiiabs #1%
202 {%
203   \expandafter\XINT_abs \romannumeral`&&@#1%
204 }%
205 \def\XINT_abs #1%
206 {%
207   \xint_UDsignfork
208   #1 { }%
209   - { #1}%
210   \krof
211 }%

```

## 4.10 `\xintFDg`

FIRST DIGIT.

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

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

```

212 \def\xintFDg {\romannumeral0\xintfdg }%
213 \def\xintfdg #1{\expandafter\XINT_fdg \romannumeral`&&@#1\xint:\Z}%
214 \def\XINT_FDg #1%
215   {\romannumeral0\expandafter\XINT_fdg\romannumeral`&&@\xintnum{#1}\xint:\Z }%
216 \def\XINT_fdg #1#2#3\Z
217 {%
218   \xint_UDzerominusfork
219   #1- { 0}%      zero
220   0#1 { #2}%    negative
221   0- { #1}%     positive

```



```
261 \def\XINT_dbl_e#1{* \xint_c_ii\if#13+\xint_c_i\fi\relax}%
```

### 4.13 \xintHalf

Attention \xintHalf non robust against non terminated input.

```
262 \def\xintHalf {\romannumeral0\xinthalf}%
263 \def\xinthalf #1{\expandafter\XINT_half_fork\romannumeral`&&@#1%
264   \xint_bye\xint_Bye345678\xint_bye
265   *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax}%
266 \def\XINT_half_fork #1%
267 {%
268   \xint_UDsignfork
269   #1\XINT_half_neg
270   -\XINT_half
271   \krof #1%
272 }%
273 \def\XINT_half_neg-{\xintiioopp\XINT_half}%
274 \def\XINT_half #1{%
275 \def\XINT_half ##1##2##3##4##5##6##7##8%
276   {\expandafter#1\the\numexpr(##1##2##3##4##5##6##7##8\XINT_half_a}%
277 }\XINT_half{ }%
278 \def\XINT_half_a#1{\xint_Bye#1\xint_bye\XINT_half_b#1}%
279 \def\XINT_half_b #1#2#3#4#5#6#7#8%
280   {\expandafter\XINT_half_e\the\numexpr(1#1#2#3#4#5#6#7#8\XINT_half_a}%
281 \def\XINT_half_e#1{* \xint_c_v+#1-\xint_c_v)\relax}%
```

### 4.14 \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.

```
282 \def\xintInc {\romannumeral0\xintinc}%
283 \def\xintinc #1{\expandafter\XINT_inc_fork\romannumeral`&&@#1%
284   \xint_bye23456789\xint_bye+\xint_c_i\relax}%
285 \def\XINT_inc_fork #1%
286 {%
287   \xint_UDsignfork
288   #1\XINT_inc_neg
289   -\XINT_inc
290   \krof #1%
291 }%
292 \def\XINT_inc_neg-#1\xint_bye#2\relax
293   {\xintiioopp\XINT_dec #1\XINT_dec_bye234567890\xint_bye}%
294 \def\XINT_inc #1{%
295 \def\XINT_inc ##1##2##3##4##5##6##7##8##9%
296   {\expandafter#1\the\numexpr##1##2##3##4##5##6##7##8##9\XINT_inc_a}%
297 }\XINT_inc{ }%
298 \def\XINT_inc_a #1#2#3#4#5#6#7#8#9%
299   {\expandafter\XINT_inc_e\the\numexpr 1#1#2#3#4#5#6#7#8#9\XINT_inc_a}%
300 \def\XINT_inc_e#1{\if#12+\xint_c_i\fi\relax}%
```

## 4.15 `\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.

```

301 \def\xintDec {\romannumeral0\xintdec}%
302 \def\xintdec #1{\expandafter\XINT_dec_fork\romannumeral`&&@#1%
303     \XINT_dec_bye234567890\xint_bye}%
304 \def\XINT_dec_fork #1%
305 {%
306     \xint_UDsignfork
307     #1\XINT_dec_neg
308     -\XINT_dec
309     \krof #1%
310 }%
311 \def\XINT_dec_neg-#1\XINT_dec_bye#2\xint_bye
312     {\expandafter-%
313     \romannumeral0\XINT_inc #1\xint_bye23456789\xint_bye+\xint_c_i\relax}%
314 \def\XINT_dec #1{%
315 \def\XINT_dec ##1##2##3##4##5##6##7##8##9%
316     {\expandafter#1\the\numexpr##1##2##3##4##5##6##7##8##9\XINT_dec_a}%
317 }\XINT_dec{ }%
318 \def\XINT_dec_a #1#2#3#4#5#6#7#8#9%
319     {\expandafter\XINT_dec_e\the\numexpr 1#1#2#3#4#5#6#7#8#9\XINT_dec_a}%
320 \def\XINT_dec_bye #1\XINT_dec_a#2#3\xint_bye
321     {\if#20-\xint_c_ii\relax+\else-\fi\xint_c_i\relax}%
322 \def\XINT_dec_e#1{\unless\if#11\xint_dothis{-\xint_c_i#1}\fi\xint_orthat\relax}%

```

## 4.16 `\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.

```

323 \def\xintDSL {\romannumeral0\xintdsl }%
324 \def\xintdsl #1{\expandafter\XINT_dsl\romannumeral`&&@#10}%
325 \def\XINT_dsl#1{%
326 \def\XINT_dsl ##1{\xint_gob_til_zero ##1\xint_dsl_zero 0#1##1}%
327 }\XINT_dsl{ }%
328 \def\xint_dsl_zero 0 0{ }%

```

## 4.17 `\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.

```

329 \def\xintDSR{\romannumeral0\xintdsr}%
330 \def\xintdsr #1{\expandafter\XINT_dsr_fork\romannumeral`&&@#1%
331     \xint_bye\xint_Bye3456789\xint_bye+\xint_c_v)/\xint_c_x-\xint_c_i\relax}%
332 \def\XINT_dsr_fork #1%
333 {%
334     \xint_UDsignfork

```

```

335     #1\XINT_dsr_neg
336     -\XINT_dsr
337     \krof #1%
338 }%
339 \def\XINT_dsr_neg-{\xintiopp\XINT_dsr}%
340 \def\XINT_dsr #1{%
341 \def\XINT_dsr ##1##2##3##4##5##6##7##8##9%
342   {\expandafter#1\the\numexpr(##1##2##3##4##5##6##7##8##9\XINT_dsr_a}%
343 }\XINT_dsr{ }%
344 \def\XINT_dsr_a#1{\xint_Bye#1\xint_bye\XINT_dsr_b#1}%
345 \def\XINT_dsr_b #1#2#3#4#5#6#7#8#9%
346   {\expandafter\XINT_dsr_e\the\numexpr(1#1#2#3#4#5#6#7#8#9\XINT_dsr_a}%
347 \def\XINT_dsr_e #1{)\relax}%

```

#### 4.18 \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.

```

348 \def\xintDSRr{\romannumeral0\xintdsrr}%
349 \def\xintdsrr #1{\expandafter\XINT_dsrr_fork\romannumeral`&&@#1%
350     \xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax}%
351 \def\XINT_dsrr_fork #1%
352 {%
353     \xint_UDsignfork
354     #1\XINT_dsrr_neg
355     -\XINT_dsrr
356     \krof #1%
357 }%
358 \def\XINT_dsrr_neg-{\xintiopp\XINT_dsrr}%
359 \def\XINT_dsrr #1{%
360 \def\XINT_dsrr ##1##2##3##4##5##6##7##8##9%
361   {\expandafter#1\the\numexpr##1##2##3##4##5##6##7##8##9\XINT_dsrr_a}%
362 }\XINT_dsrr{ }%
363 \def\XINT_dsrr_a#1{\xint_Bye#1\xint_bye\XINT_dsrr_b#1}%
364 \def\XINT_dsrr_b #1#2#3#4#5#6#7#8#9%
365   {\expandafter\XINT_dsrr_e\the\numexpr1#1#2#3#4#5#6#7#8#9\XINT_dsrr_a}%
366 \let\XINT_dsrr_e\XINT_inc_e

```

#### Blocks of eight digits

The lingua of release 1.2.

#### 4.19 \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.

```

367 \def\XINT_cuz #1{%
368 \def\XINT_cuz {\expandafter#1\the\numexpr\XINT_cuz_loop}%
369 }\XINT_cuz{ }%
370 \def\XINT_cuz_loop #1#2#3#4#5#6#7#8#9%
371 {%
372     #1#2#3#4#5#6#7#8%
373     \xint_gob_til_R #9\XINT_cuz_hitend\R
374     \ifnum #1#2#3#4#5#6#7#8>\xint_c_
375         \expandafter\XINT_cuz_cleantoend
376     \else\expandafter\XINT_cuz_loop
377     \fi #9%
378 }%
379 \def\XINT_cuz_hitend\R #1\R{\relax}%
380 \def\XINT_cuz_cleantoend #1\R{\relax #1}%

```

#### 4.20 \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

```

381 \def\XINT_cuz_byviii #1#2#3#4#5#6#7#8#9%
382 {%
383     \xint_gob_til_R #9\XINT_cuz_byviii_e \R
384     \xint_gob_til_eightzeroes #1#2#3#4#5#6#7#8\XINT_cuz_byviii_z 00000000%
385     \XINT_cuz_byviii_done #1#2#3#4#5#6#7#8#9%
386 }%
387 \def\XINT_cuz_byviii_z 00000000\XINT_cuz_byviii_done 00000000{\XINT_cuz_byviii}%
388 \def\XINT_cuz_byviii_done #1\R { #1}%
389 \def\XINT_cuz_byviii_e\R #1\XINT_cuz_byviii_done #2\R{ #2}%

```

#### 4.21 \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

```

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 1.21 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 1.21 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.

```

390 \def\XINT_unsep_loop #1!#2!#3!#4!#5!#6!#7!#8!#9!%
391 {%
392     \expandafter\XINT_unsep_clean
393     \the\numexpr #1\expandafter\XINT_unsep_clean

```

```

394 \the\numexpr #2\expandafter\XINT_unsep_clean
395 \the\numexpr #3\expandafter\XINT_unsep_clean
396 \the\numexpr #4\expandafter\XINT_unsep_clean
397 \the\numexpr #5\expandafter\XINT_unsep_clean
398 \the\numexpr #6\expandafter\XINT_unsep_clean
399 \the\numexpr #7\expandafter\XINT_unsep_clean
400 \the\numexpr #8\expandafter\XINT_unsep_clean
401 \the\numexpr #9\XINT_unsep_loop
402 }%
403 \def\XINT_unsep_clean 1{\relax}%

```

## 4.22 \XINT\_unsep\_cuzsmall

This is used as

```

\romannumeral0\XINT_unsep_cuzsmall (blocks of 1<8d>!)
\XINT_unsep_cuzsmall \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.

```

404 \def\XINT_unsep_cuzsmall
405 {%
406 \expandafter\XINT_unsep_cuzsmall_x\the\numexpr0\XINT_unsep_loop
407 }%
408 \def\XINT_unsep_cuzsmall_x #1{%
409 \def\XINT_unsep_cuzsmall_x 0##1##2##3##4##5##6##7##8%
410 {%
411 \expandafter#1\the\numexpr ##1##2##3##4##5##6##7##8\relax
412 }}\XINT_unsep_cuzsmall_x{ }%

```

## 4.23 \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.

```

413 \def\XINT_div_unsepQ_delim {\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax\Z}%
414 \def\XINT_div_unsepQ
415 {%
416 \expandafter\XINT_div_unsepQ_x\the\numexpr0\XINT_unsep_loop
417 }%
418 \def\XINT_div_unsepQ_x #1{%
419 \def\XINT_div_unsepQ_x 0##1##2##3##4##5##6##7##8##9%
420 {%
421 \xint_gob_til_Z ##9\XINT_div_unsepQ_one\Z
422 \xint_gob_til_eightzeroes ##1##2##3##4##5##6##7##8\XINT_div_unsepQ_y 00000000%
423 \expandafter#1\the\numexpr ##1##2##3##4##5##6##7##8\relax ##9%
424 }}\XINT_div_unsepQ_x{ }%
425 \def\XINT_div_unsepQ_y #1{%
426 \def\XINT_div_unsepQ_y ##1\relax ##2##3##4##5##6##7##8##9%
427 {%

```

```
428 \expandafter#1\the\numexpr ##2##3##4##5##6##7##8##9\relax
429 }}\XINT_div_unsepQ_y{ }%
430 \def\XINT_div_unsepQ_one#1\expandafter{\expandafter}%
```

#### 4.24 \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.

```
431 \def\XINT_div_unsepR
432 {%
433 \expandafter\XINT_div_unsepR_x\the\numexpr0\XINT_unsep_loop
434 }%
435 \def\XINT_div_unsepR_x#1{%
436 \def\XINT_div_unsepR_x_0{\expandafter#1\the\numexpr\XINT_cuz_loop}%
437 }\XINT_div_unsepR_x{ }%
```

#### 4.25 \XINT\_zeroes\_forviii

`\romannumeral0\XINT_zeroes_forviii #1\R\R\R\R\R\R\R\{10}0000001\W`

produces a string of  $k$  0's such that  $k + \text{length}(\#1)$  is smallest bigger multiple of eight.

```
438 \def\XINT_zeroes_forviii #1#2#3#4#5#6#7#8%
439 {%
440 \xint_gob_til_R #8\XINT_zeroes_forviii_end\R\XINT_zeroes_forviii
441 }%
442 \def\XINT_zeroes_forviii_end#1{%
443 \def\XINT_zeroes_forviii_end\R\XINT_zeroes_forviii ##1##2##3##4##5##6##7##8##9\W
444 }%
445 \expandafter#1\xint_gob_til_one ##2##3##4##5##6##7##8%
446 }}\XINT_zeroes_forviii_end{ }%
```

#### 4.26 \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>!1;!`

Prior to 1.2l 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.

```
447 \def\XINT_sepbyviii_Z #1#2#3#4#5#6#7#8%
448 {%
449 1#1#2#3#4#5#6#7#8\expandafter!\the\numexpr\XINT_sepbyviii_Z
450 }%
451 \def\XINT_sepbyviii_Z_end #1\relax {;!}%
```

## 4.27 \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.

```
452 \def\XINT_sepbyviii_andcount
453 {%
454   \expandafter\XINT_sepbyviii_andcount_a\the\numexpr\XINT_sepbyviii
455 }%
456 \def\XINT_sepbyviii #1#2#3#4#5#6#7#8%
457 {%
458   1#1#2#3#4#5#6#7#8\expandafter!\the\numexpr\XINT_sepbyviii
459 }%
460 \def\XINT_sepbyviii_end #1\relax {\relax\XINT_sepbyviii_andcount_end!}%
461 \def\XINT_sepbyviii_andcount_a {\XINT_sepbyviii_andcount_b \xint_c_\xint:}%
462 \def\XINT_sepbyviii_andcount_b #1\xint:#2!#3!#4!#5!#6!#7!#8!#9!%
463 {%
464   #2\expandafter!\the\numexpr#3\expandafter!\the\numexpr#4\expandafter
465   !\the\numexpr#5\expandafter!\the\numexpr#6\expandafter!\the\numexpr
466   #7\expandafter!\the\numexpr#8\expandafter!\the\numexpr#9\expandafter!\the\numexpr
467   \expandafter\XINT_sepbyviii_andcount_b\the\numexpr #1+\xint_c_viii\xint:%
468 }%
469 \def\XINT_sepbyviii_andcount_end #1\XINT_sepbyviii_andcount_b\the\numexpr
470   #2+\xint_c_viii\xint:#3#4\W {\expandafter\xint:\the\numexpr #2+#3\xint:}%
```

## 4.28 \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..1.2l) exclusively used in combination with \XINT\_sepandrev\_andcount or \XINT\_sepandrev.

```
471 \def\XINT_rsepbyviii #1#2#3#4#5#6#7#8%
472 {%
473   \XINT_rsepbyviii_b {#1#2#3#4#5#6#7#8}%
474 }%
475 \def\XINT_rsepbyviii_b #1#2#3#4#5#6#7#8#9%
476 {%
477   #2#3#4#5#6#7#8#9\expandafter!\the\numexpr
478   1#1\expandafter\xint:\the\numexpr 1\XINT_rsepbyviii
```

```
479 }%
480 \def\XINT_rsepbyviii_end_B #1\relax #2#3{#2\xint:}%
481 \def\XINT_rsepbyviii_end_A #1#2\expandafter #3\relax #4#5{#5!#2\xint:}%
```

#### 4.29 \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).

```
482 \def\XINT_sepandrev
483 {%
484   \expandafter\XINT_sepandrev_a\the\numexpr 1\XINT_rsepbyviii
485 }%
486 \def\XINT_sepandrev_a {\XINT_sepandrev_b {}}%
487 \def\XINT_sepandrev_b #1#2\xint:#3\xint:#4\xint:#5\xint:#6\xint:#7\xint:#8\xint:#9\xint:%
488 {%
489   \xint_gob_til_R #9\XINT_sepandrev_end\R
490   \XINT_sepandrev_b {#9!#8!#7!#6!#5!#4!#3!#2!#1}%
491 }%
492 \def\XINT_sepandrev_end\R\XINT_sepandrev_b #1#2\W {\XINT_sepandrev_done #1}%
493 \def\XINT_sepandrev_done #11#2!{ }%
```

#### 4.30 \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.

```
494 \def\XINT_sepandrev_andcount
495 {%
496   \expandafter\XINT_sepandrev_andcount_a\the\numexpr 1\XINT_rsepbyviii
497 }%
498 \def\XINT_sepandrev_andcount_a {\XINT_sepandrev_andcount_b 0!{}}%
499 \def\XINT_sepandrev_andcount_b #1!#2#3\xint:#4\xint:#5\xint:#6\xint:#7\xint:#8\xint:#9\xint:%
500 {%
501   \xint_gob_til_R #9\XINT_sepandrev_andcount_end\R
502   \expandafter\XINT_sepandrev_andcount_b \the\numexpr #1+\xint_c_i!%
503   {#9!#8!#7!#6!#5!#4!#3!#2}%
504 }%
```

```

505 \def\XINT_sepandrev_andcount_end\R
506   \expandafter\XINT_sepandrev_andcount_b\the\numexpr #1+\xint_c_i!#2#3#4\W
507 {\expandafter\XINT_sepandrev_andcount_done\the\numexpr #3+\xint_c_xiv*#1!#2}%
508 \def\XINT_sepandrev_andcount_done#1{%
509 \def\XINT_sepandrev_andcount_done##1!##21##3!{\expandafter#1\the\numexpr##1-##3\xint:}%
510 }\XINT_sepandrev_andcount_done{ }%

```

### 4.31 \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.

```

511 \def\XINT_rev_nounsep #1#2!#3!#4!#5!#6!#7!#8!#9!%
512 {%
513   \xint_gob_til_R #9\XINT_rev_nounsep_end\R
514   \XINT_rev_nounsep {#9!#8!#7!#6!#5!#4!#3!#2!#1}%
515 }%
516 \def\XINT_rev_nounsep_end\R\XINT_rev_nounsep #1#2\W {\XINT_rev_nounsep_done #1}%
517 \def\XINT_rev_nounsep_done #11{ 1}%

```

### 4.32 \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!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).

```

518 \def\XINT_unrevbyviii #11#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
519 {%
520   \xint_gob_til_R #9\XINT_unrevbyviii_a\R
521   \XINT_unrevbyviii {#9#8#7#6#5#4#3#2#1}%
522 }%
523 \def\XINT_unrevbyviii_a#1{%
524 \def\XINT_unrevbyviii_a\R\XINT_unrevbyviii ##1##2\W
525   {\expandafter#1\romannumeral`&&\xint_gob_til_sc ##1}%
526 }\XINT_unrevbyviii_a{ }%

```

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

```

527 \def\XINT_smallunrevbyviii 1#1!1#2!1#3!1#4!1#5!1#6!1#7!1#8!#9\W%
528 {%
529   \expandafter\XINT_cuz_small\xint_gob_til_sc #8#7#6#5#4#3#2#1%
530 }%

```

## 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.

### 4.33 `\xintiiAdd`

1.21: `\xintiiAdd` made robust against non terminated input.

```

531 \def\xintiiAdd    {\romannumeral0\xintiiadd }%
532 \def\xintiiadd #1{\expandafter\XINT_iiadd\romannumeral`&&@#1\xint:}%
533 \def\XINT_iiadd #1#2\xint:#3%
534 {%
535   \expandafter\XINT_add_nfork\expandafter#1\romannumeral`&&@#3\xint:#2\xint:
536 }%
537 \def\XINT_iadd #1#2\xint:#3%
538 {%
539   \expandafter\XINT_add_nfork\expandafter
540   #1\romannumeral0\xintnum{#3}\xint:#2\xint:
541 }%
542 \def\XINT_add_fork #1#2\xint:#3\xint: {\XINT_add_nfork #1#3\xint:#2\xint:}%
543 \def\XINT_add_nfork #1#2%
544 {%
545   \xint_UDzerofork
546   #1\XINT_add_firstiszero
547   #2\XINT_add_secondiszero
548   0}%
549   \krof
550   \xint_UDsignsfork
551   #1#2\XINT_add_minusminus
552   #1-\XINT_add_minusplus
553   #2-\XINT_add_plusminus
554   --\XINT_add_plusplus
555   \krof #1#2%
556 }%
557 \def\XINT_add_firstiszero #1\krof 0#2#3\xint:#4\xint:{ #2#3}%
558 \def\XINT_add_secondiszero #1\krof #20#3\xint:#4\xint:{ #2#4}%
559 \def\XINT_add_minusminus #1#2%
560   {\expandafter-\romannumeral0\XINT_add_pp_a {}}%
561 \def\XINT_add_minusplus #1#2{\XINT_sub_mm_a {}}#2%
562 \def\XINT_add_plusminus #1#2%
563   {\expandafter\XINT_opp\romannumeral0\XINT_sub_mm_a #1}%
564 \def\XINT_add_pp_a #1#2#3\xint:
565 {%
566   \expandafter\XINT_add_pp_b
567   \romannumeral0\expandafter\XINT_sepandrev_andcount
568   \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
569   #2#3\XINT_rsepbyviii_end_A 2345678%
570   \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
571   \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi

```

```

572          \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
573  \X #1%
574 }%
575 \let\XINT_add_plusplus \XINT_add_pp_a
576 \def\XINT_add_pp_b #1\xint:#2\X #3\xint:
577 {%
578   \expandafter\XINT_add_checklengths
579   \the\numexpr #1\expandafter\xint:%
580   \romannumeral0\expandafter\XINT_sepandrev_andcount
581   \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\{10}0000001\W
582   #3\XINT_rsepyviii_end_A 2345678%
583   \XINT_rsepyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
584   \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
585   \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
586   1;!1;!1;!1;!1\W #21;!1;!1;!1;!1\W
587   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\W
588 }%

```

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

```

589 \def\XINT_add_checklengths #1\xint:#2\xint:%
590 {%
591   \ifnum #2>#1
592     \expandafter\XINT_add_exchange
593   \else
594     \expandafter\XINT_add_A
595   \fi
596   #1\xint:#2\xint:%
597 }%
598 \def\XINT_add_exchange #1\xint:#2\xint:#3\W #4\W
599 {%
600   \XINT_add_A #2\xint:#1\xint:#4\W #3\W
601 }%
602 \def\XINT_add_A #1\xint:#2\xint:%
603 {%
604   \ifnum #1>\xint_c_vi
605     \expandafter\XINT_add_aa
606   \else \expandafter\XINT_add_aa_small
607   \fi
608 }%
609 \def\XINT_add_aa {\expandafter\XINT_add_out\the\numexpr\XINT_add_a \xint_c_ii}%
610 \def\XINT_add_out{\expandafter\XINT_cuz_small\romannumeral0\XINT_unrevbyviii {}}%
611 \def\XINT_add_aa_small
612   {\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).



```

613 \def\XINT_add_a #1!#2!#3!#4!#5\W
614           #6!#7!#8!#9!%
615 {%
616   \XINT_add_b
617     #1!#6!#2!#7!#3!#8!#4!#9!%
618     #5\W
619 }%
620 \def\XINT_add_b #1!#2#3!#4!%
621 {%
622   \xint_gob_til_sc #2\XINT_add_bi ;%
623   \expandafter\XINT_add_c\the\numexpr#1+1#2#3+#4-\xint_c_ii\xint:%
624 }%
625 \def\XINT_add_bi;\expandafter\XINT_add_c
626   \the\numexpr#1+#2+#3-\xint_c_ii\xint:#4!#5!#6!#7!#8!#9!\W
627 {%
628   \XINT_add_k #1#3!#5!#7!#9!%
629 }%
630 \def\XINT_add_c #1#2\xint:%
631 {%
632   1#2\expandafter!\the\numexpr\XINT_add_d #1%
633 }%
634 \def\XINT_add_d #1!#2#3!#4!%
635 {%
636   \xint_gob_til_sc #2\XINT_add_di ;%
637   \expandafter\XINT_add_e\the\numexpr#1+1#2#3+#4-\xint_c_ii\xint:%
638 }%
639 \def\XINT_add_di;\expandafter\XINT_add_e
640   \the\numexpr#1+#2+#3-\xint_c_ii\xint:#4!#5!#6!#7!#8\W
641 {%
642   \XINT_add_k #1#3!#5!#7!%
643 }%
644 \def\XINT_add_e #1#2\xint:%
645 {%
646   1#2\expandafter!\the\numexpr\XINT_add_f #1%
647 }%
648 \def\XINT_add_f #1!#2#3!#4!%
649 {%
650   \xint_gob_til_sc #2\XINT_add_fi ;%
651   \expandafter\XINT_add_g\the\numexpr#1+1#2#3+#4-\xint_c_ii\xint:%
652 }%
653 \def\XINT_add_fi;\expandafter\XINT_add_g
654   \the\numexpr#1+#2+#3-\xint_c_ii\xint:#4!#5!#6\W
655 {%
656   \XINT_add_k #1#3!#5!%
657 }%
658 \def\XINT_add_g #1#2\xint:%
659 {%
660   1#2\expandafter!\the\numexpr\XINT_add_h #1%
661 }%
662 \def\XINT_add_h #1!#2#3!#4!%
663 {%
664   \xint_gob_til_sc #2\XINT_add_hi ;%

```

```

665 \expandafter\XINT_add_i\the\numexpr#1+1#2#3+#4-\xint_c_ii\xint:%
666 }%
667 \def\XINT_add_hi;%
668 \expandafter\XINT_add_i\the\numexpr#1+#2+#3-\xint_c_ii\xint:#4\W
669 {%
670 \XINT_add_k #1#3!%
671 }%
672 \def\XINT_add_i #1#2\xint:%
673 {%
674 1#2\expandafter!\the\numexpr\XINT_add_a #1%
675 }%

676 \def\XINT_add_k #1{\if #12\expandafter\XINT_add_ke\else\expandafter\XINT_add_l \fi}%
677 \def\XINT_add_ke #11;#2\W {\XINT_add_kf #11;!}%
678 \def\XINT_add_kf 1{1\relax }%
679 \def\XINT_add_l 1#1#2{\xint_gob_til_sc #1\XINT_add_lf ;\XINT_add_m 1#1#2}%
680 \def\XINT_add_lf #1\W {1\relax 00000001!1;!}%
681 \def\XINT_add_m #1!\{\expandafter\XINT_add_n\the\numexpr\xint_c_i+#1\xint:%}
682 \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>.)

```

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

```

#### 4.34 \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.

```

684 \def\xintiiCmp {\romannumeral0\xintiicmp }%
685 \def\xintiicmp #1\expandafter\XINT_iicmp\romannumeral`&&@#1\xint:%}
686 \def\XINT_iicmp #1#2\xint:#3%
687 {%
688 \expandafter\XINT_cmp_nfork\expandafter #1\romannumeral`&&@#3\xint:#2\xint:
689 }%
690 \def\XINT_icmp #1#2\xint:#3%
691 {%
692 \expandafter\XINT_cmp_nfork\expandafter #1\romannumeral0\xintnum{#3}\xint:#2\xint:
693 }%
694 \def\XINT_cmp_nfork #1#2%
695 {%
696 \xint_UDzerofork
697 #1\XINT_cmp_firstiszero
698 #2\XINT_cmp_secondiszero
699 0{ }%
700 \krof
701 \xint_UDsignsfork
702 #1#2\XINT_cmp_minusminus
703 #1-\XINT_cmp_minusplus
704 #2-\XINT_cmp_plusminus
705 --\XINT_cmp_plusplus
706 \krof #1#2%
707 }%

```

```

708 \def\XINT_cmp_firstiszero #1\krof 0#2#3\xint:#4\xint:
709 {%
710   \xint_UDzerominusfork
711   #2-{\ 0}%
712   0#2{ 1}%
713   0-{\ -1}%
714   \krof
715 }%
716 \def\XINT_cmp_secondiszero #1\krof #20#3\xint:#4\xint:
717 {%
718   \xint_UDzerominusfork
719   #2-{\ 0}%
720   0#2{ -1}%
721   0-{\ 1}%
722   \krof
723 }%
724 \def\XINT_cmp_plusminus #1\xint:#2\xint:{ 1}%
725 \def\XINT_cmp_minusplus #1\xint:#2\xint:{ -1}%
726 \def\XINT_cmp_minusminus
727   --{\expandafter\XINT_opp\romannumeral0\XINT_cmp_plusplus {}{}}%
728 \def\XINT_cmp_plusplus #1#2#3\xint:
729 {%
730   \expandafter\XINT_cmp_pp
731   \the\numexpr\expandafter\XINT_sepbyviii_andcount
732   \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
733   #2#3\XINT_sepbyviii_end 2345678\relax
734   \xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!%
735   \xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_\W
736   #1%
737 }%
738 \def\XINT_cmp_pp #1\xint:#2\xint:#3\xint:
739 {%
740   \expandafter\XINT_cmp_checklengths
741   \the\numexpr #2\expandafter\xint:%
742   \the\numexpr\expandafter\XINT_sepbyviii_andcount
743   \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
744   #3\XINT_sepbyviii_end 2345678\relax
745   \xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!%
746   \xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_\W
747   #1;!1;!1;!1;!1;\W
748 }%
749 \def\XINT_cmp_checklengths #1\xint:#2\xint:#3\xint:
750 {%
751   \ifnum #1=#3
752     \expandafter\xint_firstoftwo
753   \else
754     \expandafter\xint_secondoftwo
755   \fi
756   \XINT_cmp_a {\XINT_cmp_distinctlengths {#1}{#3}}#2;!1;!1;!1;\W
757 }%
758 \def\XINT_cmp_distinctlengths #1#2#3\W #4\W
759 {%

```

```

760 \ifnum #1>#2
761 \expandafter\xint_firstoftwo
762 \else
763 \expandafter\xint_secondoftwo
764 \fi
765 { -1}{ 1}%
766 }%
767 \def\xINT_cmp_a 1#1!1#2!1#3!1#4!#5\W 1#6!1#7!1#8!1#9!%
768 {%
769 \xint_gob_til_sc #1\xINT_cmp_equal ;%
770 \ifnum #1>#6 \XINT_cmp_gt\fi
771 \ifnum #1<#6 \XINT_cmp_lt\fi
772 \xint_gob_til_sc #2\xINT_cmp_equal ;%
773 \ifnum #2>#7 \XINT_cmp_gt\fi
774 \ifnum #2<#7 \XINT_cmp_lt\fi
775 \xint_gob_til_sc #3\xINT_cmp_equal ;%
776 \ifnum #3>#8 \XINT_cmp_gt\fi
777 \ifnum #3<#8 \XINT_cmp_lt\fi
778 \xint_gob_til_sc #4\xINT_cmp_equal ;%
779 \ifnum #4>#9 \XINT_cmp_gt\fi
780 \ifnum #4<#9 \XINT_cmp_lt\fi
781 \XINT_cmp_a #5\W
782 }%
783 \def\xINT_cmp_lt#1{\def\xINT_cmp_lt\fi ##1\W ##2\W {\fi#1-1}}\XINT_cmp_lt{ }%
784 \def\xINT_cmp_gt#1{\def\xINT_cmp_gt\fi ##1\W ##2\W {\fi#11}}\XINT_cmp_gt{ }%
785 \def\xINT_cmp_equal #1\W #2\W { 0}%

```

#### 4.35 \xintiiSub

Entirely rewritten for 1.2.

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

```

786 \def\xintiiSub {\romannumeral0\xintiiSub }%
787 \def\xintiiSub #1{\expandafter\xINT_iisub\romannumeral`&&#1\xint:}%
788 \def\xINT_iisub #1#2\xint:#3%
789 {%
790 \expandafter\xINT_sub_nfork\expandafter

```

```

791 #1\romannumeral`&&@#3\xint:#2\xint:
792 }%
793 \def\XINT_isub #1#2\xint:#3%
794 {%
795 \expandafter\XINT_sub_nfork\expandafter
796 #1\romannumeral0\xintnum{#3}\xint:#2\xint:
797 }%
798 \def\XINT_sub_nfork #1#2%
799 {%
800 \xint_UDzerofork
801 #1\XINT_sub_firstiszero
802 #2\XINT_sub_secondiszero
803 0{}}%
804 \krof
805 \xint_UDsignsfork
806 #1#2\XINT_sub_minusminus
807 #1-\XINT_sub_minusplus
808 #2-\XINT_sub_plusminus
809 --\XINT_sub_plusplus
810 \krof #1#2%
811 }%
812 \def\XINT_sub_firstiszero #1\krof 0#2#3\xint:#4\xint:{\XINT_opp #2#3}%
813 \def\XINT_sub_secondiszero #1\krof #20#3\xint:#4\xint:{ #2#4}%
814 \def\XINT_sub_plusminus #1#2{\XINT_add_pp_a #1{}}%
815 \def\XINT_sub_plusplus #1#2%
816 {\expandafter\XINT_opp\romannumeral0\XINT_sub_mm_a #1#2}%
817 \def\XINT_sub_minusplus #1#2%
818 {\expandafter-\romannumeral0\XINT_add_pp_a { }#2}%
819 \def\XINT_sub_minusminus #1#2{\XINT_sub_mm_a { }{}}%
820 \def\XINT_sub_mm_a #1#2#3\xint:
821 {%
822 \expandafter\XINT_sub_mm_b
823 \romannumeral0\expandafter\XINT_sepandrev_andcount
824 \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
825 #2#3\XINT_rsepbyviii_end_A 2345678%
826 \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
827 \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
828 \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
829 \X #1%
830 }%
831 \def\XINT_sub_mm_b #1\xint:#2\X #3\xint:
832 {%
833 \expandafter\XINT_sub_checklengths
834 \the\numexpr #1\expandafter\xint:%
835 \romannumeral0\expandafter\XINT_sepandrev_andcount
836 \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
837 #3\XINT_rsepbyviii_end_A 2345678%
838 \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
839 \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
840 \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
841 1;!1;!1;!1;!1\W
842 #21;!1;!1;!1;!1\W

```

```

843 1;!1\R!1\R!1\R!1\R!%
844 1\R!1\R!1\R!1\R!\W
845 }%
846 \def\XINT_sub_checklengths #1\xint:#2\xint:%
847 {%
848 \ifnum #2>#1
849 \expandafter\XINT_sub_exchange
850 \else
851 \expandafter\XINT_sub_aa
852 \fi
853 }%
854 \def\XINT_sub_exchange #1\W #2\W
855 {%
856 \expandafter\XINT_opp\romannumeral0\XINT_sub_aa #2\W #1\W
857 }%
858 \def\XINT_sub_aa
859 {%
860 \expandafter\XINT_sub_out\the\numexpr\XINT_sub_a\xint_c_i
861 }%

```

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`.

```
862 \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

```

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.

```
863 \def\XINT_sub_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
864 {%
865   \XINT_sub_b
866   #1!#6!#2!#7!#3!#8!#4!#9!%
867   #5\W
868 }%
```

As 1.2l code uses `1<8digits>!` blocks one has to be careful with the carry digit 1 or 0: A `#11#2#3` pattern would result into an empty `#1` if the carry digit which is upfront is 1, rather than setting `#1=1`.

```
869 \def\XINT_sub_b #1#2#3#4!#5!%
870 {%
871   \xint_gob_til_sc #3\XINT_sub_bi ;%
872   \expandafter\XINT_sub_c\the\numexpr#1+1#5-#3#4-\xint_c_i\xint:%
873 }%
874 \def\XINT_sub_c 1#1#2\xint:%
875 {%
876   1#2\expandafter!\the\numexpr\XINT_sub_d #1%
877 }%
878 \def\XINT_sub_d #1#2#3#4!#5!%
879 {%
880   \xint_gob_til_sc #3\XINT_sub_di ;%
881   \expandafter\XINT_sub_e\the\numexpr#1+1#5-#3#4-\xint_c_i\xint:%
882 }%
883 \def\XINT_sub_e 1#1#2\xint:%
884 {%
885   1#2\expandafter!\the\numexpr\XINT_sub_f #1%
886 }%
887 \def\XINT_sub_f #1#2#3#4!#5!%
888 {%
889   \xint_gob_til_sc #3\XINT_sub_fi ;%
890   \expandafter\XINT_sub_g\the\numexpr#1+1#5-#3#4-\xint_c_i\xint:%
891 }%
892 \def\XINT_sub_g 1#1#2\xint:%
893 {%
894   1#2\expandafter!\the\numexpr\XINT_sub_h #1%
895 }%
896 \def\XINT_sub_h #1#2#3#4!#5!%
897 {%
898   \xint_gob_til_sc #3\XINT_sub_hi ;%
899   \expandafter\XINT_sub_i\the\numexpr#1+1#5-#3#4-\xint_c_i\xint:%
900 }%
901 \def\XINT_sub_i 1#1#2\xint:%
902 {%
903   1#2\expandafter!\the\numexpr\XINT_sub_a #1%
904 }%
905 \def\XINT_sub_bi;%
906   \expandafter\XINT_sub_c\the\numexpr#1+1#2-#3\xint:
907   #4!#5!#6!#7!#8!#9!\W
```

```

908 {%
909   \XINT_sub_k #1#2!#5!#7!#9!%
910 }%
911 \def\XINT_sub_di;%
912   \expandafter\XINT_sub_e\the\numexpr#1+1#2-#3\xint:
913   #4!#5!#6!#7!#8\W
914 {%
915   \XINT_sub_k #1#2!#5!#7!%
916 }%
917 \def\XINT_sub_fi;%
918   \expandafter\XINT_sub_g\the\numexpr#1+1#2-#3\xint:
919   #4!#5!#6\W
920 {%
921   \XINT_sub_k #1#2!#5!%
922 }%
923 \def\XINT_sub_hi;%
924   \expandafter\XINT_sub_i\the\numexpr#1+1#2-#3\xint:
925   #4\W
926 {%
927   \XINT_sub_k #1#2!%
928 }%

```

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.

```

929 \def\XINT_sub_k #1#2#3%
930 {%
931   \xint_gob_til_sc #3\XINT_sub_p;\XINT_sub_l #1#2#3%
932 }%
933 \def\XINT_sub_l #1%
934   {\xint_UDzerofork #1\XINT_sub_l_carry 0\XINT_sub_l_Ia\krof}%
935 \def\XINT_sub_l_Ia 1#1;!#2\W{1\relax#1;!1\XINT_sub_fix_none!}%

936 \def\XINT_sub_l_carry 1#1!{\ifcase #1
937   \expandafter \XINT_sub_l_Id
938   \or \expandafter \XINT_sub_l_Ic
939   \else\expandafter \XINT_sub_l_Ib\fi 1#1!}%
940 \def\XINT_sub_l_Ib #1;!#2\W {-\xint_c_i+#1;!1\XINT_sub_fix_none!}%
941 \def\XINT_sub_l_Ic 1#1!1#2#3!#4;!#5\W
942 {%
943   \xint_gob_til_sc #2\XINT_sub_l_Ica;%
944   1\relax 00000000!1#2#3!#4;!1\XINT_sub_fix_none!%
945 }%

```

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`

```

946 \def\XINT_sub_l_Ica#1\W

```



```

947 {%
948     1;!1\XINT_sub_fix_cuz!%
949     1;!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
950     \xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z
951 }%
952 \def\XINT_sub_l_Id 1#1!%
953     {199999999\expandafter!\the\numexpr \XINT_sub_l_Id_a}%
954 \def\XINT_sub_l_Id_a 1#1!{\ifcase #1
955     \expandafter \XINT_sub_l_Id
956     \or \expandafter \XINT_sub_l_Id_b
957     \else\expandafter \XINT_sub_l_Ib\fi 1#1!}%
958 \def\XINT_sub_l_Id_b 1#1!#2#3!#4;#5\W
959 {%
960     \xint_gob_til_sc #2\XINT_sub_l_Ida;%
961     1\relax 00000000!1#2#3!#4;!1\XINT_sub_fix_none!%
962 }%
963 \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.

```

964 \def\XINT_sub_p;\XINT_sub_l #1#2\W #3\W
965 {%
966     \xint_UDzerofork
967     #1{1;!1\XINT_sub_fix_neg!%
968     1;!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
969     \xint_bye2345678\xint_bye1099999988\relax}% A - B, B > A
970     0{1;!1\XINT_sub_fix_cuz!%
971     1;!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W}%
972     \krof
973     \xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z
974 }%

```

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.

```

975 \def\XINT_sub_fix_none;{\XINT_cuz_small}%
976 \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.

```

977 \def\XINT_sub_fix_neg;%
978 {%
979     \expandafter-\romannumeral0\expandafter
980     \XINT_sub_comp_finish\the\numexpr\XINT_sub_comp_loop
981 }%
982 \def\XINT_sub_comp_finish 0{\XINT_sub_fix_cuz;}%
983 \def\XINT_sub_comp_loop #1#2#3#4#5#6#7#8%
984 {%

```

```

985 \expandafter\XINT_sub_comp_clean
986 \the\numexpr \xint_c_xi_e_viii_mone-#1#2#3#4#5#6#7#8\XINT_sub_comp_loop
987 }%

```

#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.

```

988 \def\XINT_sub_comp_clean #1{+#1\relax}%

```

### 4.36 \xintiiMul

Completely rewritten for 1.2.

1.21: \xintiiMul made robust against non terminated input.

```

989 \def\xintiiMul {\romannumeral0\xintiimul }%
990 \def\xintiimul #1%
991 {%
992 \expandafter\XINT_iimul\romannumeral`&&@#1\xint:
993 }%
994 \def\XINT_iimul #1#2\xint:#3%
995 {%
996 \expandafter\XINT_mul_nfork\expandafter #1\romannumeral`&&@#3\xint:#2\xint:
997 }%

```

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

```

998 \def\XINT_mul_fork #1#2\xint:#3\xint: {\XINT_mul_nfork #1#3\xint:#2\xint:}%
999 \def\XINT_mul_nfork #1#2%
1000 {%
1001 \xint_UDzerofork
1002 #1\XINT_mul_zero
1003 #2\XINT_mul_zero
1004 0}%
1005 \krof
1006 \xint_UDsignsfork
1007 #1#2\XINT_mul_minusminus
1008 #1-\XINT_mul_minusplus
1009 #2-\XINT_mul_plusminus
1010 --\XINT_mul_plusplus
1011 \krof #1#2%
1012 }%
1013 \def\XINT_mul_zero #1\krof #2#3\xint:#4\xint:{ 0}%
1014 \def\XINT_mul_minusminus #1#2{\XINT_mul_plusplus {}{}}%
1015 \def\XINT_mul_minusplus #1#2%
1016 {\expandafter-\romannumeral0\XINT_mul_plusplus {}#2}%
1017 \def\XINT_mul_plusminus #1#2%
1018 {\expandafter-\romannumeral0\XINT_mul_plusplus #1{}}%
1019 \def\XINT_mul_plusplus #1#2#3\xint:
1020 {%
1021 \expandafter\XINT_mul_pre_b
1022 \romannumeral0\expandafter\XINT_sepandrev_andcount
1023 \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
1024 #2#3\XINT_rsepbyviii_end_A 2345678%
1025 \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i

```

```

1026          \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
1027          \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
1028 \W #1%
1029 }%
1030 \def\XINT_mul_pre_b #1\xint:#2\W #3\xint:
1031 {%
1032   \expandafter\XINT_mul_checklengths
1033   \the\numexpr #1\expandafter\xint:%
1034   \romannumeral0\expandafter\XINT_sepandrev_andcount
1035   \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
1036   #3\XINT_rsepyviii_end_A 2345678%
1037   \XINT_rsepyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
1038           \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
1039           \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
1040   1;! \W #21;!%
1041   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\W
1042 }%

```

Cooking recipe, 2015/10/05.

```

1043 \def\XINT_mul_checklengths #1\xint:#2\xint:%
1044 {%
1045   \ifnum #2=\xint_c_i\expandafter\XINT_mul_smallbyfirst\fi
1046   \ifnum #1=\xint_c_i\expandafter\XINT_mul_smallbysecond\fi
1047   \ifnum #2<#1
1048     \ifnum \numexpr (#2-\xint_c_i)*(#1-#2)<383
1049       \XINT_mul_exchange
1050     \fi
1051   \else
1052     \ifnum \numexpr (#1-\xint_c_i)*(#2-#1)>383
1053       \XINT_mul_exchange
1054     \fi
1055   \fi
1056   \XINT_mul_start
1057 }%
1058 \def\XINT_mul_smallbyfirst #1\XINT_mul_start 1#2!1;! \W
1059 {%
1060   \ifnum#2=\xint_c_i\expandafter\XINT_mul_oneisone\fi
1061   \ifnum#2<\xint_c_xxii\expandafter\XINT_mul_verysmall\fi
1062   \expandafter\XINT_mul_out\the\numexpr\XINT_smallmul 1#2!%
1063 }%
1064 \def\XINT_mul_smallbysecond #1\XINT_mul_start #2\W 1#3!1;!%
1065 {%
1066   \ifnum#3=\xint_c_i\expandafter\XINT_mul_oneisone\fi
1067   \ifnum#3<\xint_c_xxii\expandafter\XINT_mul_verysmall\fi
1068   \expandafter\XINT_mul_out\the\numexpr\XINT_smallmul 1#3!#2%
1069 }%
1070 \def\XINT_mul_oneisone #1!{\XINT_mul_out }%
1071 \def\XINT_mul_verysmall\expandafter\XINT_mul_out
1072           \the\numexpr\XINT_smallmul 1#1!%
1073   {\expandafter\XINT_mul_out\the\numexpr\XINT_verysmallmul 0\xint:#1!}%
1074 \def\XINT_mul_exchange #1\XINT_mul_start #2\W #31;!%
1075   {\fi\fi\XINT_mul_start #31;! \W #2}%

```

```

1076 \def\XINT_mul_start
1077   {\expandafter\XINT_mul_out\the\numexpr\XINT_mul_loop 100000000!1;!\W}%
1078 \def\XINT_mul_out
1079   {\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  $1\langle 8d\rangle!$ . 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:  $1\langle 8d\rangle!\dots 1\langle 8d\rangle!1;!$ . 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.

```

1080 \def\XINT_mul_loop #1\W #2\W 1#3!%
1081 {%
1082   \xint_gob_til_sc #3\XINT_mul_e ;%
1083   \expandafter\XINT_mul_a\the\numexpr \XINT_smallmul 1#3!#2\W
1084   #1\W #2\W
1085 }%

```

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

```

1086 \def\XINT_mul_a #1\W #2\W
1087 {%
1088   \expandafter\XINT_mul_b\the\numexpr
1089   \XINT_add_a \xint_c_ii #21;!1;!1;!\W #11;!1;!1;!\W\W
1090 }%
1091 \def\XINT_mul_b 1#1!{1#1\expandafter!\the\numexpr\XINT_mul_loop }%
1092 \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  $1\langle 8\text{digits carry}\rangle.\langle 4\text{ high digits}\rangle.\langle 4\text{ low digits}\rangle\langle 8\text{digits}\rangle$ .

It produces a block  $1\langle 8d\rangle!$  and then jump back into `\XINT_smallmul_a` with the new 8digits carry as argument. The `\XINT_smallmul_a` fetches a new  $1\langle 8d\rangle!$  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.

```

1093 \def\XINT_minimulwc_a 1#1\xint:#2\xint:#3!#4#5#6#7#8\xint:%
1094 {%
1095   \expandafter\XINT_minimulwc_b
1096   \the\numexpr \xint_c_x^ix+#1+#3*#8\xint:
1097               #3*#4#5#6#7+#2*#8\xint:
1098               #2*#4#5#6#7\xint:%
1099 }%
1100 \def\XINT_minimulwc_b 1#1#2#3#4#5#6\xint:#7\xint:%
1101 {%
1102   \expandafter\XINT_minimulwc_c
1103   \the\numexpr \xint_c_x^ix+#1#2#3#4#5+#7\xint:#6\xint:%

```

```

1104 }%
1105 \def\XINT_minimulwc_c #1#2#3#4#5#6\xint:#7\xint:#8\xint:%
1106 {%
1107     1#6#7\expandafter!%
1108     \the\numexpr\expandafter\XINT_smallmul_a
1109     \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8\xint:%
1110 }%
1111 \def\XINT_smallmul #1#2#3#4#5!{\XINT_smallmul_a 100000000\xint:#1#2#3#4\xint:#5!}%
1112 \def\XINT_smallmul_a #1\xint:#2\xint:#3!#4!%
1113 {%
1114     \xint_gob_til_sc #4\XINT_smallmul_e;%
1115     \XINT_minimulwc_a #1\xint:#2\xint:#3!#4\xint:#2\xint:#3!%
1116 }%
1117 \def\XINT_smallmul_e;\XINT_minimulwc_a #1#2#3!%
1118     {\xint_gob_til_eightzeroes #1\XINT_smallmul_f 000000001\relax #1!1;!}%
1119 \def\XINT_smallmul_f 000000001\relax 00000000!1{1\relax}%

```

```

1120 \def\XINT_verysmallmul #1\xint:#2!#3!%
1121 {%
1122     \xint_gob_til_sc #3\XINT_verysmallmul_e;%
1123     \expandafter\XINT_verysmallmul_a
1124     \the\numexpr #2*#3+#1\xint:#2!%
1125 }%
1126 \def\XINT_verysmallmul_e;\expandafter\XINT_verysmallmul_a\the\numexpr
1127     #1+#2#3\xint:#4!%
1128 {\xint_gob_til_zero #2\XINT_verysmallmul_f 0\xint_c_x^viii+#2#3!1;!}%
1129 \def\XINT_verysmallmul_f #1!1{1\relax}%
1130 \def\XINT_verysmallmul_a #1#2\xint:%
1131 {%
1132     \unless\ifnum #1#2<\xint_c_x^ix
1133     \expandafter\XINT_verysmallmul_bi\else
1134     \expandafter\XINT_verysmallmul_bj\fi
1135     \the\numexpr \xint_c_x^ix+#1#2\xint:%
1136 }%
1137 \def\XINT_verysmallmul_bj{\expandafter\XINT_verysmallmul_cj }%
1138 \def\XINT_verysmallmul_cj #1#2\xint:%
1139     {#2\expandafter!\the\numexpr\XINT_verysmallmul #1\xint:%
1140 \def\XINT_verysmallmul_bi\the\numexpr\xint_c_x^ix+#1#2#3\xint:%
1141     {#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.

```

1142 \def\XINT_minimul_a #1\xint:#2!#3#4#5#6#7!%
1143 {%
1144     \expandafter\XINT_minimul_b
1145     \the\numexpr \xint_c_x^viii+#2*#7\xint:#2*#3#4#5#6+#1*#7\xint:#1*#3#4#5#6\xint:%
1146 }%
1147 \def\XINT_minimul_b #1#2#3#4#5\xint:#6\xint:%
1148 {%
1149     \expandafter\XINT_minimul_c
1150     \the\numexpr \xint_c_x^ix+#1#2#3#4+#6\xint:#5\xint:%

```

```

1151 }%
1152 \def\XINT_minimul_c #1#2#3#4#5#6\xint:#7\xint:#8\xint:%
1153 {%
1154     1#6#7\expandafter!\the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8!%
1155 }%

```

### 4.37 \xintiDivision

Completely rewritten for 1.2.

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

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

1.2l: \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:  $A=BQ+R$ ,  $0 \leq R < |B|$ .

```

1156 \def\xintiDivision    {\romannumeral0\xintiDivision }%
1157 \def\xintiDivision    #1{\expandafter\XINT_iidivision \romannumeral`&&@#1\xint:}%
1158 \def\XINT_iidivision  #1#2\xint:#3{\expandafter\XINT_iidivision_a\expandafter #1%
1159                      \romannumeral`&&@#3\xint:#2\xint:}%

```

On regarde les signes de A et de B.

```

1160 \def\XINT_iidivision_a #1#2% #1 de A, #2 de B.
1161 {%
1162     \if0#2\xint_dothis{\XINT_iidivision_divbyzero #1#2}\fi
1163     \if0#1\xint_dothis\XINT_iidivision_aiszero\fi
1164     \if-#2\xint_dothis{\expandafter\XINT_iidivision_bneg
1165                       \romannumeral0\XINT_iidivision_bpos #1}\fi
1166     \xint_orthat{\XINT_iidivision_bpos #1#2}%
1167 }%
1168 \def\XINT_iidivision_divbyzero#1#2#3\xint:#4\xint:
1169     {\if0#1\xint_dothis{\XINT_signalcondition{DivisionUndefined}}\fi
1170     \xint_orthat{\XINT_signalcondition{DivisionByZero}}%
1171     {Division of #1#4 by #2#3}{\{0\}{0}}}%
1172 \def\XINT_iidivision_aiszero #1\xint:#2\xint:{\{0\}{0}}%
1173 \def\XINT_iidivision_bneg #1% q->-q, r unchanged
1174     {\expandafter{\romannumeral0\XINT_opp #1}}%
1175 \def\XINT_iidivision_bpos #1%
1176 {%
1177     \xint_UDsignfork
1178         #1\XINT_iidivision_aneg
1179         -{\XINT_iidivision_apos #1}%
1180     \krof
1181 }%

```

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.

```

1182 \def\XINT_iidivision_apos #1#2\xint:#3\xint:{\XINT_div_prepare {#2}{#1#3}}%
1183 \def\XINT_iidivision_aneg #1\xint:#2\xint:
1184   {\expandafter
1185     \XINT_iidivision_aneg_b\romannumeral0\XINT_div_prepare {#1}{#2}{#1}}%
1186 \def\XINT_iidivision_aneg_b #1#2{\if0\XINT_Sgn #2\xint:
1187   \expandafter\XINT_iidivision_aneg_rzero
1188   \else
1189   \expandafter\XINT_iidivision_aneg_rpos
1190   \fi {#1}{#2}}%
1191 \def\XINT_iidivision_aneg_rzero #1#2#3{{-#1}{0}}% necessarily q was >0
1192 \def\XINT_iidivision_aneg_rpos #1%
1193 {%
1194   \expandafter\XINT_iidivision_aneg_end\expandafter
1195   {\expandafter-\romannumeral0\xintinc {#1}}% q-> -(1+q)
1196 }%
1197 \def\XINT_iidivision_aneg_end #1#2#3%
1198 {%
1199   \expandafter\xint_exchangetwo_keepbraces
1200   \expandafter{\romannumeral0\XINT_sub_mm_a {}{}#3\xint:#2\xint:}{#1}% r-> b-r
1201 }%

```

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.

```

1202 \def\XINT_div_prepare #1%
1203 {%
1204   \XINT_div_prepare_a #1\R\R\R\R\R\R\R\R {10}0000001\W !{#1}%
1205 }%
1206 \def\XINT_div_prepare_a #1#2#3#4#5#6#7#8#9%
1207 {%
1208   \xint_gob_til_R #9\XINT_div_prepare_small\R
1209   \XINT_div_prepare_b #9%
1210 }%

```

$B$  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).

```

1211 \def\XINT_div_prepare_small\R #1!#2%
1212 {%
1213   \ifcase #2
1214   \or\expandafter\XINT_div_BisOne
1215   \or\expandafter\XINT_div_BisTwo
1216   \else\expandafter\XINT_div_small_a
1217   \fi {#2}%
1218 }%
1219 \def\XINT_div_BisOne #1#2{{#2}{0}}%
1220 \def\XINT_div_BisTwo #1#2%
1221 {%

```

```

1222 \expandafter\expandafter\expandafter\XINT_div_BisTwo_a
1223 \ifodd\xintLDg{#2} \expandafter1\else \expandafter0\fi {#2}%
1224 }%
1225 \def\XINT_div_BisTwo_a #1#2%
1226 {%
1227 \expandafter{\romannumeral0\XINT_half
1228 #2\xint_bye\xint_Bye345678\xint_bye
1229 *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax}{#1}%
1230 }%

```

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.

```

1231 \def\XINT_div_small_a #1#2%
1232 {%
1233 \expandafter\XINT_div_small_b
1234 \the\numexpr #1/\xint_c_i\expandafter
1235 \xint:\the\numexpr \xint_c_x^viii+#1\expandafter!%
1236 \romannumeral0%
1237 \XINT_div_small_ba #2\R\R\R\R\R\R\R\R\{10}0000001\W
1238 #2\XINT_sepbyviii_Z_end 2345678\relax
1239 }%

```

Le #2 poursuivra l'expansion par \XINT\_div\_dosmallsmall ou par \XINT\_smalldivx\_a suivi de \XINT\_sdiv\_out.

```

1240 \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  $A < 10^8$ .

```

1241 \def\XINT_div_small_ba #1#2#3#4#5#6#7#8#9%
1242 {%
1243 \xint_gob_til_R #9\XINT_div_smallsmall\R
1244 \expandafter\XINT_div_dosmalldiv
1245 \the\numexpr\expandafter\XINT_sepbyviii_Z
1246 \romannumeral0\XINT_zeroes_forviii
1247 #1#2#3#4#5#6#7#8#9%
1248 }%

```

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

```

1249 \def\XINT_div_smallsmall\R
1250 \expandafter\XINT_div_dosmalldiv
1251 \the\numexpr\expandafter\XINT_sepbyviii_Z
1252 \romannumeral0\XINT_zeroes_forviii #1\R #2\relax
1253 {\XINT_div_dosmallsmall}{#1}%
1254 \def\XINT_div_dosmallsmall #1\xint:1#2!#3%
1255 {%
1256 \expandafter\XINT_div_smallsmallend
1257 \the\numexpr (#3+#1)/#2-\xint_c_i\xint:#2\xint:#3\xint:%
1258 }%
1259 \def\XINT_div_smallsmallend #1\xint:#2\xint:#3\xint:{\expandafter
1260 {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #3-#1*#2}}%

```



Si  $A \geq 10^8$ , il est maintenant sous la forme  $1\langle 8d \rangle! \dots 1\langle 8d \rangle! 1!$  avec plus significatifs en premier. Donc on poursuit par  $\backslash\expandafter\XINT_sdiv\_out\the\numexpr\XINT\_smalldivx\_a \times 1B! 1\langle 8d \rangle! \dots 1\langle 8d \rangle! 1!$  avec  $x = \text{round}(B/2)$ ,  $1B = 10^8 + B$ .

```
1261 \def\XINT_div_dosmalldiv
1262     {\expandafter\XINT_sdiv\_out\the\numexpr\XINT\_smalldivx\_a}}%
```

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

```
1263 \def\XINT_div_prepare_b
1264     {\expandafter\XINT_div_prepare_c\romannumeral0\XINT\_zeroes\_forviii }%
1265 \def\XINT_div_prepare_c #1!%
1266 {%
1267     \XINT_div_prepare_d #1.00000000!{#1}%
1268 }%
1269 \def\XINT_div_prepare_d #1#2#3#4#5#6#7#8#9%
1270 {%
1271     \expandafter\XINT_div_prepare_e\xint\_gob\_til\_dot #1#2#3#4#5#6#7#8#9!%
1272 }%
1273 \def\XINT_div_prepare_e #1!#2!#3#4%
1274 {%
1275     \XINT_div_prepare_f #4#3\X {#1}{#3}%
1276 }%
```

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.

```
1277 \def\XINT_div_prepare_f #1#2#3#4#5#6#7#8#9\X
1278 {%
1279     \expandafter\XINT_div_prepare_g
1280     \the\numexpr #1#2#3#4#5#6#7#8+\xint\_c\_i\expandafter
1281     \xint:\the\numexpr (#1#2#3#4#5#6#7#8+\xint\_c\_i)/\xint\_c\_ii\expandafter
1282     \xint:\the\numexpr #1#2#3#4#5#6#7#8\expandafter
1283     \xint:\romannumeral0\XINT\_sepandrev\_andcount
1284     #1#2#3#4#5#6#7#8#9\XINT\_rsepbyviii\_end\_A 2345678%
1285     \XINT\_rsepbyviii\_end\_B 2345678\relax\xint\_c\_ii\xint\_c\_i
1286     \R\xint:\xint\_c\_xii \R\xint:\xint\_c\_x \R\xint:\xint\_c\_viii \R\xint:\xint\_c\_vi
1287     \R\xint:\xint\_c\_iv \R\xint:\xint\_c\_ii \R\xint:\xint\_c\_W
1288     \X
1289 }%
1290 \def\XINT_div_prepare_g #1\xint:#2\xint:#3\xint:#4\xint:#5\X #6#7#8%
1291 {%
1292     \expandafter\XINT_div_prepare_h
1293     \the\numexpr\expandafter\XINT\_sepbyviii\_andcount
1294     \romannumeral0\XINT\_zeroes\_forviii #8#7\R\R\R\R\R\R\R\{10}0000001\W
1295     #8#7\XINT\_sepbyviii\_end 2345678\relax
1296     \xint\_c\_vii!\xint\_c\_vi!\xint\_c\_v!\xint\_c\_iv!%
1297     \xint\_c\_iii!\xint\_c\_ii!\xint\_c\_i!\xint\_c\_W
1298     {#1}{#2}{#3}{#4}{#5}{#6}%
```

```

1299 }%
1300 \def\XINT_div_prepare_h #11\xint:#2\xint:#3#4#5#6#7#8%
1301 {%
1302     \XINT_div_start_a {#2}{#6}{#1}{#3}{#4}{#5}{#7}{#8}%
1303 }%

```

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

```

1304 \def\XINT_div_start_a #1#2%
1305 {%
1306     \ifnum #1 < #2
1307         \expandafter\XINT_div_zeroQ
1308     \else
1309         \expandafter\XINT_div_start_b
1310     \fi
1311     {#1}{#2}%
1312 }%
1313 \def\XINT_div_zeroQ #1#2#3#4#5#6#7%
1314 {%
1315     \expandafter\XINT_div_zeroQ_end
1316     \romannumeral0\XINT_unsep_cuzsmall
1317     #3\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax\xint:
1318 }%
1319 \def\XINT_div_zeroQ_end #1\xint:#2%
1320     {\expandafter{\expandafter0\expandafter}\XINT_div_cleanR #1#2\xint:}%

```

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

```

1321 \def\XINT_div_start_b #1#2#3#4#5#6%
1322 {%
1323     \expandafter\XINT_div_finish\the\numexpr
1324     \XINT_div_start_c {#2}\xint:#3\xint:{#6}{#1}{#2}{#4}{#5}{#6}%
1325 }%
1326 \def\XINT_div_finish
1327 {%
1328     \expandafter\XINT_div_finish_a \romannumeral`&&\XINT_div_unsepQ
1329 }%
1330 \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«c».

```

1331 \def\XINT_div_finish_b #1%
1332 {%
1333     \if0#1%
1334         \expandafter\XINT_div_finish_bRzero
1335     \else
1336         \expandafter\XINT_div_finish_bRpos
1337     \fi
1338     #1%
1339 }%
1340 \def\XINT_div_finish_bRzero 0\xint:#1#2{{#1}{0}}%
1341 \def\XINT_div_finish_bRpos #1\xint:#2#3%

```

```

1342 {%
1343   \expandafter\xint_exchangetwo_keepbraces\xINT_div_cleanR #1#3\xint:{#2}%
1344 }%
1345 \def\xINT_div_cleanR #1000000000\xint:{{#1}}%

```

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

```

1346 \def\xINT_div_start_c #1%
1347 {%
1348   \ifnum #1>\xint_c_vi
1349     \expandafter\xINT_div_start_ca
1350   \else
1351     \expandafter\xINT_div_start_cb
1352   \fi {#1}%
1353 }%
1354 \def\xINT_div_start_ca #1#2\xint:#3!#4!#5!#6!#7!#8!#9!%
1355 {%
1356   \expandafter\xINT_div_start_c\expandafter
1357   {\the\numexpr #1-\xint_c_vii}#2#3!#4!#5!#6!#7!#8!#9!\xint:%
1358 }%
1359 \def\xINT_div_start_cb #1%
1360   {\csname XINT_div_start_c_\romannumeral\numexpr#1\endcsname}%
1361 \def\xINT_div_start_c_i #1\xint:#2!%
1362   {\XINT_div_start_c_ #1#2!\xint:}%
1363 \def\xINT_div_start_c_ii #1\xint:#2!#3!%
1364   {\XINT_div_start_c_ #1#2!#3!\xint:}%
1365 \def\xINT_div_start_c_iii #1\xint:#2!#3!#4!%
1366   {\XINT_div_start_c_ #1#2!#3!#4!\xint:}%
1367 \def\xINT_div_start_c_iv #1\xint:#2!#3!#4!#5!%
1368   {\XINT_div_start_c_ #1#2!#3!#4!#5!\xint:}%
1369 \def\xINT_div_start_c_v #1\xint:#2!#3!#4!#5!#6!%
1370   {\XINT_div_start_c_ #1#2!#3!#4!#5!#6!\xint:}%
1371 \def\xINT_div_start_c_vi #1\xint:#2!#3!#4!#5!#6!#7!%
1372   {\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».

```

1373 \def\xINT_div_start_c_ 1#1!#2\xint:#3\xint:#4#5#6%
1374 {%
1375   \XINT_div_I_a {#1}{#4}{1#1!#2}{#6}{00000000}#5{#3}{#6}%
1376 }%

```

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

```

1377 \def\xINT_div_I_a #1#2%
1378 {%
1379   \expandafter\xINT_div_I_b\the\numexpr #1/#2\xint:{#1}{#2}%
1380 }%
1381 \def\xINT_div_I_b #1%
1382 {%
1383   \xint_gob_til_zero #1\xINT_div_I_czero 0\xINT_div_I_c #1%
1384 }%

```

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».

```
1385 \def\XINT_div_I_czero 0\XINT_div_I_c 0\xint:#1#2#3#4#5{1#5\XINT_div_I_g {#3}}%
1386 \def\XINT_div_I_c #1\xint:#2#3%
1387 {%
1388   \expandafter\XINT_div_I_da\the\numexpr #2-#1*#3\xint:#1\xint:{#2}{#3}%
1389 }%
```

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

```
1390 \def\XINT_div_I_da #1\xint:%
1391 {%
1392   \ifnum #1>\xint_c_ix
1393     \expandafter\XINT_div_I_dP
1394   \else
1395     \ifnum #1<\xint_c_
1396       \expandafter\expandafter\expandafter\XINT_div_I_dN
1397     \else
1398       \expandafter\expandafter\expandafter\XINT_div_I_db
1399     \fi
1400   \fi
1401 }%
```

attention très mauvaises notations avec \_b et \_db.

```
1402 \def\XINT_div_I_dN #1\xint:%
1403 {%
1404   \expandafter\XINT_div_I_b\the\numexpr #1-\xint_c_i\xint:%
1405 }%
1406 \def\XINT_div_I_db #1\xint:#2#3#4#5%
1407 {%
1408   \expandafter\XINT_div_I_dc\expandafter #1%
1409   \romannumeral0\expandafter\XINT_div_sub\expandafter
1410     {\romannumeral0\XINT_rev_nounsep {}}#4\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1411     {\the\numexpr\XINT_div_verysmallmul #1!#51;!}%
1412   \Z {#4}{#5}%
1413 }%
```

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

```
1414 \def\XINT_div_I_dc #1#2%
1415 {%
1416   \if-#2\expandafter\XINT_div_I_dd\else\expandafter\XINT_div_I_de\fi
1417   #1#2%
1418 }%
1419 \def\XINT_div_I_dd #1-\Z
1420 {%
1421   \if #11\expandafter\XINT_div_I_dz\fi
1422   \expandafter\XINT_div_I_dP\the\numexpr #1-\xint_c_i\xint: XX%
1423 }%
1424 \def\XINT_div_I_dz #1XX#2#3#4%
1425 {%
```

```

1426     1#4\XINT_div_I_g {#2}%
1427 }%
1428 \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) -> Inouveauq.nouvel alpha,
    L, K, {x'y}, x, alpha',B«c»

1429 \def\XINT_div_I_dP #1\xint:#2#3#4#5#6%
1430 {%
1431     1#6+#1\expandafter\XINT_div_I_g\expandafter
1432     {\romannumeral0\expandafter\XINT_div_sub\expandafter
1433     {\romannumeral0\XINT_rev_nounsep {#4\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1434     {\the\numexpr\XINT_div_verysmallmul #1!#51;!}%
1435     }%
1436 }%

    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»

1437 \def\XINT_div_I_g #1#2#3#4#5#6#7%
1438 {%
1439     \expandafter !\the\numexpr
1440     \ifnum#2=#3
1441         \expandafter\XINT_div_exittofinish
1442     \else
1443         \expandafter\XINT_div_I_h
1444     \fi
1445     {#4}#1\xint:#6\xint:{{#4}{#5}{#3}{#2}}{#7}%
1446 }%

    {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»

1447 \def\XINT_div_exittofinish #1#2\xint:#3\xint:#4#5%
1448 {%
1449     1\expandafter\expandafter\expandafter!\expandafter\XINT_div_unsepQ_delim
1450     \romannumeral0\XINT_div_unsepR #2#3%
1451     \xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax\R\xint:
1452 }%

    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»

1453 \def\XINT_div_I_h #1\xint:#2!#3\xint:#4#5%
1454 {%
1455     \XINT_div_II_b #1#2!\xint:{{#5}{#4}{#3}{#5}}%
1456 }%

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

1457 \def\XINT_div_II_b #1#2!#3!%
1458 {%

```

```
1459 \xint_gob_til_eightzeroes #2\XINT_div_II_skipc 00000000%
1460 \XINT_div_II_c #1{1#2}{#3}%
1461 }%
```

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

```
1462 \def\XINT_div_II_skipc 00000000\XINT_div_II_c #1#2#3#4#5\xint:#6#7%
1463 {%
1464 \XINT_div_II_k #7{#4!#5}{#6}{00000000}%
1465 }%
```

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

```
1466 \def\XINT_div_II_c #1#2#3#4%
1467 {%
1468 \expandafter\XINT_div_II_d\the\numexpr\XINT_div_xmini
1469 #1\xint:#2!#3!#4!{#1}{#2}#3!#4!%
1470 }%
```

```
1471 \def\XINT_div_xmini #1%
1472 {%
1473 \xint_gob_til_one #1\XINT_div_xmini_a 1\XINT_div_mini #1%
1474 }%
```

```
1475 \def\XINT_div_xmini_a 1\XINT_div_mini 1#1%
1476 {%
```

```
1477 \xint_gob_til_zero #1\XINT_div_xmini_b 0\XINT_div_mini 1#1%
1478 }%
```

```
1479 \def\XINT_div_xmini_b 0\XINT_div_mini 10#1#2#3#4#5#6#7%
1480 {%
```

```
1481 \xint_gob_til_zero #7\XINT_div_xmini_c 0\XINT_div_mini 10#1#2#3#4#5#6#7%
1482 }%
```

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

```
1483 \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},  $\alpha'$ , B, «c» --> nouvel  $\alpha.x',y,B,q1$ , {{x'y},x,K,L},  $\alpha'$ , B, «c»

```
1484 \def\XINT_div_II_d 1#1#2#3#4#5!#6#7#8\xint:#9%
1485 {%
```

```
1486 \expandafter\XINT_div_II_e
1487 \romannumeral0\expandafter\XINT_div_sub\expandafter
1488 {\romannumeral0\XINT_rev_nounsep }#8\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1489 {\the\numexpr\XINT_div_smallmul_a 100000000\xint:#1#2#3#4\xint:#5!#91;!}%
1490 \xint:{#6}{#7}{#9}{#1#2#3#4#5}%
1491 }%
```

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

```
1492 \def\XINT_div_II_e 1#1!%
1493 {%
1494 \xint_gob_til_eightzeroes #1\XINT_div_II_skipf 00000000%
1495 \XINT_div_II_f 1#1!%
1496 }%
```

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

```
1497 \def\XINT_div_II_skipf 00000000\XINT_div_II_f 100000000!#1\xint:#2#3#4#5#6%
1498 {%
1499   \XINT_div_II_k #6{#1}{#4}{#5}%
1500 }%
```

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.

```
1501 \def\XINT_div_II_f #1!#2!#3\xint:%
1502 {%
1503   \XINT_div_II_fa {#1!#2!}{#1!#2!#3}%
1504 }%
1505 \def\XINT_div_II_fa #1#2#3#4%
1506 {%
1507   \expandafter\XINT_div_II_g \the\numexpr\XINT_div_xmini #3\xint:#4!#1{#2}%
1508 }%
```

#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»

```
1509 \def\XINT_div_II_g 1#1#2#3#4#5!#6#7#8%
1510 {%
1511   \expandafter \XINT_div_II_h
1512   \the\numexpr 1#1#2#3#4#5+#8\expandafter\expandafter\expandafter
1513   \xint:\expandafter\expandafter\expandafter
1514   {\expandafter\xint_gob_til_exclam
1515   \romannumeral0\expandafter\XINT_div_sub\expandafter
1516   {\romannumeral0\XINT_rev_nounsep }#6\R!\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1517   {\the\numexpr\XINT_div_smallmul_a 100000000\xint:#1#2#3#4\xint:#5!#71;!}%
1518   {#7}%
1519 }%
```

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»

```
1520 \def\XINT_div_II_h 1#1\xint:#2#3#4%
1521 {%
1522   \XINT_div_II_k #4{#2}{#3}{#1}%
1523 }%
```

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

```
1524 \def\XINT_div_II_k #1#2#3#4#5%
1525 {%
1526   \expandafter\XINT_div_II_l \the\numexpr #4-\xint_c_i\xint:{#3}#1{#2}#5\xint:%
1527 }%
1528 \def\XINT_div_II_l #1\xint:#2#3#4#5!#6!%
1529 {%
1530   \XINT_div_II_m {#1}{#2}{#3}{#4}{#5}{#5}{#6}1#6!%
1531 }%
```

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

```
1532 \def\XINT_div_II_m #1#2#3#4\xint:#5#6%
1533 {%
1534     \XINT_div_I_a {#3}{#2}{#4}{#5}{#6}#1%
1535 }%
```

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

```
1536 \def\XINT_div_minimulwc_a 1#1\xint:#2\xint:#3!#4#5#6#7#8\xint:%
1537 {%
1538     \expandafter\XINT_div_minimulwc_b
1539     \the\numexpr \xint_c_x^ix+#1+#3*#8\xint:#3*#4#5#6#7+#2*#8\xint:#2*#4#5#6#7\xint:%
1540 }%
1541 \def\XINT_div_minimulwc_b 1#1#2#3#4#5#6\xint:#7\xint:%
1542 {%
1543     \expandafter\XINT_div_minimulwc_c
1544     \the\numexpr \xint_c_x^ix+#1#2#3#4#5+#7\xint:#6\xint:%
1545 }%
1546 \def\XINT_div_minimulwc_c 1#1#2#3#4#5#6\xint:#7\xint:#8\xint:%
1547 {%
1548     1#6#7\expandafter!%
1549     \the\numexpr\expandafter\XINT_div_smallmul_a
1550     \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8\xint:%
1551 }%
1552 \def\XINT_div_smallmul_a #1\xint:#2\xint:#3!1#4!%
1553 {%
1554     \xint_gob_til_sc #4\XINT_div_smallmul_e;%
1555     \XINT_div_minimulwc_a #1\xint:#2\xint:#3!#4\xint:#2\xint:#3!%
1556 }%
1557 \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 `10000000!` at the end.

```
1558 \def\XINT_div_verysmallmul #1%
1559     {\xint_gob_til_one #1\XINT_div_verysmallisone 1\XINT_div_verysmallmul_a 0\xint:#1}%
1560 \def\XINT_div_verysmallisone 1\XINT_div_verysmallmul_a 0\xint:1!1#11;!%
1561     {1\relax #1100000000!}%
1562 \def\XINT_div_verysmallmul_a #1\xint:#2!1#3!%
1563 {%
1564     \xint_gob_til_sc #3\XINT_div_verysmallmul_e;%
1565     \expandafter\XINT_div_verysmallmul_b
1566     \the\numexpr \xint_c_x^ix+#2*#3+#1\xint:#2!%
1567 }%
1568 \def\XINT_div_verysmallmul_b 1#1#2\xint:%
1569     {1#2\expandafter!\the\numexpr\XINT_div_verysmallmul_a #1\xint:}%
1570 \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.



```

1571 \def\XINT_div_sub #1#2%
1572 {%
1573     \expandafter\XINT_div_sub_clean
1574     \the\numexpr\expandafter\XINT_div_sub_a\expandafter
1575     1#2;!!;!!;!!\W #1;!!;!!;!!\W
1576 }%
1577 \def\XINT_div_sub_clean #1-#2#3\W
1578 {%
1579     \if1#2\expandafter\XINT_rev_nounsep\else\expandafter\XINT_div_sub_neg\fi
1580     {}#1\R!\R!\R!\R!\R!\R!\R!\R!\W
1581 }%
1582 \def\XINT_div_sub_neg #1\W { -}%
1583 \def\XINT_div_sub_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
1584 {%
1585     \XINT_div_sub_b #1!#6!#2!#7!#3!#8!#4!#9!#5\W
1586 }%
1587 \def\XINT_div_sub_b #1#2#3!#4!%
1588 {%
1589     \xint_gob_til_sc #4\XINT_div_sub_bi ;%
1590     \expandafter\XINT_div_sub_c\the\numexpr#1-#3+1#4-\xint_c_i\xint:%
1591 }%
1592 \def\XINT_div_sub_c 1#1#2\xint:%
1593 {%
1594     1#2\expandafter!\the\numexpr\XINT_div_sub_d #1%
1595 }%
1596 \def\XINT_div_sub_d #1#2#3!#4!%
1597 {%
1598     \xint_gob_til_sc #4\XINT_div_sub_di ;%
1599     \expandafter\XINT_div_sub_e\the\numexpr#1-#3+1#4-\xint_c_i\xint:%
1600 }%
1601 \def\XINT_div_sub_e 1#1#2\xint:%
1602 {%
1603     1#2\expandafter!\the\numexpr\XINT_div_sub_f #1%
1604 }%
1605 \def\XINT_div_sub_f #1#2#3!#4!%
1606 {%
1607     \xint_gob_til_sc #4\XINT_div_sub_fi ;%
1608     \expandafter\XINT_div_sub_g\the\numexpr#1-#3+1#4-\xint_c_i\xint:%
1609 }%
1610 \def\XINT_div_sub_g 1#1#2\xint:%
1611 {%
1612     1#2\expandafter!\the\numexpr\XINT_div_sub_h #1%
1613 }%
1614 \def\XINT_div_sub_h #1#2#3!#4!%
1615 {%
1616     \xint_gob_til_sc #4\XINT_div_sub_hi ;%
1617     \expandafter\XINT_div_sub_i\the\numexpr#1-#3+1#4-\xint_c_i\xint:%
1618 }%
1619 \def\XINT_div_sub_i 1#1#2\xint:%
1620 {%
1621     1#2\expandafter!\the\numexpr\XINT_div_sub_a #1%
1622 }%

```

```

1623 \def\XINT_div_sub_bi;%
1624   \expandafter\XINT_div_sub_c\the\numexpr#1-#2+#3\xint:#4!#5!#6!#7!#8!#9!;! \W
1625 {%
1626   \XINT_div_sub_l #1#2!#5!#7!#9!%
1627 }%
1628 \def\XINT_div_sub_di;%
1629   \expandafter\XINT_div_sub_e\the\numexpr#1-#2+#3\xint:#4!#5!#6!#7!#8\W
1630 {%
1631   \XINT_div_sub_l #1#2!#5!#7!%
1632 }%
1633 \def\XINT_div_sub_fi;%
1634   \expandafter\XINT_div_sub_g\the\numexpr#1-#2+#3\xint:#4!#5!#6\W
1635 {%
1636   \XINT_div_sub_l #1#2!#5!%
1637 }%
1638 \def\XINT_div_sub_hi;%
1639   \expandafter\XINT_div_sub_i\the\numexpr#1-#2+#3\xint:#4\W
1640 {%
1641   \XINT_div_sub_l #1#2!%
1642 }%
1643 \def\XINT_div_sub_l #1%
1644 {%
1645   \xint_UDzerofork
1646   #1{-2\relax}%
1647   0\XINT_div_sub_r
1648   \krof
1649 }%
1650 \def\XINT_div_sub_r #1!%
1651 {%
1652   -\ifnum 0#1=\xint_c_ 1\else2\fi\relax
1653 }%

```

Ici  $B < 10^8$  (et est  $> 2$ ). On exécute `\expandafter\XINT_sdiv_out\the\numexpr\XINT_smalldivx_a x.1B!1<8d>!...1<8d>!1;! avec  $x = \text{round}(B/2)$ ,  $1B = 10^8 + B$ , et A déjà en blocs  $1 < 8d > !$  (non renversés). Le \the\numexpr\XINT_smalldivx_a va produire  $Q \backslash Z R \backslash W$  avec un  $R < 10^8$ , et un Q sous forme de blocs  $1 < 8d > !$  terminé par  $1!$  et nécessitant 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.`

```

1654 \def\XINT_sdiv_out #1;!#2!%
1655   {\expandafter
1656   {\romannumeral0\XINT_unsep_cuzsmall
1657   #1\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax}%
1658   {#2}}%

```

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

```

1659 \def\XINT_smalldivx_a #1\xint:1#2!1#3!%
1660 {%
1661   \expandafter\XINT_smalldivx_b
1662   \the\numexpr (#3+#1)/#2-\xint_c_i!#1\xint:#2!#3!%
1663 }%
1664 \def\XINT_smalldivx_b #1#2!%

```

```

1665 {%
1666     \if0#1\else
1667         \xint_c_x^viii+#1#2\xint_afterfi{\expandafter!\the\numexpr}\fi
1668     \XINT_smallldiv_c #1#2!%
1669 }%
1670 \def\XINT_smallldiv_c #1!#2!\xint:#3!#4!%
1671 {%
1672     \expandafter\XINT_smallldiv_d\the\numexpr #4-#1*#3!#2\xint:#3!%
1673 }%

```

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.

```

1674 \def\XINT_smallldiv_d #1!#2!1#3#4!%
1675 {%
1676     \xint_gob_til_sc #3\XINT_smallldiv_end ;%
1677     \XINT_smallldiv_e #1!#2!1#3#4!%
1678 }%
1679 \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.

```

1680 \def\XINT_smallldiv_e #1!#2!\xint:#3!%
1681 {%
1682     \expandafter\XINT_smallldiv_f\the\numexpr
1683     \xint_c_xi_e_viii_mone+#1*\xint_c_x^viii/#3!#2!\xint:#3!#1!%
1684 }%
1685 \def\XINT_smallldiv_f 1#1#2#3#4#5#6!#7!\xint:#8!%
1686 {%
1687     \xint_gob_til_zero #1\XINT_smallldiv_fz 0%
1688     \expandafter\XINT_smallldiv_g
1689     \the\numexpr\XINT_minimul_a #2#3#4#5\xint:#6!#8!#2#3#4#5#6!#7!\xint:#8!%
1690 }%
1691 \def\XINT_smallldiv_fz 0%
1692     \expandafter\XINT_smallldiv_g\the\numexpr\XINT_minimul_a
1693     9999\xint:9999!#1!99999999!#2!0!1#3!%
1694 {%
1695     \XINT_smallldiv_i \xint:#3!\xint_c_!#2!%
1696 }%
1697 \def\XINT_smallldiv_g 1#1!1#2!#3!#4!#5!#6!%
1698 {%
1699     \expandafter\XINT_smallldiv_h\the\numexpr 1#6-#1!\xint:#2!#5!#3!#4!%
1700 }%
1701 \def\XINT_smallldiv_h 1#1#2!\xint:#3!#4!%
1702 {%
1703     \expandafter\XINT_smallldiv_i\the\numexpr #4-#3+#1-\xint_c_i\xint:#2!%
1704 }%
1705 \def\XINT_smallldiv_i #1!\xint:#2!#3!#4!\xint:#5!%
1706 {%
1707     \expandafter\XINT_smallldiv_j\the\numexpr (#1#2+#4)/#5-\xint_c_i!#3!#1#2!#4!\xint:#5!%
1708 }%

```

```
1709 \def\XINT_smallldiv_j #1!#2!%
1710 {%
1711     \xint_c_x^viii+#1+#2\expandafter!\the\numexpr\XINT_smallldiv_k
1712     #1!%
1713 }%
```

On boucle vers `\XINT_smallldiv_d`.

```
1714 \def\XINT_smallldiv_k #1!#2!#3\xint:#4!%
1715 {%
1716     \expandafter\XINT_smallldiv_d\the\numexpr #2-#1*#4!#3\xint:#4!%
1717 }%
```

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

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<8chiffres>`.

```
1718 \def\XINT_div_mini #1\xint:#2!#3!%
1719 {%
1720     \expandafter\XINT_div_mini_a\the\numexpr
1721     \xint_c_xi_e_viii_mone+#3*\xint_c_x^viii/#1!#1\xint:#2!#3!%
1722 }%
```

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.

```
1723 \def\XINT_div_mini_a 1#1#2#3#4#5#6!#7\xint:#8!%
1724 {%
1725     \xint_gob_til_zero #1\XINT_div_mini_w 0%
1726     \expandafter\XINT_div_mini_b
1727     \the\numexpr\XINT_minimul_a #2#3#4#5\xint:#6!#7!#2#3#4#5#6!#7\xint:#8!%
1728 }%
1729 \def\XINT_div_mini_w 0%
1730     \expandafter\XINT_div_mini_b\the\numexpr\XINT_minimul_a
1731     9999\xint:9999!#1!99999999!#2\xint:#3!00000000!#4!%
1732 {%
1733     \xint_c_x^viii_mone+(#4+#3)/#2!%
1734 }%
1735 \def\XINT_div_mini_b 1#1!1#2!#3!#4!#5!#6!%
1736 {%
1737     \expandafter\XINT_div_mini_c
1738     \the\numexpr 1#6-#1\xint:#2!#5!#3!#4!%
1739 }%
1740 \def\XINT_div_mini_c 1#1#2\xint:#3!#4!%
1741 {%
1742     \expandafter\XINT_div_mini_d
1743     \the\numexpr #4-#3+#1-\xint_c_i\xint:#2!%
```

```
1744 }%
1745 \def\XINT_div_mini_d #1\xint:#2!#3!#4\xint:#5!%
1746 {%
1747   \xint_c_x^viii_mone+#3+(#1#2+#5)/#4!%
1748 }%
```

## Derived arithmetic

### 4.38 `\xintiiQuo`, `\xintiiRem`

```
1749 \def\xintiiQuo {\romannumeral0\xintiiquo }%
1750 \def\xintiiRem {\romannumeral0\xintiiirem }%
1751 \def\xintiiquo
1752   {\expandafter\xint_stop_atfirstoftwo\romannumeral0\xintiidivision }%
1753 \def\xintiiirem
1754   {\expandafter\xint_stop_atsecondoftwo\romannumeral0\xintiidivision }%
```

### 4.39 `\xintiiDivRound`

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

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

```
1755 \def\xintiiDivRound {\romannumeral0\xintiidivround }%
1756 \def\xintiidivround #1{\expandafter\XINT_iidivround\romannumeral`&&@#1\xint:}%
1757 \def\XINT_iidivround #1#2\xint:#3%
1758   {\expandafter\XINT_iidivround_a\expandafter #1%
1759     \romannumeral0\xintnum{#3}\xint:#2\xint:}%
1760 \def\XINT_iidivround #1#2\xint:#3%
1761   {\expandafter\XINT_iidivround_a\expandafter #1\romannumeral`&&@#3\xint:#2\xint:}%
1762 \def\XINT_iidivround_a #1#2% #1 de A, #2 de B.
1763 {%
1764   \if0#2\xint_dothis{\XINT_iidivround_divbyzero#1#2}\fi
1765   \if0#1\xint_dothis\XINT_iidivround_aiszero\fi
1766   \if-#2\xint_dothis{\XINT_iidivround_bneg #1}\fi
1767   \xint_orthat{\XINT_iidivround_bpos #1#2}%
1768 }%
1769 \def\XINT_iidivround_divbyzero #1#2#3\xint:#4\xint:
1770   {\XINT_signalcondition{DivisionByZero}{Division of #1#4 by #2#3}{0}}%
1771 \def\XINT_iidivround_aiszero #1\xint:#2\xint:{ 0}%
1772 \def\XINT_iidivround_bpos #1%
1773 {%
1774   \xint_UDsignfork
1775     #1{\xintiiopp\XINT_iidivround_pos {}}%
1776     -{\XINT_iidivround_pos #1}%
1777   \krof
1778 }%
1779 \def\XINT_iidivround_bneg #1%
1780 {%
1781   \xint_UDsignfork
1782     #1{\XINT_iidivround_pos {}}%
1783     -{\xintiiopp\XINT_iidivround_pos #1}%
1784   \krof
1785 }%
```

```

1786 \def\XINT_iidivround_pos #1#2\xint:#3\xint:
1787 {%
1788   \expandafter\expandafter\expandafter\XINT_dsrr
1789   \expandafter\xint_firstoftwo
1790   \romannumeral0\XINT_div_prepare {#2}{#1#30}%
1791   \xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax
1792 }%

```

#### 4.40 \xintiDivTrunc

1.21: \xintiDivTrunc made robust against non terminated input.

```

1793 \def\xintiDivTrunc {\romannumeral0\xintiDivTrunc }%
1794 \def\xintiDivTrunc #1{\expandafter\XINT_iidivtrunc\romannumeral`&&@#1\xint:}%
1795 \def\XINT_iidivtrunc #1#2\xint:#3{\expandafter\XINT_iidivtrunc_a\expandafter #1%
1796   \romannumeral`&&@#3\xint:#2\xint:}%
1797 \def\XINT_iidivtrunc_a #1#2% #1 de A, #2 de B.
1798 {%
1799   \if0#2\xint_dothis{\XINT_iidivtrunc_divbyzero#1#2}\fi
1800   \if0#1\xint_dothis\XINT_iidivtrunc_aiszero\fi
1801   \if-#2\xint_dothis{\XINT_iidivtrunc_bneg #1}\fi
1802   \xint_orthat{\XINT_iidivtrunc_bpos #1#2}%
1803 }%

```

Attention to not move DivRound code beyond that point.

```

1804 \let\XINT_iidivtrunc_divbyzero\XINT_iidivround_divbyzero
1805 \let\XINT_iidivtrunc_aiszero \XINT_iidivround_aiszero
1806 \def\XINT_iidivtrunc_bpos #1%
1807 {%
1808   \xint_UDsignfork
1809     #1{\xintiopp\XINT_iidivtrunc_pos {}}%
1810     -{\XINT_iidivtrunc_pos #1}%
1811   \krof
1812 }%
1813 \def\XINT_iidivtrunc_bneg #1%
1814 {%
1815   \xint_UDsignfork
1816     #1{\XINT_iidivtrunc_pos {}}%
1817     -{\xintiopp\XINT_iidivtrunc_pos #1}%
1818   \krof
1819 }%
1820 \def\XINT_iidivtrunc_pos #1#2\xint:#3\xint:
1821   {\expandafter\xint_stop_atfirstoftwo
1822   \romannumeral0\XINT_div_prepare {#2}{#1#3}}%

```

#### 4.41 \xintiModTrunc

Renamed from \xintiMod to \xintiModTrunc at 1.2p.

```

1823 \def\xintiModTrunc {\romannumeral0\xintiModTrunc }%
1824 \def\xintiModTrunc #1{\expandafter\XINT_iimodtrunc\romannumeral`&&@#1\xint:}%
1825 \def\XINT_iimodtrunc #1#2\xint:#3{\expandafter\XINT_iimodtrunc_a\expandafter #1%

```

```

1826             \romannumeral`&&@#3\xint:#2\xint:}%
1827 \def\xINT_iimodtrunc_a #1#2% #1 de A, #2 de B.
1828 {%
1829     \if0#2\xint_dothis{\XINT_iimodtrunc_divbyzero#1#2}\fi
1830     \if0#1\xint_dothis{\XINT_iimodtrunc_aiszero}\fi
1831     \if-#2\xint_dothis{\XINT_iimodtrunc_bneg #1}\fi
1832         \xint_orthat{\XINT_iimodtrunc_bpos #1#2}%
1833 }%

```

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

```

1834 \let\xINT_iimodtrunc_divbyzero\xINT_iidivround_divbyzero
1835 \let\xINT_iimodtrunc_aiszero \XINT_iidivround_aiszero
1836 \def\xINT_iimodtrunc_bpos #1%
1837 {%
1838     \xint_UDsignfork
1839         #1{\xintiiopt\xINT_iimodtrunc_pos {}}%
1840         -{\XINT_iimodtrunc_pos #1}%
1841     \krof
1842 }%
1843 \def\xINT_iimodtrunc_bneg #1%
1844 {%
1845     \xint_UDsignfork
1846         #1{\xintiiopt\xINT_iimodtrunc_pos {}}%
1847         -{\XINT_iimodtrunc_pos #1}%
1848     \krof
1849 }%
1850 \def\xINT_iimodtrunc_pos #1#2\xint:#3\xint:
1851     {\expandafter\xint_stop_atsecondoftwo\romannumeral0\xINT_div_prepare
1852     {#2}{#1#3}}%

```

## 4.42 \xintiDivMod

**1.2p.** It is associated with floored division (like Python divmod function), and with the // operator in [xintiexpr](#).

```

1853 \def\xintiDivMod {\romannumeral0\xintiDivMod }%
1854 \def\xintiDivMod #1{\expandafter\xINT_iidivmod\romannumeral`&&@#1\xint:}%
1855 \def\xINT_iidivmod #1#2\xint:#3{\expandafter\xINT_iidivmod_a\expandafter #1%
1856             \romannumeral`&&@#3\xint:#2\xint:}%
1857 \def\xINT_iidivmod_a #1#2% #1 de A, #2 de B.
1858 {%
1859     \if0#2\xint_dothis{\XINT_iidivmod_divbyzero#1#2}\fi
1860     \if0#1\xint_dothis{\XINT_iidivmod_aiszero}\fi
1861     \if-#2\xint_dothis{\XINT_iidivmod_bneg #1}\fi
1862         \xint_orthat{\XINT_iidivmod_bpos #1#2}%
1863 }%
1864 \def\xINT_iidivmod_divbyzero #1#2\xint:#3\xint:
1865 {%
1866     \XINT_signalcondition{DivisionByZero}{Division by #2 of #1#3}{}%
1867     {{0}{0}}% à revoir...
1868 }%
1869 \def\xINT_iidivmod_aiszero #1\xint:#2\xint:{{0}{0}}%

```

```

1870 \def\XINT_iidivmod_bneg #1%
1871 {%
1872   \expandafter\XINT_iidivmod_bneg_finish
1873   \romannumeral0\xint_UDsignfork
1874     #1{\XINT_iidivmod_bpos {}}%
1875     -{\XINT_iidivmod_bpos {-#1}}%
1876   \krof
1877 }%
1878 \def\XINT_iidivmod_bneg_finish#1#2%
1879 {%
1880   \expandafter\xint_exchangetwo_keepbraces\expandafter
1881   {\romannumeral0\xintiiopt#2}{#1}%
1882 }%
1883 \def\XINT_iidivmod_bpos #1#2\xint:#3\xint:{\xintiividivision{#1#3}{#2}}%

```

#### 4.43 \xintiiDivFloor

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

```

1884 \def\xintiiDivFloor {\romannumeral0\xintiividivfloor}%
1885 \def\xintiividivfloor {\expandafter\xint_stop_atfirstoftwo
1886   \romannumeral0\xintiividivmod}%

```

#### 4.44 \xintiiMod

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

```

1887 \def\xintiiMod {\romannumeral0\xintiimod}%
1888 \def\xintiimod {\expandafter\xint_stop_atsecondoftwo
1889   \romannumeral0\xintiividivmod}%

```

#### 4.45 \xintiiSqr

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

```

1890 \def\xintiiSqr {\romannumeral0\xintiisqr }%
1891 \def\xintiisqr #1%
1892 {%
1893   \expandafter\XINT_sqr\romannumeral0\xintiiaabs{#1}\xint:
1894 }%
1895 \def\XINT_sqr #1\xint:
1896 {%
1897   \expandafter\XINT_sqr_a
1898   \romannumeral0\expandafter\XINT_sepandrev_andcount
1899   \romannumeral0\XINT_zeroes_forviii #1\R\R\R\R\R\R\R\R{10}0000001\W
1900   #1\XINT_rsepbyviii_end_A 2345678%
1901   \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
1902   \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
1903   \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
1904   \xint:
1905 }%

```



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

```

1906 \def\XINT_sqr_a #1\xint:
1907 {%
1908   \ifnum #1=\xint_c_i \expandafter\XINT_sqr_small
1909       \else\expandafter\XINT_sqr_start\fi
1910 }%
1911 \def\XINT_sqr_small #1#2#3#4#5!\xint:
1912 {%
1913   \ifnum #1#2#3#4#5<46341 \expandafter\XINT_sqr_verysmall\fi
1914   \expandafter\XINT_sqr_small_out
1915   \the\numexpr\XINT_minimul_a #1#2#3#4\xint:#5!#1#2#3#4#5!%
1916 }%
1917 \def\XINT_sqr_verysmall#1{%
1918 \def\XINT_sqr_verysmall
1919   \expandafter\XINT_sqr_small_out\the\numexpr\XINT_minimul_a ##1!##2!%
1920   {\expandafter#1\the\numexpr ##2*##2\relax}%
1921 }\XINT_sqr_verysmall{ }%
1922 \def\XINT_sqr_small_out #1!1#2!%
1923 {%
1924   \XINT_cuz #2#1\R
1925 }%

```

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`.

```

1926 \def\XINT_sqr_start #1\xint:
1927 {%
1928   \expandafter\XINT_mul_out
1929   \the\numexpr\XINT_mul_loop
1930       100000000!1;!W #11;!W #11;!%
1931   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1932 }%

```

#### 4.46 `\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 `\xintiiPow` 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.

```

1933 \def\xintiiPow {\romannumeral0\xintiiPow }%
1934 \def\xintiiPow #1#2%
1935 {%
1936   \expandafter\xint_pow\the\numexpr #2\expandafter
1937   .\romannumeral`&&@#1\xint:
1938 }%
1939 \def\xint_pow #1.#2%#3\xint:
1940 {%

```

```

1941 \xint_UDzerominusfork
1942 #2-\XINT_pow_AisZero
1943 0#2\XINT_pow_Aneg
1944 0-{\XINT_pow_Apos #2}%
1945 \krof {#1}%
1946 }%
1947 \def\XINT_pow_AisZero #1#2\xint:
1948 {%
1949 \ifcase\XINT_cntSgn #1\xint:
1950 \xint_afterfi { 1}%
1951 \or
1952 \xint_afterfi { 0}%
1953 \else
1954 \xint_afterfi
1955 {\XINT_signalcondition{DivisionByZero}{Zero to power #1}{0}}%
1956 \fi
1957 }%
1958 \def\XINT_pow_Aneg #1%
1959 {%
1960 \ifodd #1
1961 \expandafter\XINT_opp\romannumeral0%
1962 \fi
1963 \XINT_pow_Apos {}{#1}%
1964 }%
1965 \def\XINT_pow_Apos #1#2{\XINT_pow_Apos_a {#2}#1}%
1966 \def\XINT_pow_Apos_a #1#2#3%
1967 {%
1968 \xint_gob_til_xint: #3\XINT_pow_Apos_short\xint:
1969 \XINT_pow_AatleastTwo {#1}#2#3%
1970 }%
1971 \def\XINT_pow_Apos_short\xint:\XINT_pow_AatleastTwo #1#2\xint:
1972 {%
1973 \ifcase #2
1974 \xintError:thiscannothappen
1975 \or \expandafter\XINT_pow_AisOne
1976 \else\expandafter\XINT_pow_AatleastTwo
1977 \fi {#1}#2\xint:
1978 }%
1979 \def\XINT_pow_AisOne #1\xint:{ 1}%
1980 \def\XINT_pow_AatleastTwo #1%
1981 {%
1982 \ifcase\XINT_cntSgn #1\xint:
1983 \expandafter\XINT_pow_BisZero
1984 \or
1985 \expandafter\XINT_pow_I_in
1986 \else
1987 \expandafter\XINT_pow_BisNegative
1988 \fi
1989 {#1}%
1990 }%
1991 \def\XINT_pow_BisNegative #1\xint:{\XINT_signalcondition{Underflow}{Inverse power
1992 can not be represented by an integer}{0}}%

```

```
1993 \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).
```

```
1994 \def\XINT_pow_I_in #1#2\xint:
1995 {%
1996     \expandafter\XINT_pow_I_loop
1997     \the\numexpr #1\expandafter\xint:%
1998     \romannumeral0\expandafter\XINT_sepandrev
1999     \romannumeral0\XINT_zeroes_forviii #2\R\R\R\R\R\R\R\R\{10}0000001\W
2000     #2\XINT_rsepbyviii_end_A 2345678%
2001     \XINT_rsepbyviii_end_B 2345678\relax XX%
2002     \R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\W
2003     1;! \W
2004     1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
2005 }%
2006 \def\XINT_pow_I_loop #1\xint:%
2007 {%
2008     \ifnum #1 = \xint_c_i\expandafter\XINT_pow_I_exit\fi
2009     \ifodd #1
2010         \expandafter\XINT_pow_II_in
2011     \else
2012         \expandafter\XINT_pow_I_squareit
2013     \fi #1\xint:%
2014 }%
2015 \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 "butcheck-ifsmall" 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.

```
2016 \def\XINT_pow_I_squareit #1\xint:#2\W%
2017 {%
2018     \expandafter\XINT_pow_I_loop
2019     \the\numexpr #1/\xint_c_ii\expandafter\xint:%
2020     \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W
2021 }%
2022 \def\XINT_pow_mulbutcheckifsmall #1!1#2%
2023 {%
2024     \xint_gob_til_sc #2\XINT_pow_mul_small;%
2025     \XINT_mul_loop 100000000!1;! \W #1!1#2%
2026 }%
2027 \def\XINT_pow_mul_small;\XINT_mul_loop
2028     100000000!1;! \W #1!1;! \W
2029 {%
2030     \XINT_smallmul 1#1!%
2031 }%
2032 \def\XINT_pow_II_in #1\xint:#2\W
2033 {%
2034     \expandafter\XINT_pow_II_loop
2035     \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter\xint:%
2036     \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W #2\W
2037 }%
```

```

2038 \def\XINT_pow_II_loop #1\xint:%
2039 {%
2040   \ifnum #1 = \xint_c_i\expandafter\XINT_pow_II_exit\fi
2041   \ifodd #1
2042     \expandafter\XINT_pow_II_odda
2043   \else
2044     \expandafter\XINT_pow_II_even
2045   \fi #1\xint:%
2046 }%
2047 \def\XINT_pow_II_exit\ifodd #1\fi #2\xint:#3\W #4\W
2048 {%
2049   \expandafter\XINT_mul_out
2050   \the\numexpr\XINT_pow_mulbutcheckifsmall #4\W #3%
2051 }%
2052 \def\XINT_pow_II_even #1\xint:#2\W
2053 {%
2054   \expandafter\XINT_pow_II_loop
2055   \the\numexpr #1/\xint_c_ii\expandafter\xint:%
2056   \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W
2057 }%
2058 \def\XINT_pow_II_odda #1\xint:#2\W #3\W
2059 {%
2060   \expandafter\XINT_pow_II_oddb
2061   \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter\xint:%
2062   \the\numexpr\XINT_pow_mulbutcheckifsmall #3\W #2\W #2\W
2063 }%
2064 \def\XINT_pow_II_oddb #1\xint:#2\W #3\W
2065 {%
2066   \expandafter\XINT_pow_II_loop
2067   \the\numexpr #1\expandafter\xint:%
2068   \the\numexpr\XINT_pow_mulbutcheckifsmall #3\W #3\W #2\W
2069 }%

```

#### 4.47 \xintiiFac

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

An `\xintiFac` is needed by `xintexpr.sty`. Prior to 1.2o it was defined here as an alias to `\xintiFac`, then redefined by `xintfrac` to use `\xintNum`. This was incoherent. Contrarily to other similarly named macros, `\xintiiFac` uses `\numexpr` on its input. This is also incoherent with the naming scheme, alas.

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).

Note (end november 2015): I also tried out a quickly written recursive (binary split) implementation

```

\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!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 10000000!1;!\W }% core multiplication...
\catcode`_ 8
\catcode`^ 7

```

and I was quite surprised that it was only about 1.6x--2x slower in the range N=200 to 2000 than the `\xintiifac` here which attempts to be smarter...

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 `\xintiifac` 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`.

```

2070 \def\xintiifac {\romannumeral0\xintiifac }%
2071 \def\xintiifac #1{\expandafter\XINT_fac_fork\the\numexpr#1.}%
2072 \def\XINT_fac_fork #1#2.%
2073 {%
2074   \xint_UDzerominusfork
2075   #1-\XINT_fac_zero
2076   0#1\XINT_fac_neg
2077   0-\XINT_fac_checksize
2078   \krof #1#2.%
2079 }%
2080 \def\XINT_fac_zero #1.{ 1}%
2081 \def\XINT_fac_neg #1.{\XINT_signalcondition{InvalidOperation}{Factorial of
2082   negative: (#1)!}{0}}%
2083 \def\XINT_fac_checksize #1.%
2084 {%
2085   \ifnum #1>\xint_c_x^iv \xint_dothis{\XINT_fac_toobig #1.}\fi
2086   \ifnum #1>465 \xint_dothis{\XINT_fac_bigloop_a #1.}\fi
2087   \ifnum #1>101 \xint_dothis{\XINT_fac_medloop_a #1.\XINT_mul_out}\fi
2088   \xint_orthat{\XINT_fac_smallloop_a #1.\XINT_mul_out}%
2089   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W

```

```

2090 }%
2091 \def\XINT_fac_toobig #1.#2\W{\XINT_signalcondition{InvalidOperation}{Factorial
2092   of too big argument: #1 > 10000}}{0}}%
2093 \def\XINT_fac_bigloop_a #1.%
2094 {%
2095   \expandafter\XINT_fac_bigloop_b \the\numexpr
2096   #1+\xint_c_i-\xint_c_ii*(#1-464)/\xint_c_ii).#1.%
2097 }%
2098 \def\XINT_fac_bigloop_b #1.#2.%
2099 {%
2100   \expandafter\XINT_fac_medloop_a
2101     \the\numexpr #1-\xint_c_i.{\XINT_fac_bigloop_loop #1.#2.}%
2102 }%
2103 \def\XINT_fac_bigloop_loop #1.#2.%
2104 {%
2105   \ifnum #1>#2 \expandafter\XINT_fac_bigloop_exit\fi
2106   \expandafter\XINT_fac_bigloop_loop
2107   \the\numexpr #1+\xint_c_ii\expandafter.%
2108   \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_bigloop_mul #1!%
2109 }%
2110 \def\XINT_fac_bigloop_exit #1!{\XINT_mul_out}%
2111 \def\XINT_fac_bigloop_mul #1!%
2112 {%
2113   \expandafter\XINT_smallmul
2114     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2115 }%
2116 \def\XINT_fac_medloop_a #1.%
2117 {%
2118   \expandafter\XINT_fac_medloop_b
2119     \the\numexpr #1+\xint_c_i-\xint_c_iii*((#1-100)/\xint_c_iii).#1.%
2120 }%
2121 \def\XINT_fac_medloop_b #1.#2.%
2122 {%
2123   \expandafter\XINT_fac_smallloop_a
2124     \the\numexpr #1-\xint_c_i.{\XINT_fac_medloop_loop #1.#2.}%
2125 }%
2126 \def\XINT_fac_medloop_loop #1.#2.%
2127 {%
2128   \ifnum #1>#2 \expandafter\XINT_fac_loop_exit\fi
2129   \expandafter\XINT_fac_medloop_loop
2130   \the\numexpr #1+\xint_c_iii\expandafter.%
2131   \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_medloop_mul #1!%
2132 }%
2133 \def\XINT_fac_medloop_mul #1!%
2134 {%
2135   \expandafter\XINT_smallmul
2136   \the\numexpr
2137     \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2138 }%
2139 \def\XINT_fac_smallloop_a #1.%
2140 {%
2141   \csname

```

```

2142     XINT_fac_smallloop_\the\numexpr #1-\xint_c_iv*(#1/\xint_c_iv)\relax
2143   \endcsname #1.%
2144 }%
2145 \expandafter\def\csname XINT_fac_smallloop_1\endcsname #1.%
2146 {%
2147   \XINT_fac_smallloop_loop 2.#1.100000001!1;!%
2148 }%
2149 \expandafter\def\csname XINT_fac_smallloop_-2\endcsname #1.%
2150 {%
2151   \XINT_fac_smallloop_loop 3.#1.100000002!1;!%
2152 }%
2153 \expandafter\def\csname XINT_fac_smallloop_-1\endcsname #1.%
2154 {%
2155   \XINT_fac_smallloop_loop 4.#1.100000006!1;!%
2156 }%
2157 \expandafter\def\csname XINT_fac_smallloop_0\endcsname #1.%
2158 {%
2159   \XINT_fac_smallloop_loop 5.#1.1000000024!1;!%
2160 }%
2161 \def\XINT_fac_smallloop_loop #1.#2.%
2162 {%
2163   \ifnum #1>#2 \expandafter\XINT_fac_loop_exit\fi
2164   \expandafter\XINT_fac_smallloop_loop
2165   \the\numexpr #1+\xint_c_iv\expandafter.%
2166   \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_smallloop_mul #1!%
2167 }%
2168 \def\XINT_fac_smallloop_mul #1!%
2169 {%
2170   \expandafter\XINT_smallmul
2171   \the\numexpr
2172     \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2173 }%
2174 \def\XINT_fac_loop_exit #1!#2;!#3{#3#2;!}%

```

## 4.48 \XINT\_useiimessage

### 1.2o

```

2175 \def\XINT_useiimessage #1% used in LaTeX only
2176 {%
2177   \XINT_ifFlagRaised {#1}%
2178   {\@backslashchar#1
2179     (load xintfrac or use \@backslashchar xintii\xint_gobble_iv#1!)\MessageBreak}%
2180   }%
2181 }%
2182 \XINT_restorecatcodes_endinput%

```

## 5 Package [xint](#) implementation

.1	Package identification . . . . .	113	.33	<code>\xintiiifLt</code> . . . . .	125
.2	More token management . . . . .	113	.34	<code>\xintiiifZero</code> . . . . .	125
.3	(WIP) A constant needed by <code>\xintRandomDigits</code> et al. . . . .	113	.35	<code>\xintiiifNotZero</code> . . . . .	126
.4	<code>\xintLen</code> , <code>\xintiLen</code> . . . . .	114	.36	<code>\xintiiifOne</code> . . . . .	126
.5	<code>\xintiiLogTen</code> . . . . .	114	.37	<code>\xintiiifOdd</code> . . . . .	126
.6	<code>\xintReverseDigits</code> . . . . .	114	.38	<code>\xintifTrueAelseB</code> , <code>\xintifFalseAelseB</code> . . . . .	126
.7	<code>\xintiiE</code> . . . . .	115	.39	<code>\xintIsTrue</code> , <code>\xintIsFalse</code> . . . . .	127
.8	<code>\xintDecSplit</code> . . . . .	116	.40	<code>\xintNOT</code> . . . . .	127
.9	<code>\xintDecSplitL</code> . . . . .	117	.41	<code>\xintAND</code> , <code>\xintOR</code> , <code>\xintXOR</code> . . . . .	127
.10	<code>\xintDecSplitR</code> . . . . .	118	.42	<code>\xintANDof</code> . . . . .	127
.11	<code>\xintDSHr</code> . . . . .	118	.43	<code>\xintORof</code> . . . . .	128
.12	<code>\xintDSH</code> . . . . .	119	.44	<code>\xintXORof</code> . . . . .	128
.13	<code>\xintDSx</code> . . . . .	119	.45	<code>\xintiiMax</code> . . . . .	128
.14	<code>\xintiiEq</code> . . . . .	121	.46	<code>\xintiiMin</code> . . . . .	129
.15	<code>\xintiiNotEq</code> . . . . .	121	.47	<code>\xintiiMaxof</code> . . . . .	130
.16	<code>\xintiiGeq</code> . . . . .	121	.48	<code>\xintiiMinof</code> . . . . .	131
.17	<code>\xintiiGt</code> . . . . .	122	.49	<code>\xintiiSum</code> . . . . .	131
.18	<code>\xintiiLt</code> . . . . .	122	.50	<code>\xintiiPrd</code> . . . . .	131
.19	<code>\xintiiGtorEq</code> . . . . .	122	.51	<code>\xintiiSquareRoot</code> . . . . .	132
.20	<code>\xintiiLtorEq</code> . . . . .	122	.52	<code>\xintiiSqrt</code> , <code>\xintiiSqrtR</code> . . . . .	139
.21	<code>\xintiiIsZero</code> . . . . .	122	.53	<code>\xintiiBinomial</code> . . . . .	139
.22	<code>\xintiiIsNotZero</code> . . . . .	122	.54	<code>\xintiiPfactorial</code> . . . . .	145
.23	<code>\xintiiIsOne</code> . . . . .	122	.55	<code>\xintBool</code> , <code>\xintToggle</code> . . . . .	148
.24	<code>\xintiiOdd</code> . . . . .	123	.56	<code>\xintGCD</code> , <code>\xintiiGCD</code> . . . . .	148
.25	<code>\xintiiEven</code> . . . . .	123	.57	<code>\xintLCM</code> , <code>\xintiiLCM</code> . . . . .	148
.26	<code>\xintiiMON</code> . . . . .	123	.58	(WIP) <code>\xintRandomDigits</code> . . . . .	149
.27	<code>\xintiiMMON</code> . . . . .	123	.59	(WIP) <code>\XINT_eightrandomdigits</code> . . . . .	150
.28	<code>\xintSgnFork</code> . . . . .	124	.60	(WIP) <code>\xintXRandomDigits</code> . . . . .	150
.29	<code>\xintiiifSgn</code> . . . . .	124	.61	(WIP) <code>\xintiiRandRangeAtoB</code> . . . . .	150
.30	<code>\xintiiifCmp</code> . . . . .	124	.62	(WIP) <code>\xintiiRandRange</code> . . . . .	151
.31	<code>\xintiiifEq</code> . . . . .	125	.63	Adjustments for engines without uniformdeviate primitive . . . . .	152
.32	<code>\xintiiifGt</code> . . . . .	125			

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](#).

At 1.3 the macros deprecated at 1.2o got all removed.

1.3b adds randomness related macros.

```

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)

```

## 5.1 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xint}%
46 [2019/04/05 1.3e Expandable operations on big integers (JFB)]%

```

## 5.2 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_stop_atfirstofthree #1#2#3{ #1}%
51 \long\def\xint_stop_atsecondofthree #1#2#3{ #2}%
52 \long\def\xint_stop_atthirdofthree #1#2#3{ #3}%

```

## 5.3 (WIP) A constant needed by `\xintRandomDigits` et al.

```

53 \ifdefined\xint_texuniformdeviate
54   \unless\ifdefined\xint_c_nine_x^viii
55     \csname newcount\endcsname\xint_c_nine_x^viii
56     \xint_c_nine_x^viii 9000000000
57   \fi
58 \fi

```

## 5.4 `\xintLen`, `\xintiLen`

`\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`.

`\xintLen` gets redefined by `xintfrac`.

```

59 \def\xintiLen {\romannumeral0\xintilen }%
60 \def\xintilen #1{\def\xintilen ##1%
61 {%
62   \expandafter#1\the\numexpr
63   \expandafter\XINT_len_fork\romannumeral0\xintinum{##1}%
64   \xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
65   \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
66   \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye\relax
67 }}\xintilen{ }%
68 \def\xintLen {\romannumeral0\xintlen }%
69 \let\xintlen\xintilen

70 \def\XINT_len_fork #1%
71 {%
72   \expandafter\XINT_length_loop\xint_UDsignfork#1{-#1\krof
73 }%

```

## 5.5 `\xintiLogTen`

1.3e. Support for `ilog10()` function in `\xintiexpr`. See `\XINTiLogTen` in `xintfrac.sty` which also currently uses `-"7FFF8000` as value if input is zero.

```

74 \def\xintiLogTen {\the\numexpr\xintiilogten }%
75 \def\xintiilogten #1%
76 {%
77   \expandafter\XINT_iilogten\romannumeral`&&@#1%
78   \xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
79   \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
80   \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
81   \relax
82 }%
83 \def\XINT_iilogten #1{\if#10-"7FFF8000\fi -1+%
84   \expandafter\XINT_length_loop\xint_UDsignfork#1{-#1\krof}%

```

## 5.6 `\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.

```

85 \def\xintReverseDigits {\romannumeral0\xintreversedigits }%
86 \def\xintreversedigits #1%

```

```

87 {%
88   \expandafter\XINT_revdigits\romannumeral`&&@#1%
89   {\XINT_microrevsep_end\W}\XINT_microrevsep_end
90   \XINT_microrevsep_end\XINT_microrevsep_end
91   \XINT_microrevsep_end\XINT_microrevsep_end
92   \XINT_microrevsep_end\XINT_microrevsep_end\XINT_microrevsep_end\Z
93   1\Z!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
94 }%
95 \def\XINT_revdigits #1%
96 {%
97   \xint_UDsignfork
98   #1{\expandafter-\romannumeral0\XINT_revdigits_a}%
99   -{\XINT_revdigits_a #1}%
100  \krof
101 }%
102 \def\XINT_revdigits_a
103 {%
104   \expandafter\XINT_revdigits_b\expandafter{\expandafter}%
105   \the\numexpr\XINT_microrevsep
106 }%
107 \def\XINT_microrevsep #1#2#3#4#5#6#7#8#9%
108 {%
109   1#9#8#7#6#5#4#3#2#1\expandafter!\the\numexpr\XINT_microrevsep
110 }%
111 \def\XINT_microrevsep_end #1\W #2\expandafter #3\Z{\relax#2!}%
112 \def\XINT_revdigits_b #1#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
113 {%
114   \xint_gob_til_R #9\XINT_revdigits_end\R
115   \XINT_revdigits_b {#9#8#7#6#5#4#3#2#1}%
116 }%
117 \def\XINT_revdigits_end#1{%
118 \def\XINT_revdigits_end\R\XINT_revdigits_b ##1##2\W
119  {\expandafter#1\xint_gob_til_Z ##1}%
120 }\XINT_revdigits_end{ }%
121 \let\xintRev\xintReverseDigits

```

## 5.7 \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.

```

122 \def\xintiiE {\romannumeral0\xintiiE }%
123 \def\xintiiE #1#2%
124  {\expandafter\XINT_iiE_fork\the\numexpr #2\expandafter.\romannumeral`&&@#1;}%
125 \def\XINT_iiE_fork #1%
126 {%
127   \xint_UDsignfork
128   #1\XINT_iiE_neg
129   -\XINT_iiE_a
130   \krof #1%
131 }%

```

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

```
132 \def\XINT_iae_a #1.%
133 {\expandafter\XINT_dsx_append\romannumeral\XINT_rep #1\endcsname 0.}%
134 \def\XINT_iae_neg #1.#2;{ #2}%
```

## 5.8 \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.

```
135 \def\xintDecSplit {\romannumeral0\xintdecsplit }%
136 \def\xintdecsplit #1#2%
137 {%
138   \expandafter\XINT_split_finish
139   \romannumeral0\expandafter\XINT_split_xfork
140   \the\numexpr #1\expandafter.\romannumeral`&&@#2%
141   \xint_bye2345678\xint_bye..%
142 }%
143 \def\XINT_split_finish #1.#2.{{#1}{#2}}%

144 \def\XINT_split_xfork #1%
145 {%
146   \xint_UDzerominusfork
147   #1-\XINT_split_zerosplit
148   0#1\XINT_split_fromleft
149   0-\XINT_split_fromright #1}%
150 \krof
151 }%
152 \def\XINT_split_zerosplit .#1\xint_bye#2\xint_bye..{ #1..}%
153 \def\XINT_split_fromleft
154   {\expandafter\XINT_split_fromleft_a\the\numexpr\xint_c_viii-}%
155 \def\XINT_split_fromleft_a #1%
156 {%
157   \xint_UDsignfork
158   #1\XINT_split_fromleft_b
159   -{\XINT_split_fromleft_end_a #1}%
160 \krof
161 }%
162 \def\XINT_split_fromleft_b #1.#2#3#4#5#6#7#8#9%
163 {%
164   \expandafter\XINT_split_fromleft_clean
165   \the\numexpr1#2#3#4#5#6#7#8#9\expandafter
```

```

166 \XINT_split_fromleft_a\the\numexpr\xint_c_viii-#1.%
167 }%
168 \def\XINT_split_fromleft_end_a #1.%
169 {%
170 \expandafter\XINT_split_fromleft_clean
171 \the\numexpr1\csname XINT_split_fromleft_end#1\endcsname
172 }%
173 \def\XINT_split_fromleft_clean 1{ }%
174 \expandafter\def\csname XINT_split_fromleft_end7\endcsname #1%
175 {#1\XINT_split_fromleft_end_b}%
176 \expandafter\def\csname XINT_split_fromleft_end6\endcsname #1#2%
177 {#1#2\XINT_split_fromleft_end_b}%
178 \expandafter\def\csname XINT_split_fromleft_end5\endcsname #1#2#3%
179 {#1#2#3\XINT_split_fromleft_end_b}%
180 \expandafter\def\csname XINT_split_fromleft_end4\endcsname #1#2#3#4%
181 {#1#2#3#4\XINT_split_fromleft_end_b}%
182 \expandafter\def\csname XINT_split_fromleft_end3\endcsname #1#2#3#4#5%
183 {#1#2#3#4#5\XINT_split_fromleft_end_b}%
184 \expandafter\def\csname XINT_split_fromleft_end2\endcsname #1#2#3#4#5#6%
185 {#1#2#3#4#5#6\XINT_split_fromleft_end_b}%
186 \expandafter\def\csname XINT_split_fromleft_end1\endcsname #1#2#3#4#5#6#7%
187 {#1#2#3#4#5#6#7\XINT_split_fromleft_end_b}%
188 \expandafter\def\csname XINT_split_fromleft_end0\endcsname #1#2#3#4#5#6#7#8%
189 {#1#2#3#4#5#6#7#8\XINT_split_fromleft_end_b}%

190 \def\XINT_split_fromleft_end_b #1\xint_bye#2\xint_bye.{.#1}% puis .
191 \def\XINT_split_fromright #1.#2\xint_bye
192 {%
193 \expandafter\XINT_split_fromright_a
194 \the\numexpr#1-\numexpr\XINT_length_loop
195 #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
196 \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
197 \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
198 .#2\xint_bye
199 }%

200 \def\XINT_split_fromright_a #1%
201 {%
202 \xint_UDsignfork
203 #1\XINT_split_fromleft
204 -\XINT_split_fromright_Lempty
205 \krof
206 }%
207 \def\XINT_split_fromright_Lempty #1.#2\xint_bye#3..{.#2.}%

```

## 5.9 \xintDecSplitL

```

208 \def\xintDecSplitL {\romannumeral0\xintdecsplitl }%
209 \def\xintdecsplitl #1#2%
210 {%
211 \expandafter\XINT_splitl_finish
212 \romannumeral0\expandafter\XINT_split_xfork

```

```

213 \the\numexpr #1\expandafter.\romannumeral`&&@#2%
214 \xint_bye2345678\xint_bye..%
215 }%
216 \def\xINT_splitl_finish #1.#2.{ #1}%

```

## 5.10 \xintDecSplitR

```

217 \def\xintDecSplitR {\romannumeral0\xintdecsplitr }%
218 \def\xintdecsplitr #1#2%
219 {%
220 \expandafter\xINT_splitr_finish
221 \romannumeral0\expandafter\xINT_split_xfork
222 \the\numexpr #1\expandafter.\romannumeral`&&@#2%
223 \xint_bye2345678\xint_bye..%
224 }%
225 \def\xINT_splitr_finish #1.#2.{ #2}%

```

## 5.11 \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.

```

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

```

```

227 \def\xintdshr #1#2%
228 {%

```

```

229 \expandafter\xINT_dshr_fork\the\numexpr#1\expandafter.\romannumeral`&&@#2;%
230 }%

```

```

231 \def\xINT_dshr_fork #1%

```

```

232 {%

```

```

233 \xint_UDzerominusfork

```

```

234 0#\XINT_dshr_xzeroorneg

```

```

235 #1-\XINT_dshr_xzeroorneg

```

```

236 0-\XINT_dshr_xpositive

```

```

237 \krof #1%

```

```

238 }%

```

```

239 \def\xINT_dshr_xzeroorneg #1;{ 0}%

```

```

240 \def\xINT_dshr_xpositive

```

```

241 {%

```

```

242 \expandafter\xint_stop_atsecondoftwo\romannumeral0\xINT_dsx_xisPos

```

```

243 }%

```

## 5.12 \xintDSH

```

244 \def\xintDSH {\romannumeral0\xintdsh }%
245 \def\xintdsh #1#2%
246 {%

247     \expandafter\XINT_dsh_fork\the\numexpr#1\expandafter.\romannumeral`&&@#2;%
248 }%
249 \def\XINT_dsh_fork #1%
250 {%
251     \xint_UDzerominusfork
252     #1-\XINT_dsh_xiszero

253     0#1\XINT_dsx_xisNeg_checkA
254     0-\XINT_dsh_xisPos #1}%
255     \krof
256 }%
257 \def\XINT_dsh_xiszero #1.#2;{ #2}%
258 \def\XINT_dsh_xisPos
259 {%

    \expandafter\xint_stop_atfirstoftwo\romannumeral0\XINT_dsx_xisPos
260 }%

```

## 5.13 \xintDSx

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

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

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

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

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

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

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

Rewritten for 1.2i, this was old code.

```

261 \def\xintDSx {\romannumeral0\xintdsx }%
262 \def\xintdsx #1#2%
263 {%

264     \expandafter\XINT_dsx_fork\the\numexpr#1\expandafter.\romannumeral`&&@#2;%
265 }%
266 \def\XINT_dsx_fork #1%
267 {%
268     \xint_UDzerominusfork
269     #1-\XINT_dsx_xisZero
270     0#1\XINT_dsx_xisNeg_checkA
271     0-\XINT_dsx_xisPos #1}%
272     \krof
273 }%
274 \def\XINT_dsx_xisZero #1.#2;{{#2}{0}}%
275 \def\XINT_dsx_xisNeg_checkA #1.#2%
276 {%
277     \xint_gob_til_zero #2\XINT_dsx_xisNeg_Azero 0%

```

```

278 \expandafter\XINT_dsx_append\romannumeral\XINT_rep #1\endcsname 0.#2%
279 }%
280 \def\XINT_dsx_xisNeg_Azero #1;{ 0}%

281 \def\XINT_dsx_addzeros #1%
282 {\expandafter\XINT_dsx_append\romannumeral\XINT_rep#1\endcsname0.}%

283 \def\XINT_dsx_addzerosnofuss #1%
284 {\expandafter\XINT_dsx_append\romannumeral\xintreplicate{#1}0.}%
285 \def\XINT_dsx_append #1.#2;{ #2#1}%

286 \def\XINT_dsx_xisPos #1.#2%
287 {%
288 \xint_UDzerominusfork
289 #2-\XINT_dsx_AisZero
290 0#2\XINT_dsx_AisNeg
291 0-\XINT_dsx_AisPos
292 \krof #1.#2%
293 }%
294 \def\XINT_dsx_AisZero #1;{{0}{0}}%
295 \def\XINT_dsx_AisNeg #1.-#2;%
296 {%
297 \expandafter\XINT_dsx_AisNeg_checkiffirstempty
298 \romannumeral0\XINT_split_xfork #1.#2\xint_bye2345678\xint_bye..%
299 }%

300 \def\XINT_dsx_AisNeg_checkiffirstempty #1%
301 {%
302 \xint_gob_til_dot #1\XINT_dsx_AisNeg_finish_zero.%
303 \XINT_dsx_AisNeg_finish_notzero #1%
304 }%
305 \def\XINT_dsx_AisNeg_finish_zero.\XINT_dsx_AisNeg_finish_notzero.#1.%
306 {%
307 \expandafter\XINT_dsx_end
308 \expandafter {\romannumeral0\XINT_num {-#1}}{0}%
309 }%
310 \def\XINT_dsx_AisNeg_finish_notzero #1.#2.%
311 {%
312 \expandafter\XINT_dsx_end
313 \expandafter {\romannumeral0\XINT_num {#2}}{-#1}%
314 }%

315 \def\XINT_dsx_AisPos #1.#2;%
316 {%
317 \expandafter\XINT_dsx_AisPos_finish
318 \romannumeral0\XINT_split_xfork #1.#2\xint_bye2345678\xint_bye..%
319 }%

320 \def\XINT_dsx_AisPos_finish #1.#2.%
321 {%

```



```

322 \expandafter\XINT_dsx_end
323 \expandafter {\romannumeral0\XINT_num {#2}}%
324 {\romannumeral0\XINT_num {#1}}%
325 }%
326 \def\XINT_dsx_end #1#2{\expandafter{#2}{#1}}%

```

## 5.14 \xintiiEq

no \xintiieq.

```

327 \def\xintiiEq #1#2{\romannumeral0\xintiifeq{#1}{#2}{1}{0}}%

```

## 5.15 \xintiiNotEq

Pour xintexpr. Pas de version en lowercase.

```

328 \def\xintiiNotEq #1#2{\romannumeral0\xintiifeq {#1}{#2}{0}{1}}%

```

## 5.16 \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.

```

329 \def\xintiiGeq {\romannumeral0\xintiigeq }%
330 \def\xintiigeq #1{\expandafter\XINT_iigeq\romannumeral`&&@#1\xint:}%
331 \def\XINT_iigeq #1#2\xint:#3%
332 {%
333 \expandafter\XINT_geq_fork\expandafter #1\romannumeral`&&@#3\xint:#2\xint:
334 }%

335 \def\XINT_geq #1#2\xint:#3%
336 {%
337 \expandafter\XINT_geq_fork\expandafter #1\romannumeral0\xintnum{#3}\xint:#2\xint:
338 }%
339 \def\XINT_geq_fork #1#2%
340 {%
341 \xint_UDzerofork
342 #1\XINT_geq_firstiszero
343 #2\XINT_geq_secondiszero
344 0{}}%
345 \krof
346 \xint_UDsignsfork
347 #1#2\XINT_geq_minusminus
348 #1-\XINT_geq_minusplus
349 #2-\XINT_geq_plusminus
350 --\XINT_geq_plusplus
351 \krof #1#2%
352 }%
353 \def\XINT_geq_firstiszero #1\krof 0#2#3\xint:#4\xint:
354 {\xint_UDzerofork #2{ 1}0{ 0}\krof }%

```

```

355 \def\XINT_geq_secondiszero #1\krof #20#3\xint:#4\xint:{ 1}%
356 \def\XINT_geq_plusminus #1-{\XINT_geq_plusplus #1{}}%
357 \def\XINT_geq_minusplus -#1{\XINT_geq_plusplus { }#1}%
358 \def\XINT_geq_minusminus --{\XINT_geq_plusplus { }{}}%
359 \def\XINT_geq_plusplus
360   {\expandafter\XINT_geq_finish\romannumeral0\XINT_cmp_plusplus}%
361 \def\XINT_geq_finish #1{\if-#1\expandafter\XINT_geq_no
362   \else\expandafter\XINT_geq_yes\fi}%
363 \def\XINT_geq_no 1{ 0}%
364 \def\XINT_geq_yes { 1}%

```

## 5.17 \xintiiGt

```

365 \def\xintiiGt #1#2{\romannumeral0\xintiiifgt{#1}{#2}{1}{0}}%

```

## 5.18 \xintiiLt

```

366 \def\xintiiLt #1#2{\romannumeral0\xintiiiflt{#1}{#2}{1}{0}}%

```

## 5.19 \xintiiGtorEq

```

367 \def\xintiiGtorEq #1#2{\romannumeral0\xintiiiflt {#1}{#2}{0}{1}}%

```

## 5.20 \xintiiLtorEq

```

368 \def\xintiiLtorEq #1#2{\romannumeral0\xintiiifgt {#1}{#2}{0}{1}}%

```

## 5.21 \xintiiIsZero

1.09a. restyled in 1.09i. 1.1 adds \xintiiIsZero, etc... for optimization in [xintexpr](#)

```

369 \def\xintiiIsZero {\romannumeral0\xintiiiszero }%
370 \def\xintiiiszero #1{\if0\xintiiSgn{#1}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%

```

## 5.22 \xintiiIsNotZero

1.09a. restyled in 1.09i. 1.1 adds \xintiiIsZero, etc... for optimization in [xintexpr](#)

```

371 \def\xintiiIsNotZero {\romannumeral0\xintiiisnotzero }%
372 \def\xintiiisnotzero
373   #1{\if0\xintiiSgn{#1}\xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi}%

```

## 5.23 \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.

```

374 \def\xintiiIsOne {\romannumeral0\xintiiisone }%
375 \def\xintiiisone #1{\expandafter\XINT_isone\romannumeral`&&@#1XY}%
376 \def\XINT_isone #1#2#3Y%
377 {%
378   \unless\if#2X\xint_dothis{ 0}\fi
379   \unless\if#11\xint_dothis{ 0}\fi
380   \xint_orthat{ 1}%
381 }%
382 \def\XINT_isOne #1{\XINT_is_One#1XY}%

```

```

383 \def\XINT_is_One #1#2#3Y%
384 {%
385   \unless\if#2X\xint_dothis0\fi
386   \unless\if#11\xint_dothis0\fi
387   \xint_orthat1%
388 }%

```

## 5.24 \xintiiOdd

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

```

389 \def\xintiiOdd {\romannumeral0\xintiiodd }%
390 \def\xintiiodd #1%
391 {%
392   \ifodd\xintLDg{#1} %<- intentional space
393   \xint_afterfi{ 1}%
394   \else
395   \xint_afterfi{ 0}%
396   \fi
397 }%

```

## 5.25 \xintiiEven

```

398 \def\xintiiEven {\romannumeral0\xintiieven }%
399 \def\xintiieven #1%
400 {%
401   \ifodd\xintLDg{#1} %<- intentional space
402   \xint_afterfi{ 0}%
403   \else
404   \xint_afterfi{ 1}%
405   \fi
406 }%

```

## 5.26 \xintiiMON

MINUS ONE TO THE POWER N

```

407 \def\xintiiMON {\romannumeral0\xintiimon }%
408 \def\xintiimon #1%
409 {%
410   \ifodd\xintLDg {#1} %<- intentional space
411   \xint_afterfi{ -1}%
412   \else
413   \xint_afterfi{ 1}%
414   \fi
415 }%

```

## 5.27 \xintiiMMON

MINUS ONE TO THE POWER N-1

```

416 \def\xintiiMMON {\romannumeral0\xintiimmon }%
417 \def\xintiimmon #1%
418 {%

```

```

419 \ifodd\xintLDg {#1} %<- intentional space
420 \xint_afterfi{ 1}%
421 \else
422 \xint_afterfi{ -1}%
423 \fi
424 }%

```

## 5.28 `\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` (now `_stop_at...`).

```

425 \def\xintSgnFork {\romannumeral0\xintsngfork }%
426 \def\xintsngfork #1%
427 {%
428 \ifcase #1 \expandafter\xint_stop_atsecondofthree
429 \or\expandafter\xint_stop_atthirdofthree
430 \else\expandafter\xint_stop_atfirstofthree
431 \fi
432 }%

```

## 5.29 `\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 `\xint_stop_atfirstofthree`) etc for faster expansion.

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

```

433 \def\xintiiifSgn {\romannumeral0\xintiiifsgn }%
434 \def\xintiiifsgn #1%
435 {%
436 \ifcase \xintiiSgn{#1}
437 \expandafter\xint_stop_atsecondofthree
438 \or\expandafter\xint_stop_atthirdofthree
439 \else\expandafter\xint_stop_atfirstofthree
440 \fi
441 }%

```

## 5.30 `\xintiiifCmp`

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

```

442 \def\xintiiifCmp {\romannumeral0\xintiiifcmp }%
443 \def\xintiiifcmp #1#2%
444 {%
445 \ifcase\xintiiCmp {#1}{#2}
446 \expandafter\xint_stop_atsecondofthree
447 \or\expandafter\xint_stop_atthirdofthree
448 \else\expandafter\xint_stop_atfirstofthree
449 \fi
450 }%

```

### 5.31 `\xintiiifEq`

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

```
451 \def\xintiiifEq {\romannumeral0\xintiiifeq }%
452 \def\xintiiifeq #1#2%
453 {%
454   \if0\xintiiCmp{#1}{#2}%
455     \expandafter\xint_stop_atfirstoftwo
456   \else\expandafter\xint_stop_atsecondoftwo
457   \fi
458 }%
```

### 5.32 `\xintiiifGt`

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

```
459 \def\xintiiifGt {\romannumeral0\xintiiifgt }%
460 \def\xintiiifgt #1#2%
461 {%
462   \if1\xintiiCmp{#1}{#2}%
463     \expandafter\xint_stop_atfirstoftwo
464   \else\expandafter\xint_stop_atsecondoftwo
465   \fi
466 }%
```

### 5.33 `\xintiiifLt`

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

```
467 \def\xintiiifLt {\romannumeral0\xintiiiflt }%
468 \def\xintiiiflt #1#2%
469 {%
470   \ifnum\xintiiCmp{#1}{#2}<\xint_c_
471     \expandafter\xint_stop_atfirstoftwo
472   \else \expandafter\xint_stop_atsecondoftwo
473   \fi
474 }%
```

### 5.34 `\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`.

```
475 \def\xintiiifZero {\romannumeral0\xintiiifzero }%
476 \def\xintiiifzero #1%
477 {%
478   \if0\xintiiSgn{#1}%
479     \expandafter\xint_stop_atfirstoftwo
480   \else
481     \expandafter\xint_stop_atsecondoftwo
```

```
482 \fi
483 }%
```

### 5.35 `\xintiiifNotZero`

```
484 \def\xintiiifNotZero {\romannumeral0\xintiiifnotzero }%
485 \def\xintiiifnotzero #1%
486 {%
487 \if0\xintiiSgn{#1}%
488 \expandafter\xint_stop_atsecondoftwo
489 \else
490 \expandafter\xint_stop_atfirstoftwo
491 \fi
492 }%
```

### 5.36 `\xintiiifOne`

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

```
493 \def\xintiiifOne {\romannumeral0\xintiiifone }%
494 \def\xintiiifone #1%
495 {%
496 \if1\xintiiIsOne{#1}%
497 \expandafter\xint_stop_atfirstoftwo
498 \else
499 \expandafter\xint_stop_atsecondoftwo
500 \fi
501 }%
```

### 5.37 `\xintiiifOdd`

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

```
502 \def\xintiiifOdd {\romannumeral0\xintiiifodd }%
503 \def\xintiiifodd #1%
504 {%
505 \if\xintiiOdd{#1}1%
506 \expandafter\xint_stop_atfirstoftwo
507 \else
508 \expandafter\xint_stop_atsecondoftwo
509 \fi
510 }%
```

### 5.38 `\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.

```
511 \def\xintifTrueAelseB {\romannumeral0\xintiiifnotzero}%
512 \def\xintifFalseAelseB{\romannumeral0\xintiiifzero}%
```

### 5.39 `\xintIsTrue`, `\xintIsFalse`

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

```
513 %\let\xintIsTrue \xintIsNotZero
514 %\let\xintIsFalse\xintIsZero
```

### 5.40 `\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.

```
515 \def\xintNOT{\romannumeral0\xintiiszero}%
```

### 5.41 `\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.

```
516 \def\xintAND {\romannumeral0\xintand }%
517 \def\xintand #1#2{\if0\xintiiSgn{#1}\expandafter\xint_firstoftwo
518             \else\expandafter\xint_secondoftwo\fi
519             { 0}{\xintiiisnotzero{#2}}}%
520 \def\xintOR {\romannumeral0\xintor }%
521 \def\xintor #1#2{\if0\xintiiSgn{#1}\expandafter\xint_firstoftwo
522             \else\expandafter\xint_secondoftwo\fi
523             {\xintiiisnotzero{#2}}{ 1}}}%
524 \def\xintXOR {\romannumeral0\xintxor }%
525 \def\xintxor #1#2{\if\xintiiIsZero{#1}\xintiiIsZero{#2}%
526             \xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi }%
```

### 5.42 `\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.

```
527 \def\xintANDof      {\romannumeral0\xintandof }%
528 \def\xintandof     #1{\expandafter\XINT_andof_a\romannumeral`&&@#1\xint:}%
529 \def\XINT_andof_a #1{\expandafter\XINT_andof_b\romannumeral`&&@#1!}%
530 \def\XINT_andof_b #1%
531             {\xint_gob_til_xint: #1\XINT_andof_e\xint:\XINT_andof_c #1}%
532 \def\XINT_andof_c #1!%
533             {\xintifTrueAelseB {#1}{\XINT_andof_a}{\XINT_andof_no}}%
534 \def\XINT_andof_no #1\xint:{ 0}%
535 \def\XINT_andof_e #1!{ 1}%
```

### 5.43 `\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.

```
536 \def\xintORof      {\romannumeral0\xintorof }%
537 \def\xintorof     #1{\expandafter\XINT_orof_a\romannumeral`&&@#1\xint:}%
538 \def\XINT_orof_a  #1{\expandafter\XINT_orof_b\romannumeral`&&@#1!}%
539 \def\XINT_orof_b  #1%
540      {\xint_gob_til_xint: #1\XINT_orof_e\xint:\XINT_orof_c #1}%
541 \def\XINT_orof_c  #1!%
542      {\xintifTrueAelseB {#1}{\XINT_orof_yes}{\XINT_orof_a}}%
543 \def\XINT_orof_yes #1\xint:{ 1}%
544 \def\XINT_orof_e  #1!{ 0}%
```

### 5.44 `\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.

```
545 \def\xintXORof     {\romannumeral0\xintxorof }%
546 \def\xintxorof     #1{\expandafter\XINT_xorof_a\expandafter
547      0\romannumeral`&&@#1\xint:}%
548 \def\XINT_xorof_a  #1#2{\expandafter\XINT_xorof_b\romannumeral`&&@#2!#1}%
549 \def\XINT_xorof_b  #1%
550      {\xint_gob_til_xint: #1\XINT_xorof_e\xint:\XINT_xorof_c #1}%
551 \def\XINT_xorof_c  #1!#2%
552      {\xintifTrueAelseB {#1}{\if #20\xint_afterfi{\XINT_xorof_a 1}%
553      \else\xint_afterfi{\XINT_xorof_a 0}\fi}%
554      {\XINT_xorof_a #2}%
555      }%
556 \def\XINT_xorof_e  #1!#2{ #2}%
```

### 5.45 `\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.

```
557 \def\xintiiMax {\romannumeral0\xintiimax }%
558 \def\xintiimax #1%
559 {%
560   \expandafter\xint_iimax \romannumeral`&&@#1\xint:
561 }%
562 \def\xint_iimax #1\xint:#2%
563 {%
564   \expandafter\XINT_max_fork\romannumeral`&&@#2\xint:#1\xint:
565 }%
```

`#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.



```

566 \def\XINT_max_fork #1#2\xint:#3#4\xint:
567 {%
568   \xint_UDsignsfork
569     #1#3\XINT_max_minusminus % A < 0, B < 0
570     #1-\XINT_max_plusminus % B < 0, A >= 0
571     #3-\XINT_max_minusplus % A < 0, B >= 0
572     --{\xint_UDzerosfork
573       #1#3\XINT_max_zerozero % A = B = 0
574       #10\XINT_max_pluszero % B = 0, A > 0
575       #30\XINT_max_zeroplus % A = 0, B > 0
576       00\XINT_max_plusplus % A, B > 0
577     \krof }%
578   \krof
579   #3#1#2\xint:#4\xint:
580   \expandafter\xint_stop_atfirstoftwo
581   \else
582   \expandafter\xint_stop_atsecondoftwo
583   \fi
584   {#3#4}{#1#2}%
585 }%

```

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

```

586 \def\XINT_max_zerozero #1\fi{\xint_stop_atfirstoftwo }%
587 \def\XINT_max_zeroplus #1\fi{\xint_stop_atsecondoftwo }%
588 \def\XINT_max_pluszero #1\fi{\xint_stop_atfirstoftwo }%
589 \def\XINT_max_minusplus #1\fi{\xint_stop_atsecondoftwo }%
590 \def\XINT_max_plusminus #1\fi{\xint_stop_atfirstoftwo }%
591 \def\XINT_max_plusplus
592 {%
593   \if1\romannumeral0\XINT_geq_plusplus
594 }%

```

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.

```

595 \def\XINT_max_minusminus --%
596 {%
597   \unless\if1\romannumeral0\XINT_geq_plusplus{}}%
598 }%

```

## 5.46 `\xintiiMin`

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

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.

```

599 \def\xintiiMin {\romannumeral0\xintiimin }%
600 \def\xintiimin #1%
601 {%

```

```

602 \expandafter\xint_iimin \romannumeral`&&@#1\xint:
603 }%
604 \def\xint_iimin #1\xint:#2%
605 {%
606 \expandafter\XINT_min_fork\romannumeral`&&@#2\xint:#1\xint:
607 }%
608 \def\XINT_min_fork #1#2\xint:#3#4\xint:
609 {%
610 \xint_UDsignsfork
611 #1#3\XINT_min_minusminus % A < 0, B < 0
612 #1-\XINT_min_plusminus % B < 0, A >= 0
613 #3-\XINT_min_minusplus % A < 0, B >= 0
614 --{\xint_UDzerosfork
615 #1#3\XINT_min_zerozero % A = B = 0
616 #10\XINT_min_pluszero % B = 0, A > 0
617 #30\XINT_min_zeroplus % A = 0, B > 0
618 00\XINT_min_plusplus % A, B > 0
619 \krof }%
620 \krof
621 #3#1#2\xint:#4\xint:
622 \expandafter\xint_stop_atsecondoftwo
623 \else
624 \expandafter\xint_stop_atfirstoftwo
625 \fi
626 {#3#4}{#1#2}%
627 }%
628 \def\XINT_min_zerozero #1\fi{\xint_stop_atfirstoftwo }%
629 \def\XINT_min_zeroplus #1\fi{\xint_stop_atfirstoftwo }%
630 \def\XINT_min_pluszero #1\fi{\xint_stop_atsecondoftwo }%
631 \def\XINT_min_minusplus #1\fi{\xint_stop_atfirstoftwo }%
632 \def\XINT_min_plusminus #1\fi{\xint_stop_atsecondoftwo }%
633 \def\XINT_min_plusplus
634 {%
635 \if1\romannumeral0\XINT_geq_plusplus
636 }%
637 \def\XINT_min_minusminus --%
638 {%
639 \unless\if1\romannumeral0\XINT_geq_plusplus{}}%
640 }%

```

## 5.47 \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.

```

641 \def\xintiiMaxof {\romannumeral0\xintiimaxof }%
642 \def\xintiimaxof #1{\expandafter\XINT_iimaxof_a\romannumeral`&&@#1\xint:}%
643 \def\XINT_iimaxof_a #1{\expandafter\XINT_iimaxof_b\romannumeral`&&@#1!}%
644 \def\XINT_iimaxof_b #1!#2%
645 {\expandafter\XINT_iimaxof_c\romannumeral`&&@#2!{#1!}%
646 \def\XINT_iimaxof_c #1%

```

```

647      {\xint_gob_til_xint: #1\XINT_iimaxof_e\xint:\XINT_iimaxof_d #1}%
648 \def\XINT_iimaxof_d #1!%
649      {\expandafter\XINT_iimaxof_b\romannumeral0\xintiimax {#1}}%
650 \def\XINT_iimaxof_e #1!#2!{ #2}%

```

## 5.48 \xintiiMinof

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

```

651 \def\xintiiMinof      {\romannumeral0\xintiiminof }%
652 \def\xintiiminof     #1{\expandafter\XINT_iiminof_a\romannumeral`&&@#1\xint:}%
653 \def\XINT_iiminof_a #1{\expandafter\XINT_iiminof_b\romannumeral`&&@#1!}%
654 \def\XINT_iiminof_b #1!#2%
655      {\expandafter\XINT_iiminof_c\romannumeral`&&@#2!{#1}!}%
656 \def\XINT_iiminof_c #1%
657      {\xint_gob_til_xint: #1\XINT_iiminof_e\xint:\XINT_iiminof_d #1}%
658 \def\XINT_iiminof_d #1!%
659      {\expandafter\XINT_iiminof_b\romannumeral0\xintiimin {#1}}%
660 \def\XINT_iiminof_e #1!#2!{ #2}%

```

## 5.49 \xintiiSum

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

```

661 \def\xintiiSum {\romannumeral0\xintiisum }%
662 \def\xintiisum #1{\expandafter\XINT_sumexpr\romannumeral`&&@#1\xint:}%
663 \def\XINT_sumexpr {\XINT_sum_loop_a 0\Z }%
664 \def\XINT_sum_loop_a #1\Z #2%
665      {\expandafter\XINT_sum_loop_b \romannumeral`&&@#2\xint:#1\xint:\Z}%
666 \def\XINT_sum_loop_b #1%
667      {\xint_gob_til_xint: #1\XINT_sum_finished\xint:\XINT_sum_loop_c #1}%
668 \def\XINT_sum_loop_c
669      {\expandafter\XINT_sum_loop_a\romannumeral0\XINT_add_fork }%
670 \def\XINT_sum_finished\xint:\XINT_sum_loop_c\xint:\xint:#1\xint:\Z{ #1}%

```

## 5.50 \xintiiPrd

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

```

671 \def\xintiiPrd {\romannumeral0\xintiiprd }%
672 \def\xintiiprd #1{\expandafter\XINT_prdexpr\romannumeral`&&@#1\xint:}%
673 \def\XINT_prdexpr {\XINT_prod_loop_a 1\Z }%
674 \def\XINT_prod_loop_a #1\Z #2%
675      {\expandafter\XINT_prod_loop_b\romannumeral`&&@#2\xint:#1\xint:\Z}%
676 \def\XINT_prod_loop_b #1%
677      {\xint_gob_til_xint: #1\XINT_prod_finished\xint:\XINT_prod_loop_c #1}%
678 \def\XINT_prod_loop_c
679      {\expandafter\XINT_prod_loop_a\romannumeral0\XINT_mul_fork }%
680 \def\XINT_prod_finished\xint:\XINT_prod_loop_c\xint:\xint:#1\xint:\Z { #1}%

```

## 5.51 `\xintiiSquareRoot`

First done with 1.08.

1.1 added `\xintiiSquareRoot`.

1.1a added `\xintiiSqrtR`.

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

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

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

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

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.)

```

681 \def\xintiiSquareRoot {\romannumeral0\xintiisquareroot }%
682 \def\xintiisquareroot #1{\expandafter\XINT_sqrt_checkin\romannumeral`&&#1\xint:}%
683 \def\XINT_sqrt_checkin #1%
684 {%
685   \xint_UDzerominusfork
686   #1-\XINT_sqrt_iszero
687   0#1\XINT_sqrt_isneg
688   0-\XINT_sqrt
689   \krof #1%
690 }%
691 \def\XINT_sqrt_iszero #1\xint:{{1}{1}}%
692 \def\XINT_sqrt_isneg #1\xint:{\XINT_signalcondition{InvalidOperation}{square
693   root of negative: #1}{{0}{0}}}%
694 \def\XINT_sqrt #1\xint:
695 {%
696   \expandafter\XINT_sqrt_start\romannumeral0\xintlength {#1}.#1.%
697 }%
698 \def\XINT_sqrt_start #1.%
699 {%
700   \ifnum #1<\xint_c_x\xint_dothis\XINT_sqrt_small_a\fi
701   \xint_orthat\XINT_sqrt_big_a #1.%
702 }%
703 \def\XINT_sqrt_small_a #1.{\XINT_sqrt_a #1.\XINT_sqrt_small_d }%
704 \def\XINT_sqrt_big_a #1.{\XINT_sqrt_a #1.\XINT_sqrt_big_d }%
705 \def\XINT_sqrt_a #1.%
706 {%
707   \ifodd #1
708     \expandafter\XINT_sqrt_b0

```

```

709 \else
710 \expandafter\XINT_sqrt_bE
711 \fi
712 #1.%
713 }%

714 \def\XINT_sqrt_bE #1.#2#3#4%
715 {%
716 \XINT_sqrt_c {#3#4}#2{#1}#3#4%
717 }%

718 \def\XINT_sqrt_b0 #1.#2#3%
719 {%
720 \XINT_sqrt_c #3#2{#1}#3%
721 }%

722 \def\XINT_sqrt_c #1#2%
723 {%
724 \expandafter #2%
725 \the\numexpr \ifnum #1>\xint_c_ii
726 \ifnum #1>\xint_c_vi
727 \ifnum #1>12 \ifnum #1>20 \ifnum #1>30
728 \ifnum #1>42 \ifnum #1>56 \ifnum #1>72
729 \ifnum #1>90
730 10\else 9\fi \else 8\fi \else 7\fi \else 6\fi \else 5\fi
731 \else 4\fi \else 3\fi \else 2\fi \else 1\fi .%
732 }%

733 \def\XINT_sqrt_small_d #1.#2%
734 {%
735 \expandafter\XINT_sqrt_small_e
736 \the\numexpr #1\ifcase \numexpr #2/\xint_c_ii-\xint_c_i\relax
737 \or 0\or 00\or 000\or 0000\fi .%
738 }%

739 \def\XINT_sqrt_small_e #1.#2.%
740 {%
741 \expandafter\XINT_sqrt_small_ea\the\numexpr #1*#1-#2.#1.%
742 }%

743 \def\XINT_sqrt_small_ea #1%
744 {%
745 \if0#1\xint_dothis\XINT_sqrt_small_ez\fi
746 \if-#1\xint_dothis\XINT_sqrt_small_eb\fi
747 \xint_orthat\XINT_sqrt_small_f #1%
748 }%
749 \def\XINT_sqrt_small_ez 0.#1.{\expandafter{\the\numexpr#1+\xint_c_i
750 \expandafter}\expandafter{\the\numexpr #1*\xint_c_ii+\xint_c_i}}%

```

```

751 \def\XINT_sqrt_small_eb -#1.#2.%
752 {%
753   \expandafter\XINT_sqrt_small_ec \the\numexpr
754   (#1-\xint_c_i+#2)/(\xint_c_ii*#2).#1.#2.%
755 }%

756 \def\XINT_sqrt_small_ec #1.#2.#3.%
757 {%
758   \expandafter\XINT_sqrt_small_f \the\numexpr
759   -#2+\xint_c_ii*#3*#1+#1*#1\expandafter.\the\numexpr #3+#1.%
760 }%

761 \def\XINT_sqrt_small_f #1.#2.%
762 {%
763   \expandafter\XINT_sqrt_small_g
764   \the\numexpr (#1+#2)/(\xint_c_ii*#2)-\xint_c_i.#1.#2.%
765 }%

766 \def\XINT_sqrt_small_g #1#2.%
767 {%
768   \if 0#1%
769     \expandafter\XINT_sqrt_small_end
770   \else
771     \expandafter\XINT_sqrt_small_h
772   \fi
773   #1#2.%
774 }%

775 \def\XINT_sqrt_small_h #1.#2.#3.%
776 {%
777   \expandafter\XINT_sqrt_small_f
778   \the\numexpr #2-\xint_c_ii*#1*#3+#1*#1\expandafter.%
779   \the\numexpr #3-#1.%
780 }%
781 \def\XINT_sqrt_small_end #1.#2.#3.{{#3}{#2}}%

782 \def\XINT_sqrt_big_d #1.#2%
783 {%
784   \ifodd #2 \xint_dothis{\expandafter\XINT_sqrt_big_e0}\fi
785   \xint_orthat{\expandafter\XINT_sqrt_big_eE}%

786   \the\numexpr (#2-\xint_c_i)/\xint_c_ii.#1;%
787 }%

788 \def\XINT_sqrt_big_eE #1;#2#3#4#5#6#7#8#9%
789 {%
790   \XINT_sqrt_big_eE_a #1;{#2#3#4#5#6#7#8#9}%
791 }%

```

```

792 \def\XINT_sqrt_big_eE_a #1.#2;#3%
793 {%
794   \expandafter\XINT_sqrt_bigormed_f
795   \romannumeral0\XINT_sqrt_small_e #2000.#3.#1;%
796 }%

797 \def\XINT_sqrt_big_e0 #1;#2#3#4#5#6#7#8#9%
798 {%
799   \XINT_sqrt_big_e0_a #1;{#2#3#4#5#6#7#8#9}%
800 }%
801 \def\XINT_sqrt_big_e0_a #1.#2;#3#4%
802 {%
803   \expandafter\XINT_sqrt_bigormed_f
804   \romannumeral0\XINT_sqrt_small_e #20000.#3#4.#1;%
805 }%

806 \def\XINT_sqrt_bigormed_f #1#2#3;%
807 {%
808   \ifnum#3<\xint_c_ix
809     \xint_dothis {\csname XINT_sqrt_med_f\romannumeral#3\endcsname}%
810     \fi
811     \xint_orthat\XINT_sqrt_big_f #1.#2.#3;%
812 }%
813 \def\XINT_sqrt_med_fv {\XINT_sqrt_med_fa .}%
814 \def\XINT_sqrt_med_fvi {\XINT_sqrt_med_fa 0.}%
815 \def\XINT_sqrt_med_fvii {\XINT_sqrt_med_fa 00.}%
816 \def\XINT_sqrt_med_fviii{\XINT_sqrt_med_fa 000.}%

817 \def\XINT_sqrt_med_fa #1.#2.#3.#4;%
818 {%
819   \expandafter\XINT_sqrt_med_fb
820   \the\numexpr (#30#1-5#1)/(\xint_c_ii*#2).#1.#2.#3.%
821 }%

822 \def\XINT_sqrt_med_fb #1.#2.#3.#4.#5.%
823 {%
824   \expandafter\XINT_sqrt_small_ea
825   \the\numexpr (#40#2-\xint_c_ii*#3*#1)*10#2+(#1*#1-#5)\expandafter.%
826   \the\numexpr #30#2-#1.%
827 }%

828 \def\XINT_sqrt_big_f #1;#2#3#4#5#6#7#8#9%
829 {%
830   \XINT_sqrt_big_fa #1;{#2#3#4#5#6#7#8#9}%
831 }%

832 \def\XINT_sqrt_big_fa #1.#2.#3;#4%
833 {%
834   \expandafter\XINT_sqrt_big_ga
835   \the\numexpr #3-\xint_c_viii\expandafter.%
836   \romannumeral0\XINT_sqrt_med_fa 000.#1.#2.;#4.%
837 }%

```

```

838 \def\XINT_sqrt_big_ga #1.#2#3%
839 {%
840   \ifnum #1>\xint_c_viii
841     \expandafter\XINT_sqrt_big_gb\else
842     \expandafter\XINT_sqrt_big_ka
843     \fi #1.#3.#2.%
844 }%

845 \def\XINT_sqrt_big_gb #1.#2.#3.%
846 {%
847   \expandafter\XINT_sqrt_big_gc
848   \the\numexpr (\xint_c_ii*#2-\xint_c_i)*\xint_c_x^viii/(\xint_c_iv*#3).%
849   #3.#2.#1;%
850 }%

851 \def\XINT_sqrt_big_gc #1.#2.#3.%
852 {%
853   \expandafter\XINT_sqrt_big_gd
854   \romannumeral0\xintiiaadd
855     {\xintiiSub {#3000000000}{\xintDouble{\xintiiMul{#2}{#1}}00000000}%
856     {\xintiiSqr {#1}}.%
857   \romannumeral0\xintiisub{#2000000000}{#1}.%
858 }%

859 \def\XINT_sqrt_big_gd #1.#2.%
860 {%
861   \expandafter\XINT_sqrt_big_ge #2.#1.%
862 }%

863 \def\XINT_sqrt_big_ge #1;#2#3#4#5#6#7#8#9%
864   {\XINT_sqrt_big_gf #1.#2#3#4#5#6#7#8#9;}%
865 \def\XINT_sqrt_big_gf #1;#2#3#4#5#6#7#8#9%
866   {\XINT_sqrt_big_gg #1#2#3#4#5#6#7#8#9.}%

867 \def\XINT_sqrt_big_gg #1.#2.#3.#4.%
868 {%
869   \expandafter\XINT_sqrt_big_gloop
870   \expandafter\xint_c_xvi\expandafter.%
871   \the\numexpr #3-\xint_c_viii\expandafter.%
872   \romannumeral0\xintiisub {#2}{\xintiNum{#4}}.#1.%
873 }%

874 \def\XINT_sqrt_big_gloop #1.#2.%
875 {%
876   \unless\ifnum #1<#2 \xint_dothis\XINT_sqrt_big_ka \fi
877   \xint_orthat{\XINT_sqrt_big_gi #1.}#2.%
878 }%

```



```

879 \def\XINT_sqrt_big_gi #1.%
880 {%
881   \expandafter\XINT_sqrt_big_gj\romannumeral\xintreplicate{#1}0.#1.%
882 }%

883 \def\XINT_sqrt_big_gj #1.#2.#3.#4.#5.%
884 {%
885   \expandafter\XINT_sqrt_big_gk
886   \romannumeral0\xintiidivision {#4#1}%
887   {\XINT_dbl #5\xint_bye2345678\xint_bye*\xint_c_ii\relax}.%
888   #1.#5.#2.#3.%
889 }%

890 \def\XINT_sqrt_big_gk #1#2.#3.#4.%
891 {%
892   \expandafter\XINT_sqrt_big_gl
893   \romannumeral0\xintiiaadd {#2#3}{\xintiisqr{#1}}.%
894   \romannumeral0\xintiisub {#4#3}{#1}.%
895 }%

896 \def\XINT_sqrt_big_gl #1.#2.%
897 {%
898   \expandafter\XINT_sqrt_big_gm #2.#1.%
899 }%

900 \def\XINT_sqrt_big_gm #1.#2.#3.#4.#5.%
901 {%
902   \expandafter\XINT_sqrt_big_gn

903   \romannumeral0\XINT_split_fromleft\xint_c_ii*#3.#5\xint_bye2345678\xint_bye.%.%
904   #1.#2.#3.#4.%
905 }%

906 \def\XINT_sqrt_big_gn #1.#2.#3.#4.#5.#6.%
907 {%
908   \expandafter\XINT_sqrt_big_gloop
909   \the\numexpr \xint_c_ii*#5\expandafter.%
910   \the\numexpr #6-#5\expandafter.%
911   \romannumeral0\xintiisub{#4}{\xintiinum{#1}}.#3.#2.%
912 }%

913 \def\XINT_sqrt_big_ka #1.#2.#3.#4.%
914 {%
915   \expandafter\XINT_sqrt_big_kb

916   \romannumeral0\XINT_dsx_addzeros {#1}#3;.%
917   \romannumeral0\xintiisub
918   {\XINT_dsx_addzerosnofuss {\xint_c_ii*#1}#2;}%
919   {\xintiinum{#4}}.%

```

```

920 }%
921 \def\XINT_sqrt_big_kb #1.#2.%
922 {%
923   \expandafter\XINT_sqrt_big_kc #2.#1.%
924 }%

925 \def\XINT_sqrt_big_kc #1%
926 {%
927   \if0#1\xint_dothis\XINT_sqrt_big_kz\fi
928   \xint_orthat\XINT_sqrt_big_kloop #1%
929 }%
930 \def\XINT_sqrt_big_kz 0.#1.%
931 {%
932   \expandafter\XINT_sqrt_big_kend
933   \romannumeral0%
934   \xintinc{\XINT_dbl#1\xint_bye2345678\xint_bye*\xint_c_ii\relax}.#1.%
935 }%
936 \def\XINT_sqrt_big_kend #1.#2.%
937 {%
938   \expandafter{\romannumeral0\xintinc{#2}}{#1}%
939 }%

940 \def\XINT_sqrt_big_kloop #1.#2.%
941 {%
942   \expandafter\XINT_sqrt_big_ke
943   \romannumeral0\xintiidivision{#1}%
944   {\romannumeral0\XINT_dbl #2\xint_bye2345678\xint_bye*\xint_c_ii\relax}{#2}%
945 }%

946 \def\XINT_sqrt_big_ke #1%
947 {%
948   \if0\XINT_Sgn #1\xint:
949     \expandafter \XINT_sqrt_big_end
950   \else \expandafter \XINT_sqrt_big_kf
951   \fi {#1}%
952 }%

953 \def\XINT_sqrt_big_kf #1#2#3%
954 {%
955   \expandafter\XINT_sqrt_big_kg
956   \romannumeral0\xintiisub {#3}{#1}.%
957   \romannumeral0\xintiisub {#2}{\xintiiSqr {#1}}.%
958 }%
959 \def\XINT_sqrt_big_kg #1.#2.%
960 {%
961   \expandafter\XINT_sqrt_big_kloop #2.#1.%
962 }%

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

```

## 5.52 \xintiisqrt, \xintiisqrtR

```
964 \def\xintiisqrt {\romannumeral0\xintiisqrt }%
965 \def\xintiisqrt {\expandafter\XINT_sqrt_post\romannumeral0\xintiisquareroot }%
966 \def\XINT_sqrt_post #1#2{\XINT_dec #1\XINT_dec_bye234567890\xint_bye}%
967 \def\xintiisqrtR {\romannumeral0\xintiisqrtR }%
968 \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-> $\#1 - 1$ .

```
969 \def\XINT_sqrtr_post #1#2%
970 {\xintiiflt {#2}{#1}{ #1}{\XINT_dec #1\XINT_dec_bye234567890\xint_bye}}%
```

## 5.53 \xintiibinomial

2015/11/28-29 for 1.2f.

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 binomial(x,y), if  $y < 0$  or  $x < y$  !

I really lack some kind of infinity or NaN value.

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

```
971 \def\xintiibinomial {\romannumeral0\xintiibinomial }%
972 \def\xintiibinomial #1#2%
973 {%
974   \expandafter\XINT_binom_pre\the\numexpr #1\expandafter.\the\numexpr #2.%
975 }%
976 \def\XINT_binom_pre #1.#2.%
977 {%
978   \expandafter\XINT_binom_fork \the\numexpr#1-#2.#2.#1.%
979 }%
```

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.

```
980 \def\XINT_binom_fork #1#2.#3#4.#5#6.%
981 {%
982   \if-#5\xint_dothis{\XINT_signalcondition{InvalidOperation}{Binomial with
983     negative first arg: #5#6}{0}}\fi
984   \if-#1\xint_dothis{ 0}\fi
985   \if-#3\xint_dothis{ 0}\fi
986   \if0#1\xint_dothis{ 1}\fi
987   \if0#3\xint_dothis{ 1}\fi
988   \ifnum #5#6 > \xint_c_x^viii_mone\xint_dothis
989     {\XINT_signalcondition{InvalidOperation}{Binomial with too
990       large argument: 99999999 < #5#6}{0}}\fi
991   \ifnum #1#2 > #3#4 \xint_dothis{\XINT_binom_a #1#2.#3#4.}\fi
992   \xint_orthat{\XINT_binom_a #3#4.#1#2.}%
993 }%
```

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

```

994 \def\xINT_binom_a #1.#2.%
995 {%
996   \expandafter\xINT_binom_b\the\numexpr \xint_c_i+#1.1.#2.100000001!1!;!0!%
997 }%

```

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

```

998 \def\xINT_binom_b #1.%
999 {%
1000   \ifnum #1>9999 \xint_dothis\xINT_binom_vbigloop \fi
1001   \ifnum #1>463 \xint_dothis\xINT_binom_bigloop \fi
1002   \ifnum #1>98 \xint_dothis\xINT_binom_medloop \fi
1003   \ifnum #1>29 \xint_dothis\xINT_binom_smallloop \fi
1004   \xint_orthat\xINT_binom_vsmallloop #1.%
1005 }%

```

$y.j.k$ . 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.

```

1006 \def\xINT_binom_smallloop #1.#2.#3.%
1007 {%
1008   \ifcase\numexpr #3-#2\relax
1009     \expandafter\xINT_binom_end_
1010   \or \expandafter\xINT_binom_end_i
1011   \or \expandafter\xINT_binom_end_ii
1012   \or \expandafter\xINT_binom_end_iii
1013   \else\expandafter\xINT_binom_smallloop_a
1014   \fi #1.#2.#3.%
1015 }%

```

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`.

```

1016 \def\xINT_binom_smallloop_a #1.#2.#3.%
1017 {%
1018   \expandafter\xINT_binom_smallloop_b
1019   \the\numexpr #1+\xint_c_iv\expandafter.%
1020   \the\numexpr #2+\xint_c_iv\expandafter.%
1021   \the\numexpr #3\expandafter.%
1022   \the\numexpr\expandafter\xINT_binom_div
1023   \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1024   !\romannumeral0\expandafter\xINT_binom_mul
1025   \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1026 }%
1027 \def\xINT_binom_smallloop_b #1.%

```

```

1028 {%
1029     \ifnum #1>98 \expandafter\XINT_binom_medloop \else
1030                 \expandafter\XINT_binom_smallloop \fi #1.%
1031 }%
```

Ici on prend trois par trois.

```

1032 \def\XINT_binom_medloop #1.#2.#3.%
1033 {%
1034     \ifcase\numexpr #3-#2\relax
1035         \expandafter\XINT_binom_end_
1036     \or \expandafter\XINT_binom_end_i
1037     \or \expandafter\XINT_binom_end_ii
1038     \else\expandafter\XINT_binom_medloop_a
1039     \fi #1.#2.#3.%
1040 }%
1041 \def\XINT_binom_medloop_a #1.#2.#3.%
1042 {%
1043     \expandafter\XINT_binom_medloop_b
1044     \the\numexpr #1+\xint_c_iii\expandafter.%
1045     \the\numexpr #2+\xint_c_iii\expandafter.%
1046     \the\numexpr #3\expandafter.%
1047     \the\numexpr\expandafter\XINT_binom_div
1048         \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
1049     !\romannumeral0\expandafter\XINT_binom_mul
1050     \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1051 }%
1052 \def\XINT_binom_medloop_b #1.%
1053 {%
1054     \ifnum #1>463 \expandafter\XINT_binom_bigloop \else
1055                 \expandafter\XINT_binom_medloop \fi #1.%
1056 }%
```

Ici on prend deux par deux.

```

1057 \def\XINT_binom_bigloop #1.#2.#3.%
1058 {%
1059     \ifcase\numexpr #3-#2\relax
1060         \expandafter\XINT_binom_end_
1061     \or \expandafter\XINT_binom_end_i
1062     \else\expandafter\XINT_binom_bigloop_a
1063     \fi #1.#2.#3.%
1064 }%
1065 \def\XINT_binom_bigloop_a #1.#2.#3.%
1066 {%
1067     \expandafter\XINT_binom_bigloop_b
1068     \the\numexpr #1+\xint_c_ii\expandafter.%
1069     \the\numexpr #2+\xint_c_ii\expandafter.%
1070     \the\numexpr #3\expandafter.%
1071     \the\numexpr\expandafter\XINT_binom_div
1072         \the\numexpr #2*(#2+\xint_c_i)\expandafter
1073     !\romannumeral0\expandafter\XINT_binom_mul
1074     \the\numexpr #1*(#1+\xint_c_i)!%
1075 }%
```

```

1076 \def\XINT_binom_bigloop_b #1.%
1077 {%
1078     \ifnum #1>9999 \expandafter\XINT_binom_vbigloop \else
1079                 \expandafter\XINT_binom_bigloop \fi #1.%
1080 }%

```

Et finalement un par un.

```

1081 \def\XINT_binom_vbigloop #1.#2.#3.%
1082 {%
1083     \ifnum #3=#2
1084         \expandafter\XINT_binom_end_
1085     \else\expandafter\XINT_binom_vbigloop_a
1086     \fi #1.#2.#3.%
1087 }%
1088 \def\XINT_binom_vbigloop_a #1.#2.#3.%
1089 {%
1090     \expandafter\XINT_binom_vbigloop
1091     \the\numexpr #1+\xint_c_i\expandafter.%
1092     \the\numexpr #2+\xint_c_i\expandafter.%
1093     \the\numexpr #3\expandafter.%
1094     \the\numexpr\expandafter\XINT_binom_div\the\numexpr #2\expandafter
1095     !\romannumeral0\XINT_binom_mul #1!%
1096 }%

```

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.

```

1097 \def\XINT_binom_vsmallloop #1.#2.#3.%
1098 {%
1099     \ifcase\numexpr #3-#2\relax
1100         \expandafter\XINT_binom_vsmallend_
1101     \or \expandafter\XINT_binom_vsmallend_i
1102     \or \expandafter\XINT_binom_vsmallend_ii
1103     \or \expandafter\XINT_binom_vsmallend_iii
1104     \else\expandafter\XINT_binom_vsmallloop_a
1105     \fi #1.#2.#3.%
1106 }%
1107 \def\XINT_binom_vsmallloop_a #1.%
1108 {%
1109     \ifnum #1>26 \expandafter\XINT_binom_smallloop_a \else
1110         \expandafter\XINT_binom_vsmallloop_b \fi #1.%
1111 }%
1112 \def\XINT_binom_vsmallloop_b #1.#2.#3.%
1113 {%
1114     \expandafter\XINT_binom_vsmallloop
1115     \the\numexpr #1+\xint_c_iv\expandafter.%
1116     \the\numexpr #2+\xint_c_iv\expandafter.%
1117     \the\numexpr #3\expandafter.%
1118     \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1119     \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter

```

```

1120 !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1121 }%
1122 \def\xINT_binom_mul #1!#21!;!0!%
1123 {%
1124 \expandafter\xINT_rev_nounsep\expandafter{\expandafter}%
1125 \the\numexpr\expandafter\xINT_smallmul
1126 \the\numexpr\xint_c_x^viii+#1\expandafter
1127 !\romannumeral0\xINT_rev_nounsep {}1;!#2%
1128 \R!\R!\R!\R!\R!\R!\R!\R!\W
1129 \R!\R!\R!\R!\R!\R!\R!\R!\W
1130 1;!%
1131 }%
1132 \def\xINT_binom_div #1!1;!%
1133 {%
1134 \expandafter\xINT_smalldivx_a
1135 \the\numexpr #1/\xint_c_ii\expandafter\xint:
1136 \the\numexpr \xint_c_x^viii+#1!%
1137 }%

```

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

```

1138 \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.

```

1139 \def\xINT_binom_end_iii #1.#2.#3.%
1140 {%
1141 \expandafter\xINT_binom_finish
1142 \the\numexpr\expandafter\xINT_binom_div
1143 \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1144 !\romannumeral0\expandafter\xINT_binom_mul
1145 \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1146 }%
1147 \def\xINT_binom_end_ii #1.#2.#3.%
1148 {%
1149 \expandafter\xINT_binom_finish
1150 \the\numexpr\expandafter\xINT_binom_div
1151 \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
1152 !\romannumeral0\expandafter\xINT_binom_mul
1153 \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1154 }%
1155 \def\xINT_binom_end_i #1.#2.#3.%
1156 {%
1157 \expandafter\xINT_binom_finish
1158 \the\numexpr\expandafter\xINT_binom_div
1159 \the\numexpr #2*(#2+\xint_c_i)\expandafter
1160 !\romannumeral0\expandafter\xINT_binom_mul
1161 \the\numexpr #1*(#1+\xint_c_i)!%
1162 }%
1163 \def\xINT_binom_end_ #1.#2.#3.%
1164 {%

```

```

1165 \expandafter\XINT_binom_finish
1166 \the\numexpr\expandafter\XINT_binom_div\the\numexpr #2\expandafter
1167 !\romannumeral0\XINT_binom_mul #1!%
1168 }%

1169 \def\XINT_binom_finish #1;!0!%
1170 {\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).

```

1171 \def\XINT_binom_vsmallend_iii #1.%
1172 {%
1173 \ifnum #1>26 \expandafter\XINT_binom_end_iii \else
1174 \expandafter\XINT_binom_vsmallend_iiib \fi #1.%
1175 }%
1176 \def\XINT_binom_vsmallend_iiib #1.#2.#3.%
1177 {%
1178 \expandafter\XINT_binom_vsmallfinish
1179 \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1180 \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1181 !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1182 }%
1183 \def\XINT_binom_vsmallend_ii #1.%
1184 {%
1185 \ifnum #1>27 \expandafter\XINT_binom_end_ii \else
1186 \expandafter\XINT_binom_vsmallend_iib \fi #1.%
1187 }%
1188 \def\XINT_binom_vsmallend_iib #1.#2.#3.%
1189 {%
1190 \expandafter\XINT_binom_vsmallfinish
1191 \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1192 \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
1193 !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1194 }%
1195 \def\XINT_binom_vsmallend_i #1.%
1196 {%
1197 \ifnum #1>28 \expandafter\XINT_binom_end_i \else
1198 \expandafter\XINT_binom_vsmallend_ib \fi #1.%
1199 }%
1200 \def\XINT_binom_vsmallend_ib #1.#2.#3.%
1201 {%
1202 \expandafter\XINT_binom_vsmallfinish
1203 \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1204 \the\numexpr #2*(#2+\xint_c_i)\expandafter
1205 !\the\numexpr #1*(#1+\xint_c_i)!%
1206 }%
1207 \def\XINT_binom_vsmallend_ #1.%
1208 {%
1209 \ifnum #1>29 \expandafter\XINT_binom_end_ \else
1210 \expandafter\XINT_binom_vsmallend_b \fi #1.%
1211 }%
1212 \def\XINT_binom_vsmallend_b #1.#2.#3.%
1213 {%

```





```

1252 }%
1253 \def\XINT_pfac_b #1.%
1254 {%
1255     \ifnum #1>9999 \xint_dothis\XINT_pfac_vbigloop \fi
1256     \ifnum #1>463  \xint_dothis\XINT_pfac_bigloop  \fi
1257     \ifnum #1>98   \xint_dothis\XINT_pfac_medloop   \fi
1258                 \xint_orthat\XINT_pfac_smallloop #1.%
1259 }%
1260 \def\XINT_pfac_smallloop #1.#2.%
1261 {%
1262     \ifcase\numexpr #2-#1\relax
1263         \expandafter\XINT_pfac_end_
1264     \or \expandafter\XINT_pfac_end_i
1265     \or \expandafter\XINT_pfac_end_ii
1266     \or \expandafter\XINT_pfac_end_iii
1267     \else\expandafter\XINT_pfac_smallloop_a
1268     \fi #1.#2.%
1269 }%
1270 \def\XINT_pfac_smallloop_a #1.#2.%
1271 {%
1272     \expandafter\XINT_pfac_smallloop_b
1273     \the\numexpr #1+\xint_c_iv\expandafter.%
1274     \the\numexpr #2\expandafter.%
1275     \the\numexpr\expandafter\XINT_smallmul
1276     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1277 }%
1278 \def\XINT_pfac_smallloop_b #1.%
1279 {%
1280     \ifnum #1>98 \expandafter\XINT_pfac_medloop \else
1281                 \expandafter\XINT_pfac_smallloop \fi #1.%
1282 }%
1283 \def\XINT_pfac_medloop #1.#2.%
1284 {%
1285     \ifcase\numexpr #2-#1\relax
1286         \expandafter\XINT_pfac_end_
1287     \or \expandafter\XINT_pfac_end_i
1288     \or \expandafter\XINT_pfac_end_ii
1289     \else\expandafter\XINT_pfac_medloop_a
1290     \fi #1.#2.%
1291 }%
1292 \def\XINT_pfac_medloop_a #1.#2.%
1293 {%
1294     \expandafter\XINT_pfac_medloop_b
1295     \the\numexpr #1+\xint_c_iii\expandafter.%
1296     \the\numexpr #2\expandafter.%
1297     \the\numexpr\expandafter\XINT_smallmul
1298     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1299 }%
1300 \def\XINT_pfac_medloop_b #1.%
1301 {%
1302     \ifnum #1>463 \expandafter\XINT_pfac_bigloop \else
1303                 \expandafter\XINT_pfac_medloop \fi #1.%

```

```

1304 }%
1305 \def\XINT_pfac_bigloop #1.#2.%
1306 {%
1307     \ifcase\numexpr #2-#1\relax
1308         \expandafter\XINT_pfac_end_
1309     \or \expandafter\XINT_pfac_end_i
1310     \else\expandafter\XINT_pfac_bigloop_a
1311     \fi #1.#2.%
1312 }%
1313 \def\XINT_pfac_bigloop_a #1.#2.%
1314 {%
1315     \expandafter\XINT_pfac_bigloop_b
1316     \the\numexpr #1+\xint_c_ii\expandafter.%
1317     \the\numexpr #2\expandafter.%
1318     \the\numexpr\expandafter
1319     \XINT_smallmul\the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
1320 }%
1321 \def\XINT_pfac_bigloop_b #1.%
1322 {%
1323     \ifnum #1>9999 \expandafter\XINT_pfac_vbigloop \else
1324         \expandafter\XINT_pfac_bigloop \fi #1.%
1325 }%
1326 \def\XINT_pfac_vbigloop #1.#2.%
1327 {%
1328     \ifnum #2=#1
1329         \expandafter\XINT_pfac_end_
1330     \else\expandafter\XINT_pfac_vbigloop_a
1331     \fi #1.#2.%
1332 }%
1333 \def\XINT_pfac_vbigloop_a #1.#2.%
1334 {%
1335     \expandafter\XINT_pfac_vbigloop
1336     \the\numexpr #1+\xint_c_i\expandafter.%
1337     \the\numexpr #2\expandafter.%
1338     \the\numexpr\expandafter\XINT_smallmul\the\numexpr\xint_c_x^viii+#1!%
1339 }%
1340 \def\XINT_pfac_end_iii #1.#2.%
1341 {%
1342     \expandafter\XINT_mul_out
1343     \the\numexpr\expandafter\XINT_smallmul
1344     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1345 }%
1346 \def\XINT_pfac_end_ii #1.#2.%
1347 {%
1348     \expandafter\XINT_mul_out
1349     \the\numexpr\expandafter\XINT_smallmul
1350     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1351 }%
1352 \def\XINT_pfac_end_i #1.#2.%
1353 {%
1354     \expandafter\XINT_mul_out
1355     \the\numexpr\expandafter\XINT_smallmul

```

```

1356 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
1357 }%
1358 \def\XINT_pfac_end_ #1.#2.%
1359 {%
1360 \expandafter\XINT_mul_out
1361 \the\numexpr\expandafter\XINT_smallmul\the\numexpr \xint_c_x^viii+#1!%
1362 }%

```

## 5.55 \xintBool, \xintToggle

### 1.09c

```

1363 \def\xintBool #1{\romannumeral`&&@%
1364 \csname if#1\endcsname\expandafter1\else\expandafter0\fi }%
1365 \def\xintToggle #1{\romannumeral`&&@\iftoggle{#1}{1}{0}}%

```

## 5.56 \xintGCD, \xintiigcd

Copied over from `\xintiigcd` of `xintgcd` at 1.3d to support `gcd()` function in `\xintiexpr`.

```

1366 \def\xintiigcd {\romannumeral0\xintiigcd }%
1367 \def\xintiigcd #1{\expandafter\XINT_iigcd\romannumeral0\xintiiaabs#1\xint:}%
1368 \def\XINT_iigcd #1#2\xint:#3%
1369 {%
1370 \expandafter\XINT_gcd_fork\expandafter#1%
1371 \romannumeral0\xintiiaabs#3\xint:#1#2\xint:
1372 }%
1373 \def\XINT_gcd_fork #1#2%
1374 {%
1375 \xint_UDzerofork
1376 #1\XINT_gcd_Aiszero
1377 #2\XINT_gcd_Biszero
1378 0\XINT_gcd_loop
1379 \krof
1380 #2%
1381 }%
1382 \def\XINT_gcd_AisZero #1\xint:#2\xint:{ #1}%
1383 \def\XINT_gcd_BisZero #1\xint:#2\xint:{ #2}%
1384 \def\XINT_gcd_loop #1\xint:#2\xint:
1385 {%
1386 \expandafter\expandafter\expandafter\XINT_gcd_CheckRem
1387 \expandafter\xint_secondoftwo
1388 \romannumeral0\XINT_div_prepare {#1}{#2}\xint:#1\xint:
1389 }%
1390 \def\XINT_gcd_CheckRem #1%
1391 {%
1392 \xint_gob_til_zero #1\XINT_gcd_end0\XINT_gcd_loop #1%
1393 }%
1394 \def\XINT_gcd_end0\XINT_gcd_loop #1\xint:#2\xint:{ #2}%

```

## 5.57 \xintLCM, \xintiilcm

```

1395 \def\xintiilcm {\romannumeral0\xintiilcm}%
1396 \def\xintiilcm #1{\expandafter\XINT_iilcm\romannumeral0\xintiiaabs#1\xint:}%

```

```

1397 \def\XINT_iilcm #1#2\xint:#3%
1398 {%
1399     \expandafter\XINT_lcm_fork\expandafter#1%
1400         \romannumeral0\xintiabs#3\xint:#1#2\xint:
1401 }%
1402 \def\XINT_lcm_fork #1#2%
1403 {%
1404     \xint_UDzerofork
1405     #1\XINT_lcm_iszero
1406     #2\XINT_lcm_iszero
1407     0\XINT_lcm_notzero
1408     \krof
1409     #2%
1410 }%
1411 \def\XINT_lcm_iszero #1\xint:#2\xint:{ 0}%
1412 \def\XINT_lcm_notzero #1\xint:#2\xint:
1413 {%
1414     \expandafter\XINT_lcm_end\romannumeral0%
1415     \expandafter\expandafter\expandafter\XINT_gcd_CheckRem
1416     \expandafter\xint_secondoftwo
1417     \romannumeral0\XINT_div_prepare {#1}{#2}\xint:#1\xint:
1418     \xint:#1\xint:#2\xint:
1419 }%
1420 \def\XINT_lcm_end #1\xint:#2\xint:#3\xint:{\xintiimul {#2}{\xintiiQuo{#3}{#1}}}%

```

## 5.58 (WIP) \xintRandomDigits

1.3b. See user manual. Whether this will be part of `xintkernel`, `xintcore`, or `xint` is yet to be decided.

```

1421 \def\xintRandomDigits{\romannumeral0\xintrandomdigits}%
1422 \def\xintrandomdigits#1%
1423 {%
1424     \csname xint_gob_andstop_\expandafter\XINT_randomdigits\the\numexpr#1\xint:
1425 }%
1426 \def\XINT_randomdigits#1\xint:
1427 {%
1428     \expandafter\XINT_randomdigits_a
1429     \the\numexpr(#1+\xint_c_iii)/\xint_c_viii\xint:#1\xint:
1430 }%
1431 \def\XINT_randomdigits_a#1\xint:#2\xint:
1432 {%
1433     \romannumeral\numexpr\xint_c_viii*#1-#2\csname XINT_%
1434     \romannumeral\XINT_replicate #1\endcsname \csname
1435     XINT_rdg\endcsname
1436 }%
1437 \def\XINT_rdg
1438 {%
1439     \expandafter\XINT_rdg_aux\the\numexpr%
1440         \xint_c_nine_x^viii%
1441         -\xint_texuniformdeviate\xint_c_ii^vii%
1442         -\xint_c_ii^vii*\xint_texuniformdeviate\xint_c_ii^vii%
1443         -\xint_c_ii^xiv*\xint_texuniformdeviate\xint_c_ii^vii%
1444         -\xint_c_ii^xxi*\xint_texuniformdeviate\xint_c_ii^vii%

```

```

1445             +\xint_texuniformdeviate\xint_c_x^viii%
1446             \relax%
1447 }%
1448 \def\xINT_rdg_aux#1{\XINT_rdg\endcsname}%
1449 \let\xINT_XINT_rdg\endcsname

```

## 5.59 (WIP) \XINT\_eightrandomdigits

### 1.3b.

```

1450 \def\xINT_eightrandomdigits
1451 {%
1452     \expandafter\xint_gobble_i\the\numexpr%
1453         \xint_c_nine_x^viii%
1454             -\xint_texuniformdeviate\xint_c_ii^vii%
1455             -\xint_c_ii^vii*\xint_texuniformdeviate\xint_c_ii^vii%
1456             -\xint_c_ii^xiv*\xint_texuniformdeviate\xint_c_ii^vii%
1457             -\xint_c_ii^xxi*\xint_texuniformdeviate\xint_c_ii^vii%
1458             +\xint_texuniformdeviate\xint_c_x^viii%
1459     \relax%
1460 }%

```

## 5.60 (WIP) \xintXRandomDigits

### 1.3b.

```

1461 \def\xintXRandomDigits#1%
1462 {%
1463     \csname xint_gobble_\expandafter\XINT_xrandomdigits\the\numexpr#1\xint:
1464 }%
1465 \def\XINT_xrandomdigits#1\xint:
1466 {%
1467     \expandafter\XINT_xrandomdigits_a
1468     \the\numexpr(#1+\xint_c_iii)/\xint_c_viii\xint:#1\xint:
1469 }%
1470 \def\XINT_xrandomdigits_a#1\xint:#2\xint:
1471 {%
1472     \romannumeral\numexpr\xint_c_viii*#1-#2\expandafter\endcsname
1473     \romannumeral`&&\romannumeral
1474         \XINT_replicate #1\endcsname\XINT_eightrandomdigits
1475 }%

```

## 5.61 (WIP) \xintiiRandRangeAtoB

### 1.3b. Support for randrange() function.

Wee do it f-expandably for matters of \xintNewExpr etc... The \xintexpr will add \xintNum wrapper to possible fractional input. But \xintiiexpr will call as is.

TODO: ? implement third argument (STEP) TODO: \xintNum wrapper (which truncates) not so good in floatexpr. Use round?

It is an error if  $b \leq a$ , as in Python.

```

1476 \def\xintiiRandRangeAtoB{\romannumeral`&&\xintiiRandRangeAtoB}%
1477 \def\xintiiRandRangeAtoB#1%

```

```

1478 {%
1479   \expandafter\XINT_randrangeAtoB_a\romannumeral`&&@#1\xint:
1480 }%
1481 \def\XINT_randrangeAtoB_a#1\xint:#2%
1482 {%
1483   \xintiiadd{\expandafter\XINT_randrange
1484             \romannumeral0\xintiisub{#2}{#1}\xint:}%
1485   {#1}%
1486 }%

```

## 5.62 (WIP) \xintiiRandRange

### 1.3b. Support for randrange().

```

1487 \def\xintiiRandRange{\romannumeral`&&@\xintiirandrange}%
1488 \def\xintiirandrange#1%
1489 {%
1490   \expandafter\XINT_randrange\romannumeral`&&@#1\xint:
1491 }%
1492 \def\XINT_randrange #1%
1493 {%
1494   \xint_UDzerominusfork
1495   #1-\XINT_randrange_err:empty
1496   0#1\XINT_randrange_err:empty
1497   0-\XINT_randrange_a
1498   \krof #1%
1499 }%
1500 \def\XINT_randrange_err:empty#1\xint:
1501 {%
1502   \XINT_expandableerror{Empty range for randrange.} 0%
1503 }%
1504 \def\XINT_randrange_a #1\xint:
1505 {%
1506   \expandafter\XINT_randrange_b\romannumeral0\xintlength{#1}.#1\xint:
1507 }%
1508 \def\XINT_randrange_b #1.%
1509 {%
1510   \ifnum#1<\xint_c_x\xint_dothis{\the\numexpr\XINT_uniformdeviate{}}\fi
1511   \xint_orthat{\XINT_randrange_c #1.}%
1512 }%
1513 \def\XINT_randrange_c #1.#2#3#4#5#6#7#8#9%
1514 {%
1515   \expandafter\XINT_randrange_d
1516   \the\numexpr\expandafter\XINT_uniformdeviate\expandafter
1517   {\expandafter}\the\numexpr\xint_c_i+#2#3#4#5#6#7#8#9\xint:\xint:
1518   #2#3#4#5#6#7#8#9\xint:#1\xint:
1519 }%

```

This raises following annex question: immediately after setting the seed is it possible for `\xintUniformDeviate{N}` where  $N > 0$  has exactly eight digits to return either 0 or  $N-1$ ? It could be that this is never the case, then there is a bias in `randrange()`. Of course there are anyhow only  $2^{28}$  seeds so `randrange(10^X)` is by necessity biased when executed immediately after setting the seed, if  $X$  is at least 9.

```

1520 \def\XINT_randrange_d #1\xint:#2\xint:
1521 {%
1522   \ifnum#1=\xint_c_\xint_dothis\XINT_randrange_Z\fi
1523   \ifnum#1=#2 \xint_dothis\XINT_randrange_A\fi
1524   \xint_orthat\XINT_randrange_e #1\xint:
1525 }%
1526 \def\XINT_randrange_e #1\xint:#2\xint:#3\xint:
1527 {%
1528   \the\numexpr#1\expandafter\relax
1529   \romannumeral0\xintrandomdigits{#2-\xint_c_viii}%
1530 }%

```

This is quite unlikely to get executed but if it does it must pay attention to leading zeros, hence the `\xintinum`. We don't have to be overly obstinate about removing overheads...

```

1531 \def\XINT_randrange_Z 0\xint:#1\xint:#2\xint:
1532 {%
1533   \xintinum{\xintRandomDigits{#1-\xint_c_viii}}%
1534 }%

```

Here too, overhead is not such a problem. The idea is that we got by extraordinary same first 8 digits as upper range bound so we pick at random the remaining needed digits in one go and compare with the upper bound. If too big, we start again with another random 8 leading digits in given range. No need to aim at any kind of efficiency for the check and loop back.

```

1535 \def\XINT_randrange_A #1\xint:#2\xint:#3\xint:
1536 {%
1537   \expandafter\XINT_randrange_B
1538   \romannumeral0\xintrandomdigits{#2-\xint_c_viii}\xint:
1539   #3\xint:#2.#1\xint:
1540 }%
1541 \def\XINT_randrange_B #1\xint:#2\xint:#3.#4\xint:
1542 {%
1543   \xintiiiflt{#1}{#2}{\XINT_randrange_E}{\XINT_randrange_again}%
1544   #4#1\xint:#3.#4#2\xint:
1545 }%
1546 \def\XINT_randrange_E #1\xint:#2\xint:{ #1}%
1547 \def\XINT_randrange_again #1\xint:{\XINT_randrange_c}%

```

## 5.63 Adjustments for engines without uniformdeviate primitive

### 1.3b.

```

1548 \ifdefined\xint_texuniformdeviate
1549 \else
1550   \def\xintrandomdigits#1%
1551   {%
1552     \XINT_expandableerror
1553     {No uniformdeviate at engine level, returning 0.} 0%
1554   }%
1555   \let\xintXRandomDigits\xintRandomDigits
1556   \def\XINT_randrange#1\xint:
1557   {%
1558     \XINT_expandableerror

```



*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintfrac, xintexpr, xinttrig, xintlog*

```
1559      {No uniformdeviate at engine level, returning 0.} 0%
1560    }%
1561 \fi
1562 \XINT_restorecatcodes_endinput%
```

## 6 Package xintbinhex implementation

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

The commenting is currently (2019/04/05) very sparse.

The macros from 1.08 (2013/06/07) remained unchanged until their complete rewrite at 1.2m (2019/07/31).

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.

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

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

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

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2  \catcode13=5    % ^^M
3  \endlinechar=13 %
4  \catcode123=1  % {
5  \catcode125=2  % }
6  \catcode64=11 % @
7  \catcode35=6  % #
8  \catcode44=12 % ,
9  \catcode45=12 % -
10 \catcode46=12 % .
11 \catcode58=12 % :
12 \let\z\endgroup
13 \expandafter\let\expandafter\x\csname ver@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\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 % xintbinhex 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{xintbinhex}%
46 [2019/04/05 1.3e Expandable binary and hexadecimal conversions (JFB)]%

```

## 6.3 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 {\endcsname0000}%
81 \expandafter\def\csname XINT_cshtb_1\endcsname {\endcsname0001}%
82 \expandafter\def\csname XINT_cshtb_2\endcsname {\endcsname0010}%
83 \expandafter\def\csname XINT_cshtb_3\endcsname {\endcsname0011}%
84 \expandafter\def\csname XINT_cshtb_4\endcsname {\endcsname0100}%
85 \expandafter\def\csname XINT_cshtb_5\endcsname {\endcsname0101}%
86 \expandafter\def\csname XINT_cshtb_6\endcsname {\endcsname0110}%
87 \expandafter\def\csname XINT_cshtb_7\endcsname {\endcsname0111}%
88 \expandafter\def\csname XINT_cshtb_8\endcsname {\endcsname1000}%
89 \expandafter\def\csname XINT_cshtb_9\endcsname {\endcsname1001}%
90 \def\XINT_cshtb_A {\endcsname1010}%
91 \def\XINT_cshtb_B {\endcsname1011}%
92 \def\XINT_cshtb_C {\endcsname1100}%
93 \def\XINT_cshtb_D {\endcsname1101}%
94 \def\XINT_cshtb_E {\endcsname1110}%
95 \def\XINT_cshtb_F {\endcsname1111}%
96 \let\XINT_cshtb_none \endcsname

```

## 6.4 Helper macros

### 6.4.1 \XINT\_zeroes\_foriv

```

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

```

### 6.5 \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
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 }%

```

## 6.6 \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\endcename{!\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 { }%

```

## 6.7 \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 }%
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+1000000000\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!%
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{ }%

```

## 6.8 \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 }%

```

## 6.9 \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\xintbintohehex }%
400 \def\xintbintohehex #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 }%
```

## 6.10 \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 { }%
```

## 6.11 \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.

```
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%
```

## 7 Package [xintgcd](#) implementation

.1	Catcodes, $\varepsilon$ -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 (2019/04/05) 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  $\mathbb{E}$ TeX's `\loop`.

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

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  $AU+BV=D$ .

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

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

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

```

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

```

```

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

```

## 7.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintgcd}%
46 [2019/04/05 1.3e Euclidean algorithm with xint package (JFB)]%

```

## 7.3 \xintGCD, \xintiGCD

### 1.3d.

Removed some braces in favor of `\xint:` delimiter at 1.3d (but `\xintiGCD` was already robust against non-delimited `\numexpr` inputs thanks to using `\xintiiabs{...}`) and refactored the whole `\XINT_iigcd_fork`.

```

47 \def\xintGCD {\romannumeral0\xintgcd }%
48 \def\xintgcd #1#2{\xintiigcd {\xintNum{#1}}{\xintNum{#2}}}%
49 \def\xintiGCD {\romannumeral0\xintiigcd }%

```

This abuses the way `\xintiiabs` expands.

```

50 \def\xintiigcd #1{\expandafter\XINT_iigcd\romannumeral0\xintiiabs#1\xint:}%
51 \def\XINT_iigcd #1#2\xint:#3%
52 {%
53     \expandafter\XINT_gcd_fork\expandafter#1%
54         \romannumeral0\xintiiabs#3\xint:#1#2\xint:
55 }%

```

First argument now in second position (after `\xint:`) but its first digit is also the `#1`.

```

56 \def\XINT_gcd_fork #1#2%
57 {%
58     \xint_UDzerofork
59     #1\XINT_gcd_Aiszero
60     #2\XINT_gcd_Biszero
61     0\XINT_gcd_loop
62     \krof
63     #2%
64 }%
65 \def\XINT_gcd_AisZero #1\xint:#2\xint:{ #1}%
66 \def\XINT_gcd_BisZero #1\xint:#2\xint:{ #2}%

```

`\XINT_div_prepare{#1}{#2}` divides #2 by #1, and outputs {Quotient}{Remainder}.

```

67 \def\XINT_gcd_loop #1\xint:#2\xint:
68 {%
69   \expandafter\expandafter\expandafter\XINT_gcd_CheckRem
70   \expandafter\xint_secondoftwo
71   \romannumeral0\XINT_div_prepare {#1}{#2}\xint:#1\xint:
72 }%
73 \def\XINT_gcd_CheckRem #1%
74 {%
75   \xint_gob_til_zero #1\XINT_gcd_end0\XINT_gcd_loop #1%
76 }%
77 \def\XINT_gcd_end0\XINT_gcd_loop #1\xint:#2\xint:{ #2}%

```

## 7.4 `\xintLCM`, `\xintiilCM`

See comments of `\xintiilgcd` for the refactoring done at 1.3d. No time to make `\xintiilCM` code more efficient now.

Macros `\xintLCM`, `\xintlcm` only for backwards compatibility.

```

78 \def\xintLCM {\romannumeral0\xintlcm}%
79 \def\xintlcm #1#2{\xintiilcm{\xintNum{#1}}{\xintNum{#2}}}%
80 \def\xintiilCM {\romannumeral0\xintiilcm}%
81 \def\xintiilcm #1{\expandafter\XINT_iilcm\romannumeral0\xintiilabs#1\xint:}%
82 \def\XINT_iilcm #1#2\xint:#3%
83 {%
84   \expandafter\XINT_lcm_fork\expandafter#1%
85   \romannumeral0\xintiilabs#3\xint:#1#2\xint:
86 }%
87 \def\XINT_lcm_fork #1#2%
88 {%
89   \xint_UDzerofork
90   #1\XINT_lcm_iszero
91   #2\XINT_lcm_iszero
92   0\XINT_lcm_notzero
93   \krof
94   #2%
95 }%
96 \def\XINT_lcm_iszero #1\xint:#2\xint:{ 0}%
97 \def\XINT_lcm_notzero #1\xint:#2\xint:
98 {%
99   \expandafter\XINT_lcm_end\romannumeral0%
100   \expandafter\expandafter\expandafter\XINT_gcd_CheckRem
101   \expandafter\xint_secondoftwo
102   \romannumeral0\XINT_div_prepare {#1}{#2}\xint:#1\xint:
103   \xint:#1\xint:#2\xint:
104 }%
105 \def\XINT_lcm_end #1\xint:#2\xint:#3\xint:{\xintiimul {#2}{\xintiilQuo{#3}{#1}}}%

```

## 7.5 `\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 1.2l but overlooked the other blemish that `\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\xintiiadd{\XINT_mul_plusplus{}}{#1\xint:#4\xint:}{#6}}%
235     {\romannumeral0\xintiiadd{\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}}%

```

## 7.6 \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\xintiiabs{\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:  
 $\{N\}\{A\}\{D=r(n)\}\{B\}\{q_1\}\{r_1\}\{q_2\}\{r_2\}\{q_3\}\{r_3\}\dots\{q_N\}\{r_N=0\}$

```
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}}%
```

$\{n\}\{rn\}\{an\}\{\{qn\}\{rn\}\}\dots\{\{A\}\{B\}\}\}\Z$   
 $a(n) = r(n-1)$ . Pour  $n=0$  on a juste  $\{0\}\{B\}\{A\}\{\{A\}\{B\}\}\}\Z$   
 $\XINT\_div\_prepare \{u\}\{v\}$  divise  $v$  par  $u$

```
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 }%
```

$\{n+1\}\{q(n+1)\}\{r(n+1)\}\{rn\}\{\{qn\}\{rn\}\}\dots$

```
274 \def\XINT_euc_b #1.#2#3#4%
275 {%
276   \XINT_euc_c #3\Z {#1}{#3}{#4}\{\{#2\}\{#3\}\}%
277 }%
```

$r(n+1)\Z \{n+1\}\{r(n+1)\}\{r(n)\}\{\{q(n+1)\}\{r(n+1)\}\}\{\{qn\}\{rn\}\}\dots$   
 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 }%
```

$\{n+1\}\{r(n+1)\}\{r(n)\}\{\{q(n+1)\}\{r(n+1)\}\}\dots\}\Z$  Ici  $r(n+1) = 0$ . On arrête on se prépare à inverser  
 $\{n+1\}\{0\}\{r(n)\}\{\{q(n+1)\}\{r(n+1)\}\}\dots\{\{q_1\}\{r_1\}\}\{\{A\}\{B\}\}\}\Z$   
 On veut renvoyer:  $\{N=n+1\}\{A\}\{D=r(n)\}\{B\}\{q_1\}\{r_1\}\{q_2\}\{r_2\}\{q_3\}\{r_3\}\dots\{q_N\}\{r_N=0\}$

```
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}}%

```

## 7.7 \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}{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\xintiiadd {\xintiiMul {#6}{#2}}{#8}}%
323     {\romannumeral0\xintiiadd {\xintiiMul {#5}{#2}}{#7}}%
324     {#1}{#2}{#3}{#4}{#5}{#6}%
325 }%

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

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

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

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

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

334 \def\XINT_bezalg_e #1#2\Z
335 {%
336     \xint_gob_til_zero #1\XINT_bezalg_end0\XINT_bezalg_a
337 }%

    Ici r(n+1) = 0. On arrête on se prépare à inverser.
    {n+1}{r(n+1)}{r(n)}{\alpha(n+1)}{\beta(n+1)}{\alpha(n)}{\beta(n)}
    {q,r,alpha,beta(n+1)}...{A}{B}}{\Z
    On veut renvoyer
    {N}{A}{0}{1}{D=r(n)}{B}{1}{0}{q1}{r1}{alpha1=q1}{beta1=1}
    {q2}{r2}{alpha2}{beta2}...{qN}{rN=0}{alphaN=A/D}{betaN=B/D}

338 \def\XINT_bezalg_end0\XINT_bezalg_a #1#2#3#4#5#6#7#8\Z
339 {%
340     \expandafter\XINT_bezalg_end_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}}%

```

## 7.8 \xintGCDof

1.2l 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}%
```

## 7.9 \xintLCMof

New with 1.09a

1.2l 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}%
```

## 7.10 \xintTypesetEuclideanAlgorithm

### TYPESETTING

Organisation:

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

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

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

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

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

```
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   \beginngroup
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 }%

```

## 7.11 \xintTypesetBezoutAlgorithm

Pour Bezout on a:  $\{N\}\{A\}\{0\}\{1\}\{D=r(n)\}\{B\}\{1\}\{0\}\{q1\}\{r1\}\{\alpha1=q1\}\{\beta1=1\}$   
 $\{q2\}\{r2\}\{\alpha2\}\{\beta2\}\dots\{qN\}\{rN=0\}\{\alphaN=A/D\}\{\betaN=B/D\}$

Donc  $4N+8$  termes:  $U1 = N$ ,  $U2 = A$ ,  $U5 = D$ ,  $U6 = B$ ,  $q1 = U9$ ,  $qn = U\{4n+5\}$ ,  $n$  au moins 1  
 $rn = U\{4n+6\}$ ,  $n$  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%
```

## 8 Package **xintfrac** implementation

.1	Catcodes, $\varepsilon$ -TeX and reload detection . . .	180	.44	<code>\xintPFactorial</code> . . . . .	213
.2	Package identification . . . . .	180	.45	<code>\xintPrd</code> . . . . .	214
.3	<code>\XINT_cntSgnFork</code> . . . . .	181	.46	<code>\xintDiv</code> . . . . .	214
.4	<code>\xintLen</code> . . . . .	181	.47	<code>\xintDivFloor</code> . . . . .	215
.5	<code>\XINT_outfrac</code> . . . . .	181	.48	<code>\xintDivTrunc</code> . . . . .	215
.6	<code>\XINT_inFrac</code> . . . . .	182	.49	<code>\xintDivRound</code> . . . . .	215
.7	<code>\XINT_frac_gen</code> . . . . .	184	.50	<code>\xintModTrunc</code> . . . . .	215
.8	<code>\XINT_factortens</code> . . . . .	186	.51	<code>\xintDivMod</code> . . . . .	216
.9	<code>\xintEq</code> , <code>\xintNotEq</code> , <code>\xintGt</code> , <code>\xintLt</code> , <code>\xintGtorEq</code> , <code>\xintLtorEq</code> , <code>\xintIsZero</code> , <code>\xintIsNotZero</code> , <code>\xintOdd</code> , <code>\xintEven</code> , <code>\xintifSgn</code> , <code>\xintifCmp</code> , <code>\xintifEq</code> , <code>\xin-</code> <code>tifGt</code> , <code>\xintifLt</code> , <code>\xintifZero</code> , <code>\xin-</code> <code>tifNotZero</code> , <code>\xintifOne</code> , <code>\xintifOdd</code> . . . . .	187	.52	<code>\xintMod</code> . . . . .	217
.10	<code>\xintRaw</code> . . . . .	189	.53	<code>\xintIsOne</code> . . . . .	218
.11	<code>\xintiLogTen</code> . . . . .	189	.54	<code>\xintGeq</code> . . . . .	218
.12	<code>\xintPRaw</code> . . . . .	190	.55	<code>\xintMax</code> . . . . .	219
.13	<code>\xintRawWithZeros</code> . . . . .	190	.56	<code>\xintMaxof</code> . . . . .	220
.14	<code>\xintDecToString</code> . . . . .	191	.57	<code>\xintMin</code> . . . . .	220
.15	<code>\xintFloor</code> , <code>\xintiFloor</code> . . . . .	191	.58	<code>\xintMinof</code> . . . . .	221
.16	<code>\xintCeil</code> , <code>\xintiCeil</code> . . . . .	191	.59	<code>\xintCmp</code> . . . . .	221
.17	<code>\xintNumerator</code> . . . . .	192	.60	<code>\xintAbs</code> . . . . .	223
.18	<code>\xintDenominator</code> . . . . .	192	.61	<code>\xintOpp</code> . . . . .	223
.19	<code>\xintFrac</code> . . . . .	192	.62	<code>\xintInv</code> . . . . .	223
.20	<code>\xintSignedFrac</code> . . . . .	193	.63	<code>\xintSgn</code> . . . . .	224
.21	<code>\xintFwOver</code> . . . . .	193	.64	Floating point macros . . . . .	224
.22	<code>\xintSignedFwOver</code> . . . . .	194	.65	<code>\xintFloat</code> . . . . .	224
.23	<code>\xintREZ</code> . . . . .	194	.66	<code>\XINTinFloat</code> , <code>\XINTinFloatS</code> , <code>\XIN-</code> <code>TiLogTen</code> . . . . .	226
.24	<code>\xintE</code> . . . . .	195	.67	<code>\xintPFloat</code> . . . . .	232
.25	<code>\xintIrr</code> , <code>\xintPIrr</code> . . . . .	195	.68	<code>\XINTinFloatFracdigits</code> . . . . .	234
.26	<code>\xintifInt</code> . . . . .	197	.69	<code>\xintFloatAdd</code> , <code>\XINTinFloatAdd</code> . . . . .	234
.27	<code>\xintIsInt</code> . . . . .	197	.70	<code>\xintFloatSub</code> , <code>\XINTinFloatSub</code> . . . . .	235
.28	<code>\xintJrr</code> . . . . .	197	.71	<code>\xintFloatMul</code> , <code>\XINTinFloatMul</code> . . . . .	236
.29	<code>\xintTFrac</code> . . . . .	199	.72	<code>\XINTinFloatInv</code> . . . . .	237
.30	<code>\xintTrunc</code> , <code>\xintiTrunc</code> . . . . .	199	.73	<code>\xintFloatDiv</code> , <code>\XINTinFloatDiv</code> . . . . .	237
.31	<code>\xintTTrunc</code> . . . . .	202	.74	<code>\xintFloatPow</code> , <code>\XINTinFloatPow</code> . . . . .	238
.32	<code>\xintNum</code> . . . . .	202	.75	<code>\xintFloatPower</code> , <code>\XINTinFloatPower</code> . . . . .	241
.33	<code>\xintRound</code> , <code>\xintiRound</code> . . . . .	202	.76	<code>\xintFloatFac</code> , <code>\XINTFloatFac</code> . . . . .	245
.34	<code>\xintXTrunc</code> . . . . .	203	.77	<code>\xintFloatPFactorial</code> , <code>\XINTinFloatP-</code> <code>Factorial</code> . . . . .	250
.35	<code>\xintDigits</code> . . . . .	209	.78	<code>\xintFloatBinomial</code> , <code>\XINTinFloatBino-</code> <code>mial</code> . . . . .	254
.36	<code>\xintAdd</code> . . . . .	209	.79	<code>\xintFloatSqrt</code> , <code>\XINTinFloatSqrt</code> . . . . .	256
.37	<code>\xintSub</code> . . . . .	211	.80	<code>\xintFloatE</code> , <code>\XINTinFloatE</code> . . . . .	258
.38	<code>\xintSum</code> . . . . .	211	.81	<code>\XINTinFloatMod</code> . . . . .	259
.39	<code>\xintMul</code> . . . . .	211	.82	<code>\XINTinFloatDivFloor</code> . . . . .	259
.40	<code>\xintSqr</code> . . . . .	212	.83	<code>\XINTinFloatDivMod</code> . . . . .	259
.41	<code>\xintPow</code> . . . . .	212	.84	<code>\xintifFloatInt</code> . . . . .	260
.42	<code>\xintFac</code> . . . . .	213	.85	<code>\xintFloatIsInt</code> . . . . .	260
.43	<code>\xintBinomial</code> . . . . .	213	.86	(WIP) <code>\XINTinRandomFloatS</code> , <code>\XINTinRan-</code> <code>domFloatSdigits</code> . . . . .	260
			.87	(WIP) <code>\XINTinRandomFloatSixteen</code> . . . . .	261

The commenting is currently (2019/04/05) very sparse.

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

The code for reload detection was initially copied from ΗΕΙΚΟ ΟΒΕΡΔΙΕΚ'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

```

## 8.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintfrac}%
46 [2019/04/05 1.3e Expandable operations on fractions (JFB)]%

```

### 8.3 \XINT\_cntSgnFork

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

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

### 8.4 \xintLen

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

```
54 \def\xintLen {\romannumeral0\xintlen }%
55 \def\xintlen #1%
56 {%
57   \expandafter\XINT_flen\romannumeral0\XINT_infrac {#1}%
58 }%
59 \def\XINT_flen#1{\def\XINT_flen ##1##2##3%
60 {%
61   \expandafter#1%

62   \the\numexpr \XINT_abs##1+%
63   \XINT_len_fork ##2##3\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
64   \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
65   \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye-\xint_c_i
66   \relax
67 }}\XINT_flen{ }%
```

### 8.5 \XINT\_outfrac

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

```
68 \def\XINT_outfrac #1#2#3%
69 {%
70   \ifcase\XINT_cntSgn #3\xint:
71     \expandafter \XINT_outfrac_divisionbyzero
72   \or
73     \expandafter \XINT_outfrac_P
74   \else
75     \expandafter \XINT_outfrac_N
76   \fi
77   {#2}{#3}[#1]%
```

```

78 }%
79 \def\XINT_outfrac_divisionbyzero #1#2%
80 {%
81   \XINT_signalcondition{DivisionByZero}{Division of #1 by #2}{0/1[0]}%
82 }%
83 \def\XINT_outfrac_P#1{%
84 \def\XINT_outfrac_P ##1##2%
85   {\if0\XINT_Sgn ##1\xint:\expandafter\XINT_outfrac_Zero\fi##1/##2}%
86 }\XINT_outfrac_P{ }%
87 \def\XINT_outfrac_Zero #1[#2]{ 0/1[0]}%
88 \def\XINT_outfrac_N #1#2%
89 {%
90   \expandafter\XINT_outfrac_N_a\expandafter
91   {\romannumeral0\XINT_opp #2}{\romannumeral0\XINT_opp #1}%
92 }%
93 \def\XINT_outfrac_N_a #1#2%
94 {%
95   \expandafter\XINT_outfrac_P\expandafter {#2}{#1}%
96 }%

```

## 8.6 \XINT\_inFrac

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

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

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

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

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

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

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

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

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

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

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

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`.

```

97 \def\XINT_inFrac {\romannumeral0\XINT_infrac }%
98 \def\XINT_infrac #1%
99 {%
100   \expandafter\XINT_infrac_fork\romannumeral`&&@#1\xint:\XINT_W\XINT_W\XINT_T
101 }%
102 \def\XINT_infrac_fork #1[#2%
103 {%
104   \xint_UDXINTWfork
105   #2\XINT_frac_gen           % input has no brackets [N]
106   \XINT_W\XINT_infrac_res_a % there is some [N], must be strict A[N] or A/B[N] input
107   \krof
108   #1[#2%
109 }%
110 \def\XINT_infrac_res_a #1%
111 {%
112   \xint_gob_til_zero #1\XINT_infrac_res_zero 0\XINT_infrac_res_b #1%
113 }%

```

Note that input exponent is here ignored and forced to be zero.

```

114 \def\XINT_infrac_res_zero 0\XINT_infrac_res_b #1\XINT_T {{0}{0}{1}}%
115 \def\XINT_infrac_res_b #1/#2%
116 {%
117   \xint_UDXINTWfork
118   #2\XINT_infrac_res_ca      % it was A[N] input
119   \XINT_W\XINT_infrac_res_cb % it was A/B[N] input
120   \krof
121   #1/#2%
122 }%

```

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.

```

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

```

## 8.7 \XINT\_frac\_gen

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

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

```
129 \def\XINT_frac_gen #1/#2%
130 {%
131   \xint_UDXINTWfork
132   #2\XINT_frac_gen_A      % there was no /
133   \XINT_W\XINT_frac_gen_B % there was a /
134   \krof
135   #1/#2%
136 }%
```

Note that #1 is only expanded so far up to decimal mark or "e".

```
137 \def\XINT_frac_gen_A #1\xint:\XINT_W [\XINT_W {\XINT_frac_gen_C 0~1!#1ee.\XINT_W }%
138 \def\XINT_frac_gen_B #1/#2\xint:\XINT_W[\XINT_W
139 {%
140   \expandafter\XINT_frac_gen_Ba
141   \romannumeral`&&@#2ee.\XINT_W\XINT_Z #1ee.%\XINT_W
142 }%
143 \def\XINT_frac_gen_Ba #1.#2%
144 {%
145   \xint_UDXINTWfork
146   #2\XINT_frac_gen_Bb
147   \XINT_W\XINT_frac_gen_Bc
148   \krof
149   #1.#2%
150 }%
151 \def\XINT_frac_gen_Bb #1e#2e#3\XINT_Z
152   {\expandafter\XINT_frac_gen_C\the\numexpr #2+\xint_c_~#1!}%
153 \def\XINT_frac_gen_Bc #1.#2e%
154 {%
155   \expandafter\XINT_frac_gen_Bd\romannumeral`&&@#2.#1e%
156 }%
157 \def\XINT_frac_gen_Bd #1.#2e#3e#4\XINT_Z
158 {%
159   \expandafter\XINT_frac_gen_C\the\numexpr #3-%
160   \numexpr\XINT_length_loop
161   #1\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
162   \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
163   \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
164   ~#2#1!%
165 }%
166 \def\XINT_frac_gen_C #1!#2.#3%
167 {%
168   \xint_UDXINTWfork
```



```

169     #3\XINT_frac_gen_Ca
170     \XINT_W\XINT_frac_gen_Cb
171     \krof
172     #1!#2.#3%
173 }%
174 \def\XINT_frac_gen_Ca #1~#2!#3e#4e#5\XINT_T
175 {%
176     \expandafter\XINT_frac_gen_F\the\numexpr #4-#1\expandafter
177     ~\romannumeral0\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
178     #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z~#3~%
179 }%
180 \def\XINT_frac_gen_Cb #1.#2e%
181 {%
182     \expandafter\XINT_frac_gen_Cc\romannumeral`&&@#2.#1e%
183 }%
184 \def\XINT_frac_gen_Cc #1.#2~#3!#4e#5e#6\XINT_T
185 {%
186     \expandafter\XINT_frac_gen_F\the\numexpr #5-#2-%

187     \numexpr\XINT_length_loop
188     #1\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
189     \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
190     \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye

191     \relax\expandafter~%
192     \romannumeral0\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
193     #3\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z
194     ~#4#1~%
195 }%
196 \def\XINT_frac_gen_F #1~#2%
197 {%
198     \xint_UDzerominusfork
199     #2-\XINT_frac_gen_Gdivbyzero
200     0#2{\XINT_frac_gen_G -{}}%
201     0-{\XINT_frac_gen_G }#2}%
202     \krof #1~%
203 }%
204 \def\XINT_frac_gen_Gdivbyzero #1~#2~%
205 {%
206     \expandafter\XINT_frac_gen_Gdivbyzero_a
207     \romannumeral0\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
208     #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z~#1~%
209 }%
210 \def\XINT_frac_gen_Gdivbyzero_a #1~#2~%
211 {%
212     \XINT_signalcondition{DivisionByZero}{Division of #1 by zero}{#2}{#1}{0}%
213 }%
214 \def\XINT_frac_gen_G #1#2#3~#4~#5~%
215 {%
216     \expandafter\XINT_frac_gen_Ga
217     \romannumeral0\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
218     #1#5\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z~#3~#2#4}%

```

```

219 }%
220 \def\XINT_frac_gen_Ga #1#2~#3~%
221 {%
222     \xint_gob_til_zero #1\XINT_frac_gen_zero 0%
223     {#3}{#1#2}%
224 }%
225 \def\XINT_frac_gen_zero 0#1#2#3{{0}{0}{1}}%

```

## 8.8 \XINT\_factortens

This is the core macro for `\xintREZ`. To be used as `\romannumeral0\XINT_factortens{...}`. Output is A.N. (formerly `{A}{N}`) where A is the integer stripped from trailing zeroes and N is the number of removed zeroes. Only for positive strict integers!

Completely rewritten at 1.3a to replace a double `\xintReverseOrder` by a direct `\numexpr` governed expansion to the end and back, à la 1.2. I should comment more... and perhaps improve again in future.

Testing shows significant gain at 100 digits or more.

```

226 \def\XINT_factortens #1{\expandafter\XINT_factortens_z
227     \romannumeral0\XINT_factortens_a#1%
228     \XINT_factortens_b123456789.}%
229 \def\XINT_factortens_z.\XINT_factortens_y{ }%
230 \def\XINT_factortens_a #1#2#3#4#5#6#7#8#9%
231     {\expandafter\XINT_factortens_x
232     \the\numexpr 1#1#2#3#4#5#6#7#8#9\XINT_factortens_a}%
233 \def\XINT_factortens_b#1\XINT_factortens_a#2#3.%
234     {.\XINT_factortens_cc 000000000-#2.}%
235 \def\XINT_factortens_x1#1.#2{#2#1}%
236 \def\XINT_factortens_y{.\XINT_factortens_y}%
237 \def\XINT_factortens_cc #1#2#3#4#5#6#7#8#9%
238     {\if#90\xint_dothis
239     {\expandafter\XINT_factortens_d\the\numexpr #8#7#6#5#4#3#2#1\relax
240     \xint_c_i 2345678.}\fi
241     \xint_orthat{\XINT_factortens_yy{#1#2#3#4#5#6#7#8#9}}}%
242 \def\XINT_factortens_yy #1#2.{.\XINT_factortens_y#1.0.}%
243 \def\XINT_factortens_c #1#2#3#4#5#6#7#8#9%
244     {\if#90\xint_dothis
245     {\expandafter\XINT_factortens_d\the\numexpr #8#7#6#5#4#3#2#1\relax
246     \xint_c_i 2345678.}\fi
247     \xint_orthat{.\XINT_factortens_y #1#2#3#4#5#6#7#8#9.}}}%
248 \def\XINT_factortens_d #1#2#3#4#5#6#7#8#9%
249     {\if#10\expandafter\XINT_factortens_e\fi
250     \XINT_factortens_f #9#9#8#7#6#5#4#3#2#1.}%
251 \def\XINT_factortens_f #1#2\xint_c_i#3.#4.#5.%
252     {\expandafter\XINT_factortens_g\the\numexpr#1+#5.#3.}%
253 \def\XINT_factortens_g #1.#2.{.\XINT_factortens_y#2.#1.}%
254 \def\XINT_factortens_e #1..#2.%
255     {\expandafter.\expandafter\XINT_factortens_c
256     \the\numexpr\xint_c_ix+#2.}%

```

## 8.9 `\xintEq`, `\xintNotEq`, `\xintGt`, `\xintLt`, `\xintGtorEq`, `\xintLtorEq`, `\xintIsZero`, `\xintIsNotZero`, `\xintOdd`, `\xintEven`, `\xintifSgn`, `\xintifCmp`, `\xintifEq`, `\xintifGt`, `\xintifLt`, `\xintifZero`, `\xintifNotZero`, `\xintifOne`, `\xintifOdd`

Moved here at 1.3. Formerly these macros were already defined in `xint.sty` or even `xintcore.sty`. They are slim wrappers of macros defined elsewhere in `xintfrac`.

```

257 \def\xintEq    {\romannumeral0\xinteq }%
258 \def\xinteq   #1#2{\xintifeq{#1}{#2}{1}{0}}%
259 \def\xintNotEq#1#2{\romannumeral0\xintifeq {#1}{#2}{0}{1}}%
260 \def\xintGt   {\romannumeral0\xintgt }%
261 \def\xintgt   #1#2{\xintifgt{#1}{#2}{1}{0}}%
262 \def\xintLt   {\romannumeral0\xintlt }%
263 \def\xintlt   #1#2{\xintiflt{#1}{#2}{1}{0}}%
264 \def\xintGtorEq #1#2{\romannumeral0\xintiflt {#1}{#2}{0}{1}}%
265 \def\xintLtorEq #1#2{\romannumeral0\xintifgt {#1}{#2}{0}{1}}%
266 \def\xintIsZero  {\romannumeral0\xintiszero }%
267 \def\xintiszero #1{\if0\xintSgn{#1}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%
268 \def\xintIsNotZero{\romannumeral0\xintisnotzero }%
269 \def\xintisnotzero
270     #1{\if0\xintSgn{#1}\xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi}%
271 \def\xintOdd     {\romannumeral0\xintodd }%
272 \def\xintodd #1%
273 {%
274     \ifodd\xintLDg{\xintNum{#1}} %<- intentional space
275     \xint_afterfi{ 1}%
276     \else
277     \xint_afterfi{ 0}%
278     \fi
279 }%
280 \def\xintEven    {\romannumeral0\xinteven }%
281 \def\xinteven #1%
282 {%
283     \ifodd\xintLDg{\xintNum{#1}} %<- intentional space
284     \xint_afterfi{ 0}%
285     \else
286     \xint_afterfi{ 1}%
287     \fi
288 }%
289 \def\xintifSgn{\romannumeral0\xintifsgn }%
290 \def\xintifsgn #1%
291 {%
292     \ifcase \xintSgn{#1}
293         \expandafter\xint_stop_atsecondofthree
294         \or\expandafter\xint_stop_atthirdofthree
295         \else\expandafter\xint_stop_atfirstofthree
296     \fi
297 }%
298 \def\xintifCmp{\romannumeral0\xintifcmp }%
299 \def\xintifcmp #1#2%
300 {%
301     \ifcase\xintCmp {#1}{#2}
302         \expandafter\xint_stop_atsecondofthree

```

```

303         \or\expandafter\xint_stop_atthirdofthree
304         \else\expandafter\xint_stop_atfirstofthree
305     \fi
306 }%
307 \def\xintifEq {\romannumeral0\xintifeq }%
308 \def\xintifeq #1#2%
309 {%
310     \if0\xintCmp{#1}{#2}%
311         \expandafter\xint_stop_atfirstoftwo
312     \else\expandafter\xint_stop_atsecondoftwo
313     \fi
314 }%
315 \def\xintifGt {\romannumeral0\xintifgt }%
316 \def\xintifgt #1#2%
317 {%
318     \if1\xintCmp{#1}{#2}%
319         \expandafter\xint_stop_atfirstoftwo
320     \else\expandafter\xint_stop_atsecondoftwo
321     \fi
322 }%
323 \def\xintifLt {\romannumeral0\xintiflt }%
324 \def\xintiflt #1#2%
325 {%
326     \ifnum\xintCmp{#1}{#2}<\xint_c_
327         \expandafter\xint_stop_atfirstoftwo
328     \else \expandafter\xint_stop_atsecondoftwo
329     \fi
330 }%
331 \def\xintifZero {\romannumeral0\xintifzero }%
332 \def\xintifzero #1%
333 {%
334     \if0\xintSgn{#1}%
335         \expandafter\xint_stop_atfirstoftwo
336     \else
337         \expandafter\xint_stop_atsecondoftwo
338     \fi
339 }%
340 \def\xintifNotZero{\romannumeral0\xintifnotzero }%
341 \def\xintifnotzero #1%
342 {%
343     \if0\xintSgn{#1}%
344         \expandafter\xint_stop_atsecondoftwo
345     \else
346         \expandafter\xint_stop_atfirstoftwo
347     \fi
348 }%
349 \def\xintifOne {\romannumeral0\xintifone }%
350 \def\xintifone #1%
351 {%
352     \if1\xintIsOne{#1}%
353         \expandafter\xint_stop_atfirstoftwo
354     \else

```

```

355     \expandafter\xint_stop_atsecondoftwo
356   \fi
357 }%
358 \def\xintifOdd {\romannumeral0\xintifodd }%
359 \def\xintifodd #1%
360 {%
361   \if\xintOdd{#1}1%
362     \expandafter\xint_stop_atfirstoftwo
363   \else
364     \expandafter\xint_stop_atsecondoftwo
365   \fi
366 }%

```

## 8.10 \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.

```

367 \def\xintRaw {\romannumeral0\xintraaw }%
368 \def\xintraaw
369 {%
370   \expandafter\XINT_raw\romannumeral0\XINT_infrac
371 }%
372 \def\XINT_raw #1#2#3{ #2/#3[#1]}%

```

## 8.11 \xintiLogTen

New at 1.3e

```

373 \def\xintiLogTen {\the\numexpr\xintiLogten}%
374 \def\xintiLogten
375 {%
376   \expandafter\XINT_ilogten\romannumeral0\xintraaw
377 }%
378 \def\XINT_ilogten #1%
379 {%
380   \xint_UDzerominusfork
381     0#1\XINT_ilogten_p
382     #1-\XINT_ilogten_z
383     0-{\XINT_ilogten_p#1}%
384   \krof
385 }%
386 \def\XINT_ilogten_z #1[#2]{-"7FFF8000\relax}%
387 \def\XINT_ilogten_p #1/#2[#3]%
388 {%
389   #3+\expandafter\XINT_ilogten_a
390     \the\numexpr\xintLength{#1}\expandafter.\the\numexpr\xintLength{#2}.#1.#2.%
391 }%
392 \def\XINT_ilogten_a #1.#2.%
393 {%
394   #1-#2\ifnum#1>#2
395     \expandafter\XINT_ilogten_aa
396   \else

```

```

397     \expandafter\XINT_ilogten_ab
398     \fi #1.#2.%
399 }%
400 \def\XINT_ilogten_aa #1.#2.#3.#4.%
401 {%
402     \xintiiiflt{#3}{\XINT_dsx_addzerosnofuss{#1-#2}#4;}{-1}{}\relax
403 }%
404 \def\XINT_ilogten_ab #1.#2.#3.#4.%
405 {%
406     \xintiiiflt{\XINT_dsx_addzerosnofuss{#2-#1}#3;}{#4}{-1}{}\relax
407 }%

```

## 8.12 \xintPRaw

### 1.09b

```

408 \def\xintPRaw {\romannumeral0\xintpraw }%
409 \def\xintpraw
410 {%
411     \expandafter\XINT_praw\romannumeral0\XINT_infrac
412 }%
413 \def\XINT_praw #1%
414 {%
415     \ifnum #1=\xint_c_ \expandafter\XINT_praw_a\fi \XINT_praw_A {#1}%
416 }%
417 \def\XINT_praw_A #1#2#3%
418 {%
419     \if\XINT_isOne{#3}1\expandafter\xint_firstoftwo
420         \else\expandafter\xint_secondoftwo
421     \fi { #2[#1]}{ #2/#3[#1]}%
422 }%
423 \def\XINT_praw_a\XINT_praw_A #1#2#3%
424 {%
425     \if\XINT_isOne{#3}1\expandafter\xint_firstoftwo
426         \else\expandafter\xint_secondoftwo
427     \fi { #2}{ #2/#3}%
428 }%

```

## 8.13 \xintRawWithZeros

This was called \xintRaw in versions earlier than 1.07

```

429 \def\xintRawWithZeros {\romannumeral0\xintrawwithzeros }%
430 \def\xintrawwithzeros
431 {%
432     \expandafter\XINT_rawz_fork\romannumeral0\XINT_infrac
433 }%

434 \def\XINT_rawz_fork #1%
435 {%
436     \ifnum#1<\xint_c_
437         \expandafter\XINT_rawz_Ba

```

```

438 \else
439 \expandafter\XINT_rawz_A
440 \fi
441 #1.%
442 }%
443 \def\XINT_rawz_A #1.#2#3{\XINT_dsx_addzeros{#1}#2;/#3}%
444 \def\XINT_rawz_Ba -#1.#2#3{\expandafter\XINT_rawz_Bb
445 \expandafter{\romannumeral0\XINT_dsx_addzeros{#1}#3;}{#2}}%
446 \def\XINT_rawz_Bb #1#2{ #2/#1}%

```

## 8.14 \xintDecToString

1.3. This is a backport from `polexpr 0.4`. It is definitely not in final form, consider it to be an unstable macro.

```

447 \def\xintDecToString{\romannumeral0\xintdectostring}%
448 \def\xintdectostring#1{\expandafter\XINT_dectostr\romannumeral0\xintraew{#1}}%
449 \def\XINT_dectostr #1/#2[#3]{\xintiifZero {#1}%
450 \XINT_dectostr_z
451 {\if1\XINT_isOne{#2}\expandafter\XINT_dectostr_a
452 \else\expandafter\XINT_dectostr_b
453 \fi}%
454 #1/#2[#3]}%
455 }%
456 \def\XINT_dectostr_z#1[#2]{ 0}%
457 \def\XINT_dectostr_a#1/#2[#3]{%
458 \ifnum#3<\xint_c_\xint_dothis{\xinttrunc{-#3}{#1[#3]}}\fi
459 \xint_orthat{\xintiie{#1}{#3}}%
460 }%
461 \def\XINT_dectostr_b#1/#2[#3]{% just to handle this somehow
462 \ifnum#3<\xint_c_\xint_dothis{\xinttrunc{-#3}{#1[#3]}/#2}\fi
463 \xint_orthat{\xintiie{#1}{#3}/#2}%
464 }%

```

## 8.15 \xintFloor, \xintiFloor

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

```

465 \def\xintFloor {\romannumeral0\xintfloor }%
466 \def\xintfloor #1% devrais-je faire \xintREZ?
467 {\expandafter\XINT_ifloor \romannumeral0\xintraewwithzeros {#1}./1[0]}%
468 \def\xintiFloor {\romannumeral0\xintifloor }%
469 \def\xintifloor #1%
470 {\expandafter\XINT_ifloor \romannumeral0\xintraewwithzeros {#1}.}%
471 \def\XINT_ifloor #1/#2.{\xintiiquo {#1}{#2}}%

```

## 8.16 \xintCeil, \xintiCeil

1.09a

```

472 \def\xintCeil {\romannumeral0\xintceil }%
473 \def\xintceil #1{\xintiopp {\xintFloor {\xintOpp{#1}}}}%

```

```
474 \def\xintiCeil {\romannumeral0\xintceil }%
475 \def\xintceil #1{\xintiopp {\xintiFloor {\xintOpp{#1}}}}%
```

### 8.17 \xintNumerator

```
476 \def\xintNumerator {\romannumeral0\xintnumerator }%
477 \def\xintnumerator
478 {%
479   \expandafter\XINT_numer\romannumeral0\XINT_infrac
480 }%
481 \def\XINT_numer #1%
482 {%
483   \ifcase\XINT_cntSgn #1\xint:
484     \expandafter\XINT_numer_B
485   \or
486     \expandafter\XINT_numer_A
487   \else
488     \expandafter\XINT_numer_B
489   \fi
490   {#1}%
491 }%
492 \def\XINT_numer_A #1#2#3{\XINT_dsx_addzeros{#1}#2;}%
493 \def\XINT_numer_B #1#2#3{ #2}%
```

### 8.18 \xintDenominator

```
494 \def\xintDenominator {\romannumeral0\xintdenominator }%
495 \def\xintdenominator
496 {%
497   \expandafter\XINT_denom_fork\romannumeral0\XINT_infrac
498 }%
499 \def\XINT_denom_fork #1%
500 {%
501   \ifnum#1<\xint_c_
502     \expandafter\XINT_denom_B
503   \else
504     \expandafter\XINT_denom_A
505   \fi
506   #1.%
507 }%
508 \def\XINT_denom_A #1.#2#3{ #3}%
509 \def\XINT_denom_B -#1.#2#3{\XINT_dsx_addzeros{#1}#3;}%
```

### 8.19 \xintFrac

Useless typesetting macro.

```
510 \def\xintFrac {\romannumeral0\xintfrac }%
511 \def\xintfrac #1%
512 {%
513   \expandafter\XINT_fracfrac_A\romannumeral0\XINT_infrac {#1}%
514 }%
515 \def\XINT_fracfrac_A #1{\XINT_fracfrac_B #1\Z }%
```



```

516 \catcode`^=7
517 \def\XINT_fracfrac_B #1#2\Z
518 {%
519   \xint_gob_til_zero #1\XINT_fracfrac_C 0\XINT_fracfrac_D {10^{#1#2}}}%
520 }%
521 \def\XINT_fracfrac_C 0\XINT_fracfrac_D #1#2#3%
522 {%
523   \if1\XINT_isOne {#3}%
524   \xint_afterfi {\expandafter\xint_stop_atfirstoftwo\xint_gobble_ii }%
525   \fi
526   \space
527   \frac {#2}{#3}%
528 }%
529 \def\XINT_fracfrac_D #1#2#3%
530 {%
531   \if1\XINT_isOne {#3}\XINT_fracfrac_E\fi
532   \space
533   \frac {#2}{#3}#1%
534 }%
535 \def\XINT_fracfrac_E \fi\space\frac #1#2{\fi \space #1\cdot }%

```

## 8.20 \xintSignedFrac

```

536 \def\xintSignedFrac {\romannumeral0\xintsignedfrac }%
537 \def\xintsignedfrac #1%
538 {%
539   \expandafter\XINT_sgnfrac_a\romannumeral0\XINT_infrac {#1}%
540 }%
541 \def\XINT_sgnfrac_a #1#2%
542 {%
543   \XINT_sgnfrac_b #2\Z {#1}%
544 }%
545 \def\XINT_sgnfrac_b #1%
546 {%
547   \xint_UDsignfork
548   #1\XINT_sgnfrac_N
549   -{\XINT_sgnfrac_P #1}%
550   \krof
551 }%
552 \def\XINT_sgnfrac_P #1\Z #2%
553 {%
554   \XINT_fracfrac_A {#2}{#1}%
555 }%
556 \def\XINT_sgnfrac_N
557 {%
558   \expandafter-\romannumeral0\XINT_sgnfrac_P
559 }%

```

## 8.21 \xintFwOver

```

560 \def\xintFwOver {\romannumeral0\xintfwover }%
561 \def\xintfwover #1%
562 {%
563   \expandafter\XINT_fwover_A\romannumeral0\XINT_infrac {#1}%

```

```

564 }%
565 \def\XINT_fwover_A #1{\XINT_fwover_B #1\Z }%
566 \def\XINT_fwover_B #1#2\Z
567 {%
568   \xint_gob_til_zero #1\XINT_fwover_C 0\XINT_fwover_D {10^{#1#2}}%
569 }%
570 \catcode`^=11
571 \def\XINT_fwover_C #1#2#3#4#5%
572 {%
573   \if0\XINT_isOne {#5}\xint_afterfi { {#4\over #5}}%
574   \else\xint_afterfi { #4}%
575   \fi
576 }%
577 \def\XINT_fwover_D #1#2#3%
578 {%
579   \if0\XINT_isOne {#3}\xint_afterfi { {#2\over #3}}%
580   \else\xint_afterfi { #2\cdot }%
581   \fi
582   #1%
583 }%

```

## 8.22 \xintSignedFwOver

```

584 \def\xintSignedFwOver {\romannumeral0\xintsignedfwover }%
585 \def\xintsignedfwover #1%
586 {%
587   \expandafter\XINT_sgnfwover_a\romannumeral0\XINT_infrac {#1}%
588 }%
589 \def\XINT_sgnfwover_a #1#2%
590 {%
591   \XINT_sgnfwover_b #2\Z {#1}%
592 }%
593 \def\XINT_sgnfwover_b #1%
594 {%
595   \xint_UDsignfork
596   #1\XINT_sgnfwover_N
597   -{\XINT_sgnfwover_P #1}%
598   \krof
599 }%
600 \def\XINT_sgnfwover_P #1\Z #2%
601 {%
602   \XINT_fwover_A {#2}{#1}%
603 }%
604 \def\XINT_sgnfwover_N
605 {%
606   \expandafter-\romannumeral0\XINT_sgnfwover_P
607 }%

```

## 8.23 \xintREZ

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

The macro really doing the job \XINT\_factortens was redone at 1.3a. But speed gain really noticeable only beyond about 100 digits.

```

608 \def\xintREZ {\romannumeral0\xintrez }%
609 \def\xintrez
610 {%
611   \expandafter\XINT_rez_A\romannumeral0\XINT_infrac
612 }%
613 \def\XINT_rez_A #1#2%
614 {%
615   \XINT_rez_AB #2\Z {#1}%
616 }%
617 \def\XINT_rez_AB #1%
618 {%
619   \xint_UDzerominusfork
620   #1-\XINT_rez_zero
621   0#1\XINT_rez_neg
622   0-{\XINT_rez_B #1}%
623   \krof
624 }%
625 \def\XINT_rez_zero #1\Z #2#3{ 0/1[0]}%
626 \def\XINT_rez_neg {\expandafter-\romannumeral0\XINT_rez_B }%
627 \def\XINT_rez_B #1\Z
628 {%
629   \expandafter\XINT_rez_C\romannumeral0\XINT_factortens {#1}%
630 }%
631 \def\XINT_rez_C #1.#2.#3#4%
632 {%
633   \expandafter\XINT_rez_D\romannumeral0\XINT_factortens {#4}#3+#2.#1.%
634 }%
635 \def\XINT_rez_D #1.#2.#3.%
636 {%
637   \expandafter\XINT_rez_E\the\numexpr #3-#2.#1.%
638 }%
639 \def\XINT_rez_E #1.#2.#3.{ #3/#2[#1]}%

```

## 8.24 \xintE

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

```

640 \def\xintE {\romannumeral0\xinte }%
641 \def\xinte #1%
642 {%
643   \expandafter\XINT_e \romannumeral0\XINT_infrac {#1}%
644 }%
645 \def\XINT_e #1#2#3#4%
646 {%
647   \expandafter\XINT_e_end\the\numexpr #1+#4.{#2}{#3}%
648 }%
649 \def\XINT_e_end #1.#2#3{ #2/#3[#1]}%

```

## 8.25 \xintIrr, \xintPIrr

`\xintPIrr` (partial Irr, which ignores the decimal part) added at 1.3.

```

650 \def\xintIrr {\romannumeral0\xintirr }%
651 \def\xintPIrr{\romannumeral0\xintpirr }%
652 \def\xintirr #1%
653 {%
654   \expandafter\XINT_irr_start\romannumeral0\xintraawithzeros {#1}\Z
655 }%
656 \def\xintpirr #1%
657 {%
658   \expandafter\XINT_pirr_start\romannumeral0\xintraaw{#1}%
659 }%
660 \def\XINT_irr_start #1#2/#3\Z
661 {%
662   \if0\XINT_isOne {#3}%
663     \xint_afterfi
664       {\xint_UDsignfork
665         #1\XINT_irr_negative
666         -{\XINT_irr_nonneg #1}%
667         \krof}%
668   \else
669     \xint_afterfi{\XINT_irr_denomisine #1}%
670   \fi
671   #2\Z {#3}%
672 }%
673 \def\XINT_pirr_start #1#2/#3[%
674 {%
675   \if0\XINT_isOne {#3}%
676     \xint_afterfi
677       {\xint_UDsignfork
678         #1\XINT_irr_negative
679         -{\XINT_irr_nonneg #1}%
680         \krof}%
681   \else
682     \xint_afterfi{\XINT_irr_denomisine #1}%
683   \fi
684   #2\Z {#3}[%
685 }%
686 \def\XINT_irr_denomisine #1\Z #2{ #1/1}% changed in 1.08
687 \def\XINT_irr_negative #1\Z #2{\XINT_irr_D #1\Z #2\Z -}%
688 \def\XINT_irr_nonneg #1\Z #2{\XINT_irr_D #1\Z #2\Z \space}%
689 \def\XINT_irr_D #1#2\Z #3#4\Z
690 {%
691   \xint_UDzerosfork
692     #3#1\XINT_irr_indeterminate
693     #30\XINT_irr_divisionbyzero
694     #10\XINT_irr_zero
695     00\XINT_irr_loop_a
696   \krof
697   {#3#4}{#1#2}{#3#4}{#1#2}%
698 }%
699 \def\XINT_irr_indeterminate #1#2#3#4#5%
700 {%
701   \XINT_signalcondition{DivisionUndefined}{indeterminate: 0/0}{0/1}%

```

```

702 }%
703 \def\XINT_irr_divisionbyzero #1#2#3#4#5%
704 {%
705   \XINT_signalcondition{DivisionByZero}{vanishing denominator: #5#2/0}{0/1}%
706 }%
707 \def\XINT_irr_zero #1#2#3#4#5{ 0/1}% changed in 1.08
708 \def\XINT_irr_loop_a #1#2%
709 {%
710   \expandafter\XINT_irr_loop_d
711   \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
712 }%
713 \def\XINT_irr_loop_d #1#2%
714 {%
715   \XINT_irr_loop_e #2\Z
716 }%
717 \def\XINT_irr_loop_e #1#2\Z
718 {%
719   \xint_gob_til_zero #1\XINT_irr_loop_exit0\XINT_irr_loop_a {#1#2}%
720 }%
721 \def\XINT_irr_loop_exit0\XINT_irr_loop_a #1#2#3#4%
722 {%
723   \expandafter\XINT_irr_loop_exitb\expandafter
724   {\romannumeral0\xintiigo {#3}{#2}}%
725   {\romannumeral0\xintiigo {#4}{#2}}%
726 }%
727 \def\XINT_irr_loop_exitb #1#2%
728 {%
729   \expandafter\XINT_irr_finish\expandafter {#2}{#1}%
730 }%
731 \def\XINT_irr_finish #1#2#3{#3#1/#2}% changed in 1.08

```

## 8.26 \xintifInt

```

732 \def\xintifInt {\romannumeral0\xintifint }%
733 \def\xintifint #1{\expandafter\XINT_ifint\romannumeral0\xintrawwithzeros {#1}.}%
734 \def\XINT_ifint #1/#2.%
735 {%
736   \if 0\xintiiRem {#1}{#2}%
737   \expandafter\xint_stop_atfirstoftwo
738   \else
739   \expandafter\xint_stop_atsecondoftwo
740   \fi
741 }%

```

## 8.27 \xintIsInt

Added at 1.3d only, for `isint()` *xintexpr* function.

```

742 \def\xintIsInt {\romannumeral0\xintisint }%
743 \def\xintisint #1%
744   {\expandafter\XINT_ifint\romannumeral0\xintrawwithzeros {#1}.10}%

```

## 8.28 \xintJrr

```

745 \def\xintJrr {\romannumeral0\xintjrr }%
746 \def\xintjrr #1%
747 {%
748   \expandafter\XINT_jrr_start\romannumeral0\xintraawwithzeros {#1}\Z
749 }%
750 \def\XINT_jrr_start #1#2/#3\Z
751 {%
752   \if0\XINT_isOne {#3}\xint_afterfi
753     {\xint_UDsignfork
754       #1\XINT_jrr_negative
755       -{\XINT_jrr_nonneg #1}%
756       \krof}%
757   \else
758     \xint_afterfi{\XINT_jrr_denomisone #1}%
759   \fi
760   #2\Z {#3}%
761 }%
762 \def\XINT_jrr_denomisone #1\Z #2{ #1/1}% changed in 1.08
763 \def\XINT_jrr_negative #1\Z #2{\XINT_jrr_D #1\Z #2\Z -}%
764 \def\XINT_jrr_nonneg #1\Z #2{\XINT_jrr_D #1\Z #2\Z \space}%
765 \def\XINT_jrr_D #1#2\Z #3#4\Z
766 {%
767   \xint_UDzerosfork
768     #3#1\XINT_jrr_indeterminate
769     #30\XINT_jrr_divisionbyzero
770     #10\XINT_jrr_zero
771     00\XINT_jrr_loop_a
772   \krof
773   {#3#4}{#1#2}1001%
774 }%
775 \def\XINT_jrr_indeterminate #1#2#3#4#5#6#7%
776 {%
777   \XINT_signalcondition{DivisionUndefined}{indeterminate: 0/0}{0/1}%
778 }%
779 \def\XINT_jrr_divisionbyzero #1#2#3#4#5#6#7%
780 {%
781   \XINT_signalcondition{DivisionByZero}{Vanishing denominator: #7#2/0}{0/1}%
782 }%
783 \def\XINT_jrr_zero #1#2#3#4#5#6#7{ 0/1}% changed in 1.08
784 \def\XINT_jrr_loop_a #1#2%
785 {%
786   \expandafter\XINT_jrr_loop_b
787   \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
788 }%
789 \def\XINT_jrr_loop_b #1#2#3#4#5#6#7%
790 {%
791   \expandafter \XINT_jrr_loop_c \expandafter
792     {\romannumeral0\xintiiadd{\XINT_mul_fork #4\xint:#1\xint:}{#6}}%
793     {\romannumeral0\xintiiadd{\XINT_mul_fork #5\xint:#1\xint:}{#7}}%
794     {#2}{#3}{#4}{#5}%
795 }%
796 \def\XINT_jrr_loop_c #1#2%

```

```

797 {%
798   \expandafter \XINT_jrr_loop_d \expandafter{#2}{#1}%
799 }%
800 \def\XINT_jrr_loop_d #1#2#3#4%
801 {%
802   \XINT_jrr_loop_e #3\Z {#4}{#2}{#1}%
803 }%
804 \def\XINT_jrr_loop_e #1#2\Z
805 {%
806   \xint_gob_til_zero #1\XINT_jrr_loop_exit0\XINT_jrr_loop_a {#1#2}%
807 }%
808 \def\XINT_jrr_loop_exit0\XINT_jrr_loop_a #1#2#3#4#5#6%
809 {%
810   \XINT_irr_finish {#3}{#4}%
811 }%

```

## 8.29 \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`.

```

812 \def\xintTFrac {\romannumeral0\xinttfrac }%
813 \def\xinttfrac #1{\expandafter\XINT_tfrac_fork\romannumeral0\xintraewithzeros {#1}\Z }%
814 \def\XINT_tfrac_fork #1%
815 {%
816   \xint_UDzerominusfork
817     #1-\XINT_tfrac_zero
818     0#1{\xintiiopp\XINT_tfrac_P }%
819     0-{\XINT_tfrac_P #1}%
820   \krof
821 }%
822 \def\XINT_tfrac_zero #1\Z { 0/1[0]}%
823 \def\XINT_tfrac_P #1/#2\Z {\expandafter\XINT_rez_AB
824   \romannumeral0\xintiirem{#1}{#2}\Z {0}{#2}}%

```

## 8.30 \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.

```

825 \def\xintTrunc {\romannumeral0\xinttrunc }%
826 \def\xintiTrunc {\romannumeral0\xintitrunc}%
827 \def\xinttrunc #1{\expandafter\XINT_trunc\the\numexpr#1.\XINT_trunc_G}%
828 \def\xintitrunc #1{\expandafter\XINT_trunc\the\numexpr#1.\XINT_itrunc_G}%

```

```

829 \def\XINT_trunc #1.#2#3%
830 {%
831   \expandafter\XINT_trunc_a\romannumeral0\XINT_infrac{#3}#1.#2%
832 }%
833 \def\XINT_trunc_a #1#2#3#4.#5%
834 {%
835   \if0\XINT_Sgn#2\xint:\xint_dothis\XINT_trunc_zero\fi
836   \if1\XINT_is_One#3XY\xint_dothis\XINT_trunc_sp_b\fi
837   \xint_orthat\XINT_trunc_b #1+#4.{#2}{#3}#5#4.%
838 }%
839 \def\XINT_trunc_zero #1.#2.{ 0}%

840 \def\XINT_trunc_b      {\expandafter\XINT_trunc_B\the\numexpr}%
841 \def\XINT_trunc_sp_b  {\expandafter\XINT_trunc_sp_B\the\numexpr}%

842 \def\XINT_trunc_B #1%
843 {%
844   \xint_UDsignfork
845   #1\XINT_trunc_C
846   -\XINT_trunc_D
847   \krof #1%
848 }%

849 \def\XINT_trunc_sp_B #1%
850 {%
851   \xint_UDsignfork
852   #1\XINT_trunc_sp_C
853   -\XINT_trunc_sp_D
854   \krof #1%
855 }%

856 \def\XINT_trunc_C -#1.#2#3%
857 {%
858   \expandafter\XINT_trunc_CE
859   \romannumeral0\XINT_dsx_addzeros{#1}#3;.{#2}%
860 }%
861 \def\XINT_trunc_CE #1.#2{\XINT_trunc_E #2.{#1}}%

862 \def\XINT_trunc_sp_C -#1.#2#3{\XINT_trunc_sp_Ca #2.#1.}%
863 \def\XINT_trunc_sp_Ca #1%
864 {%
865   \xint_UDsignfork
866   #1{\XINT_trunc_sp_Cb -}%
867   -{\XINT_trunc_sp_Cb \space#1}%
868   \krof
869 }%
870 \def\XINT_trunc_sp_Cb #1#2.#3.%
871 {%
872   \expandafter\XINT_trunc_sp_Cc

```



```

873 \romannumeral0\expandafter\XINT_split_fromright_a
874 \the\numexpr#3-\numexpr\XINT_length_loop
875 #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
876 \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
877 \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
878 .#2\xint_bye2345678\xint_bye..#1%
879 }%

880 \def\XINT_trunc_sp_Cc #1%
881 {%
882 \if.#1\xint_dothis{\XINT_trunc_sp_Cd 0.}\fi
883 \xint_orthat {\XINT_trunc_sp_Cd #1}%
884 }%
885 \def\XINT_trunc_sp_Cd #1.#2.#3%
886 {%
887 \XINT_trunc_sp_F #3#1.%
888 }%
889 \def\XINT_trunc_D #1.#2%
890 {%
891 \expandafter\XINT_trunc_E
892 \romannumeral0\XINT_dsx_addzeros {#1}#2;.%
893 }%
894 \def\XINT_trunc_sp_D #1.#2#3%
895 {%
896 \expandafter\XINT_trunc_sp_E
897 \romannumeral0\XINT_dsx_addzeros {#1}#2;.%
898 }%
899 \def\XINT_trunc_E #1%
900 {%
901 \xint_UDsignfork
902 #1{\XINT_trunc_F -}%
903 -{\XINT_trunc_F \space#1}%
904 \krof
905 }%
906 \def\XINT_trunc_sp_E #1%
907 {%
908 \xint_UDsignfork
909 #1{\XINT_trunc_sp_F -}%
910 -{\XINT_trunc_sp_F\space#1}%
911 \krof
912 }%
913 \def\XINT_trunc_F #1#2.#3#4%
914 {\expandafter#4\romannumeral`&&\expandafter\xint_firstoftwo
915 \romannumeral0\XINT_div_prepare {#3}{#2}.#1}%
916 \def\XINT_trunc_sp_F #1#2.#3{#3#2.#1}%
917 \def\XINT_itrunc_G #1#2.#3#4.{\if#10\xint_dothis{ 0}\fi\xint_orthat{#3#1}#2}%
918 \def\XINT_trunc_G #1.#2#3.%
919 {%
920 \expandafter\XINT_trunc_H
921 \the\numexpr\romannumeral0\xintlength {#1}-#3.#3.{#1}#2%
922 }%
923 \def\XINT_trunc_H #1.#2.%

```

```

924 {%
925   \ifnum #1 > \xint_c_
926     \xint_afterfi {\XINT_trunc_Ha {#2}}%
927   \else
928     \xint_afterfi {\XINT_trunc_Hb {-#1}}% -0,--1,--2, ....
929   \fi
930 }%
931 \def\XINT_trunc_Ha{\expandafter\XINT_trunc_Haa\romannumeral0\xintdecsplit}%
932 \def\XINT_trunc_Haa #1#2#3{#3#1.#2}%
933 \def\XINT_trunc_Hb #1#2#3%
934 {%
935   \expandafter #3\expandafter0\expandafter.%

936   \romannumeral\xintreplicate{#1}0#2%
937 }%

```

### 8.31 \xintTTrunc

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

```

938 \def\xintTTrunc {\romannumeral0\xintttrunc }%
939 \def\xintttrunc {\xintitrunc\xint_c_}%

```

### 8.32 \xintNum

```

940 \let\xintnum \xintttrunc

```

### 8.33 \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.

```

941 \def\xintRound {\romannumeral0\xintround }%
942 \def\xintiRound {\romannumeral0\xintiround }%
943 \def\xintround #1{\expandafter\XINT_round\the\numexpr #1.\XINT_round_A}%
944 \def\xintiround #1{\expandafter\XINT_round\the\numexpr #1.\XINT_iround_A}%
945 \def\XINT_round #1.{\expandafter\XINT_round_aa\the\numexpr #1+\xint_c_i.#1.}%
946 \def\XINT_round_aa #1.#2.#3#4%
947 {%
948   \expandafter\XINT_round_a\romannumeral0\XINT_infrac{#4}#1.#3#2.%
949 }%
950 \def\XINT_round_a #1#2#3#4.%
951 {%
952   \if0\XINT_Sgn#2\xint:\xint_dothis\XINT_trunc_zero\fi
953   \if1\XINT_is_One#3XY\xint_dothis\XINT_trunc_sp_b\fi
954   \xint_orthat\XINT_trunc_b #1+#4.{#2}{#3}%
955 }%
956 \def\XINT_round_A{\expandafter\XINT_trunc_G\romannumeral0\XINT_round_B}%
957 \def\XINT_iround_A{\expandafter\XINT_itrunc_G\romannumeral0\XINT_round_B}%

```

```
958 \def\XINT_round_B #1.%
959   {\XINT_dsrr #1\xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax.}%
```

### 8.34 \xintXTrunc

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

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

```
960 \def\xintXTrunc #1#2%
961 {%
962   \expandafter\XINT_xtrunc_a
963   \the\numexpr #1\expandafter.\romannumeral0\xintraw
964 }%
965 \def\XINT_xtrunc_a #1.% ?? faire autre chose
966 {%
967   \expandafter\XINT_xtrunc_b\the\numexpr\ifnum#1<\xint_c_i \xint_c_i-\fi #1.%
968 }%
```

```
969 \def\XINT_xtrunc_b #1.#2{\XINT_xtrunc_c #2{#1}}%
```

```
970 \def\XINT_xtrunc_c #1%
971 {%
972   \xint_UDzerominusfork
973     #1-\XINT_xtrunc_zero
974     0#1{-\XINT_xtrunc_d {}}%
975     0-{\XINT_xtrunc_d #1}%
976   \krof
977 }%[
978 \def\XINT_xtrunc_zero #1#2]{0.\romannumeral\xintreplicate{#1}0}%
```

```
979 \def\XINT_xtrunc_d #1#2#3/#4[#5]%
980 {%
981   \XINT_xtrunc_prepare_a#4\R\R\R\R\R\R\R {10}0000001\W
982   !{#4};{#5}{#2}{#1#3}%
983 }%
984 \def\XINT_xtrunc_prepare_a #1#2#3#4#5#6#7#8#9%
985 {%
986   \xint_gob_til_R #9\XINT_xtrunc_prepare_small\R
987   \XINT_xtrunc_prepare_b #9%
988 }%
989 \def\XINT_xtrunc_prepare_small\R #1!#2;%
990 {%
991   \ifcase #2
992   \or\expandafter\XINT_xtrunc_BisOne
993   \or\expandafter\XINT_xtrunc_BisTwo
994   \or
```

```

995 \or\expandafter\XINT_xtrunc_BisFour
996 \or\expandafter\XINT_xtrunc_BisFive
997 \or
998 \or
999 \or\expandafter\XINT_xtrunc_BisEight
1000 \fi\XINT_xtrunc_BisSmall {#2}%
1001 }%

1002 \def\XINT_xtrunc_BisOne\XINT_xtrunc_BisSmall #1#2#3#4%
1003 {\XINT_xtrunc_sp_e {#2}{#4}{#3}}%
1004 \def\XINT_xtrunc_BisTwo\XINT_xtrunc_BisSmall #1#2#3#4%
1005 {%
1006 \expandafter\XINT_xtrunc_sp_e\expandafter
1007 {\the\numexpr #2-\xint_c_i\expandafter}\expandafter
1008 {\romannumeral0\xintiimul 5{#4}{#3}}%
1009 }%
1010 \def\XINT_xtrunc_BisFour\XINT_xtrunc_BisSmall #1#2#3#4%
1011 {%
1012 \expandafter\XINT_xtrunc_sp_e\expandafter
1013 {\the\numexpr #2-\xint_c_ii\expandafter}\expandafter
1014 {\romannumeral0\xintiimul {25}{#4}{#3}}%
1015 }%
1016 \def\XINT_xtrunc_BisFive\XINT_xtrunc_BisSmall #1#2#3#4%
1017 {%
1018 \expandafter\XINT_xtrunc_sp_e\expandafter
1019 {\the\numexpr #2-\xint_c_i\expandafter}\expandafter
1020 {\romannumeral0\xintdouble {#4}{#3}}%
1021 }%
1022 \def\XINT_xtrunc_BisEight\XINT_xtrunc_BisSmall #1#2#3#4%
1023 {%
1024 \expandafter\XINT_xtrunc_sp_e\expandafter
1025 {\the\numexpr #2-\xint_c_iii\expandafter}\expandafter
1026 {\romannumeral0\xintiimul {125}{#4}{#3}}%
1027 }%
1028 \def\XINT_xtrunc_BisSmall #1%
1029 {%
1030 \expandafter\XINT_xtrunc_e\expandafter
1031 {\expandafter\XINT_xtrunc_small_a
1032 \the\numexpr #1/\xint_c_ii\expandafter
1033 .\the\numexpr \xint_c_x^viii+#1!}%
1034 }%

1035 \def\XINT_xtrunc_small_a #1.#2!#3%
1036 {%
1037 \expandafter\XINT_div_small_b\the\numexpr #1\expandafter
1038 \xint:\the\numexpr #2\expandafter!%
1039 \romannumeral0\XINT_div_small_ba #3\R\R\R\R\R\R\R\R\{10}0000001\W
1040 #3\XINT_sepbyviii_Z_end 2345678\relax
1041 }%

1042 \def\XINT_xtrunc_prepare_b

```

```

1043   {\expandafter\XINT_xtrunc_prepare_c\romannumeral0\XINT_zeroes_forviii }%
1044 \def\XINT_xtrunc_prepare_c #1!%
1045 {%
1046   \XINT_xtrunc_prepare_d #1.00000000!{#1}%
1047 }%
1048 \def\XINT_xtrunc_prepare_d #1#2#3#4#5#6#7#8#9%
1049 {%
1050   \expandafter\XINT_xtrunc_prepare_e
1051   \xint_gob_til_dot #1#2#3#4#5#6#7#8#9!%
1052 }%
1053 \def\XINT_xtrunc_prepare_e #1!#2!#3#4%
1054 {%
1055   \XINT_xtrunc_prepare_f #4#3\X {#1}{#3}%
1056 }%
1057 \def\XINT_xtrunc_prepare_f #1#2#3#4#5#6#7#8#9\X
1058 {%
1059   \expandafter\XINT_xtrunc_prepare_g\expandafter
1060   \XINT_div_prepare_g
1061   \the\numexpr #1#2#3#4#5#6#7#8+\xint_c_i\expandafter
1062   \R\xint:\the\numexpr (#1#2#3#4#5#6#7#8+\xint_c_i)/\xint_c_ii\expandafter
1063   \xint:\the\numexpr #1#2#3#4#5#6#7#8\expandafter
1064   \xint:\romannumeral0\XINT_sepandrev_andcount
1065   #1#2#3#4#5#6#7#8#9\XINT_rsepyviii_end_A 2345678%
1066   \XINT_rsepyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
1067   \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
1068   \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
1069   \X
1070 }%

1071 \def\XINT_xtrunc_prepare_g #1;{\XINT_xtrunc_e {#1}}%

1072 \def\XINT_xtrunc_e #1#2%
1073 {%
1074   \ifnum #2<\xint_c_
1075     \expandafter\XINT_xtrunc_I
1076   \else
1077     \expandafter\XINT_xtrunc_II
1078   \fi #2\xint:{#1}%
1079 }%

1080 \def\XINT_xtrunc_I -#1\xint:#2#3#4%
1081 {%
1082   \expandafter\XINT_xtrunc_I_a\romannumeral0#2{#4}{#2}{#1}{#3}%
1083 }%

1084 \def\XINT_xtrunc_I_a #1#2#3#4#5%
1085 {%
1086   \expandafter\XINT_xtrunc_I_b\the\numexpr #4-#5\xint:#4\xint:{#5}{#2}{#3}{#1}%
1087 }%

```

```

1088 \def\XINT_xtrunc_I_b #1%
1089 {%
1090   \xint_UDsignfork
1091   #1\XINT_xtrunc_IA_c
1092   -\XINT_xtrunc_IB_c
1093   \krof #1%
1094 }%

1095 \def\XINT_xtrunc_IA_c -#1\xint:#2\xint:#3#4#5#6%
1096 {%
1097   \expandafter\XINT_xtrunc_IA_d
1098   \the\numexpr#2-\xintLength{#6}\xint:{#6}%
1099   \expandafter\XINT_xtrunc_IA_xd
1100   \the\numexpr (#1+\xint_c_ii^v)/\xint_c_ii^vi-\xint_c_i\xint:#1\xint:{#5}{#4}%
1101 }%

1102 \def\XINT_xtrunc_IA_d #1%
1103 {%
1104   \xint_UDsignfork
1105   #1\XINT_xtrunc_IAA_e
1106   -\XINT_xtrunc_IAB_e
1107   \krof #1%
1108 }%

1109 \def\XINT_xtrunc_IAA_e -#1\xint:#2%
1110 {%
1111   \romannumeral0\XINT_split_fromleft
1112   #1.#2\xint_gobble_i\xint_bye2345678\xint_bye..%
1113 }%

1114 \def\XINT_xtrunc_IAB_e #1\xint:#2%
1115 {%
1116   0.\romannumeral\XINT_rep#1\endcsname0#2%
1117 }%

1118 \def\XINT_xtrunc_IA_xd #1\xint:#2\xint:%
1119 {%
1120   \expandafter\XINT_xtrunc_IA_xe\the\numexpr #2-\xint_c_ii^vi*#1\xint:#1\xint:%
1121 }%

1122 \def\XINT_xtrunc_IA_xe #1\xint:#2\xint:#3#4%
1123 {%
1124   \XINT_xtrunc_loop {#2}{#4}{#3}{#1}%
1125 }%

1126 \def\XINT_xtrunc_IB_c #1\xint:#2\xint:#3#4#5#6%
1127 {%
1128   \expandafter\XINT_xtrunc_IB_d
1129   \romannumeral0\XINT_split_xfork #1.#6\xint_bye2345678\xint_bye..{#3}%
1130 }%

```



```

1178 \expandafter\XINT_xtrunc_finish_a\romannumeral0#2{#1}%
1179 }%
1180 \def\XINT_xtrunc_finish_a #1#2#3%
1181 {%
1182 \romannumeral\xintreplicate{#3-\xintLength{#1}}0#1%
1183 }%

1184 \def\XINT_xtrunc_sp_e #1%
1185 {%
1186 \ifnum #1<\xint_c_
1187 \expandafter\XINT_xtrunc_sp_I
1188 \else
1189 \expandafter\XINT_xtrunc_sp_II
1190 \fi #1\xint:%
1191 }%

1192 \def\XINT_xtrunc_sp_I -#1\xint:#2#3%
1193 {%
1194 \expandafter\XINT_xtrunc_sp_I_a\the\numexpr #1-#3\xint:#1\xint:{#3}{#2}%
1195 }%

1196 \def\XINT_xtrunc_sp_I_a #1%
1197 {%
1198 \xint_UDsignfork
1199 #1\XINT_xtrunc_sp_IA_b
1200 -\XINT_xtrunc_sp_IB_b
1201 \krof #1%
1202 }%

1203 \def\XINT_xtrunc_sp_IA_b -#1\xint:#2\xint:#3#4%
1204 {%
1205 \expandafter\XINT_xtrunc_sp_IA_c
1206 \the\numexpr#2-\xintLength{#4}\xint:{#4}\romannumeral\XINT_rep#1\endcsname0%
1207 }%

1208 \def\XINT_xtrunc_sp_IA_c #1%
1209 {%
1210 \xint_UDsignfork
1211 #1\XINT_xtrunc_sp_IAA
1212 -\XINT_xtrunc_sp_IAB
1213 \krof #1%
1214 }%

1215 \def\XINT_xtrunc_sp_IAA -#1\xint:#2%
1216 {%
1217 \romannumeral0\XINT_split_fromleft
1218 #1.#2\xint_gobble_i\xint_bye2345678\xint_bye..%
1219 }%

```



```

1220 \def\XINT_xtrunc_sp_IAB #1\xint:#2%
1221 {%
1222     0.\romannumeral\XINT_rep#1\endcsname0#2%
1223 }%

1224 \def\XINT_xtrunc_sp_IB_b #1\xint:#2\xint:#3#4%
1225 {%
1226     \expandafter\XINT_xtrunc_sp_IB_c
1227     \romannumeral0\XINT_split_xfork #1.#4\xint_bye2345678\xint_bye..{#3}%
1228 }%

1229 \def\XINT_xtrunc_sp_IB_c #1.#2.#3%
1230 {%
1231     \expandafter\XINT_xtrunc_sp_IA_c\the\numexpr#3-\xintLength {#1}\xint:{#1}%
1232 }%

1233 \def\XINT_xtrunc_sp_II #1\xint:#2#3%
1234 {%
1235     #2\romannumeral\XINT_rep#1\endcsname0.\romannumeral\XINT_rep#3\endcsname0%
1236 }%

```

### 8.35 \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.

```

1237 \mathchardef\XINTdigits 16
1238 \def\xintDigits #1#2%
1239     {\afterassignment \xint_gobble_i \mathchardef\XINTdigits=}
1240 \def\xinttheDigits {\number\XINTdigits }%

```

### 8.36 \xintAdd

Big change at 1.3:  $a/b+c/d$  uses  $\text{lcm}(b,d)$  as denominator.

```

1241 \def\xintAdd {\romannumeral0\xintadd }%
1242 \def\xintadd #1{\expandafter\XINT_fadd\romannumeral0\xintraw {#1}}%
1243 \def\XINT_fadd #1{\xint_gob_til_zero #1\XINT_fadd_Azero 0\XINT_fadd_a #1}%
1244 \def\XINT_fadd_Azero #1{\xintraw }%
1245 \def\XINT_fadd_a #1/#2[#3]#4%
1246     {\expandafter\XINT_fadd_b\romannumeral0\xintraw {#4}{#3}{#1}{#2}}%
1247 \def\XINT_fadd_b #1{\xint_gob_til_zero #1\XINT_fadd_Bzero 0\XINT_fadd_c #1}%
1248 \def\XINT_fadd_Bzero #1]#2#3#4{ #3/#4[#2]}%
1249 \def\XINT_fadd_c #1/#2[#3]#4%
1250 {%
1251     \expandafter\XINT_fadd_Aa\the\numexpr #4-#3.{#3}{#4}{#1}{#2}%
1252 }%
1253 \def\XINT_fadd_Aa #1%
1254 {%
1255     \xint_UDzerominusfork
1256     #1-\XINT_fadd_B

```

```

1257      0#1\XINT_fadd_Bb
1258      0-\XINT_fadd_Ba
1259      \krof #1%
1260 }%
1261 \def\XINT_fadd_B #1.#2#3#4#5#6#7{\XINT_fadd_C {#4}{#5}{#7}{#6}[#3]}%
1262 \def\XINT_fadd_Ba #1.#2#3#4#5#6#7%
1263 {%
1264      \expandafter\XINT_fadd_C\expandafter
1265      {\romannumeral0\XINT_dsx_addzeros {#1}#6;}%
1266      {#7}{#5}{#4}[#2]}%
1267 }%
1268 \def\XINT_fadd_Bb -#1.#2#3#4#5#6#7%
1269 {%
1270      \expandafter\XINT_fadd_C\expandafter
1271      {\romannumeral0\XINT_dsx_addzeros {#1}#4;}%
1272      {#5}{#7}{#6}[#3]}%
1273 }%
1274 \def\XINT_fadd_iszero #1[#2]{ 0/1[0]}% ou [#2] origine1?
1275 \def\XINT_fadd_C #1#2#3%
1276 {%
1277      \expandafter\XINT_fadd_D_b
1278      \romannumeral0\XINT_div_prepare{#2}{#3}{#2}{#2}{#3}{#1}%
1279 }%

```

Basically a clone of the `\XINT_irr_loop_a` loop. I should modify the output of `\XINT_div_prepare` perhaps to be optimized for checking if remainder vanishes.

```

1280 \def\XINT_fadd_D_a #1#2%
1281 {%
1282      \expandafter\XINT_fadd_D_b
1283      \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
1284 }%
1285 \def\XINT_fadd_D_b #1#2{\XINT_fadd_D_c #2\Z}%
1286 \def\XINT_fadd_D_c #1#2\Z
1287 {%
1288      \xint_gob_til_zero #1\XINT_fadd_D_exit0\XINT_fadd_D_a {#1#2}%
1289 }%
1290 \def\XINT_fadd_D_exit0\XINT_fadd_D_a #1#2#3%
1291 {%
1292      \expandafter\XINT_fadd_E
1293      \romannumeral0\xintiiquo {#3}{#2}.{#2}%
1294 }%
1295 \def\XINT_fadd_E #1.#2#3%
1296 {%
1297      \expandafter\XINT_fadd_F
1298      \romannumeral0\xintiimul{#1}{#3}.{\xintiiquo{#3}{#2}}{#1}%
1299 }%
1300 \def\XINT_fadd_F #1.#2#3#4#5%
1301 {%
1302      \expandafter\XINT_fadd_G
1303      \romannumeral0\xintiiaadd{\xintiimul{#2}{#4}}{\xintiimul{#3}{#5}}/#1%
1304 }%
1305 \def\XINT_fadd_G #1{%

```

```
1306 \def\XINT_fadd_G ##1{\if0##1\expandafter\XINT_fadd_iszero\fi#1##1}%
1307 }\XINT_fadd_G{ }%
```

### 8.37 \xintSub

Since 1.3 will use least common multiple of denominators.

```
1308 \def\xintSub {\romannumeral0\xintsub }%
1309 \def\xintsub #1{\expandafter\XINT_fsub\romannumeral0\xintra #1}%
1310 \def\XINT_fsub #1{\xint_gob_til_zero #1\XINT_fsub_Azero 0\XINT_fsub_a #1}%
1311 \def\XINT_fsub_Azero #1{\xintopp }%
1312 \def\XINT_fsub_a #1/#2[#3]#4%
1313   {\expandafter\XINT_fsub_b\romannumeral0\xintra #4}{#3}{#1}{#2}%
1314 \def\XINT_fsub_b #1{\xint_UDzerominusfork
1315   #1-\XINT_fadd_Bzero
1316   0#1\XINT_fadd_c
1317   0-\XINT_fadd_c -#1}%
1318 \krof }%
```

### 8.38 \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.

```
1319 \def\xintSum {\romannumeral0\xintsum }%
1320 \def\xintsum #1{\expandafter\XINT_fsumexpr\romannumeral`&&#1\xint:}%
1321 \def\XINT_fsumexpr {\XINT_fsum_loop_a {0/1[0]}}%
1322 \def\XINT_fsum_loop_a #1#2%
1323 {%
1324   \expandafter\XINT_fsum_loop_b \romannumeral`&&#2\xint:#{#1}%
1325 }%
1326 \def\XINT_fsum_loop_b #1%
1327 {%
1328   \xint_gob_til_xint: #1\XINT_fsum_finished\xint:\XINT_fsum_loop_c #1%
1329 }%
1330 \def\XINT_fsum_loop_c #1\xint:#2%
1331 {%
1332   \expandafter\XINT_fsum_loop_a\expandafter{\romannumeral0\xintadd {#2}{#1}}%
1333 }%
1334 \def\XINT_fsum_finished #1\xint:\xint:#2{ #2}%
```

### 8.39 \xintMul

```
1335 \def\xintMul {\romannumeral0\xintmul }%
1336 \def\xintmul #1{\expandafter\XINT_fmula\romannumeral0\xintra #1}.}%
1337 \def\XINT_fmula #1{\xint_gob_til_zero #1\XINT_fmula_zero 0\XINT_fmula_a #1}%
1338 \def\XINT_fmula_a #1[#2].#3%
1339   {\expandafter\XINT_fmula_b\romannumeral0\xintra #3}{#1}{#2}.}%
1340 \def\XINT_fmula_b #1{\xint_gob_til_zero #1\XINT_fmula_zero 0\XINT_fmula_c #1}%
1341 \def\XINT_fmula_c #1/#2[#3]#4/#5[#6].}%
1342 {%
1343   \expandafter\XINT_fmula_d
```

```

1344 \expandafter{\the\numexpr #3+#6\expandafter}%
1345 \expandafter{\romannumeral0\xintiimul {#5}{#2}}%
1346 {\romannumeral0\xintiimul {#4}{#1}}%
1347 }%
1348 \def\XINT_fmud #1#2#3%
1349 {%
1350 \expandafter \XINT_fmud_e \expandafter{#3}{#1}{#2}%
1351 }%
1352 \def\XINT_fmud_e #1#2{\XINT_outfrac {#2}{#1}}%
1353 \def\XINT_fmud_zero #1.#2{ 0/1[0]}%

```

## 8.40 \xintSqr

### 1.1 modifs comme xintMul.

```

1354 \def\xintSqr {\romannumeral0\xintsqr }%
1355 \def\xintsqr #1{\expandafter\XINT_fsqr\romannumeral0\xintraw {#1}}%
1356 \def\XINT_fsqr #1{\xint_gob_til_zero #1\XINT_fsqr_zero 0\XINT_fsqr_a #1}%
1357 \def\XINT_fsqr_a #1/#2[#3]%
1358 {%
1359 \expandafter\XINT_fsqr_b
1360 \expandafter{\the\numexpr #3+#3\expandafter}%
1361 \expandafter{\romannumeral0\xintiisqr {#2}}%
1362 {\romannumeral0\xintiisqr {#1}}%
1363 }%
1364 \def\XINT_fsqr_b #1#2#3{\expandafter \XINT_fmud_e \expandafter{#3}{#1}{#2}}%
1365 \def\XINT_fsqr_zero #1{ 0/1[0]}%

```

## 8.41 \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.

```

1366 \def\xintipow #1#2%
1367 {%
1368 \expandafter\xint_pow\the\numexpr \xintNum{#2}\expandafter
1369 .\romannumeral0\xintnum{#1}\xint:
1370 }%
1371 \def\xintPow {\romannumeral0\xintpow }%
1372 \def\xintpow #1%
1373 {%
1374 \expandafter\XINT_fpow\expandafter {\romannumeral0\XINT_infrac {#1}}%
1375 }%
1376 \def\XINT_fpow #1#2%
1377 {%
1378 \expandafter\XINT_fpow_fork\the\numexpr \xintNum{#2}\relax\Z #1%
1379 }%
1380 \def\XINT_fpow_fork #1#2\Z
1381 {%
1382 \xint_UDzerominusfork
1383 #1-\XINT_fpow_zero

```

```

1384      0#1\XINT_fpow_neg
1385      0-{\XINT_fpow_pos #1}%
1386      \krof
1387      {#2}%
1388 }%
1389 \def\XINT_fpow_zero #1#2#3#4{ 1/1[0]}%
1390 \def\XINT_fpow_pos #1#2#3#4#5%
1391 {%
1392      \expandafter\XINT_fpow_pos_A\expandafter
1393      {\the\numexpr #1#2*#3\expandafter}\expandafter
1394      {\romannumeral0\xintiipow {#5}{#1#2}}%
1395      {\romannumeral0\xintiipow {#4}{#1#2}}%
1396 }%
1397 \def\XINT_fpow_neg #1#2#3#4%
1398 {%
1399      \expandafter\XINT_fpow_pos_A\expandafter
1400      {\the\numexpr -#1*#2\expandafter}\expandafter
1401      {\romannumeral0\xintiipow {#3}{#1}}%
1402      {\romannumeral0\xintiipow {#4}{#1}}%
1403 }%
1404 \def\XINT_fpow_pos_A #1#2#3%
1405 {%
1406      \expandafter\XINT_fpow_pos_B\expandafter {#3}{#1}{#2}%
1407 }%
1408 \def\XINT_fpow_pos_B #1#2{\XINT_outfrac {#2}{#1}}%

```

## 8.42 \xintFac

Factorial coefficients: variant which can be chained with other `xintfrac` macros. `\xintiFac` deprecated at 1.2o and removed at 1.3; `\xintFac` used by `xintexpr.sty`.

```

1409 \def\xintFac {\romannumeral0\xintfac}%
1410 \def\xintfac #1{\expandafter\XINT_fac_fork\the\numexpr\xintNum{#1}.[0]}%

```

## 8.43 \xintBinomial

1.2f. Binomial coefficients. `\xintiBinomial` deprecated at 1.2o and removed at 1.3; `\xintBinomial` needed by `xintexpr.sty`.

```

1411 \def\xintBinomial {\romannumeral0\xintbinomial}%
1412 \def\xintbinomial #1#2%
1413 {%
1414      \expandafter\XINT_binom_pre
1415      \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.[0]}%
1416 }%

```

## 8.44 \xintPFactorial

1.2f. Partial factorial. For needs of `xintexpr.sty`.

```

1417 \def\xintipfactorial #1#2%
1418 {%
1419      \expandafter\XINT_pfac_fork

```

```

1420 \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.%
1421 }%
1422 \def\xintPFactorial {\romannumeral0\xintpfactorial}%
1423 \def\xintpfactorial #1#2%
1424 {%
1425 \expandafter\XINT_pfac_fork
1426 \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.[0]%
1427 }%

```

## 8.45 \xintPrd

There was (not documented anymore since 1.09d, 2013/10/22) a macro `\xintPrdExpr`, but it has been deleted at 1.21

```

1428 \def\xintPrd {\romannumeral0\xintprd }%
1429 \def\xintprd #1{\expandafter\XINT_fprdexpr \romannumeral`&&@#1\xint:}%
1430 \def\XINT_fprdexpr {\XINT_fprod_loop_a {1/1[0]}}%
1431 \def\XINT_fprod_loop_a #1#2%
1432 {%
1433 \expandafter\XINT_fprod_loop_b \romannumeral`&&@#2\xint:{#1}%
1434 }%
1435 \def\XINT_fprod_loop_b #1%
1436 {%
1437 \xint_gob_til_xint: #1\XINT_fprod_finished\xint:\XINT_fprod_loop_c #1%
1438 }%
1439 \def\XINT_fprod_loop_c #1\xint:#2%
1440 {%
1441 \expandafter\XINT_fprod_loop_a\expandafter{\romannumeral0\xintmul {#1}{#2}}%
1442 }%
1443 \def\XINT_fprod_finished#1\xint:\xint:#2{ #2}%

```

## 8.46 \xintDiv

```

1444 \def\xintDiv {\romannumeral0\xintdiv }%
1445 \def\xintdiv #1%
1446 {%
1447 \expandafter\XINT_fdiv\expandafter {\romannumeral0\XINT_infrac {#1}}%
1448 }%
1449 \def\XINT_fdiv #1#2%
1450 {\expandafter\XINT_fdiv_A\romannumeral0\XINT_infrac {#2}#1}%
1451 \def\XINT_fdiv_A #1#2#3#4#5#6%
1452 {%
1453 \expandafter\XINT_fdiv_B
1454 \expandafter{\the\numexpr #4-#1\expandafter}%
1455 \expandafter{\romannumeral0\xintiimul {#2}{#6}}%
1456 {\romannumeral0\xintiimul {#3}{#5}}%
1457 }%
1458 \def\XINT_fdiv_B #1#2#3%
1459 {%
1460 \expandafter\XINT_fdiv_C
1461 \expandafter{#3}{#1}{#2}%
1462 }%
1463 \def\XINT_fdiv_C #1#2{\XINT_outfrac {#2}{#1}}%

```

## 8.47 `\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`.

```
1464 \def\xintDivFloor      {\romannumeral0\xintdivfloor }%
1465 \def\xintdivfloor #1#2{\xintifloor{\xintDiv {#1}{#2}}}%
```

## 8.48 `\xintDivTrunc`

1.1. `\xintttrunc` rather than `\xintitrunc0` in 1.1a

```
1466 \def\xintDivTrunc      {\romannumeral0\xintdivtrunc }%
1467 \def\xintdivtrunc #1#2{\xintttrunc {\xintDiv {#1}{#2}}}%
```

## 8.49 `\xintDivRound`

1.1

```
1468 \def\xintDivRound      {\romannumeral0\xintdivround }%
1469 \def\xintdivround #1#2{\xintiround 0{\xintDiv {#1}{#2}}}%
```

## 8.50 `\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`.

At 1.3, uses least common multiple denominator, like `\xintMod` (next).

```
1470 \def\xintModTrunc {\romannumeral0\xintmodtrunc }%
1471 \def\xintmodtrunc #1{\expandafter\XINT_modtrunc_a\romannumeral0\xintra{#1}.}%
1472 \def\XINT_modtrunc_a #1#2.#3%
1473   {\expandafter\XINT_modtrunc_b\expandafter #1\romannumeral0\xintra{#3}#2.}%
1474 \def\XINT_modtrunc_b #1#2% #1 de A, #2 de B.
1475 {%
1476   \if0#2\xint_dothis{\XINT_modtrunc_divbyzero #1#2}\fi
1477   \if0#1\xint_dothis\XINT_modtrunc_aiszero\fi
1478   \if-#2\xint_dothis{\XINT_modtrunc_bneg #1}\fi
1479   \xint_orthat{\XINT_modtrunc_bpos #1#2}%
1480 }%
1481 \def\XINT_modtrunc_divbyzero #1#2[#3]#4.%
1482 {%
1483   \XINT_signalcondition{DivisionByZero}{Division by #2[#3] of #1#4}{0/1[0]}%
1484 }%
1485 \def\XINT_modtrunc_aiszero #1.{ 0/1[0]}%
1486 \def\XINT_modtrunc_bneg #1%
1487 {%
1488   \xint_UDsignfork
1489     #1{\xintiiopp\XINT_modtrunc_pos {}}%
1490     -{\XINT_modtrunc_pos #1}%
1491   \krof
1492 }%
1493 \def\XINT_modtrunc_bpos #1%
```

```

1494 {%
1495   \xint_UDsignfork
1496     #1{\xintiiopp\XINT_modtrunc_pos {}}%
1497     -{\XINT_modtrunc_pos #1}%
1498   \krof
1499 }%

```

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 >= 0).

```

1500 \def\XINT_modtrunc_pos #1#2/#3[#4]#5/#6[#7].%
1501 {%
1502   \expandafter\XINT_modtrunc_pos_a
1503   \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.%
1504   \romannumeral0\expandafter\XINT_mod_D_b
1505   \romannumeral0\XINT_div_prepare{#3}{#6}{#3}{#3}{#6}%
1506   {#1#5}{#7-#4}{#2}{#4-#7}%
1507 }%
1508 \def\XINT_modtrunc_pos_a #1.#2#3#4{\xintiirem {#3}{#4}/#2[#1]}%

```

## 8.51 \xintDivMod

1.2p. `\xintDivMod{q1}{q2}` outputs  $\{\text{floor}(q1/q2)\}\{q1 - q2*\text{floor}(q1/q2)\}$ . Attention that it relies on `\xintiiE{x}{e}` returning x if e < 0.

Modified (like `\xintAdd` and `\xintSub`) at 1.3 to use a l.c.m for final denominator of the "mod" part.

```

1509 \def\xintDivMod {\romannumeral0\xintdivmod }%
1510 \def\xintdivmod #1{\expandafter\XINT_divmod_a\romannumeral0\xintra{#1}.}%
1511 \def\XINT_divmod_a #1#2.#3%
1512   {\expandafter\XINT_divmod_b\expandafter #1\romannumeral0\xintra{#3}#2.}%
1513 \def\XINT_divmod_b #1#2% #1 de A, #2 de B.
1514 {%
1515   \if0#2\xint_dothis{\XINT_divmod_divbyzero #1#2}\fi
1516   \if0#1\xint_dothis\XINT_divmod_aiszero\fi
1517   \if-#2\xint_dothis{\XINT_divmod_bneg #1}\fi
1518   \xint_orthat{\XINT_divmod_bpos #1#2}%
1519 }%
1520 \def\XINT_divmod_divbyzero #1#2[#3]#4.%
1521 {%
1522   \XINT_signalcondition{DivisionByZero}{Division by #2[#3] of #1#4}{}%
1523   {{0}{0/1[0]}}% à revoir...
1524 }%
1525 \def\XINT_divmod_aiszero #1.{{0}{0/1[0]}}%
1526 \def\XINT_divmod_bneg #1% f // -g = (-f) // g, f % -g = -((-f) % g)
1527 {%
1528   \expandafter\XINT_divmod_bneg_finish
1529   \romannumeral0\xint_UDsignfork
1530     #1{\XINT_divmod_bpos {}}%
1531     -{\XINT_divmod_bpos {-#1}}%
1532   \krof
1533 }%
1534 \def\XINT_divmod_bneg_finish#1#2%

```



```

1535 {%
1536   \expandafter\xint_exchangetwo_keepbraces\expandafter
1537   {\romannumeral0\xintiioopp#2}{#1}%
1538 }%
1539 \def\xINT_divmod_bpos #1#2/#3[#4]#5/#6[#7].%
1540 {%
1541   \expandafter\xINT_divmod_bpos_a
1542   \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.%
1543   \romannumeral0\expandafter\xINT_mod_D_b
1544   \romannumeral0\xINT_div_prepare{#3}{#6}{#3}{#3}{#6}%
1545   {#1#5}{#7-#4}{#2}{#4-#7}%
1546 }%
1547 \def\xINT_divmod_bpos_a #1.#2#3#4%
1548 {%
1549   \expandafter\xINT_divmod_bpos_finish
1550   \romannumeral0\xintiidivision{#3}{#4}{/#2[#1]}%
1551 }%
1552 \def\xINT_divmod_bpos_finish #1#2#3{#1}{#2#3}%

```

## 8.52 \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`.

Modified (like `\xintAdd` and `\xintSub`) at 1.3 to use a l.c.m for final denominator.

```

1553 \def\xintMod {\romannumeral0\xintmod }%
1554 \def\xintmod #1{\expandafter\xINT_mod_a\romannumeral0\xintra{#1}.}%
1555 \def\xINT_mod_a #1#2.#3%
1556   {\expandafter\xINT_mod_b\expandafter #1\romannumeral0\xintra{#3}#2.}%
1557 \def\xINT_mod_b #1#2% #1 de A, #2 de B.
1558 {%
1559   \if0#2\xint_dothis{\XINT_mod_divbyzero #1#2}\fi
1560   \if0#1\xint_dothis\xINT_mod_aiszero\fi
1561   \if-#2\xint_dothis{\XINT_mod_bneg #1}\fi
1562   \xint_orthat{\XINT_mod_bpos #1#2}%
1563 }%

```

Attention to not move `ModTrunc` code beyond that point.

```

1564 \let\xINT_mod_divbyzero\xINT_modtrunc_divbyzero
1565 \let\xINT_mod_aiszero \XINT_modtrunc_aiszero
1566 \def\xINT_mod_bneg #1% f % -g = -((-f) % g), for g > 0
1567 {%
1568   \xintiioopp\xint_UDsignfork
1569   #1{\XINT_mod_bpos {}}%
1570   -{\XINT_mod_bpos {-#1}}%
1571   \krof
1572 }%
1573 \def\xINT_mod_bpos #1#2/#3[#4]#5/#6[#7].%
1574 {%
1575   \expandafter\xINT_mod_bpos_a
1576   \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.%

```

```

1577 \romannumeral0\expandafter\XINT_mod_D_b
1578 \romannumeral0\XINT_div_prepare{#3}{#6}{#3}{#3}{#6}%
1579 {#1#5}{#7-#4}{#2}{#4-#7}%
1580 }%
1581 \def\XINT_mod_D_a #1#2%
1582 {%
1583 \expandafter\XINT_mod_D_b
1584 \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
1585 }%
1586 \def\XINT_mod_D_b #1#2{\XINT_mod_D_c #2\Z}%
1587 \def\XINT_mod_D_c #1#2\Z
1588 {%
1589 \xint_gob_til_zero #1\XINT_mod_D_exit0\XINT_mod_D_a {#1#2}%
1590 }%
1591 \def\XINT_mod_D_exit0\XINT_mod_D_a #1#2#3%
1592 {%
1593 \expandafter\XINT_mod_E
1594 \romannumeral0\xintiipro {#3}{#2}.{#2}%
1595 }%
1596 \def\XINT_mod_E #1.#2#3%
1597 {%
1598 \expandafter\XINT_mod_F
1599 \romannumeral0\xintiimul{#1}{#3}.{\xintiiQuo{#3}{#2}}{#1}%
1600 }%
1601 \def\XINT_mod_F #1.#2#3#4#5#6#7%
1602 {%
1603 {#1}{\xintiiE{\xintiiMul{#4}{#3}}{#5}}%
1604 {\xintiiE{\xintiiMul{#6}{#2}}{#7}}%
1605 }%
1606 \def\XINT_mod_bpos_a #1.#2#3#4{\xintiiirem {#3}{#4}/#2[#1]}%

```

### 8.53 \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.

```

1607 \def\xintIsOne {\romannumeral0\xintisone }%
1608 \def\xintisone #1{\expandafter\XINT_fracisone
1609 \romannumeral0\xintrawwithzeros{#1}\Z }%
1610 \def\XINT_fracisone #1/#2\Z
1611 {\if0\xintiiCmp {#1}{#2}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%

```

### 8.54 \xintGeq

```

1612 \def\xintGeq {\romannumeral0\xintgeq }%
1613 \def\xintgeq #1%
1614 {%
1615 \expandafter\XINT_fgeq\expandafter {\romannumeral0\xintabs {#1}}%
1616 }%
1617 \def\XINT_fgeq #1#2%
1618 {%
1619 \expandafter\XINT_fgeq_A \romannumeral0\xintabs {#2}#1%
1620 }%

```

```

1621 \def\XINT_fgeq_A #1%
1622 {%
1623   \xint_gob_til_zero #1\XINT_fgeq_Zii 0%
1624   \XINT_fgeq_B #1%
1625 }%
1626 \def\XINT_fgeq_Zii 0\XINT_fgeq_B #1[#2]#3[#4]{ 1}%
1627 \def\XINT_fgeq_B #1/#2[#3]#4#5/#6[#7]%
1628 {%
1629   \xint_gob_til_zero #4\XINT_fgeq_Zi 0%
1630   \expandafter\XINT_fgeq_C\expandafter
1631   {\the\numexpr #7-#3\expandafter}\expandafter
1632   {\romannumeral0\xintiimul {#4#5}{#2}}%
1633   {\romannumeral0\xintiimul {#6}{#1}}%
1634 }%
1635 \def\XINT_fgeq_Zi 0#1#2#3#4#5#6#7{ 0}%
1636 \def\XINT_fgeq_C #1#2#3%
1637 {%
1638   \expandafter\XINT_fgeq_D\expandafter
1639   {#3}{#1}{#2}%
1640 }%
1641 \def\XINT_fgeq_D #1#2#3%
1642 {%
1643   \expandafter\XINT_cntSgnFork\romannumeral`&&\expandafter\XINT_cntSgn
1644   \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\xint:
1645   { 0}{\XINT_fgeq_E #2\Z {#3}{#1}}{ 1}%
1646 }%
1647 \def\XINT_fgeq_E #1%
1648 {%
1649   \xint_UDsignfork
1650     #1\XINT_fgeq_Fd
1651     -{\XINT_fgeq_Fn #1}%
1652   \krof
1653 }%

1654 \def\XINT_fgeq_Fd #1\Z #2#3%
1655 {%
1656   \expandafter\XINT_fgeq_Fe
1657   \romannumeral0\XINT_dsx_addzeros {#1}#3;\xint:#2\xint:
1658 }%
1659 \def\XINT_fgeq_Fe #1\xint:#2#3\xint:{\XINT_geq_plusplus #2#1\xint:#3\xint:}%
1660 \def\XINT_fgeq_Fn #1\Z #2#3%
1661 {%
1662   \expandafter\XINT_fgeq_Fo
1663   \romannumeral0\XINT_dsx_addzeros {#1}#2;\xint:#3\xint:
1664 }%
1665 \def\XINT_fgeq_Fo #1#2\xint:#3\xint:{\XINT_geq_plusplus #1#3\xint:#2\xint:}%

8.55 \xintMax

1666 \def\xintMax {\romannumeral0\xintmax }%
1667 \def\xintmax #1%
1668 {%
1669   \expandafter\XINT_fmax\expandafter {\romannumeral0\xintraw {#1}}%
1670 }%

```

```

1671 \def\XINT_fmax #1#2%
1672 {%
1673   \expandafter\XINT_fmax_A\romannumeral0\xintra #2}#1%
1674 }%
1675 \def\XINT_fmax_A #1#2/#3[#4]#5#6/#7[#8]%
1676 {%
1677   \xint_UDsignsfork
1678     #1#5\XINT_fmax_minusminus
1679     -#5\XINT_fmax_firstneg
1680     #1-\XINT_fmax_secondneg
1681     --\XINT_fmax_nonneg_a
1682   \krof
1683   #1#5{#2/#3[#4]}{#6/#7[#8]}%
1684 }%
1685 \def\XINT_fmax_minusminus --%
1686   {\expandafter-\romannumeral0\XINT_fmin_nonneg_b }%
1687 \def\XINT_fmax_firstneg #1-#2#3{ #1#2}%
1688 \def\XINT_fmax_secondneg -#1#2#3{ #1#3}%
1689 \def\XINT_fmax_nonneg_a #1#2#3#4%
1690 {%
1691   \XINT_fmax_nonneg_b {#1#3}{#2#4}%
1692 }%
1693 \def\XINT_fmax_nonneg_b #1#2%
1694 {%
1695   \if0\romannumeral0\XINT_fgeq_A #1#2%
1696     \xint_afterfi{ #1}%
1697   \else \xint_afterfi{ #2}%
1698   \fi
1699 }%

```

## 8.56 \xintMaxof

1.21 protects \xintMaxof against items with non terminated \the\numexpr expressions.  
The macro is not compatible with an empty list.

```

1700 \def\xintMaxof      {\romannumeral0\xintmaxof }%
1701 \def\xintmaxof      #1{\expandafter\XINT_maxof_a\romannumeral`&&@#1\xint:}%
1702 \def\XINT_maxof_a #1{\expandafter\XINT_maxof_b\romannumeral0\xintra{#1}!}%
1703 \def\XINT_maxof_b #1!#2%
1704   {\expandafter\XINT_maxof_c\romannumeral`&&@#2!{#1}!}%
1705 \def\XINT_maxof_c #1%
1706   {\xint_gob_til_xint: #1\XINT_maxof_e\xint:\XINT_maxof_d #1}%
1707 \def\XINT_maxof_d #1!%
1708   {\expandafter\XINT_maxof_b\romannumeral0\xintmax {#1}}%
1709 \def\XINT_maxof_e #1!#2!{ #2}%

```

## 8.57 \xintMin

```

1710 \def\xintMin {\romannumeral0\xintmin }%
1711 \def\xintmin #1%
1712 {%
1713   \expandafter\XINT_fmin\expandafter {\romannumeral0\xintra {#1}}%
1714 }%
1715 \def\XINT_fmin #1#2%

```

```

1716 {%
1717   \expandafter\XINT_fmin_A\romannumeral0\xintra w {#2}#1%
1718 }%
1719 \def\XINT_fmin_A #1#2/#3[#4]#5#6/#7[#8]%
1720 {%
1721   \xint_UDsignsfork
1722     #1#5\XINT_fmin_minusminus
1723     -#5\XINT_fmin_firstneg
1724     #1-\XINT_fmin_secondneg
1725     --\XINT_fmin_nonneg_a
1726   \krof
1727   #1#5{#2/#3[#4]}{#6/#7[#8]}%
1728 }%
1729 \def\XINT_fmin_minusminus --%
1730   {\expandafter-\romannumeral0\XINT_fmax_nonneg_b }%
1731 \def\XINT_fmin_firstneg #1-#2#3{ -#3}%
1732 \def\XINT_fmin_secondneg -#1#2#3{ -#2}%
1733 \def\XINT_fmin_nonneg_a #1#2#3#4%
1734 {%
1735   \XINT_fmin_nonneg_b {#1#3}{#2#4}%
1736 }%
1737 \def\XINT_fmin_nonneg_b #1#2%
1738 {%
1739   \if0\romannumeral0\XINT_fgeq_A #1#2%
1740     \xint_afterfi{ #2}%
1741   \else \xint_afterfi{ #1}%
1742   \fi
1743 }%

```

## 8.58 \xintMinof

1.21 protects \xintMinof against items with non terminated \the\numexpr expressions.  
The macro is not compatible with an empty list.

```

1744 \def\xintMinof      {\romannumeral0\xintminof }%
1745 \def\xintminof      #1{\expandafter\XINT_minof_a\romannumeral`&&@#1\xint:}%
1746 \def\XINT_minof_a #1{\expandafter\XINT_minof_b\romannumeral0\xintra w{#1}!}%
1747 \def\XINT_minof_b #1!#2%
1748     {\expandafter\XINT_minof_c\romannumeral`&&@#2!{#1}!}%
1749 \def\XINT_minof_c #1%
1750     {\xint_gob_til_xint: #1\XINT_minof_e\xint:\XINT_minof_d #1}%
1751 \def\XINT_minof_d #1!%
1752     {\expandafter\XINT_minof_b\romannumeral0\xintmin {#1}}%
1753 \def\XINT_minof_e #1!#2!{ #2}%

```

## 8.59 \xintCmp

```

1754 \def\xintCmp {\romannumeral0\xintcmp }%
1755 \def\xintcmp #1%
1756 {%
1757   \expandafter\XINT_fcmp\expandafter {\romannumeral0\xintra w {#1}}%
1758 }%
1759 \def\XINT_fcmp #1#2%
1760 {%

```

```

1761 \expandafter\XINT_fcmp_A\romannumeral0\xintra {#2}#1%
1762 }%
1763 \def\XINT_fcmp_A #1#2/#3[#4]#5#6/#7[#8]%
1764 {%
1765 \xint_UDsignsfork
1766 #1#5\XINT_fcmp_minusminus
1767 -#5\XINT_fcmp_firstneg
1768 #1-\XINT_fcmp_secondneg
1769 --\XINT_fcmp_nonneg_a
1770 \krof
1771 #1#5{#2/#3[#4]}{#6/#7[#8]}%
1772 }%
1773 \def\XINT_fcmp_minusminus --#1#2{\XINT_fcmp_B #2#1}%
1774 \def\XINT_fcmp_firstneg #1-#2#3{ -1}%
1775 \def\XINT_fcmp_secondneg -#1#2#3{ 1}%
1776 \def\XINT_fcmp_nonneg_a #1#2%
1777 {%
1778 \xint_UDzerosfork
1779 #1#2\XINT_fcmp_zerozero
1780 0#2\XINT_fcmp_firstzero
1781 #10\XINT_fcmp_secondzero
1782 00\XINT_fcmp_pos
1783 \krof
1784 #1#2%
1785 }%
1786 \def\XINT_fcmp_zerozero #1#2#3#4{ 0}%
1787 \def\XINT_fcmp_firstzero #1#2#3#4{ -1}%
1788 \def\XINT_fcmp_secondzero #1#2#3#4{ 1}%
1789 \def\XINT_fcmp_pos #1#2#3#4%
1790 {%
1791 \XINT_fcmp_B #1#3#2#4%
1792 }%
1793 \def\XINT_fcmp_B #1/#2[#3]#4/#5[#6]%
1794 {%
1795 \expandafter\XINT_fcmp_C\expandafter
1796 {\the\numexpr #6-#3\expandafter}\expandafter
1797 {\romannumeral0\xintiimul {#4}{#2}}%
1798 {\romannumeral0\xintiimul {#5}{#1}}%
1799 }%
1800 \def\XINT_fcmp_C #1#2#3%
1801 {%
1802 \expandafter\XINT_fcmp_D\expandafter
1803 {#3}{#1}{#2}%
1804 }%
1805 \def\XINT_fcmp_D #1#2#3%
1806 {%
1807 \expandafter\XINT_cntSgnFork\romannumeral`&&\expandafter\XINT_cntSgn
1808 \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\xint:
1809 { -1}{\XINT_fcmp_E #2\Z {#3}{#1}}{ 1}%
1810 }%
1811 \def\XINT_fcmp_E #1%
1812 {%

```

```

1813 \xint_UDsignfork
1814 #1\XINT_fcmp_Fd
1815 -{\XINT_fcmp_Fn #1}%
1816 \krof
1817 }%

1818 \def\XINT_fcmp_Fd #1\Z #2#3%
1819 {%
1820 \expandafter\XINT_fcmp_Fe
1821 \romannumeral0\XINT_dsx_addzeros {#1}#3;\xint:#2\xint:
1822 }%
1823 \def\XINT_fcmp_Fe #1\xint:#2#3\xint:{\XINT_cmp_plusplus #2#1\xint:#3\xint:}%
1824 \def\XINT_fcmp_Fn #1\Z #2#3%
1825 {%
1826 \expandafter\XINT_fcmp_Fo
1827 \romannumeral0\XINT_dsx_addzeros {#1}#2;\xint:#3\xint:
1828 }%
1829 \def\XINT_fcmp_Fo #1#2\xint:#3\xint:{\XINT_cmp_plusplus #1#3\xint:#2\xint:}%

```

## 8.60 \xintAbs

```

1830 \def\xintAbs {\romannumeral0\xintabs }%
1831 \def\xintabs #1{\expandafter\XINT_abs\romannumeral0\xintraw {#1}}%

```

## 8.61 \xintOpp

```

1832 \def\xintOpp {\romannumeral0\xintopp }%
1833 \def\xintopp #1{\expandafter\XINT_opp\romannumeral0\xintraw {#1}}%

```

## 8.62 \xintInv

### 1.3d.

```

1834 \def\xintInv {\romannumeral0\xintinv }%
1835 \def\xintinv #1{\expandafter\XINT_inv\romannumeral0\xintraw {#1}}%
1836 \def\XINT_inv #1%
1837 {%
1838 \xint_UDzerominusfork
1839 #1-\XINT_inv_iszero
1840 0#1\XINT_inv_a
1841 0-{\XINT_inv_a {}}%
1842 \krof #1%
1843 }%
1844 \def\XINT_inv_iszero #1]%
1845 {\XINT_signalcondition{DivisionByZero}{Division of 1 by zero (#1)}}{\0/1[0]}%
1846 \def\XINT_inv_a #1#2/#3[#4#5]%
1847 {%
1848 \xint_UDzerominusfork
1849 #4-\XINT_inv_expiszero
1850 0#4\XINT_inv_b
1851 0-{\XINT_inv_b -#4}%
1852 \krof #5.{#1#3/#2}%
1853 }%
1854 \def\XINT_inv_expiszero #1.#2{ #2[0]}%
1855 \def\XINT_inv_b #1.#2{ #2[#1]}%

```

### 8.63 `\xintSgn`

```
1856 \def\xintSgn {\romannumeral0\xintsgn }%
1857 \def\xintsgn #1{\expandafter\XINT_sgn\romannumeral0\xintraw {#1}\xint:}%
```

### 8.64 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-1` 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  $\TeX$  implementation which has extra cost of fetching long sequences of tokens.

### 8.65 `\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...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...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...0eN with P-1 zeroes, not 10.0...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`.

```

1858 \def\xintFloat    {\romannumeral0\xintfloat }%
1859 \def\xintfloat #1{\XINT_float_chkopt #1\xint:}%
1860 \def\XINT_float_chkopt #1%
1861 {%
1862   \ifx [#1\expandafter\XINT_float_opt
1863     \else\expandafter\XINT_float_noopt
1864   \fi #1%
1865 }%
1866 \def\XINT_float_noopt #1\xint:%
1867 {%
1868   \expandafter\XINT_float_post
1869   \romannumeral0\XINTinfloat[\XINTdigits]{#1}\XINTdigits.%
1870 }%
1871 \def\XINT_float_opt [\xint:#1]%
1872 {%
1873   \expandafter\XINT_float_opt_a\the\numexpr #1.%
1874 }%
1875 \def\XINT_float_opt_a #1.#2%
1876 {%
1877   \expandafter\XINT_float_post
1878   \romannumeral0\XINTinfloat[#1]{#2}#1.%
1879 }%
1880 \def\XINT_float_post #1%
1881 {%
1882   \xint_UDzerominusfork
1883   #1-\XINT_float_zero
1884   0#1\XINT_float_neg
1885   0-\XINT_float_pos
1886   \krof #1%
1887 }%[

```

```

1888 \def\XINT_float_zero #1#2.{ 0.e0}%
1889 \def\XINT_float_neg-{\expandafter\romannumeral0\XINT_float_pos}%
1890 \def\XINT_float_pos #1#2[#3]#4.%
1891 {%
1892   \expandafter\XINT_float_pos_done\the\numexpr#3+#4-\xint_c_i.#1.#2;%
1893 }%
1894 \def\XINT_float_pos_done #1.#2;{ #2e#1}%

```

## 8.66 \XINTinFloat, \XINTinFloatS, \XINTiLogTen

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`.

1.3e adds `\XINTiLogTen`.

```

1895 \def\XINTinFloat {\romannumeral0\XINTinfloat }%
1896 \def\XINTinfloat
1897   {\expandafter\XINT_infloat_clean\romannumeral0\XINT_infloat}%

```

Attention que ici le fait que l'on grabbe `#1` est important car il pourrait y avoir un zéro (en particulier dans le cas où input est nul).

```

1898 \def\XINT_infloat_clean #1%
1899   {\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`

```

1900 \def\XINT_infloat_clean_a !#1.#2[#3]%
1901 {%
1902   \expandafter\XINT_infloat_done
1903   \the\numexpr #3-#1\expandafter.%
1904   \romannumeral0\XINT_dsx_addzeros {#1}#2;;%
1905 }%
1906 \def\XINT_infloat_done #1.#2;{ #2[#1]}%

```

variant which allows output with shorter mantissas.

```

1907 \def\XINTinFloatS {\romannumeral0\XINTinfloatS}%
1908 \def\XINTinfloatS

```

```

1909   {\expandafter\XINT_infloatS_clean\romannumeral0\XINT_infloat}%
1910 \def\XINT_infloatS_clean #1%
1911   {\if #1!\xint_dothis\XINT_infloatS_clean_a\fi\xint_orthat{ }#1}%
1912 \def\XINT_infloatS_clean_a !#1.{ }%

```

1.3e ajoute \XINTiLogTen. Le comportement pour un input nul est non encore finalisé. Il changera lorsque NaN, +Inf, -Inf existeront.

```

1913 \def\XINTFloatiLogTen {\the\numexpr\XINTfloatilogten}%
1914 \def\XINTfloatilogten [#1]#2%
1915   {\expandafter\XINT_floatilogten\romannumeral0\XINT_infloat[#1]{#2}#1.}%
1916 \def\XINT_floatilogten #1{%
1917   \if #10\xint_dothis\XINT_floatilogten_z\fi
1918   \if #1!\xint_dothis\XINT_floatilogten_a\fi
1919   \xint_orthat\XINT_floatilogten_b #1%
1920 }%
1921 \def\XINT_floatilogten_z 0[0]#1.{-7FFF8000\relax}%
1922 \def\XINT_floatilogten_a !#1.#2[#3]#4.{#3-#1+#4-1\relax}%
1923 \def\XINT_floatilogten_b #1[#2]#3.{#2+#3-1\relax}%

```

début de la routine proprement dite, l'argument optionnel est obligatoire.

```

1924 \def\XINT_infloat [#1]#2%
1925 {%
1926   \expandafter\XINT_infloat_a\the\numexpr #1\expandafter.%
1927   \romannumeral0\XINT_infrac {#2}%
1928 }%

```

#1=P, #2=n, #3=A, #4=B.

```

1929 \def\XINT_infloat_a #1.#2#3#4%
1930 {%

```

micro boost au lieu d'utiliser \XINT\_isOne{#4}, mais pas bon style.

```

1931   \if1\XINT_is_One#4XY%
1932     \expandafter\XINT_infloat_sp
1933   \else\expandafter\XINT_infloat_fork
1934   \fi #3.{#1}{#2}{#4}%
1935 }%

```

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.

```

1936 \def\XINT_infloat_sp #1%
1937 {%
1938   \xint_UDzerominusfork
1939   #1-\XINT_infloat_spzero
1940   0#1\XINT_infloat_spneg
1941   0-\XINT_infloat_sppos
1942   \krof #1%
1943 }%

```

Attention surtout pas 0/1[0] ici.

```
1944 \def\XINT_infloat_spzero 0.#1#2#3{ 0[0]}%
1945 \def\XINT_infloat_spneg-%
1946     {\expandafter\XINT_infloat_spnegend\romannumeral0\XINT_infloat_spos}%
1947 \def\XINT_infloat_spnegend #1%
1948     {\if#1!\expandafter\XINT_infloat_spneg_needzeros\fi -#1}%
1949 \def\XINT_infloat_spneg_needzeros -!#1.{!#1.-}%
```

in: A.{P}{N}{1}  
out: P-L.A.P.N.

```
1950 \def\XINT_infloat_spos #1.#2#3#4%
1951 {%
1952     \expandafter\XINT_infloat_sp_b\the\numexpr#2-\xintLength{#1}.#1.#2.#3.%
1953 }%
```

#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.

```
1954 \def\XINT_infloat_sp_b #1%
1955 {%
1956     \xint_UDzerominusfork
1957     #1-\XINT_infloat_sp_quick
1958     0#1\XINT_infloat_sp_c
1959     0-\XINT_infloat_sp_needzeros
1960     \krof #1%
1961 }%
```

Ici P=L. Le cas usuel dans \xintfloatexpr.

```
1962 \def\XINT_infloat_sp_quick 0.#1.#2.#3.{ #1[#3]}%
```

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.

```
1963 \def\XINT_infloat_sp_needzeros #1.#2.#3.#4.{!#1.#2[#4]}%
```

L-P=#1.A=#2#3.P=#4.N=#5.  
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.

```
1964 \def\XINT_infloat_sp_c -#1.#2#3.#4.#5.%
1965 {%
1966     \expandafter\XINT_infloat_Y
1967     \the\numexpr #5+#1\expandafter.%
1968     \romannumeral0\expandafter\XINT_infloat_sp_round
1969     \romannumeral0\XINT_split_fromleft
1970     (\xint_c_i+#4).#2#3\xint_bye2345678\xint_bye..#2%
1971 }%
1972 \def\XINT_infloat_sp_round #1.#2.%
```

```
1973 {%
1974   \XINT_dsrr#1\xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax.%
1975 }%
```

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.

```
1976 \def\xINT_infloat_fork #1%
1977 {%
1978   \xint_UDsignfork
1979   #1\xINT_infloat_J
1980   -\XINT_infloat_K
1981   \krof #1%
1982 }%
1983 \def\xINT_infloat_J-{\expandafter-\romannumeral0\xINT_infloat_K }%
```

A. $\{P\}_n\{B\}$  avec B>1.

```
1984 \def\xINT_infloat_K #1.#2%
1985 {%
1986   \expandafter\xINT_infloat_L
1987   \the\numexpr\xintLength{#1}\expandafter.\the\numexpr #2+\xint_c_iv.{#1}{#2}%
1988 }%
```

$|A|.P+4.\{A\}_n\{B\}$ . We check if A already has length  $\leq P+4$ .

```
1989 \def\xINT_infloat_L #1.#2.%
1990 {%
1991   \ifnum #1>#2
1992     \expandafter\xINT_infloat_Ma
1993   \else
1994     \expandafter\xINT_infloat_Mb
1995   \fi #1.#2.%
1996 }%
```

$|A|.P+4.\{A\}_n\{B\}$ . We will keep only the first P+4 digits of A, denoted A'' in what follows.  
output: u=-0.A'' .junk.P+4. $|A|.\{A\}_n\{B\}$

```
1997 \def\xINT_infloat_Ma #1.#2.#3%
1998 {%
1999   \expandafter\xINT_infloat_MtoN\expandafter-\expandafter0\expandafter.%
2000   \romannumeral0\xINT_split_fromleft#2.#3\xint_bye2345678\xint_bye..%
2001   #2.#1.{#3}%
2002 }%
```

$|A|.P+4.\{A\}_n\{B\}$ .

Here A is short. We set  $u = P+4 - |A|$ , and  $A'' = A$  ( $A' = 10^u A$ )  
output: u.A'' ..P+4. $|A|.\{A\}_n\{B\}$

```
2003 \def\xINT_infloat_Mb #1.#2.#3%
2004 {%
2005   \expandafter\xINT_infloat_MtoN\the\numexpr#2-#1.%
2006   #3..#2.#1.{#3}%
2007 }%
```

```

input u.A'' .junk.P+4. |A|. {A}{P}{n}{B}
output |B|.P+4. {B}u.A'' .P. |A|.n. {A}{B}

```

```

2008 \def\XINT_infloat_MtoN #1.#2.#3.#4.#5.#6#7#8#9%
2009 {%
2010   \expandafter\XINT_infloat_N
2011   \the\numexpr\xintLength{#9}.#4.{#9}#1.#2.#7.#5.#8.{#6}{#9}%
2012 }%
2013 \def\XINT_infloat_N #1.#2.%
2014 {%
2015   \ifnum #1>#2
2016     \expandafter\XINT_infloat_Oa
2017   \else
2018     \expandafter\XINT_infloat_Ob
2019   \fi #1.#2.%
2020 }%

```

```

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}

```

```

2021 \def\XINT_infloat_Oa #1.#2.#3%
2022 {%
2023   \expandafter\XINT_infloat_P\expandafter-\expandafter0\expandafter.%
2024   \romannumeral0\XINT_split_fromleft#2.#3\xint_bye2345678\xint_bye..%
2025   #1.%
2026 }%

```

```

output v=P+4-|B|>=0.B'' .junk. |B|.u.A'' .P. |A|.n. {A}{B}

```

```

2027 \def\XINT_infloat_Ob #1.#2.#3%
2028 {%
2029   \expandafter\XINT_infloat_P\the\numexpr#2-#1.#3..#1.%
2030 }%

```

```

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.

```

2031 \def\XINT_infloat_P #1#2.#3.#4.#5.#6#7.#8.#9.%
2032 {%
2033   \csname XINT_infloat_Q\if-#1\else\if-#6\else q\fi\fi\expandafter\endcsname
2034   \romannumeral0\xintiigo
2035   {\romannumeral0\XINT_dsx_addzerosnofuss
2036     {#6#7-#1#2+#9+\xint_c_iii\if-#1\else\if-#6\else-\xint_c_ii\fi\fi}#8;}%
2037   {#3}.#9.#5.%
2038 }%

```

«quick» branch.

```

2039 \def\XINT_infloat_Qq #1.#2.%
2040 {%
2041   \expandafter\XINT_infloat_Rq

```

```

2042 \romannumeral0\XINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..#2.%
2043 }%
2044 \def\XINT_infloat_Rq #1.#2#3.%
2045 {%
2046 \ifnum#2<\xint_c_v
2047 \expandafter\XINT_infloat_SEq
2048 \else\expandafter\XINT_infloat_SUP
2049 \fi
2050 {\if.#3.\xint_c_\else\xint_c_i\fi}#1.%
2051 }%

```

standard branch which will have to handle undecided rounding, if too close to a mid-value.

```

2052 \def\XINT_infloat_Q #1.#2.%
2053 {%
2054 \expandafter\XINT_infloat_R
2055 \romannumeral0\XINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..#2.%
2056 }%
2057 \def\XINT_infloat_R #1.#2#3#4#5.%
2058 {%
2059 \if.#5.\expandafter\XINT_infloat_Sa\else\expandafter\XINT_infloat_Sb\fi
2060 #2#3#4#5.#1.%
2061 }%

```

trailing digits.Q.P.|B|.|A|.n.{A}{B}  
 #1=trailing digits (they may have leading zeros.)

```

2062 \def\XINT_infloat_Sa #1.%
2063 {%
2064 \ifnum#1>500 \xint_dothis\XINT_infloat_SUP\fi
2065 \ifnum#1<499 \xint_dothis\XINT_infloat_SEq\fi
2066 \xint_orthat\XINT_infloat_X\xint_c_
2067 }%
2068 \def\XINT_infloat_Sb #1.%
2069 {%
2070 \ifnum#1>5009 \xint_dothis\XINT_infloat_SUP\fi
2071 \ifnum#1<4990 \xint_dothis\XINT_infloat_SEq\fi
2072 \xint_orthat\XINT_infloat_X\xint_c_i
2073 }%

```

epsilon #2=Q.#3=P.#4=|B|. #5=|A|. #6=n.{A}{B}  
 exposant final est  $n+|A|-|B|-P+\text{epsilon}$

```

2074 \def\XINT_infloat_SEq #1#2.#3.#4.#5.#6.#7#8%
2075 {%
2076 \expandafter\XINT_infloat_SY
2077 \the\numexpr #6+#5-#4-#3+#1.#2.%
2078 }%
2079 \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.

```

2080 \def\XINT_infloat_SUP #1#2#3.#4.#5.#6.#7.#8#9%
2081 {%

```

```

2082 \expandafter\XINT_infloat_Y
2083 \the\numexpr#7+#6-#5-#4+#1\expandafter.%
2084 \romannumeral0\xintinc{#2#3}.#2%
2085 }%

```

$\epsilon Q.P.|B|.|A|.n.\{A\}\{B\}$

`\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.

$\#1=\epsilon$ ,  $\#2\#3=Q$ ,  $\#4=P$ ,  $\#5=|B|$ ,  $\#6=|A|$ ,  $\#7=n$ ,  $\#8=A$ ,  $\#9=B$

```

2086 \def\XINT_infloat_X #1#2#3.#4.#5.#6.#7.#8#9%
2087 {%
2088 \expandafter\XINT_infloat_Y
2089 \the\numexpr #7+#6-#5-#4+#1\expandafter.%
2090 \romannumeral`&&\romannumeral0\xintiiiflt
2091 {\xintDSH{#6-#5-#4+#1}\xintDouble{#8}}}%
2092 {\xintiiMul{\xintInc{\xintDouble{#2#3}}}\{#9}}}%
2093 \xint_firstofone
2094 \xintinc{#2#3}.#2%
2095 }%

```

check for rounding up to next power of ten.

```

2096 \def\XINT_infloat_Y #1{%
2097 \def\XINT_infloat_Y ##1.##2##3.##4%
2098 {%
2099 \if##49\if##21\expandafter\expandafter\expandafter\XINT_infloat_Z\fi\fi
2100 #1##2##3[##1]%
2101 }}\XINT_infloat_Y{ }%

```

$\#1=1$ ,  $\#2=0$ .

```

2102 \def\XINT_infloat_Z #1#2#3[#4]%
2103 {%
2104 \expandafter\XINT_infloat_ZZ\the\numexpr#4+\xint_c_i.#3.%
2105 }%
2106 \def\XINT_infloat_ZZ #1.#2.{ 1#2[#1]}%

```

## 8.67 `\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  $-5 < N \leq 5$ . 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).

```

2107 \def\xintPFloat {\romannumeral0\xintpfloat }%
2108 \def\xintpfloat #1{\XINT_pfloat_chkopt #1\xint:}%
2109 \def\XINT_pfloat_chkopt #1%

```



```

2110 {%
2111   \ifx [#1\expandafter\XINT_pfloat_opt
2112     \else\expandafter\XINT_pfloat_noopt
2113   \fi #1%
2114 }%
2115 \def\XINT_pfloat_noopt #1\xint:%
2116 {%
2117   \expandafter\XINT_pfloat_a
2118   \romannumeral0\xintfloat [\XINTdigits]{#1};\XINTdigits.%
2119 }%

2120 \def\XINT_pfloat_opt [\xint:#1]%
2121 {%
2122   \expandafter\XINT_pfloat_opt_a \the\numexpr #1.%
2123 }%
2124 \def\XINT_pfloat_opt_a #1.#2%
2125 {%
2126   \expandafter\XINT_pfloat_a\romannumeral0\xintfloat [#1]{#2};#1.%
2127 }%
2128 \def\XINT_pfloat_a #1%
2129 {%
2130   \xint_UDzerominusfork
2131     #1-\XINT_pfloat_zero
2132     0#1\XINT_pfloat_neg
2133     0-\XINT_pfloat_pos
2134   \krof #1%
2135 }%

2136 \def\XINT_pfloat_zero #1;#2.{ 0.}%
2137 \def\XINT_pfloat_neg-{\expandafter-\romannumeral0\XINT_pfloat_pos }%

2138 \def\XINT_pfloat_pos #1.#2e#3;#4.%
2139 {%
2140   \ifnum #3>\xint_c_v \xint_dothis\XINT_pfloat_no\fi
2141   \ifnum #3<-\xint_c_v \xint_dothis\XINT_pfloat_no\fi
2142   \ifnum #3<\xint_c_ \xint_dothis\XINT_pfloat_N\fi
2143   \ifnum #3>\numexpr #4-\xint_c_i\relax \xint_dothis\XINT_pfloat_Ps\fi
2144   \xint_orthat\XINT_pfloat_P #1#2e#3;%
2145 }%
2146 \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.

2147 \def\XINT_pfloat_N #1e-#2;%
2148 {%
2149   \csname XINT_pfloat_N_\romannumeral#2\endcsname #1%
2150 }%
2151 \def\XINT_pfloat_N_i { 0.}%
2152 \def\XINT_pfloat_N_ii { 0.0}%
2153 \def\XINT_pfloat_N_iii{ 0.00}%
2154 \def\XINT_pfloat_N_iv { 0.000}%
2155 \def\XINT_pfloat_N_v { 0.0000}%

```

```

2156 \def\XINT_pfloat_P #1e#2;%
2157 {%
2158   \csname XINT_pfloat_P_\romannumeral#2\endcsname #1%
2159 }%
2160 \def\XINT_pfloat_P_ #1{ #1.}%
2161 \def\XINT_pfloat_P_i #1#2{ #1#2.}%
2162 \def\XINT_pfloat_P_ii #1#2#3{ #1#2#3.}%
2163 \def\XINT_pfloat_P_iii#1#2#3#4{ #1#2#3#4.}%
2164 \def\XINT_pfloat_P_iv #1#2#3#4#5{ #1#2#3#4#5.}%
2165 \def\XINT_pfloat_P_v #1#2#3#4#5#6{ #1#2#3#4#5#6.}%

2166 \def\XINT_pfloat_Ps #1e#2;%
2167 {%
2168   \csname XINT_pfloat_Ps_\romannumeral#2\endcsname #100000;%
2169 }%
2170 \def\XINT_pfloat_Psi #1#2#3;{ #1#2.}%
2171 \def\XINT_pfloat_Psii #1#2#3#4;{ #1#2#3.}%
2172 \def\XINT_pfloat_Psiii#1#2#3#4#5;{ #1#2#3#4.}%
2173 \def\XINT_pfloat_Psiv #1#2#3#4#5#6;{ #1#2#3#4#5.}%
2174 \def\XINT_pfloat_Psv #1#2#3#4#5#6#7;{ #1#2#3#4#5#6.}%

```

## 8.68 \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.

```

2175 \def\XINTinFloatFracdigits {\romannumeral0\xINTinfloatfracdigits }%
2176 \def\XINTinfloatfracdigits #1%
2177 {%
2178   \expandafter\XINT_infloatfracdg_a\expandafter {\romannumeral0\xintfrac{#1}}%
2179 }%
2180 \def\XINT_infloatfracdg_a {\XINTinfloat [\XINTdigits]}%

```

## 8.69 \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.

```

2181 \def\xintFloatAdd      {\romannumeral0\xintfloatadd }%
2182 \def\xintfloatadd     #1{\XINT_fladd_chkopt \xintfloat #1\xint:}%
2183 \def\XINTinFloatAdd   {\romannumeral0\XINTinfloatadd }%
2184 \def\XINTinfloatadd   #1{\XINT_fladd_chkopt \XINTinfloatS #1\xint:}%
2185 \def\XINT_fladd_chkopt #1#2%

```

```

2186 {%
2187   \ifx [#2\expandafter\XINT_fladd_opt
2188     \else\expandafter\XINT_fladd_noopt
2189   \fi #1#2%
2190 }%
2191 \def\XINT_fladd_noopt #1#2\xint:#3%
2192 {%
2193   #1[\XINTdigits]%
2194   {\expandafter\XINT_FL_add_a
2195     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{#3}}%
2196 }%
2197 \def\XINT_fladd_opt #1[\xint:#2]%#3#4%
2198 {%
2199   \expandafter\XINT_fladd_opt_a\the\numexpr #2.#1%
2200 }%
2201 \def\XINT_fladd_opt_a #1.#2#3#4%
2202 {%
2203   #2[#1]{\expandafter\XINT_FL_add_a\romannumeral0\XINTinfloat[#1]{#3}#1.{#4}}%
2204 }%
2205 \def\XINT_FL_add_a #1%
2206 {%
2207   \xint_gob_til_zero #1\XINT_FL_add_zero 0\XINT_FL_add_b #1%
2208 }%

2209 \def\XINT_FL_add_zero #1.#2{#2}%[[
2210 \def\XINT_FL_add_b #1]#2.#3%
2211 {%
2212   \expandafter\XINT_FL_add_c\romannumeral0\XINTinfloat[#2]{#3}#2.#1)%
2213 }%

2214 \def\XINT_FL_add_c #1%
2215 {%
2216   \xint_gob_til_zero #1\XINT_FL_add_zero 0\XINT_FL_add_d #1%
2217 }%

2218 \def\XINT_FL_add_d #1[#2]#3.#4[#5]%
2219 {%
2220   \ifnum\numexpr #2-#3-#5>\xint_c_\xint_dothis\xint_firstoftwo\fi
2221   \ifnum\numexpr #5-#3-#2>\xint_c_\xint_dothis\xint_secondoftwo\fi
2222   \xint_orthat\xintAdd {#1[#2]}{#4[#5]}%
2223 }%

```

## 8.70 \xintFloatSub, \XINTinFloatSub

First done 1.07.

Starting with 1.2f the arguments undergo an initial rounding to the target precision P not P+2.

```

2224 \def\xintFloatSub      {\romannumeral0\xintfloatsub }%
2225 \def\xintfloatsub     #1{\XINT_fsub_chkopt \xintfloat #1\xint:}%
2226 \def\XINTinFloatSub   {\romannumeral0\XINTinfloatsub }%
2227 \def\XINTinfloatsub   #1{\XINT_fsub_chkopt \XINTinfloatS #1\xint:}%

```

```

2228 \def\XINT_flsub_chkopt #1#2%
2229 {%
2230   \ifx [#2\expandafter\XINT_flsub_opt
2231     \else\expandafter\XINT_flsub_noopt
2232   \fi #1#2%
2233 }%
2234 \def\XINT_flsub_noopt #1#2\xint:#3%
2235 {%
2236   #1[\XINTdigits]%
2237   {\expandafter\XINT_FL_add_a
2238     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{\xintOpp{#3}}}%
2239 }%
2240 \def\XINT_flsub_opt #1[\xint:#2]%#3#4%
2241 {%
2242   \expandafter\XINT_flsub_opt_a\the\numexpr #2.#1%
2243 }%
2244 \def\XINT_flsub_opt_a #1.#2#3#4%
2245 {%
2246   #2[#1]{\expandafter\XINT_FL_add_a\romannumeral0\XINTinfloat[#1]{#3}#1.{\xintOpp{#4}}}%
2247 }%

```

## 8.71 \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).

```

2248 \def\xintFloatMul {\romannumeral0\xintfloatmul }%
2249 \def\xintfloatmul #1{\XINT_flmul_chkopt \xintfloat #1\xint:}%
2250 \def\XINTinFloatMul {\romannumeral0\XINTinfloatmul }%
2251 \def\XINTinfloatmul #1{\XINT_flmul_chkopt \XINTinfloatS #1\xint:}%
2252 \def\XINT_flmul_chkopt #1#2%
2253 {%
2254   \ifx [#2\expandafter\XINT_flmul_opt
2255     \else\expandafter\XINT_flmul_noopt
2256   \fi #1#2%
2257 }%
2258 \def\XINT_flmul_noopt #1#2\xint:#3%
2259 {%
2260   #1[\XINTdigits]%
2261   {\expandafter\XINT_FL_mul_a
2262     \romannumeral0\XINTinfloatS[\XINTdigits]{#2}\XINTdigits.{#3}}%
2263 }%
2264 \def\XINT_flmul_opt #1[\xint:#2]%#3#4%
2265 {%
2266   \expandafter\XINT_flmul_opt_a\the\numexpr #2.#1%
2267 }%
2268 \def\XINT_flmul_opt_a #1.#2#3#4%
2269 {%

```

```

2270 #2[#1]{\expandafter\XINT_FL_mul_a\romannumeral0\XINTinfloatS[#1]{#3}#1.{#4}}%
2271 }%
2272 \def\XINT_FL_mul_a #1[#2]#3.#4%
2273 {%
2274 \expandafter\XINT_FL_mul_b\romannumeral0\XINTinfloatS[#3]{#4}#1[#2]%
2275 }%

```

```

2276 \def\XINT_FL_mul_b #1[#2]#3[#4]{\xintiiMul{#3}{#1}/1[#4+#2]}%

```

## 8.72 \XINTinFloatInv

Added belatedly at 1.3e, to support inv() function. We use Short output, for rare inv(\xintexpr 1/3\relax) case. I need to think the whole thing out at some later date.

```

2277 \def\XINTinFloatInv#1{\XINTinFloatS[\XINTdigits]{\xintInv{#1}}}%

```

## 8.73 \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.

```

2278 \def\xintFloatDiv {\romannumeral0\xintfloatdiv }%
2279 \def\xintfloatdiv #1{\XINT_fldiv_chkopt \xintfloat #1\xint:}%
2280 \def\XINTinFloatDiv {\romannumeral0\XINTinfloatdiv }%
2281 \def\XINTinfloatdiv #1{\XINT_fldiv_chkopt \XINTinfloatS #1\xint:}%
2282 \def\XINT_fldiv_chkopt #1#2%
2283 {%
2284 \ifx [#2\expandafter\XINT_fldiv_opt
2285 \else\expandafter\XINT_fldiv_noopt
2286 \fi #1#2%
2287 }%

```

```

2288 \def\XINT_fldiv_noopt #1#2\xint:#3%
2289 {%
2290 #1[\XINTdigits]%
2291 {\expandafter\XINT_FL_div_a
2292 \romannumeral0\XINTinfloatS[\XINTdigits]{#3}\XINTdigits.{#2}}%
2293 }%
2294 \def\XINT_fldiv_opt #1[\xint:#2]#3#4%
2295 {%
2296 \expandafter\XINT_fldiv_opt_a\the\numexpr #2.#1%
2297 }%

```

```

2298 \def\XINT_fldiv_opt_a #1.#2#3#4%
2299 {%
2300 #2[#1]{\expandafter\XINT_FL_div_a\romannumeral0\XINTinfloatS[#1]{#4}#1.{#3}}%

```

```

2301 }%
2302 \def\XINT_FL_div_a #1[#2]#3.#4%
2303 {%
2304     \expandafter\XINT_FL_div_b\romannumeral0\XINTinfloatS[#3]{#4}/#1e#2%
2305 }%

```

```

2306 \def\XINT_FL_div_b #1[#2]{#1e#2}%

```

## 8.74 \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).

```

2307 \def\xintFloatPow {\romannumeral0\xintfloatpow}%
2308 \def\xintfloatpow #1{\XINT_flpow_chkopt \xintfloat #1\xint:}%
2309 \def\XINTinFloatPow {\romannumeral0\XINTinfloatpow}%
2310 \def\XINTinfloatpow #1{\XINT_flpow_chkopt \XINTinfloatS #1\xint:}%
2311 \def\XINT_flpow_chkopt #1#2%
2312 {%
2313     \ifx [#2\expandafter\XINT_flpow_opt
2314         \else\expandafter\XINT_flpow_noopt
2315         \fi
2316     #1#2%
2317 }%
2318 \def\XINT_flpow_noopt #1#2\xint:#3%
2319 {%
2320     \expandafter\XINT_flpow_checkB_a
2321     \the\numexpr #3.\XINTdigits.{#2}{#1[\XINTdigits]}%
2322 }%
2323 \def\XINT_flpow_opt #1[\xint:#2]%
2324 {%
2325     \expandafter\XINT_flpow_opt_a\the\numexpr #2.#1%
2326 }%
2327 \def\XINT_flpow_opt_a #1.#2#3#4%
2328 {%
2329     \expandafter\XINT_flpow_checkB_a\the\numexpr #4.#1.{#3}{#2[#1]}%
2330 }%
2331 \def\XINT_flpow_checkB_a #1%
2332 {%
2333     \xint_UDzerominusfork
2334     #1-\XINT_flpow_BisZero
2335     0#1{\XINT_flpow_checkB_b -}%
2336     0-{\XINT_flpow_checkB_b }{#1}%
2337     \krof
2338 }%
2339 \def\XINT_flpow_BisZero .#1.#2#3{#3{1[0]}}%

```

```

2340 \def\XINT_flpow_checkB_b #1#2.#3.%
2341 {%
2342     \expandafter\XINT_flpow_checkB_c
2343     \the\numexpr\xintLength{#2}+\xint_c_iii.#3.#2.{#1}%
2344 }%

2345 \def\XINT_flpow_checkB_c #1.#2.%
2346 {%
2347     \expandafter\XINT_flpow_checkB_d\the\numexpr#1+#2.#1.#2.%
2348 }%

    1.2f rounds input to P digits, first.

2349 \def\XINT_flpow_checkB_d #1.#2.#3.#4.#5#6%
2350 {%
2351     \expandafter \XINT_flpow_aa
2352     \romannumeral0\XINTinfloat [#3]{#6}{#2}{#1}{#4}{#5}%
2353 }%

2354 \def\XINT_flpow_aa #1[#2]#3%
2355 {%
2356     \expandafter\XINT_flpow_ab\the\numexpr #2-#3\expandafter.%
2357     \romannumeral\XINT_rep #3\endcsname0.#1.%
2358 }%

2359 \def\XINT_flpow_ab #1.#2.#3.{\XINT_flpow_a #3#2[#1]}%

2360 \def\XINT_flpow_a #1%
2361 {%
2362     \xint_UDzerominusfork
2363     #1-\XINT_flpow_zero
2364     0#1{\XINT_flpow_b \iftrue}%
2365     0-{\XINT_flpow_b \iffalse#1}%
2366     \krof
2367 }%
2368 \def\XINT_flpow_zero #1[#2]#3#4#5#6%
2369 {%
2370     #6{\if 1#5\xint_dothis {0[0]}\fi
2371         \xint_orthat
2372         {\XINT_signalcondition{DivisionByZero}{0 to the power #4}{0[0]}}%
2373         }%
2374 }%

2375 \def\XINT_flpow_b #1#2[#3]#4#5%
2376 {%
2377     \XINT_flpow_loopI #5.#3.#2.#4.{#1\ifodd #5 \xint_c_i\fi\fi}%
2378 }%

2379 \def\XINT_flpow_truncate #1.#2.#3.%
2380 {%
2381     \expandafter\XINT_flpow_truncate_a
2382     \romannumeral0\XINT_split_fromleft
2383     #3.#2\xint_bye2345678\xint_bye..#1.#3.%
2384 }%

```

```

2385 \def\XINT_flpow_truncate_a #1.#2.#3.{#3+\xintLength{#2}.#1.}%
2386 \def\XINT_flpow_loopI #1.%
2387 {%
2388   \ifnum #1=\xint_c_i\expandafter\XINT_flpow_ItoIII\fi
2389   \ifodd #1
2390     \expandafter\XINT_flpow_loopI_odd
2391   \else
2392     \expandafter\XINT_flpow_loopI_even
2393   \fi
2394   #1.%
2395 }%

2396 \def\XINT_flpow_ItoIII\ifodd #1\fi #2.#3.#4.#5.#6%
2397 {%
2398   \expandafter\XINT_flpow_III\the\numexpr #6+\xint_c_.#3.#4.#5.%
2399 }%

2400 \def\XINT_flpow_loopI_even #1.#2.#3.%#4.%
2401 {%
2402   \expandafter\XINT_flpow_loopI
2403   \the\numexpr #1/\xint_c_ii\expandafter.%
2404   \the\numexpr\expandafter\XINT_flpow_truncate
2405   \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.%
2406 }%
2407 \def\XINT_flpow_loopI_odd #1.#2.#3.#4.%
2408 {%
2409   \expandafter\XINT_flpow_loopII
2410   \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter.%
2411   \the\numexpr\expandafter\XINT_flpow_truncate
2412   \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#2.#3.%
2413 }%
2414 \def\XINT_flpow_loopII #1.%
2415 {%
2416   \ifnum #1 = \xint_c_i\expandafter\XINT_flpow_ItoIII\fi
2417   \ifodd #1
2418     \expandafter\XINT_flpow_loopII_odd
2419   \else
2420     \expandafter\XINT_flpow_loopII_even
2421   \fi
2422   #1.%
2423 }%
2424 \def\XINT_flpow_loopII_even #1.#2.#3.%#4.%
2425 {%
2426   \expandafter\XINT_flpow_loopII
2427   \the\numexpr #1/\xint_c_ii\expandafter.%
2428   \the\numexpr\expandafter\XINT_flpow_truncate
2429   \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.%
2430 }%
2431 \def\XINT_flpow_loopII_odd #1.#2.#3.#4.#5.#6.%
2432 {%
2433   \expandafter\XINT_flpow_loopII_odda
2434   \the\numexpr\expandafter\XINT_flpow_truncate

```



```

2435 \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2436 #1.#2.#3.%
2437 }%
2438 \def\XINT_flpow_loopII_odda #1.#2.#3.#4.#5.#6.%
2439 {%
2440 \expandafter\XINT_flpow_loopII
2441 \the\numexpr #4/\xint_c_ii-\xint_c_i\expandafter.%
2442 \the\numexpr\expandafter\XINT_flpow_truncate
2443 \the\numexpr\xint_c_ii*#5\expandafter.\romannumeral0\xintiisqr{#6}.#3.%
2444 #1.#2.%
2445 }%

2446 \def\XINT_flpow_IItoIII\ifodd #1\fi #2.#3.#4.#5.#6.#7.#8%
2447 {%
2448 \expandafter\XINT_flpow_III\the\numexpr #8+\xint_c_\expandafter.%
2449 \the\numexpr\expandafter\XINT_flpow_truncate
2450 \the\numexpr#3+#6\expandafter.\romannumeral0\xintiimul{#4}{#7}.#5.%
2451 }%

```

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 ulp(Z) (and worst cases can only be slightly worse than 0.51 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 ulp(Z), and at any rate it is less than 0.52 ulp(Z).

```

2452 \def\XINT_flpow_III #1.#2.#3.#4.#5%
2453 {%
2454 \expandafter\XINT_flpow_IIIend
2455 \xint_UDsignfork
2456 #5{{1/#3[-#2]}}%
2457 -{{#3[#2]}}%
2458 \krof #1%
2459 }%

2460 \def\XINT_flpow_IIIend #1#2#3%
2461 {#3{\if#21\xint_afterfi{\expandafter-\romannumeral`&&@\fi#1}}%

```

## 8.75 `\xintFloatPower`, `\XINTinFloatPower`

1.07. The core loop has been re-organized in 1.09j for some slight efficiency gain. The exponent B is given to `\xintNum`. The `^` in expressions is mapped to this routine.

Same modifications as in `\xintFloatPow` for 1.2f.

1.2f adds a special private macro for allowing half-integral exponents for use with `^` 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.

```
2462 \def\xintFloatPower    {\romannumeral0\xintfloatpower}%
2463 \def\xintfloatpower #1{\XINT_flpower_chkopt \xintfloat #1\xint:}%
2464 \def\XINTinFloatPower  {\romannumeral0\XINTinfloatpower }%
2465 \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`.

```
2466 \def\XINTinFloatPowerH {\romannumeral0\XINTinfloatpowerh }%

2467 \def\XINTinfloatpowerh #1#2%
2468 {%
2469   \expandafter\XINT_flpowerh_a\romannumeral0\xintitrunc1{#2};%
2470   \XINTdigits.{#1}{\XINTinfloatS[\XINTdigits]}%
2471 }%

2472 \def\XINT_flpowerh_a #1;%
2473 {%
2474   \if0\xintLDg{#1}\expandafter\XINT_flpowerh_int
2475   \else\expandafter\XINT_flpowerh_b
2476   \fi #1.%
2477 }%
2478 \def\XINT_flpowerh_int #1%
2479 {%
2480   \if0#1\expandafter\XINT_flpower_BisZero
2481   \else\expandafter\XINT_flpowerh_i
2482   \fi #1%
2483 }%
2484 \def\XINT_flpowerh_i #10.{\expandafter\XINT_flpower_checkB_a#1.}%
2485 \def\XINT_flpowerh_b #1.%
2486 {%
2487   \expandafter\XINT_flpowerh_c\romannumeral0\xintdsrr{\xintDouble{#1}}.%
2488 }%
2489 \def\XINT_flpowerh_c #1.%
2490 {%
2491   \ifodd\xintLDg{#1} %<- intentional space
2492   \expandafter\XINT_flpowerh_d\else\expandafter\XINT_flpowerh_e
2493   \fi #1.%
2494 }%
```

```

2495 \def\XINT_flpowerh_d #1.\XINTdigits.#2#3%
2496 {%
2497   \XINT_flpower_checkB_a #1.\XINTdigits.{#2}\XINT_flpowerh_finish
2498 }%
2499 \def\XINT_flpowerh_finish #1%
2500   {\XINTinfloatS[\XINTdigits]{\XINTinFloatSqrt[\XINTdigits+\xint_c_iii]{#1}}}%
2501 \def\XINT_flpowerh_e #1.%
2502   {\expandafter\XINT_flpower_checkB_a\romannumeral0\xinthalff{#1}.}%

```

Start of macro. Check for optional argument.

```

2503 \def\XINT_flpower_chkopt #1#2%
2504 {%
2505   \ifx [#2\expandafter\XINT_flpower_opt
2506     \else\expandafter\XINT_flpower_noopt
2507     \fi
2508   #1#2%
2509 }%
2510 \def\XINT_flpower_noopt #1#2\xint:#3%
2511 {%
2512   \expandafter\XINT_flpower_checkB_a
2513   \romannumeral0\xintnum{#3}.\XINTdigits.{#2}{#1[\XINTdigits]}%
2514 }%
2515 \def\XINT_flpower_opt #1[\xint:#2]%
2516 {%
2517   \expandafter\XINT_flpower_opt_a\the\numexpr #2.#1%
2518 }%
2519 \def\XINT_flpower_opt_a #1.#2#3#4%
2520 {%
2521   \expandafter\XINT_flpower_checkB_a
2522   \romannumeral0\xintnum{#4}.#1.{#3}{#2[#1]}%
2523 }%
2524 \def\XINT_flpower_checkB_a #1%
2525 {%
2526   \xint_UDzerominusfork
2527   #1-{\XINT_flpower_BisZero 0}%
2528   0#1{\XINT_flpower_checkB_b -}%
2529   0-{\XINT_flpower_checkB_b }{#1}%
2530   \krof
2531 }%
2532 \def\XINT_flpower_BisZero 0.#1.#2#3{#3{1[0]}}%
2533 \def\XINT_flpower_checkB_b #1#2.#3.%
2534 {%
2535   \expandafter\XINT_flpower_checkB_c
2536   \the\numexpr\xintLength{#2}+\xint_c_iii.#3.#2.{#1}%
2537 }%

2538 \def\XINT_flpower_checkB_c #1.#2.%
2539 {%
2540   \expandafter\XINT_flpower_checkB_d\the\numexpr#1+#2.#1.#2.%
2541 }%

```

```

2542 \def\XINT_flpower_checkB_d #1.#2.#3.#4.#5#6%
2543 {%
2544   \expandafter \XINT_flpower_aa
2545   \romannumeral0\XINTinfloat [#3]{#6}{#2}{#1}{#4}{#5}%
2546 }%

2547 \def\XINT_flpower_aa #1[#2]#3%
2548 {%
2549   \expandafter\XINT_flpower_ab\the\numexpr #2-#3\expandafter.%
2550   \romannumeral\XINT_rep #3\endcsname0.#1.%
2551 }%
2552 \def\XINT_flpower_ab #1.#2.#3.{\XINT_flpower_a #3#2[#1]}%
2553 \def\XINT_flpower_a #1%
2554 {%
2555   \xint_UDzerominusfork
2556   #1-\XINT_flpow_zero
2557   0#1{\XINT_flpower_b \iftrue}%
2558   0-{\XINT_flpower_b \iffalse#1}%
2559   \krof
2560 }%
2561 \def\XINT_flpower_b #1#2[#3]#4#5%
2562 {%
2563   \XINT_flpower_loopI #5.#3.#2.#4.{#1\xintiioOdd{#5}\fi}%
2564 }%
2565 \def\XINT_flpower_loopI #1.%
2566 {%
2567   \if1\XINT_isOne {#1}\xint_dothis\XINT_flpower_ItoIII\fi
2568   \ifodd\xintLDg{#1} %<- intentional space
2569   \xint_dothis{\expandafter\XINT_flpower_loopI_odd}\fi
2570   \xint_orthat{\expandafter\XINT_flpower_loopI_even}%

2571   \romannumeral0\XINT_half
2572   #1\xint_bye\xint_Bye345678\xint_bye
2573   *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax.%
2574 }%
2575 \def\XINT_flpower_ItoIII #1.#2.#3.#4.#5%
2576 {%
2577   \expandafter\XINT_flpow_III\the\numexpr #5+\xint_c_.#2.#3.#4.%
2578 }%
2579 \def\XINT_flpower_loopI_even #1.#2.#3.#4.%
2580 {%
2581   \expandafter\XINT_flpower_toloopI
2582   \the\numexpr\expandafter\XINT_flpow_truncate
2583   \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#1.%
2584 }%
2585 \def\XINT_flpower_toloopI #1.#2.#3.#4.{\XINT_flpower_loopI #4.#1.#2.#3.}%
2586 \def\XINT_flpower_loopI_odd #1.#2.#3.#4.%
2587 {%
2588   \expandafter\XINT_flpower_toloopII
2589   \the\numexpr\expandafter\XINT_flpow_truncate
2590   \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.%
2591   #1.#2.#3.%

```

```

2592 }%
2593 \def\XINT_flpower_toloopII #1.#2.#3.#4.{\XINT_flpower_loopII #4.#1.#2.#3.}%
2594 \def\XINT_flpower_loopII #1.%
2595 {%
2596   \if1\XINT_isOne{#1}\xint_dothis\XINT_flpower_IItoIII\fi
2597   \ifodd\xintLDg{#1} %<- intentional space
2598     \xint_dothis{\expandafter\XINT_flpower_loopII_odd}\fi
2599   \xint_orthat{\expandafter\XINT_flpower_loopII_even}%

2600   \romannumeral0\XINT_half#1\xint_bye\xint_Bye345678\xint_bye
2601   *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax.%
2602 }%
2603 \def\XINT_flpower_loopII_even #1.#2.#3.#4.%
2604 {%
2605   \expandafter\XINT_flpower_toloopII
2606   \the\numexpr\expandafter\XINT_flpow_truncate
2607   \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#1.%
2608 }%
2609 \def\XINT_flpower_loopII_odd #1.#2.#3.#4.#5.#6.%
2610 {%
2611   \expandafter\XINT_flpower_loopII_odda
2612   \the\numexpr\expandafter\XINT_flpow_truncate
2613   \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2614   #1.#2.#3.%
2615 }%
2616 \def\XINT_flpower_loopII_odda #1.#2.#3.#4.#5.#6.%
2617 {%
2618   \expandafter\XINT_flpower_toloopII
2619   \the\numexpr\expandafter\XINT_flpow_truncate
2620   \the\numexpr\xint_c_ii*#5\expandafter.\romannumeral0\xintiisqr{#6}.#3.%
2621   #4.#1.#2.%
2622 }%
2623 \def\XINT_flpower_IItoIII #1.#2.#3.#4.#5.#6.#7%
2624 {%
2625   \expandafter\XINT_flpow_III\the\numexpr #7+\xint_c_\expandafter.%
2626   \the\numexpr\expandafter\XINT_flpow_truncate
2627   \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2628 }%

```

## 8.76 \xintFloatFac, \XINTFloatFac

Done at 1.2. At 1.3e \XINTinFloatFac outputs using \XINTinFloatS.

```

2629 \def\xintFloatFac {\romannumeral0\xintfloatfac}%
2630 \def\xintfloatfac #1{\XINT_flfac_chkopt \xintfloat #1\xint:}%
2631 \def\XINTinFloatFac {\romannumeral0\XINTinfloatfac}%
2632 \def\XINTinfloatfac #1{\XINT_flfac_chkopt \XINTinfloatS #1\xint:}%
2633 \def\XINT_flfac_chkopt #1#2%
2634 {%
2635   \ifx [#2\expandafter\XINT_flfac_opt
2636     \else\expandafter\XINT_flfac_noopt
2637   \fi
2638   #1#2%

```

```

2639 }%
2640 \def\XINT_flfac_noopt #1#2\xint:
2641 {%
2642   \expandafter\XINT_FL_fac_fork_a
2643   \the\numexpr \xintNum{#2}.\xint_c_i \XINTdigits\XINT_FL_fac_out{#1[\XINTdigits]}%
2644 }%
2645 \def\XINT_flfac_opt #1[\xint:#2]%
2646 {%
2647   \expandafter\XINT_flfac_opt_a\the\numexpr #2.#1%
2648 }%
2649 \def\XINT_flfac_opt_a #1.#2#3%
2650 {%
2651   \expandafter\XINT_FL_fac_fork_a\the\numexpr \xintNum{#3}.\xint_c_i {#1}\XINT_FL_fac_out{#2[#1]}%
2652 }%
2653 \def\XINT_FL_fac_fork_a #1%
2654 {%
2655   \xint_UDzerominusfork
2656   #1-\XINT_FL_fac_iszero
2657   0#1\XINT_FL_fac_isneg
2658   0-\XINT_FL_fac_fork_b #1}%
2659 \krof
2660 }%
2661 \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.

```

2662 \def\XINT_FL_fac_isneg #1.#2#3#4#5%
2663 {%
2664   #5{\XINT_signalcondition{InvalidOperation}
2665     {Factorial of negative: (-#1)!}}{0[0]}}%
2666 }%
2667 \def\XINT_FL_fac_fork_b #1.%
2668 {%
2669   \ifnum #1>\xint_c_x^viii_mone\xint_dothis\XINT_FL_fac_toobig\fi
2670   \ifnum #1>\xint_c_x^iv\xint_dothis\XINT_FL_fac_vbig \fi
2671   \ifnum #1>465 \xint_dothis\XINT_FL_fac_big\fi
2672   \ifnum #1>101 \xint_dothis\XINT_FL_fac_med\fi
2673   \xint_orthat\XINT_FL_fac_small
2674   #1.%
2675 }%
2676 \def\XINT_FL_fac_toobig #1.#2#3#4#5%
2677 {%
2678   #5{\XINT_signalcondition{InvalidOperation}
2679     {Factorial of too big: (#1)!}}{0[0]}}%
2680 }%

```

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<sup>-P</sup> 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 ulp(X''). (ulp= unit in the last (significant) place). Let N be the number of such operations, the formula for Q deduces from the previous explanations is that 8Q should be at least P+9+k, with k the number of digits of N (in base 10). Note that 1.2 version used P+10+k, for 1.2f I reduced to P+9+k. Also, k should be the

number of digits of the number  $N$  of multiplications done, hence for  $n \leq 10000$  we can take  $N = n/2$ , or  $N/3$ , or  $N/4$ . This is rounded above by `numexpr` and always an overestimate of the actual number of approximate multiplications done (the first ones are exact). (vérifier ce que je raconte, j'ai la flemme là).

We then want  $\text{ceil}((P+k+n)/8)$ . Using `\numexpr` rounding division (ARRRRRGGGHHHH), if  $m$  is a positive integer,  $\text{ceil}(m/8)$  can be computed as  $(m+3)/8$ . Thus with  $m = P+10+k$ , this gives  $Q = (P+13+k)/8$ . The routine actually computes  $8(Q-1)$  for use in `\XINT_FL_fac_addzeros`.

With 1.2f the formula is  $m = P+9+k$ ,  $Q = (P+12+k)/8$ , and we use now  $4 = 12 - 8$  rather than the earlier  $5 = 13 - 8$ . Whatever happens, the value computed in `\XINT_FL_fac_increaseP` is at least 8. There will always be an extra block.

Note: with `Digits:=32`; Maple gives for  $200!$ :

```
> factorial(200.);
375
```

```
0.78865786736479050355236321393218 10
```

My 1.2f routine (and also 1.2) outputs:

```
7.8865786736479050355236321393219e374
```

and this is the correct rounding because for 40 digits it computes

```
7.886578673647905035523632139321850622951e374
```

Maple's result (contrarily to `xint`) is thus not the correct rounding but still it is less than 0.6 ulp wrong.

```
2681 \def\XINT_FL_fac_vbig
2682   {\expandafter\XINT_FL_fac_vbigloop_a
2683     \the\numexpr \XINT_FL_fac_increaseP \xint_c_i   }%
2684 \def\XINT_FL_fac_big
2685   {\expandafter\XINT_FL_fac_bigloop_a
2686     \the\numexpr \XINT_FL_fac_increaseP \xint_c_ii  }%
2687 \def\XINT_FL_fac_med
2688   {\expandafter\XINT_FL_fac_medloop_a
2689     \the\numexpr \XINT_FL_fac_increaseP \xint_c_iii }%
2690 \def\XINT_FL_fac_small
2691   {\expandafter\XINT_FL_fac_smallloop_a
2692     \the\numexpr \XINT_FL_fac_increaseP \xint_c_iv   }%
2693 \def\XINT_FL_fac_increaseP #1#2.#3#4%
2694 {%
2695   #2\expandafter.\the\numexpr\xint_c_viii*%
2696   ((\xint_c_iv+#4+\expandafter\XINT_FL_fac_countdigits
2697     \the\numexpr #2/(#1*#3)\relax 87654321\Z)/\xint_c_viii).%
2698 }%
2699 \def\XINT_FL_fac_countdigits #1#2#3#4#5#6#7#8{\XINT_FL_fac_countdone }%
2700 \def\XINT_FL_fac_countdone  #1#2\Z {#1}%
2701 \def\XINT_FL_fac_out #1;![#2]#3%
2702   {#3{\romannumeral0\XINT_mul_out
2703     #1;!1\R!1\R!1\R!1\R!%
2704     1\R!1\R!1\R!1\R!\W [#2]}}%
2705 \def\XINT_FL_fac_vbigloop_a #1.#2.%
2706 {%
2707   \XINT_FL_fac_bigloop_a \xint_c_x^iv.#2.%
2708   {\expandafter\XINT_FL_fac_vbigloop_loop\the\numexpr 100010001\expandafter.%
2709     \the\numexpr \xint_c_x^viii+#1.}%
2710 }%
2711 \def\XINT_FL_fac_vbigloop_loop #1.#2.%
2712 {%
```

```

2713 \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2714 \expandafter\XINT_FL_fac_vbigloop_loop
2715 \the\numexpr #1+\xint_c_i\expandafter.%
2716 \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_mul #1!%
2717 }%
2718 \def\XINT_FL_fac_bigloop_a #1.%
2719 {%
2720 \expandafter\XINT_FL_fac_bigloop_b \the\numexpr
2721 #1+\xint_c_i-\xint_c_ii*((#1-464)/\xint_c_ii).#1.%
2722 }%
2723 \def\XINT_FL_fac_bigloop_b #1.#2.#3.%
2724 {%
2725 \expandafter\XINT_FL_fac_medloop_a
2726 \the\numexpr #1-\xint_c_i.#3.{\XINT_FL_fac_bigloop_loop #1.#2.}%
2727 }%
2728 \def\XINT_FL_fac_bigloop_loop #1.#2.%
2729 {%
2730 \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2731 \expandafter\XINT_FL_fac_bigloop_loop
2732 \the\numexpr #1+\xint_c_ii\expandafter.%
2733 \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_bigloop_mul #1!%
2734 }%
2735 \def\XINT_FL_fac_bigloop_mul #1!%
2736 {%
2737 \expandafter\XINT_FL_fac_mul
2738 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2739 }%
2740 \def\XINT_FL_fac_medloop_a #1.%
2741 {%
2742 \expandafter\XINT_FL_fac_medloop_b
2743 \the\numexpr #1+\xint_c_i-\xint_c_iii*((#1-100)/\xint_c_iii).#1.%
2744 }%
2745 \def\XINT_FL_fac_medloop_b #1.#2.#3.%
2746 {%
2747 \expandafter\XINT_FL_fac_smallloop_a
2748 \the\numexpr #1-\xint_c_i.#3.{\XINT_FL_fac_medloop_loop #1.#2.}%
2749 }%
2750 \def\XINT_FL_fac_medloop_loop #1.#2.%
2751 {%
2752 \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2753 \expandafter\XINT_FL_fac_medloop_loop
2754 \the\numexpr #1+\xint_c_iii\expandafter.%
2755 \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_medloop_mul #1!%
2756 }%
2757 \def\XINT_FL_fac_medloop_mul #1!%
2758 {%
2759 \expandafter\XINT_FL_fac_mul
2760 \the\numexpr
2761 \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2762 }%
2763 \def\XINT_FL_fac_smallloop_a #1.%
2764 {%

```



```

2765 \csname
2766 XINT_FL_fac_smallloop_\the\numexpr #1-\xint_c_iv*(#1/\xint_c_iv)\relax
2767 \endcsname #1.%
2768 }%
2769 \expandafter\def\csname XINT_FL_fac_smallloop_1\endcsname #1.#2.%
2770 {%
2771 \XINT_FL_fac_addzeros #2.100000001!.{2.#1.}{#2}%
2772 }%
2773 \expandafter\def\csname XINT_FL_fac_smallloop_-2\endcsname #1.#2.%
2774 {%
2775 \XINT_FL_fac_addzeros #2.100000002!.{3.#1.}{#2}%
2776 }%
2777 \expandafter\def\csname XINT_FL_fac_smallloop_-1\endcsname #1.#2.%
2778 {%
2779 \XINT_FL_fac_addzeros #2.100000006!.{4.#1.}{#2}%
2780 }%
2781 \expandafter\def\csname XINT_FL_fac_smallloop_0\endcsname #1.#2.%
2782 {%
2783 \XINT_FL_fac_addzeros #2.100000024!.{5.#1.}{#2}%
2784 }%
2785 \def\XINT_FL_fac_addzeros #1.%
2786 {%
2787 \ifnum #1=\xint_c_viii \expandafter\XINT_FL_fac_addzeros_exit\fi
2788 \expandafter\XINT_FL_fac_addzeros
2789 \the\numexpr #1-\xint_c_viii.100000000!%
2790 }%

```

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.

```

2791 \def\XINT_FL_fac_addzeros_exit #1.#2.#3#4{\XINT_FL_fac_smallloop_loop #3#21;![-#4]}%
2792 \def\XINT_FL_fac_smallloop_loop #1.#2.%
2793 {%
2794 \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2795 \expandafter\XINT_FL_fac_smallloop_loop
2796 \the\numexpr #1+\xint_c_iv\expandafter.%
2797 \the\numexpr #2\expandafter.\romannumeral0\XINT_FL_fac_smallloop_mul #1!%
2798 }%
2799 \def\XINT_FL_fac_smallloop_mul #1!%
2800 {%
2801 \expandafter\XINT_FL_fac_mul
2802 \the\numexpr
2803 \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2804 }%[
2805 \def\XINT_FL_fac_loop_exit #1!#2]#3{#3#2]}%
2806 \def\XINT_FL_fac_mul 1#1!%
2807 {\expandafter\XINT_FL_fac_mul_a\the\numexpr\XINT_FL_fac_smallmul 10!{#1]}%
2808 \def\XINT_FL_fac_mul_a #1-#2%
2809 {%
2810 \if#21\xint_afterfi{\expandafter\space\xint_gob_til_exclam}\else
2811 \expandafter\space\fi #11;!%
2812 }%

```

```

2813 \def\XINT_FL_fac_minimulwc_a #1#2#3#4#5!#6#7#8#9%
2814 {%
2815   \XINT_FL_fac_minimulwc_b {#1#2#3#4}{#5}{#6#7#8#9}%
2816 }%
2817 \def\XINT_FL_fac_minimulwc_b #1#2#3#4!#5%
2818 {%
2819   \expandafter\XINT_FL_fac_minimulwc_c
2820   \the\numexpr \xint_c_x^ix+#5+#2*#4!{{#1}{#2}{#3}{#4}}%
2821 }%
2822 \def\XINT_FL_fac_minimulwc_c 1#1#2#3#4#5#6!#7%
2823 {%
2824   \expandafter\XINT_FL_fac_minimulwc_d {#1#2#3#4#5}#7{#6}%
2825 }%
2826 \def\XINT_FL_fac_minimulwc_d #1#2#3#4#5%
2827 {%
2828   \expandafter\XINT_FL_fac_minimulwc_e
2829   \the\numexpr \xint_c_x^ix+#1+#2*#5+#3*#4!{#2}{#4}%
2830 }%
2831 \def\XINT_FL_fac_minimulwc_e 1#1#2#3#4#5#6!#7#8#9%
2832 {%
2833   1#6#9\expandafter!%
2834   \the\numexpr\expandafter\XINT_FL_fac_smallmul
2835   \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#7*#8!%
2836 }%
2837 \def\XINT_FL_fac_smallmul 1#1!#21#3!%
2838 {%
2839   \xint_gob_til_sc #3\XINT_FL_fac_smallmul_end;%
2840   \XINT_FL_fac_minimulwc_a #2!#3!{#1}{#2}%
2841 }%

```

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.)

```

2842 \def\XINT_FL_fac_smallmul_end;\XINT_FL_fac_minimulwc_a #1!;!#2#3[#4]%
2843 {%
2844   \ifnum #2=\xint_c_
2845     \expandafter\xint_firstoftwo\else
2846     \expandafter\xint_secondoftwo
2847   \fi
2848   {-2\relax[#4]}%
2849   {1#2\expandafter!\expandafter-\expandafter1\expandafter
2850     [\the\numexpr #4+\xint_c_viii]}%
2851 }%

```

## 8.77 `\xintFloatPFactorial`, `\XINTinFloatPFactorial`

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

1.2h (2016/11/20) now avoids raising `\xintError:OutOfRangePFac` if the condition  $0 \leq a \leq b < 10^8$  is violated. Same as for `\xintiPFactorial`.

```

2852 \def\xintFloatPFfactorial {\romannumeral0\xintfloatpfactorial}%
2853 \def\xintfloatpfactorial #1{\XINT_flpfac_chkopt \xintfloat #1\xint:}%
2854 \def\XINTinFloatPFfactorial {\romannumeral0\XINTinfloatpfactorial }%
2855 \def\XINTinfloatpfactorial #1{\XINT_flpfac_chkopt \XINTinfloat #1\xint:}%
2856 \def\XINT_flpfac_chkopt #1#2%
2857 {%
2858   \ifx [#2\expandafter\XINT_flpfac_opt
2859     \else\expandafter\XINT_flpfac_noopt
2860   \fi
2861   #1#2%
2862 }%
2863 \def\XINT_flpfac_noopt #1#2\xint:#3%
2864 {%
2865   \expandafter\XINT_FL_pfac_fork
2866   \the\numexpr \xintNum{#2}\expandafter.%
2867   \the\numexpr \xintNum{#3}.\xint_c_i{\XINTdigits}{#1[\XINTdigits]]}%
2868 }%
2869 \def\XINT_flpfac_opt #1[\xint:#2]%
2870 {%
2871   \expandafter\XINT_flpfac_opt_b\the\numexpr #2.#1%
2872 }%
2873 \def\XINT_flpfac_opt_b #1.#2#3#4%
2874 {%
2875   \expandafter\XINT_FL_pfac_fork
2876   \the\numexpr \xintNum{#3}\expandafter.%
2877   \the\numexpr \xintNum{#4}.\xint_c_i{#1}{#2[#1]}%
2878 }%
2879 \def\XINT_FL_pfac_fork #1#2.#3#4.%
2880 {%
2881   \unless\ifnum #1#2<#3#4 \xint_dothis\XINT_FL_pfac_one\fi
2882   \if-#3\xint_dothis\XINT_FL_pfac_neg \fi
2883   \if-#1\xint_dothis\XINT_FL_pfac_zero\fi
2884   \ifnum #3#4>\xint_c_x^viii_mone\xint_dothis\XINT_FL_pfac_outofrange\fi
2885   \xint_orthat \XINT_FL_pfac_increaseP #1#2.#3#4.%
2886 }%
2887 \def\XINT_FL_pfac_outofrange #1.#2.#3#4#5%
2888 {%
2889   #5{\XINT_signalcondition{InvalidOperation}}
2890   {pfactorial second arg too big: 9999999 < #2}{\0[0]}}%
2891 }%
2892 \def\XINT_FL_pfac_one #1.#2.#3#4#5{#5{1[0]}}%
2893 \def\XINT_FL_pfac_zero #1.#2.#3#4#5{#5{0[0]}}%
2894 \def\XINT_FL_pfac_neg -#1.-#2.%
2895 {%
2896   \ifnum #1>\xint_c_x^viii\xint_dothis\XINT_FL_pfac_outofrange\fi
2897   \xint_orthat {%
2898     \ifodd\numexpr#2-#1\relax\xint_afterfi{\expandafter-\romannumeral`&&@}\fi
2899     \expandafter\XINT_FL_pfac_increaseP}%
2900   \the\numexpr #2-\xint_c_i\expandafter.\the\numexpr#1-\xint_c_i.%
2901 }%

```

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 `\xintPFfactorial` 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.

```

2902 \def\XINT_FL_pfac_increaseP #1.#2.#3#4%
2903 {%
2904   \expandafter\XINT_FL_pfac_a
2905   \the\numexpr \xint_c_viii*((\xint_c_iv+#4+\expandafter
2906     \XINT_FL_fac_countdigits\the\numexpr (#2-#1-\xint_c_i)%
2907     /\ifnum #2>\xint_c_x^iv #3\else(#3*\xint_c_ii)\fi\relax
2908     87654321\Z)/\xint_c_viii).#1.#2.%
2909 }%
2910 \def\XINT_FL_pfac_a #1.#2.#3.%
2911 {%
2912   \expandafter\XINT_FL_pfac_b\the\numexpr \xint_c_i+#2\expandafter.%
2913   \the\numexpr#3\expandafter.%
2914   \romannumeral0\XINT_FL_pfac_addzeroes #1.100000001!1;![-#1]%
2915 }%
2916 \def\XINT_FL_pfac_addzeroes #1.%
2917 {%
2918   \ifnum #1=\xint_c_viii \expandafter\XINT_FL_pfac_addzeroes_exit\fi
2919   \expandafter\XINT_FL_pfac_addzeroes\the\numexpr #1-\xint_c_viii.100000000!%
2920 }%
2921 \def\XINT_FL_pfac_addzeroes_exit #1.{ }%
2922 \def\XINT_FL_pfac_b #1.%
2923 {%
2924   \ifnum #1>9999 \xint_dothis\XINT_FL_pfac_vbigloop \fi
2925   \ifnum #1>463 \xint_dothis\XINT_FL_pfac_bigloop \fi
2926   \ifnum #1>98 \xint_dothis\XINT_FL_pfac_medloop \fi
2927   \xint_orthat\XINT_FL_pfac_smallloop #1.%
2928 }%
2929 \def\XINT_FL_pfac_smallloop #1.#2.%
2930 {%
2931   \ifcase\numexpr #2-#1\relax
2932     \expandafter\XINT_FL_pfac_end_
2933   \or \expandafter\XINT_FL_pfac_end_i
2934   \or \expandafter\XINT_FL_pfac_end_ii
2935   \or \expandafter\XINT_FL_pfac_end_iii
2936   \else\expandafter\XINT_FL_pfac_smallloop_a
2937   \fi #1.#2.%
2938 }%
2939 \def\XINT_FL_pfac_smallloop_a #1.#2.%
2940 {%
2941   \expandafter\XINT_FL_pfac_smallloop_b
2942   \the\numexpr #1+\xint_c_iv\expandafter.%
2943   \the\numexpr #2\expandafter.%
2944   \romannumeral0\expandafter\XINT_FL_fac_mul
2945   \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2946 }%
2947 \def\XINT_FL_pfac_smallloop_b #1.%
2948 {%

```

```

2949 \ifnum #1>98 \expandafter\XINT_FL_pfac_medloop \else
2950 \expandafter\XINT_FL_pfac_smallloop \fi #1.%
2951 }%
2952 \def\XINT_FL_pfac_medloop #1.#2.%
2953 {%
2954 \ifcase\numexpr #2-#1\relax
2955 \expandafter\XINT_FL_pfac_end_
2956 \or \expandafter\XINT_FL_pfac_end_i
2957 \or \expandafter\XINT_FL_pfac_end_ii
2958 \else\expandafter\XINT_FL_pfac_medloop_a
2959 \fi #1.#2.%
2960 }%
2961 \def\XINT_FL_pfac_medloop_a #1.#2.%
2962 {%
2963 \expandafter\XINT_FL_pfac_medloop_b
2964 \the\numexpr #1+\xint_c_iii\expandafter.%
2965 \the\numexpr #2\expandafter.%
2966 \romannumeral0\expandafter\XINT_FL_fac_mul
2967 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2968 }%
2969 \def\XINT_FL_pfac_medloop_b #1.%
2970 {%
2971 \ifnum #1>463 \expandafter\XINT_FL_pfac_bigloop \else
2972 \expandafter\XINT_FL_pfac_medloop \fi #1.%
2973 }%
2974 \def\XINT_FL_pfac_bigloop #1.#2.%
2975 {%
2976 \ifcase\numexpr #2-#1\relax
2977 \expandafter\XINT_FL_pfac_end_
2978 \or \expandafter\XINT_FL_pfac_end_i
2979 \else\expandafter\XINT_FL_pfac_bigloop_a
2980 \fi #1.#2.%
2981 }%
2982 \def\XINT_FL_pfac_bigloop_a #1.#2.%
2983 {%
2984 \expandafter\XINT_FL_pfac_bigloop_b
2985 \the\numexpr #1+\xint_c_ii\expandafter.%
2986 \the\numexpr #2\expandafter.%
2987 \romannumeral0\expandafter\XINT_FL_fac_mul
2988 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2989 }%
2990 \def\XINT_FL_pfac_bigloop_b #1.%
2991 {%
2992 \ifnum #1>9999 \expandafter\XINT_FL_pfac_vbigloop \else
2993 \expandafter\XINT_FL_pfac_bigloop \fi #1.%
2994 }%
2995 \def\XINT_FL_pfac_vbigloop #1.#2.%
2996 {%
2997 \ifnum #2=#1
2998 \expandafter\XINT_FL_pfac_end_
2999 \else\expandafter\XINT_FL_pfac_vbigloop_a
3000 \fi #1.#2.%

```

```

3001 }%
3002 \def\XINT_FL_pfac_vbigloop_a #1.#2.%
3003 {%
3004     \expandafter\XINT_FL_pfac_vbigloop
3005     \the\numexpr #1+\xint_c_i\expandafter.%
3006     \the\numexpr #2\expandafter.%
3007     \romannumeral0\expandafter\XINT_FL_fac_mul
3008     \the\numexpr\xint_c_x^viii+#1!%
3009 }%
3010 \def\XINT_FL_pfac_end_iii #1.#2.%
3011 {%
3012     \expandafter\XINT_FL_fac_out
3013     \romannumeral0\expandafter\XINT_FL_fac_mul
3014     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
3015 }%
3016 \def\XINT_FL_pfac_end_ii #1.#2.%
3017 {%
3018     \expandafter\XINT_FL_fac_out
3019     \romannumeral0\expandafter\XINT_FL_fac_mul
3020     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
3021 }%
3022 \def\XINT_FL_pfac_end_i #1.#2.%
3023 {%
3024     \expandafter\XINT_FL_fac_out
3025     \romannumeral0\expandafter\XINT_FL_fac_mul
3026     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
3027 }%
3028 \def\XINT_FL_pfac_end_ #1.#2.%
3029 {%
3030     \expandafter\XINT_FL_fac_out
3031     \romannumeral0\expandafter\XINT_FL_fac_mul
3032     \the\numexpr \xint_c_x^viii+#1!%
3033 }%

```

## 8.78 \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  $4 \cdot 10^{-2}$ , 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.

```

3034 \def\xintFloatBinomial    {\romannumeral0\xintfloatbinomial}%
3035 \def\xintfloatbinomial    #1{\XINT_flbinom_chkopt \xintfloat #1\xint:}%
3036 \def\XINTinFloatBinomial  {\romannumeral0\XINTinfloatbinomial }%
3037 \def\XINTinfloatbinomial #1{\XINT_flbinom_chkopt \XINTinfloat #1\xint:}%
3038 \def\XINT_flbinom_chkopt #1#2%
3039 {%
3040     \ifx [#2\expandafter\XINT_flbinom_opt
3041         \else\expandafter\XINT_flbinom_noopt

```

```

3042     \fi #1#2%
3043 }%
3044 \def\XINT_flbinom_noopt #1#2\xint:#3%
3045 {%
3046     \expandafter\XINT_FL_binom_a
3047     \the\numexpr\xintNum{#2}\expandafter.\the\numexpr\xintNum{#3}.\XINTdigits.#1%
3048 }%
3049 \def\XINT_flbinom_opt #1[\xint:#2]#3#4%
3050 {%
3051     \expandafter\XINT_FL_binom_a
3052     \the\numexpr\xintNum{#3}\expandafter.\the\numexpr\xintNum{#4}\expandafter.%
3053     \the\numexpr #2.#1%
3054 }%
3055 \def\XINT_FL_binom_a #1.#2.%
3056 {%
3057     \expandafter\XINT_FL_binom_fork \the\numexpr #1-#2.#2.#1.%
3058 }%
3059 \def\XINT_FL_binom_fork #1#2.#3#4.#5#6.%
3060 {%
3061     \if-#5\xint_dothis \XINT_FL_binom_neg\fi
3062     \if-#1\xint_dothis \XINT_FL_binom_zero\fi
3063     \if-#3\xint_dothis \XINT_FL_binom_zero\fi
3064     \if0#1\xint_dothis \XINT_FL_binom_one\fi
3065     \if0#3\xint_dothis \XINT_FL_binom_one\fi
3066     \ifnum #5#6>\xint_c_x^viii_mone \xint_dothis\XINT_FL_binom_toobig\fi
3067     \ifnum #1#2>#3#4 \xint_dothis\XINT_FL_binom_ab \fi
3068         \xint_orthat\XINT_FL_binom_aa
3069     #1#2.#3#4.#5#6.%
3070 }%
3071 \def\XINT_FL_binom_neg #1.#2.#3.#4.#5%
3072 {%
3073     #5[#4]{\XINT_signalcondition{InvalidOperation}
3074         {binomial with first arg negative: #3}}{0[0]}}%
3075 }%
3076 \def\XINT_FL_binom_toobig #1.#2.#3.#4.#5%
3077 {%
3078     #5[#4]{\XINT_signalcondition{InvalidOperation}
3079         {binomial with first arg too big: 9999999 < #3}}{0[0]}}%
3080 }%
3081 \def\XINT_FL_binom_one #1.#2.#3.#4.#5{#5[#4]{1[0]}}%
3082 \def\XINT_FL_binom_zero #1.#2.#3.#4.#5{#5[#4]{0[0]}}%
3083 \def\XINT_FL_binom_aa #1.#2.#3.#4.#5%
3084 {%
3085     #5[#4]{\xintDiv{\XINT_FL_pfac_increaseP
3086         #2.#3.\xint_c_iv{#4+\xint_c_i}{\XINTinfloat[#4+\xint_c_iii]}}%
3087         {\XINT_FL_fac_fork_b
3088         #1.\xint_c_iv{#4+\xint_c_i}\XINT_FL_fac_out{\XINTinfloat[#4+\xint_c_iii]}}}%
3089 }%
3090 \def\XINT_FL_binom_ab #1.#2.#3.#4.#5%
3091 {%
3092     #5[#4]{\xintDiv{\XINT_FL_pfac_increaseP
3093         #1.#3.\xint_c_iv{#4+\xint_c_i}{\XINTinfloat[#4+\xint_c_iii]}}%

```

```

3094     {\XINT_FL_fac_fork_b
3095     #2.\xint_c_iv{#4+\xint_c_i}\XINT_FL_fac_out{\XINTinfloat[#4+\xint_c_iii]}}}%
3096 }%
```

## 8.79 \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.

```

3097 \def\xintFloatSqrt      {\romannumeral0\xintfloatsqrt }%
3098 \def\xintfloatsqrt     #1{\XINT_flsqrt_chkopt \xintfloat #1\xint:}%
3099 \def\XINTinFloatSqrt   {\romannumeral0\XINTinfloatsqrt }%
3100 \def\XINTinfloatsqrt  #1{\XINT_flsqrt_chkopt \XINTinfloat #1\xint:}%
3101 \def\XINT_flsqrt_chkopt #1#2%
3102 {%
3103   \ifx [#2\expandafter\XINT_flsqrt_opt
3104     \else\expandafter\XINT_flsqrt_noopt
3105   \fi #1#2%
3106 }%
3107 \def\XINT_flsqrt_noopt #1#2\xint:%
3108 {%
3109   \expandafter\XINT_FL_sqrt_a
3110     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.#1%
3111 }%
3112 \def\XINT_flsqrt_opt #1[\xint:#2]#3%
3113 {%
3114   \expandafter\XINT_flsqrt_opt_a\the\numexpr #2.#1%
3115 }%
3116 \def\XINT_flsqrt_opt_a #1.#2#3%
3117 {%
3118   \expandafter\XINT_FL_sqrt_a\romannumeral0\XINTinfloat[#1]{#3}#1.#2%
3119 }%
3120 \def\XINT_FL_sqrt_a #1%
3121 {%
3122   \xint_UDzerominusfork
3123   #1-\XINT_FL_sqrt_iszero
3124   0#1\XINT_FL_sqrt_isneg
3125   0-{\XINT_FL_sqrt_pos #1}%
3126   \krof
3127 }%[
3128 \def\XINT_FL_sqrt_iszero #1]#2.#3{#3[#2]{0[0]}}%
3129 \def\XINT_FL_sqrt_isneg #1]#2.#3%
3130 }%
```



```

3131 #3[#2]{\XINT_signalcondition{InvalidOperation}
3132           {Square root of negative: -#1}}{\{0[0]}}%
3133 }%

3134 \def\XINT_FL_sqrt_pos #1[#2]#3.%
3135 {%
3136   \expandafter\XINT_flsqrt
3137   \the\numexpr #3\ifodd #2 \xint_dothis {+\xint_c_iii.(#2+\xint_c_i).0}\fi
3138   \xint_orthat {+\xint_c_ii.#2.}\}#100.#3.%
3139 }%

3140 \def\XINT_flsqrt #1.#2.%
3141 {%
3142   \expandafter\XINT_flsqrt_a
3143   \the\numexpr #2/\xint_c_ii-(#1-\xint_c_i)/\xint_c_ii.#1.%
3144 }%

3145 \def\XINT_flsqrt_a #1.#2.#3#4.#5.%
3146 {%
3147   \expandafter\XINT_flsqrt_b
3148   \the\numexpr (#2-\xint_c_i)/\xint_c_ii\expandafter.%
3149   \romannumeral0\XINT_sqrt_start #2.#4#3.#5.#2.#4#3.#5.#1.%
3150 }%

3151 \def\XINT_flsqrt_b #1.#2#3%
3152 {%
3153   \expandafter\XINT_flsqrt_c
3154   \romannumeral0\xintiisub
3155   {\XINT_dsx_addzeros {#1}#2;}%
3156   {\xintiiDivRound{\XINT_dsx_addzeros {#1}#3;}%
3157   {\XINT_db1#2\xint_bye2345678\xint_bye*\xint_c_ii\relax}}.%
3158 }%

3159 \def\XINT_flsqrt_c #1.#2.%
3160 {%
3161   \expandafter\XINT_flsqrt_d
3162   \romannumeral0\XINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..%
3163 }%

3164 \def\XINT_flsqrt_d #1.#2#3.%
3165 {%
3166   \ifnum #2=\xint_c_v
3167   \expandafter\XINT_flsqrt_f\else\expandafter\XINT_flsqrt_finish\fi
3168   #2#3.#1.%
3169 }%

3170 \def\XINT_flsqrt_finish #1#2.#3.#4.#5.#6.#7.#8{#8[#6]{#3#1[#7]}}%

```

```

3171 \def\XINT_flsqrt_f 5#1.%
3172   {\expandafter\XINT_flsqrt_g\romannumeral0\xintinum{#1}\relax.}%
3173 \def\XINT_flsqrt_g #1#2#3.{\if\relax#2\xint_dothis{\XINT_flsqrt_h #1}\fi
3174   \xint_orthat{\XINT_flsqrt_finish 5.}}%
3175 \def\XINT_flsqrt_h #1{\ifnum #1<\xint_c_iii\xint_dothis{\XINT_flsqrt_again}\fi
3176   \xint_orthat{\XINT_flsqrt_finish 5.}}%

3177 \def\XINT_flsqrt_again #1.#2.%
3178 {%
3179   \expandafter\XINT_flsqrt_again_a\the\numexpr #2+\xint_c_viii.%
3180 }%

3181 \def\XINT_flsqrt_again_a #1.#2.#3.%
3182 {%
3183   \expandafter\XINT_flsqrt_b
3184   \the\numexpr (#1-\xint_c_i)/\xint_c_ii\expandafter.%
3185   \romannumeral0\XINT_sqrt_start #1.#200000000.#3.%
3186   #1.#200000000.#3.%
3187 }%

```

## 8.80 \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].

```

3188 \def\xintFloatE   {\romannumeral0\xintfloate }%
3189 \def\xintfloate #1{\XINT_floate_chkopt #1\xint:}%
3190 \def\XINT_floate_chkopt #1%
3191 {%
3192   \ifx [#1\expandafter\XINT_floate_opt
3193     \else\expandafter\XINT_floate_noopt
3194   \fi #1%
3195 }%
3196 \def\XINT_floate_noopt #1\xint:
3197 {%
3198   \expandafter\XINT_floate_post
3199   \romannumeral0\XINTinfloat[\XINTdigits]{#1}\XINTdigits.%
3200 }%
3201 \def\XINT_floate_opt [\xint:#1]%
3202 {%
3203   \expandafter\XINT_floate_opt_a\the\numexpr #1.%
3204 }%
3205 \def\XINT_floate_opt_a #1.#2%
3206 {%
3207   \expandafter\XINT_floate_post
3208   \romannumeral0\XINTinfloat[#1]{#2}#1.%
3209 }%
3210 \def\XINT_floate_post #1%
3211 {%
3212   \xint_UDzerominusfork

```

```

3213      #1-\XINT_floate_zero
3214      0#1\XINT_floate_neg
3215      0-\XINT_floate_pos
3216      \krof #1%
3217 }%[
3218 \def\XINT_floate_zero #1]#2.#3{ 0.e0}%
3219 \def\XINT_floate_neg-{\expandafter-\romannumeral0\XINT_floate_pos}%

3220 \def\XINT_floate_pos #1#2[#3]#4.#5%
3221 {%
3222     \expandafter\XINT_float_pos_done\the\numexpr#3+#4+#5-\xint_c_i.#1.#2;%
3223 }%
3224 \def\XINTinFloatE {\romannumeral0\XINTinfloate }%
3225 \def\XINTinfloate
3226     {\expandafter\XINT_infloate\romannumeral0\XINTinfloat[\XINTdigits]}%
3227 \def\XINT_infloate #1[#2]#3%
3228     {\expandafter\XINT_infloate_end\the\numexpr #3+#2.{#1}}%
3229 \def\XINT_infloate_end #1.#2{ #2[#1]}%

```

## 8.81 \XINTinFloatMod

1.1. Pour emploi dans xintexpr. Code shortened at 1.2p.

```

3230 \def\XINTinFloatMod {\romannumeral0\XINTinfloatmod [\XINTdigits]}%
3231 \def\XINTinfloatmod [#1]#2#3%
3232 {%
3233     \XINTinfloat[#1]{\xintMod
3234         {\romannumeral0\XINTinfloat[#1]{#2}}%
3235         {\romannumeral0\XINTinfloat[#1]{#3}}}%
3236 }%

```

## 8.82 \XINTinFloatDivFloor

1.2p. Formerly // and /: in \xintfloatexpr used \xintDivFloor and \xintMod, hence did not round their operands to float precision beforehand.

```

3237 \def\XINTinFloatDivFloor {\romannumeral0\XINTinfloatdivfloor [\XINTdigits]}%
3238 \def\XINTinfloatdivfloor [#1]#2#3%
3239 {%
3240     \xintdivfloor
3241         {\romannumeral0\XINTinfloat[#1]{#2}}%
3242         {\romannumeral0\XINTinfloat[#1]{#3}}%
3243 }%

```

## 8.83 \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 \csname.

Hésitation sur le quotient, faut-il l'arrondir immédiatement ? Finalement non, le produire comme un integer.

```

3244 \def\XINTinFloatDivMod {\romannumeral0\XINTinfloatdivmod [\XINTdigits]}%

```

```

3245 \def\XINTinfloatdivmod [#1]#2#3%
3246 {%
3247   \expandafter\XINT_infloatdivmod
3248   \romannumeral0\xintdivmod
3249     {\romannumeral0\XINTinfloat[#1]{#2}}%
3250     {\romannumeral0\XINTinfloat[#1]{#3}}%
3251   {#1}%
3252 }%
3253 \def\XINT_infloatdivmod #1#2#3{ #1,\XINTinFloat[#3]{#2}}%

```

## 8.84 \xintifFloatInt

1.3a for `ifint()` function in `\xintfloatexpr`.

```

3254 \def\xintifFloatInt {\romannumeral0\xintiffloatint}%
3255 \def\xintiffloatint #1{\expandafter\XINT_iffloatint
3256   \romannumeral0\xintrez{\XINTinFloat[\XINTdigits]{#1}}}%
3257 \def\XINT_iffloatint #1#2/1[#3]%
3258 {%
3259   \if 0#1\xint_dothis\xint_stop_atfirstoftwo\fi
3260   \ifnum#3<\xint_c_\xint_dothis\xint_stop_atsecondoftwo\fi
3261   \xint_orthat\xint_stop_atfirstoftwo
3262 }%

```

## 8.85 \xintFloatIsInt

1.3d for `isint()` function in `\xintfloatexpr`.

```

3263 \def\xintFloatIsInt {\romannumeral0\xintfloatisint}%
3264 \def\xintfloatisint #1{\expandafter\XINT_iffloatint
3265   \romannumeral0\xintrez{\XINTinFloat[\XINTdigits]{#1}}10}%

```

## 8.86 (WIP) \XINTinRandomFloatS, \XINTinRandomFloatSdigits

1.3b. Support for `random()` function.

Thus as it is a priori only for `xintexpr` usage, it expands inside `\csname` context, but as we need to get rid of initial zeros we use `\xintRandomDigits` not `\xintXRandomDigits` (`\expanded` would have a use case here).

And anyway as we want to be able to use `random()` in `\xintdeffunc/\xintNewExpr`, it is good to have `f`-expandable macros, so we add the small overhead to make it `f`-expandable.

We don't have to be very efficient in removing leading zeroes, as there is only 10% chance for each successive one. Besides we use (current) internal storage format of the type `A[N]`, where `A` is not required to be with `\xintDigits` digits, so `N` will simply be `-\xintDigits` and needs no adjustment.

In case we use in future with `#1` something else than `\xintDigits` we do the `0-(#1)` construct.

I had some qualms about doing a random float like this which means that when there are leading zeros in the random digits the (virtual) mantissa ends up with trailing zeros. That did not feel right but I checked `random()` in Python (which of course uses radix 2), and indeed this is what happens there.

```

3266 \def\XINTinRandomFloatS{\romannumeral0\XINTinrandomfloatS}%
3267 \def\XINTinRandomFloatSdigits{\XINTinRandomFloatS[\XINTdigits]}%
3268 \def\XINTinrandomfloatS[#1]%

```

```

3269 {%
3270   \expandafter\XINT_inrandomfloatS\the\numexpr\xint_c_-(#1)\xint:
3271 }%
3272 \def\XINT_inrandomfloatS-#1\xint:
3273 {%
3274   \expandafter\XINT_inrandomfloatS_a
3275   \romannumeral0\xintrandomdigits{#1}[-#1]%
3276 }%

```

We add one macro to handle a tiny bit faster 90of cases, after all we also use one extra macro for the completely improbable all 0 case.

```

3277 \def\XINT_inrandomfloatS_a#1%
3278 {%
3279   \if#10\xint_dothis{\XINT_inrandomfloatS_b}\fi
3280   \xint_orthat{ #1}%
3281 }%[
3282 \def\XINT_inrandomfloatS_b#1%
3283 {%
3284   \if#1[\xint_dothis{\XINT_inrandomfloatS_zero}\fi% ]
3285   \if#10\xint_dothis{\XINT_inrandomfloatS_b}\fi
3286   \xint_orthat{ #1}%
3287 }%[
3288 \def\XINT_inrandomfloatS_zero#1]{ 0[0]}%

```

## 8.87 (WIP) \XINTinRandomFloatSixteen

### 1.3b. Support for grand() function.

```

3289 \def\XINTinRandomFloatSixteen%
3290 {%
3291   \romannumeral0\expandafter\XINT_inrandomfloatS_a
3292   \romannumeral`&&\expandafter\XINT_eightrandomdigits
3293   \romannumeral`&&\XINT_eightrandomdigits[-16]%
3294 }%
3295 \XINT_restorecatcodes_endinput%

```

## 9 Package [xintseries](#) implementation

.1	Catcodes, $\varepsilon$ -TeX and reload detection . . .	262	.7	<code>\xintRationalSeries</code> . . . . .	265
.2	Package identification . . . . .	263	.8	<code>\xintRationalSeriesX</code> . . . . .	266
.3	<code>\xintSeries</code> . . . . .	263	.9	<code>\xintFxpPtPowerSeries</code> . . . . .	267
.4	<code>\xintiSeries</code> . . . . .	263	.10	<code>\xintFxpPtPowerSeriesX</code> . . . . .	268
.5	<code>\xintPowerSeries</code> . . . . .	264	.11	<code>\xintFloatPowerSeries</code> . . . . .	268
.6	<code>\xintPowerSeriesX</code> . . . . .	265	.12	<code>\xintFloatPowerSeriesX</code> . . . . .	270

The commenting is currently (2019/04/05) very sparse.

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

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

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

```

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

```

```

38     \aftergroup\endinput % xintseries already loaded.
39     \fi
40     \fi
41     \fi
42 \z%
43 \XINTsetupcatcodes% defined in xintkernel.sty

```

## 9.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintseries}%
46 [2019/04/05 1.3e Expandable partial sums with xint package (JFB)]%

```

## 9.3 \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 }%

```

## 9.4 \xintiSeries

```

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

```

```

84     \xint_afterfi {\XINT_iserie_loop {#1}{0}{#2}{#3}}%
85   \fi
86 }%
87 \def\XINT_iserie_loop #1#2#3#4%
88 {%
89   \ifnum #3>#1 \else \XINT_iserie_exit \fi
90   \expandafter\XINT_iserie_loop\expandafter
91   {\the\numexpr #1+1\expandafter }\expandafter
92   {\romannumeral0\xintiiadd {#2}{#4{#1}}}%
93   {#3}{#4}%
94 }%
95 \def\XINT_iserie_exit \fi #1#2#3#4#5#6#7#8%
96 {%
97   \fi\xint_gobble_ii #6%
98 }%

```

## 9.5 \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 }%

```

## 9.6 \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 }%

```

## 9.7 \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 }%

```

## 9.8 \xintRationalSeriesX

*a,b,initial,ratiofunction,x*

This computes  $F(a,x)+\dots+F(b,x)$  on the basis of the value of  $F(a,x)$  and the ratios  $F(n,x)/F(n-1,x)$ . The argument  $x$  is first expanded and it is the value resulting from this which is used then throughout. The initial term  $F(a,x)$  must be defined as one-parameter macro which will be given  $x$ . Modified in 1.06 to give the indices first to a `\numexpr` rather than expanding twice. I just use `\the\numexpr` and maintain the previous code after that. 1.08a adds the forgotten optimization following that previous change.

```

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

```

```

204 \fi
205 }%
206 \def\XINT_ratseriesx_pre #1#2#3#4#5%
207 {%
208 \XINT_ratseries_loop {#2}{1}{#3}{#4{#1}}{#5{#1}}%
209 }%

```

## 9.9 \xintFxpPowerSeries

I am not two happy with this piece of code. Will make it more economical another day. Modified in 1.06 to give the indices first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that. 1.08a: forgot last time some optimization from the change to \numexpr.

```

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

```

```

249   {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
250   {\romannumeral0\xintiadd {#4}{\xintiTrunc {#7}{\xintMul {#6{#2}}{#1}}}%
251   {#1}{#3}{#5}{#6}{#7}%
252 }%
253 \def\xINT_fppowseries_exit_i\fi\expandafter\xINT_fppowseries_loop_ii
254   {\fi \expandafter\xINT_fppowseries_exit_ii }%
255 \def\xINT_fppowseries_exit_ii #1#2#3#4#5#6#7%
256 {%
257   \xinttrunc {#7}
258   {\xintiadd {#4}{\xintiTrunc {#7}{\xintMul {#6{#2}}{#1}}[-#7]}%
259 }%

```

## 9.10 \xintFxpPowerSeriesX

*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\xintFxpPowerSeriesX {\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 }%

```

## 9.11 \xintFloatPowerSeries

1.08a. I still have to re-visit \xintFxpPowerSeries; 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 }%

```

## 9.12 \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%

```

## 10 Package xintcfrac implementation

<p>.1 Catcodes, <math>\varepsilon</math>-T<sub>E</sub>X and reload detection . . . 271</p> <p>.2 Package identification . . . . . 272</p> <p>.3 <code>\xintCFrac</code> . . . . . 272</p> <p>.4 <code>\xintGCFrac</code> . . . . . 273</p> <p>.5 <code>\xintGGCFrac</code> . . . . . 275</p> <p>.6 <code>\xintGctoGCx</code> . . . . . 276</p> <p>.7 <code>\xintFtoCs</code> . . . . . 276</p> <p>.8 <code>\xintFtoCx</code> . . . . . 277</p> <p>.9 <code>\xintFtoC</code> . . . . . 277</p> <p>.10 <code>\xintFGtoC</code> . . . . . 278</p> <p>.11 <code>\xintFGtoC</code> . . . . . 278</p> <p>.12 <code>\xintFtoCC</code> . . . . . 279</p> <p>.13 <code>\xintCtoF</code>, <code>\xintCstoF</code> . . . . . 280</p> <p>.14 <code>\xintiCstoF</code> . . . . . 281</p> <p>.15 <code>\xintGctoF</code> . . . . . 281</p>	<p>.16 <code>\xintiGctoF</code> . . . . . 283</p> <p>.17 <code>\xintCtoCv</code>, <code>\xintCstoCv</code> . . . . . 284</p> <p>.18 <code>\xintiCstoCv</code> . . . . . 285</p> <p>.19 <code>\xintGctoCv</code> . . . . . 285</p> <p>.20 <code>\xintiGctoCv</code> . . . . . 287</p> <p>.21 <code>\xintFtoCv</code> . . . . . 288</p> <p>.22 <code>\xintFtoCCv</code> . . . . . 288</p> <p>.23 <code>\xintCntoF</code> . . . . . 288</p> <p>.24 <code>\xintGcntoF</code> . . . . . 289</p> <p>.25 <code>\xintCntoCs</code> . . . . . 290</p> <p>.26 <code>\xintCntoGC</code> . . . . . 290</p> <p>.27 <code>\xintGcntoGC</code> . . . . . 291</p> <p>.28 <code>\xintCstoGC</code> . . . . . 292</p> <p>.29 <code>\xintGctoGC</code> . . . . . 292</p>
--	---

The commenting is currently (2019/04/05) 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`.

### 10.1 Catcodes, $\varepsilon$ -T<sub>E</sub>X and reload detection

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

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

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2 \catcode13=5 % ^^M
3 \endlinechar=13 %
4 \catcode123=1 % {
5 \catcode125=2 % }
6 \catcode64=11 % @
7 \catcode35=6 % #
8 \catcode44=12 % ,
9 \catcode45=12 % -
10 \catcode46=12 % .
11 \catcode58=12 % :
12 \let\z\endgroup
13 \expandafter\let\expandafter\x\csname ver@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 xintfrac.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 % xintfrac already loaded.
39   \fi
40 \fi
41 \fi
42 \z%
43 \XINTsetupcatcodes% defined in xintkernel.sty

```

## 10.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintfrac}%
46 [2019/04/05 1.3e Expandable continued fractions with xint package (JFB)]%

```

## 10.3 \xintCFrac

```

47 \def\xintCFrac {\romannumeral0\xintfrac }%
48 \def\xintfrac #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\xintraawithzeros {#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\xintraawithzeros {#1}\Z
68   \relax\hfill
69 }%
70 \def\XINT_cfrac_optc #1%
71 {%
72   \expandafter\XINT_cfrac_A\romannumeral0\xintraawithzeros {#1}\Z

```



```

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

10.4 \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}%

```

## 10.5 \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}%
```

## 10.6 \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}%
```

## 10.7 \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\xintraawwithzeros {#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}%

```

## 10.8 \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}%

```

## 10.9 \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 {}}%
```

## 10.10 \xintFtoGC

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

## 10.11 \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\xintraawithzeros {#1}\Z
322 }%
323 \def\XINT_fgtc_a #1/#2\Z #3%
324 {%
325   \expandafter\XINT_fgtc_b\romannumeral0\xintraawithzeros {#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 }%

```

## 10.12 \xintFtoCC

```

363 \def\xintFtoCC {\romannumeral0\xintftocc }%
364 \def\xintftocc #1%
365 {%
366   \expandafter\XINT_ftcc_A\expandafter {\romannumeral0\xintraewithzeros {#1}}%
367 }%
368 \def\XINT_ftcc_A #1%
369 {%
370   \expandafter\XINT_ftcc_B
371   \romannumeral0\xintraewithzeros {\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\xintraewithzeros {\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 }%

```

### 10.13 \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\xintraewithzeros {#5}.{#1}{#2}{#3}{#4}%
453 }%
454 \def\XINT_ctf_loop_b #1/#2.#3#4#5#6%
455 {%

```



```

456 \expandafter\XINT_ctf_loop_c\expandafter
457 {\romannumeral0\XINT_mul_fork #2\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]

```

#### 10.14 \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]

```

#### 10.15 \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{\xintrawithzeros {#2/#3}}% 1.09b removes [0]

```

## 10.16 \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{\xintrawwithzeros {#4/#5}}% 1.09b removes [0]

```

## 10.17 \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\xintrawwithzeros {#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\xintraawwithzeros {#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}%

```

### 10.18 \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\xintraawwithzeros {#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]

```

### 10.19 \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}%

```

## 10.20 \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\xintraewithzeros {#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\xintraewithzeros {#2/#3}}%
821 }%
822 \def\XINT_igctcv_end_b #1#2{ #2{#1}}% 1.09b removes [0]

```

## 10.21 \xintFtoCv

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

```

823 \def\xintFtoCv {\romannumeral0\xintftocv }%
824 \def\xintftocv #1%
825 {%
826   \xinticstocv {\xintFtoCs {#1}}%
827 }%

```

## 10.22 \xintFtoCCv

```

828 \def\xintFtoCCv {\romannumeral0\xintftoccv }%
829 \def\xintftoccv #1%
830 {%
831   \xintigctocv {\xintFtoCC {#1}}%
832 }%

```

## 10.23 \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 }%

```

## 10.24 \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 }%

```

## 10.25 \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

```

## 10.26 \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/{ }%

```

## 10.27 \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/{ }%

```

## 10.28 \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}%

```

## 10.29 \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}}/%

```

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, xinttrig, xintlog*

```
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%
```

## 11 Package xintexpr implementation

### Contents

11.1	Old comments . . . . .	295
11.2	Catcodes, $\varepsilon$ -TeX and reload detection . . . . .	296
11.3	Package identification . . . . .	297
11.4	<code>\xintexpr</code> , <code>\xintiexpr</code> , <code>\xintfloatexpr</code> , <code>\xintiieexpr</code> . . . . .	297
11.5	<code>\xintexprpro</code> , <code>\xintiiepro</code> . . . . .	297
11.6	<code>\xintiexprpro</code> , <code>\xintfloatexprpro</code> . . . . .	297
11.7	<code>\XINT_expr_wrap</code> , <code>\XINT_iiexpr_wrap</code> , <code>\XINT_flexpr_wrap</code> . . . . .	298
11.8	<code>\XINT_expr_usethe</code> , <code>\XINT_protectii</code> . . . . .	298
11.9	<code>\XINT_expr_print</code> , <code>\XINT_iiexpr_print</code> , <code>\XINT_flexpr_print</code> . . . . .	298
11.10	<code>\xinttheexpr</code> , <code>\xinttheiexpr</code> , <code>\xintthefloatexpr</code> , <code>\xinttheiieexpr</code> . . . . .	298
11.11	<code>\thexintexpr</code> , <code>\thexintiexpr</code> , <code>\thexintfloatexpr</code> , <code>\thexintiieexpr</code> . . . . .	299
11.12	<code>\xinteval</code> , <code>\xintieval</code> , <code>\xintfloateval</code> , <code>\xintiieval</code> . . . . .	299
11.13	<code>\xintthe</code> . . . . .	299
11.14	<code>\xintbareeval</code> , <code>\xintbarefloateval</code> , <code>\xintbareieval</code> . . . . .	299
11.15	<code>\xintthebareeval</code> , <code>\xintthebarefloateval</code> , <code>\xintthebareieval</code> . . . . .	299
11.16	<code>\xintboolexpr</code> , <code>\XINT_boolexpr_print</code> , <code>\xinttheboolexpr</code> , <code>\thexintboolexpr</code> . . . . .	299
11.17	<code>\xintifboolexpr</code> , <code>\xintifboolfloatexpr</code> , <code>\xintifbooliieexpr</code> . . . . .	300
11.18	<code>\xintifsgnexpr</code> , <code>\xintifsgnfloatexpr</code> , <code>\xintifsgniieexpr</code> . . . . .	300
11.19	<code>\xintthecoords</code> . . . . .	300
11.20	Locking and unlocking . . . . .	301
11.21	Hooks for the functioning of <code>\xintNewExpr</code> and <code>\xintdeffunc</code> . . . . .	301
11.22	Macros handling csv lists on output (for <code>\XINT_expr_print</code> et al. routines) . . . . .	301
11.23	<code>\XINT_expr_getnext</code> : fetching some number then an operator . . . . .	303
11.24	<code>\XINT_expr_scan_nbr_or_func</code> : the integer or decimal number or hexa-decimal number or function name or variable name or special hacky things big parser . . . . .	304
11.25	<code>\XINT_expr_getop</code> : finding the next operator or closing parenthesis or end of expression . . . . .	311
11.26	Expansion spanning; opening and closing parentheses . . . . .	313
11.27	<code> </code> , <code>  </code> , <code>&amp;</code> , <code>&amp;&amp;</code> , <code>&lt;</code> , <code>&gt;</code> , <code>=</code> , <code>==</code> , <code>&lt;=</code> , <code>&gt;=</code> , <code>!=</code> , <code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>^</code> , <code>**</code> , <code>//</code> , <code>/:</code> , <code>...</code> , <code>..[</code> , <code>]</code> , <code>]</code> , <code>[[:</code> , <code>:</code> , and <code>++</code> operators . . . . .	314
11.28	Macros for list selectors: <code>[list][N]</code> , <code>[list][:b]</code> , <code>[list][a:]</code> , <code>[list][a:b]</code> . . . . .	321
11.29	Macros for a..b list generation . . . . .	326
11.30	Macros for a..[d]..b list generation . . . . .	328
11.31	The comma as binary operator . . . . .	330
11.32	The minus as prefix operator of variable precedence level . . . . .	330
11.33	? as two-way and ?? as three-way conditionals with braced branches . . . . .	331
11.34	! as postfix factorial operator . . . . .	332
11.35	The A/B[N] mechanism . . . . .	332
11.36	<code>\XINT_expr_op_`</code> for recognizing functions . . . . .	333
11.37	The <code>\bool()</code> , <code>\togl()</code> , <code>\protect()</code> pseudo “functions” . . . . .	333
11.38	The <code>\break()</code> function . . . . .	333
11.39	The <code>\qraw()</code> , <code>\qint()</code> , <code>\qfrac()</code> , and <code>\qfloat()</code> “functions” . . . . .	334
11.40	The <code>\random()</code> and <code>\grand()</code> “functions” . . . . .	334
11.41	<code>\XINT_expr_op__</code> for recognizing variables . . . . .	334
11.42	User defined variables: <code>\xintdefvar</code> , <code>\xintdefiivar</code> , <code>\xintdeffloatvar</code> . . . . .	335
11.43	<code>\xintunassignvar</code> . . . . .	336
11.44	<code>seq</code> and the implementation of dummy variables . . . . .	337
11.45	<code>\add()</code> , <code>\mul()</code> . . . . .	344
11.46	<code>\subs()</code> . . . . .	345

11.47	<code>\rseq()</code>	345
11.48	<code>\iter()</code>	347
11.49	<code>\rrseq()</code>	349
11.50	<code>\iterr()</code>	350
11.51	Macros handling csv lists for functions with multiple comma separated arguments in expressions	352
11.52	Auxiliary wrappers for function macros	357
11.53	The <code>num()</code> , <code>reduce()</code> , <code>preduce()</code> , <code>abs()</code> , <code>sgn()</code> , <code>frac()</code> , <code>floor()</code> , <code>ceil()</code> , <code>sqr()</code> , <code>sqrt()</code> , <code>sqrtr()</code> , <code>float()</code> , <code>sfloat()</code> , <code>ilog10()</code> , <code>inv()</code> , <code>round()</code> , <code>trunc()</code> , <code>mod()</code> , <code>quo()</code> , <code>rem()</code> , <code>divmod()</code> , <code>gcd()</code> , <code>lcm()</code> , <code>max()</code> , <code>min()</code> , <code>`+`()</code> , <code>`*`()</code> , <code>?</code> , <code>!</code> , <code>not()</code> , <code>all()</code> , <code>any()</code> , <code>xor()</code> , <code>if()</code> , <code>ifsgn()</code> , <code>ifint()</code> , <code>ifone()</code> , <code>even()</code> , <code>odd()</code> , <code>isint()</code> , <code>isone()</code> , <code>first()</code> , <code>last()</code> , <code>len()</code> , <code>reversed()</code> , <code>factorial()</code> , <code>binomial()</code> and <code>randrange()</code> functions	358
11.54	f-expandable versions of the <code>\xintSeqB::csv</code> and alike routines, for <code>\xintNewExpr</code>	370
11.55	<code>\xintdeffunc</code> , <code>\xintdefiifunc</code> , <code>\xintdeffloatfunc</code>	372
11.56	<code>\xintdefefunc</code> , <code>\xintdefiiefunc</code> , <code>\xintdeffloatefunc</code>	373
11.57	<code>\xintunassignexprfunc</code> , <code>\xintunassigniexprfunc</code> , <code>\xintunassignfloatexprfunc</code>	376
11.58	<code>\xintNewFunction</code>	377
11.59	<code>\xintNewExpr</code> , <code>\xintNewIExpr</code> , <code>\xintNewFloatExpr</code> , <code>\xintNewIExpr</code>	378

This is release 1.3e of [2019/04/05].

## 11.1 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 [13fp-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 [13fp](#) 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.

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

The code for reload detection was initially copied from ΗΕΙΚΟ ΟΒΕΡΔΙΕΚ'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%
```

### 11.3 Package identification

```
55 \XINT_providespackage
56 \ProvidesPackage{xintexpr}%
57 [2019/04/05 1.3e Expandable expression parser (JFB)]%
58 \catcode`! 11
59 \let\XINT_Cmp \xintiiCmp
```

### 11.4 `\xintexpr`, `\xintiexpr`, `\xintfloatexpr`, `\xintiexpr`

ATTENTION! 1.3d renamed `\xinteval` to `\xintexpro` etc...

```
60 \def\xintexpr      {\romannumeral0\xintexpro      }%
61 \def\xintiexpr    {\romannumeral0\xintiexpro      }%
62 \def\xintfloatexpr {\romannumeral0\xintfloatexpro }%
63 \def\xintiexpr    {\romannumeral0\xintiexpro      }%
```

### 11.5 `\xintexpro`, `\xintiexpro`

ATTENTION! 1.3d renamed `\xinteval` to `\xintexpro` etc...

```
64 \def\xintexpro    {\expandafter\XINT_expr_wrap\romannumeral0\xintbareeval }%
65 \def\xintiexpro   {\expandafter\XINT_iiexpr_wrap\romannumeral0\xintbareiieval }%
```

### 11.6 `\xintiexpro`, `\xintfloatexpro`

Optional argument since 1.1.

ATTENTION! 1.3d renamed `\xinteval` to `\xintexpro` etc...

Some renaming of macros at 1.3e here.

```
66 \def\xintiexpro #1%
67 {%
68     \ifx [#1\expandafter\XINT_iexpr_withopt\else\expandafter\XINT_iexpr_noopt
69     \fi #1%
70 }%
71 \def\XINT_iexpr_noopt
72 {%
73     \expandafter\XINT_iexpr_preprint\expandafter 0%
74     \romannumeral0\xintbareeval
75 }%
76 \def\XINT_iexpr_withopt [#1]%
77 {%
78     \expandafter\XINT_iexpr_preprint\expandafter
79     {\the\numexpr \xint_zapspace #1 \xint_gobble_i\expandafter}%
80     \romannumeral0\xintbareeval
81 }%
82 \def\XINT_iexpr_preprint #1#2%
```

```

83 {%
84   \expandafter\XINT_expr_wrap
85   \csname .=\xintRound::csv {#1}{\XINT_expr_unlock #2}\endcsname
86 }%
87 \def\xintfloatexpro #1%
88 {%
89   \ifx [#1\expandafter\XINT_flexpr_withopt\else\expandafter\XINT_flexpr_noopt
90   \fi #1%
91 }%
92 \def\XINT_flexpr_noopt
93 {%
94   \expandafter\XINT_flexpr_preprint\expandafter\xinttheDigits
95   \romannumeral0\xintbarefloateval
96 }%
97 \def\XINT_flexpr_withopt [#1]%
98 {%
99   \expandafter\XINT_flexpr_preprint\expandafter
100  {\the\numexpr\xint_zapspace #1 \xint_gobble_i\expandafter}%
101  \romannumeral0\xintbarefloateval
102 }%
103 \def\XINT_flexpr_preprint #1#2%
104 {%
105   \expandafter\XINT_expr_wrap
106   \csname .;#1.=\XINTinFloat::csv {#1}{\XINT_expr_unlock #2}\endcsname
107 }%

```

### 11.7 \XINT\_expr\_wrap, \XINT\_iiexpr\_wrap, \XINT\_flexpr\_wrap

1.3e removes some leading space tokens which served nothing. There is no \XINT\_iiexpr\_wrap, because \XINT\_expr\_wrap is used directly.

```

108 \def\XINT_expr_wrap   {\XINT_expr_usethe\XINT_protectii\XINT_expr_print}%
109 \def\XINT_iiexpr_wrap {\XINT_expr_usethe\XINT_protectii\XINT_iiexpr_print}%
110 \def\XINT_flexpr_wrap {\XINT_expr_usethe\XINT_protectii\XINT_flexpr_print}%

```

### 11.8 \XINT\_expr\_usethe, \XINT\_protectii

```

111 \def\XINT_protectii #1{\noexpand\XINT_protectii\noexpand #1\noexpand }%
112 \protected\def\XINT_expr_usethe\XINT_protectii {\xintError:missing_xintthe!}%

```

### 11.9 \XINT\_expr\_print, \XINT\_iiexpr\_print, \XINT\_flexpr\_print

```

113 \def\XINT_expr_print  #1{\xintSPRaw::csv {\XINT_expr_unlock #1}}%
114 \def\XINT_iiexpr_print #1{\xintCSV::csv  {\XINT_expr_unlock #1}}%
115 \def\XINT_flexpr_print #1%
116 {%
117   \expandafter\xintPFloat::csv
118   \romannumeral`&&\expandafter\XINT_expr_unlock_sp\string #1!%
119 }%
120 \def\XINT_expr_unlock_sp #1.;#2.=#3!{{#2}{#3}}%

```

### 11.10 \xinttheexpr, \xinttheiexpr, \xintthefloatexpr, \xinttheiexpr

The reason why \xinttheiexpr et \xintthefloatexpr are handled differently is that they admit an optional argument which acts via a custom «printing» stage.

```

121 \def\xinttheexpr
122   {\romannumeral`&&\expandafter\XINT_expr_print\romannumeral0\xintbareeval}%
123 \def\xinttheiexpr
124   {\romannumeral`&&\expandafter\xint_gobble_iii\romannumeral`&&\xintiexpr}%
125 \def\xintthefloatexpr
126   {\romannumeral`&&\expandafter\xint_gobble_iii\romannumeral`&&\xintfloatexpr}%
127 \def\xinttheiiepr
128   {\romannumeral`&&\expandafter\XINT_iiexpr_print\romannumeral0\xintbareiieval}%

```

### 11.11 \thexintexpr, \thexintiexpr, \thexintfloatexpr, \thexintiiepr

New with 1.2h. I have been for the last three years very strict regarding macros with \xint or \XINT, but well.

```

129 \let\thexintexpr      \xinttheexpr
130 \let\thexintiexpr    \xinttheiexpr
131 \let\thexintfloatexpr\xintthefloatexpr
132 \let\thexintiiepr    \xinttheiiepr

```

### 11.12 \xinteval, \xintieval, \xintfloateval, \xintiieval

```

133 \def\xinteval #1%
134   {\romannumeral`&&\expandafter\XINT_expr_print\romannumeral0\xintbareeval#1\relax}%
135 \def\xintieval #1%
136   {\romannumeral`&&\expandafter\xint_gobble_iii\romannumeral`&&\xintiexpr#1\relax}%
137 \def\xintfloateval #1%
138   {\romannumeral`&&\expandafter\xint_gobble_iii\romannumeral`&&\xintfloatexpr#1\relax}%
139 \def\xintiieval #1%
140   {\romannumeral`&&\expandafter\XINT_iiexpr_print\romannumeral0\xintbareiieval#1\relax}%

```

### 11.13 \xintthe

```

141 \def\xintthe #1{\romannumeral`&&\expandafter\xint_gobble_iii\romannumeral`&&#1}%

```

### 11.14 \xintbareeval, \xintbarefloateval, \xintbareiieval

```

142 \def\xintbareeval
143   {\expandafter\XINT_expr_until_end_a\romannumeral`&&\XINT_expr_getnext }%
144 \def\xintbarefloateval
145   {\expandafter\XINT_flexpr_until_end_a\romannumeral`&&\XINT_expr_getnext }%
146 \def\xintbareiieval
147   {\expandafter\XINT_iiexpr_until_end_a\romannumeral`&&\XINT_expr_getnext }%

```

### 11.15 \xintthebareeval, \xintthebarefloateval, \xintthebareiieval

```

148 \def\xintthebareeval      {\expandafter\XINT_expr_unlock\romannumeral0\xintbareeval}%
149 \def\xintthebarefloateval {\expandafter\XINT_expr_unlock\romannumeral0\xintbarefloateval}%
150 \def\xintthebareiieval    {\expandafter\XINT_expr_unlock\romannumeral0\xintbareiieval}%

```

### 11.16 \xintboolexpr, \XINT\_boolexpr\_print, \xinttheboolexpr, \thexintboolexpr

ATTENTION! 1.3d renamed \xinteval to \xintexpro etc...

```

151 \def\xintboolexpr
152 {%
153   \romannumeral0\expandafter\expandafter\expandafter

```

```

154 \XINT_boolexpr_done\expandafter\xint_gobble_iv\romannumeral0\xintexpro
155 }%
156 \def\XINT_boolexpr_done {!\XINT_expr_usethe\XINT_protectii\XINT_boolexpr_print}%
157 \def\XINT_boolexpr_print #1{\xintIsTrue::csv {\XINT_expr_unlock #1}}%
158 \def\xinttheboolexpr
159 {%
160 \romannumeral`&&\expandafter\expandafter\expandafter
161 \XINT_boolexpr_print\expandafter\xint_gobble_iv\romannumeral0\xintexpro
162 }%
163 \let\thexintboolexpr\xinttheboolexpr

```

### 11.17 \xintifboolexpr, \xintifboolfloatexpr, \xintifbooliiexpr

Do not work with comma separated expressions.

```

164 \def\xintifboolexpr #1{\romannumeral0\xintiiifnotzero {\xinttheexpr #1\relax}}%
165 \def\xintifboolfloatexpr #1{\romannumeral0\xintiiifnotzero {\xintthefloatexpr #1\relax}}%
166 \def\xintifbooliiexpr #1{\romannumeral0\xintiiifnotzero {\xinttheiiexpr #1\relax}}%

```

### 11.18 \xintifsgnexpr, \xintifsgnfloatexpr, \xintifsgniiexpr

#### 1.3d.

Do not work with comma separated expressions.

```

167 \def\xintifsgnexpr #1{\romannumeral0\xintiiifsgn {\xinttheexpr #1\relax}}%
168 \def\xintifsgnfloatexpr #1{\romannumeral0\xintiiifsgn {\xintthefloatexpr #1\relax}}%
169 \def\xintifsgniiexpr #1{\romannumeral0\xintiiifsgn {\xinttheiiexpr #1\relax}}%

```

### 11.19 \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!

```

170 \def\xintthecoords #1{\romannumeral`&&\expandafter\expandafter\expandafter
171 \XINT_thecoords_a
172 \expandafter\xint_gobble_iii\romannumeral0#1}%
173 \def\XINT_thecoords_a #1#2% #1=print macro, indispensable for scientific notation
174 {\expandafter\XINT_expr_unlock\csname.=\expandafter\XINT_thecoords_b
175 \romannumeral`&&@#1#2,!,!,^\endcsname }%
176 \def\XINT_thecoords_b #1#2,#3#4,%
177 {\xint_gob_til! #3\XINT_thecoords_c ! (#1#2, #3#4)\XINT_thecoords_b }%
178 \def\XINT_thecoords_c #1^{}%

```

## 11.20 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...).

```

179 \def\xint_gob_til_! #1!{% ! with catcode 11
180 \def\xint_expr_lockscan#1{% not used for decimal numbers in xintexpr 1.2
181 \def\xint_expr_lockscan##1!{\expandafter#1\csname .##1\endcsname}%
182 }\xint_expr_lockscan{ }%
183 \def\xint_expr_lockit#1{%
184 \def\xint_expr_lockit##1!{\expandafter#1\csname .##1\endcsname}%
185 }\xint_expr_lockit{ }%
186 \def\xint_expr_unlock_hex_in #1% expanded inside \csname..\endcsname
187   {\expandafter\xint_expr_inhex\romannumeral`&&\xint_expr_unlock#1;}%
188 \def\xint_expr_inhex #1.#2#3;% expanded inside \csname..\endcsname
189 {%
190   \if#2>%
191     \xintHexToDec{#1}%
192   \else
193     \xintiiMul{\xintiiPow{625}{\xintLength{#3}}}{\xintHexToDec{#1#3}}%
194     [\the\numexpr-4*\xintLength{#3}]%
195   \fi
196 }%
197 \def\xint_expr_unlock   {\expandafter\xint_expr_unlock_a\string }%
198 \def\xint_expr_unlock_a #1.={}%
199 \def\xint_expr_unexpectedtoken {\xintError:ignored }%
200 \let\xint_expr_done\space

```

## 11.21 Hooks for the functioning of `\xintNewExpr` and `\xintdeffunc`

This is new with 1.3. See `\XINT_expr_redefinemacros`.

```

201 \let\xint:NEhook:one\empty
202 \let\xint:NEhook:two\empty
203 \let\xint:NEhook:csv\empty
204 \def\xint:NEhook:twosp #1,#2,!#3{#3{#1}{#2}}%

```

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

11.22.1 `\XINT::_end` . . . . . 302

11.22.2	\xintCSV::csv . . . . .	302
11.22.3	\xintSPRaw, \xintSPRaw::csv . . . . .	302
11.22.4	\xintIsTrue::csv . . . . .	302
11.22.5	\xintRound::csv . . . . .	302
11.22.6	\XINTinFloat::csv . . . . .	303
11.22.7	\xintPFloat::csv . . . . .	303

Changed completely for 1.1, which adds the optional arguments to \xintexpr and \xintfloatexpr.

### 11.22.1 \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

```
205 \def\xint_::_end #1,#2{\xint_gobble_i #2}%
```

### 11.22.2 \xintCSV::csv

```
206 \def\xintCSV::csv #1{\expandafter\xint_csv::_a\romannumeral`&&@#1,^}%
207 \def\xint_csv::_a {\XINT_csv::_b {}}%
208 \def\xint_csv::_b #1#2,{\expandafter\xint_csv::_c \romannumeral`&&@#2,{#1}}%
209 \def\xint_csv::_c #1{\if ^#1\expandafter\xint_::_end\fi\xint_csv::_d #1}%
210 \def\xint_csv::_d #1,#2{\XINT_csv::_b {#2, #1}}% possibly, item #1 is empty.
```

### 11.22.3 \xintSPRaw, \xintSPRaw::csv

```
211 \def\xintSPRaw {\romannumeral0\xintspraw }%
212 \def\xintspraw #1{\expandafter\xint_spraw\romannumeral`&&@#1[\W]}%
213 \def\xint_spraw #1[#2#3]{\xint_gob_til_W #2\xint_spraw_a\W\xint_spraw_p #1[#2#3]}%
214 \def\xint_spraw_a\W\xint_spraw_p #1[\W]{ #1}%
215 \def\xint_spraw_p #1[\W]{\xintpraw {#1}}%
216 \def\xintSPRaw::csv #1{\romannumeral0\expandafter\xint_spraw::_a\romannumeral`&&@#1,^}%
217 \def\xint_spraw::_a {\XINT_spraw::_b {}}%
218 \def\xint_spraw::_b #1#2,{\expandafter\xint_spraw::_c \romannumeral`&&@#2,{#1}}%
219 \def\xint_spraw::_c #1{\if ,#1\xint_dothis\xint_spraw::_e\fi
220     \if ^#1\xint_dothis\xint_::_end\fi
221     \xint_orthat\xint_spraw::_d #1}%
222 \def\xint_spraw::_d #1,{\expandafter\xint_spraw::_e\romannumeral0\xint_spraw #1[\W],}%
223 \def\xint_spraw::_e #1,#2{\XINT_spraw::_b {#2, #1}}%
```

### 11.22.4 \xintIsTrue::csv

```
224 \def\xintIsTrue::csv #1{\romannumeral0\expandafter\xint_istrue::_a\romannumeral`&&@#1,^}%
225 \def\xint_istrue::_a {\XINT_istrue::_b {}}%
226 \def\xint_istrue::_b #1#2,{\expandafter\xint_istrue::_c \romannumeral`&&@#2,{#1}}%
227 \def\xint_istrue::_c #1{\if ,#1\xint_dothis\xint_istrue::_e\fi
228     \if ^#1\xint_dothis\xint_::_end\fi
229     \xint_orthat\xint_istrue::_d #1}%
230 \def\xint_istrue::_d #1,{\expandafter\xint_istrue::_e\romannumeral0\xintisnotzero {#1},}%
231 \def\xint_istrue::_e #1,#2{\XINT_istrue::_b {#2, #1}}%
```

### 11.22.5 \xintRound::csv

1.3e Emploi d'un point comme délimiteur. Dans le futur donner une signification à un #1 négatif dans \XINT\_round::\_a ?

```

232 \def\XINT_:::_end #1,#2#3{\xint_gobble_i #3}%
233 \def\xintRound:::csv #1#2{\romannumeral0\expandafter\XINT_round:::_a
234   \the\numexpr#1\expandafter.\romannumeral`&&@#2,^,}%
235 \def\XINT_round:::_a #1.{\XINT_round:::_b #1.{}%
236 \def\XINT_round:::_b #1.#2#3,{\expandafter\XINT_round:::_c \romannumeral`&&@#3,{#1}{#2}}%
237 \def\XINT_round:::_c #1{\if ,#1\xint_dothis\XINT_round:::_e\fi
238   \if ^#1\xint_dothis\XINT_:::_end\fi
239   \xint_orthat\XINT_round:::_d #1}%
240 \def\XINT_round:::_d #1,#2{%
241   \expandafter\XINT_round:::_e\romannumeral0\ifnum#2>\xint_c_
242   \expandafter\xintround\else\expandafter\xintiround\fi {#2}{#1},{#2}}%
243 \def\XINT_round:::_e #1,#2#3{\XINT_round:::_b #2.{#3, #1}}%

```

### 11.22.6 \XINTinFloat:::csv

1.3e adds support for a negative specifier (\XINT\_infloat:::\_a inserted, by luck formerly it started straight with \XINT\_infloat:::\_b ...).

```

244 \def\XINTinFloat:::csv #1#2{\romannumeral0\expandafter\XINT_infloat:::_a
245   \the\numexpr #1\expandafter.\romannumeral`&&@#2,^,}%
246 \def\XINT_infloat:::_a #1#2.%
247   {\expandafter\XINT_infloat:::_b\the\numexpr\if#1-\XINTdigits\fi#1#2.{}%
248 \def\XINT_infloat:::_b #1.#2#3,{\XINT_infloat:::_c #3,{#1}{#2}}%
249 \def\XINT_infloat:::_c #1{\if ,#1\xint_dothis\XINT_infloat:::_e\fi
250   \if ^#1\xint_dothis\XINT_:::_end\fi
251   \xint_orthat\XINT_infloat:::_d #1}%
252 \def\XINT_infloat:::_d #1,#2%
253   {\expandafter\XINT_infloat:::_e\romannumeral0\XINTinfloat [#2]{#1},{#2}}%
254 \def\XINT_infloat:::_e #1,#2#3{\XINT_infloat:::_b #2.{#3, #1}}%

```

### 11.22.7 \xintPFloat:::csv

Also extended at 1.3e to handle negative optional specifier for digits precision. This macro formats output.

```

255 \def\xintPFloat:::csv #1#2{\romannumeral0\expandafter\XINT_pfloat:::_a
256   \the\numexpr #1\expandafter.\romannumeral`&&@#2,^,}%
257 \def\XINT_pfloat:::_a #1#2.%
258   {\expandafter\XINT_pfloat:::_b\the\numexpr\if#1-\XINTdigits\fi#1#2.{}%
259 \def\XINT_pfloat:::_b #1.#2#3,{\expandafter\XINT_pfloat:::_c \romannumeral`&&@#3,{#1}{#2}}%
260 \def\XINT_pfloat:::_c #1{\if ,#1\xint_dothis\XINT_pfloat:::_e\fi
261   \if ^#1\xint_dothis\XINT_:::_end\fi
262   \xint_orthat\XINT_pfloat:::_d #1}%
263 \def\XINT_pfloat:::_d #1,#2%
264   {\expandafter\XINT_pfloat:::_e\romannumeral0\XINT_pfloat_opt [\xint:#2]{#1},{#2}}%
265 \def\XINT_pfloat:::_e #1,#2#3{\XINT_pfloat:::_b #2.{#3, #1}}%

```

## 11.23 \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.

```

266 \def\XINT_expr_getnext #1%
267 {%
268   \expandafter\XINT_expr_getnext_a\romannumeral`&&@#1%
269 }%
270 \def\XINT_expr_getnext_a #1%
271 {% screens out sub-expressions and \count or \dimen registers/variables
272   \xint_gob_til_! #1\XINT_expr_subexpr !% recall this ! has catcode 11
273   \ifcat\relax#1% \count or \numexpr etc... token or count, dimen, skip cs
274     \expandafter\XINT_expr_countetc
275   \else
276     \expandafter\expandafter\expandafter\XINT_expr_getnextfork\expandafter\string
277   \fi
278   #1%
279 }%
280 \def\XINT_expr_subexpr !#1\fi !{\expandafter\XINT_expr_getop\xint_gobble_iii }%

```

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

```

281 \def\XINT_expr_countetc #1%
282 {%
283   \ifx\count#1\else\ifx\dimen#1\else\ifx\numexpr#1\else\ifx\dimexpr#1\else
284   \ifx\skip#1\else\ifx\glueexpr#1\else\ifx\fontdimen#1\else\ifx\ht#1\else
285   \ifx\dp#1\else\ifx\wd#1\else\ifx\fontcharht#1\else\ifx\fontcharwd#1\else
286   \ifx\fontcharhp#1\else\ifx\fontcharic#1\else
287     \XINT_expr_unpackvar
288     \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
289     \expandafter\XINT_expr_getnext\number #1%
290 }%
291 \def\XINT_expr_unpackvar\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
292   \expandafter\XINT_expr_getnext\number #1%
293   {\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
294   \expandafter\XINT_expr_getop\csname .=#1\endcsname }%
295 \begingroup
296 \lccode`*=`#
297 \lowercase{\endgroup
298 \def\XINT_expr_getnextfork #1{%
299   \if#1*\xint_dothis {\XINT_expr_scan_macropar *}\fi
300   \if#1[\xint_dothis {\xint_c_xviii ({})}\fi
301   \if#1+\xint_dothis \XINT_expr_getnext \fi
302   \if#1.\xint_dothis {\XINT_expr_startdec}\fi
303   \if#1-\xint_dothis -\fi
304   \if#1(\xint_dothis {\xint_c_xviii ({})}\fi
305   \xint_orthat {\XINT_expr_scan_nbr_or_func #1}%
306 }%
307 \def\XINT_expr_scan_macropar #1#2{\expandafter\XINT_expr_getop\csname .=#1#2\endcsname }%

```

## 11.24 \XINT\_expr\_scan\_nbr\_or\_func: the integer or decimal number or hexa-decimal number or function name or variable name or special hacky things big parser

11.24.1	Integral part (skipping zeroes)	306
11.24.2	Fractional part	307
11.24.3	Scientific notation	308



11.24.4	Hexadecimal numbers . . . . .	309
11.24.5	\XINT_expr_scanfunc: parsing names of functions and variables . . . . .	310

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 picks only one token (could have been braced stuff), here it will be + or \*, and via \XINT\_expr\_op\_` this then becomes a suitable \XINT\_{expr|iiexpr|fexpr}\_func\_+ (or \*). Documentation says to use `+(...), but `+(...) is also valid. The opening parenthesis must be there, it is not allowed to come from expansion.

```

308 \catcode96 11 % `
309 \def\XINT_expr_scan_nbr_or_func #1% this #1 has necessarily here catcode 12
310 {%
311   \if )#1\xint_dothis \XINT_expr_gotnil \fi
312   \if "#1\xint_dothis \XINT_expr_scanhex_I\fi
313   \if `#1\xint_dothis {\XINT_expr_onliteral_`}\fi
314   \ifnum \xint_c_ix<1#1 \xint_dothis \XINT_expr_startint\fi
315   \xint_orthat \XINT_expr_scanfunc #1%
316 }%
317 \def\XINT_expr_gotnil{\expandafter\XINT_expr_getop\csname.= \endcsname}%
318 \def\XINT_expr_onliteral_` #1#2#3({\xint_c_xviii `#{2}})%
319 \catcode96 12 % `
320 \def\XINT_expr_startint #1%
321 {%
322   \if #10\expandafter\XINT_expr_gobz_a\else\XINT_expr_scanint_a\fi #1%
323 }%
324 \def\XINT_expr_scanint_a #1#2%
325   {\expandafter\XINT_expr_getop\csname.=#1%
326   \expandafter\XINT_expr_scanint_b\romannumeral`&&@#2}%
327 \def\XINT_expr_gobz_a #1%
328   {\expandafter\XINT_expr_getop\csname.=%
329   \expandafter\XINT_expr_gobz_scanint_b\romannumeral`&&@#1}%
330 \def\XINT_expr_startdec #1%

```

```
331     {\expandafter\XINT_expr_getop\csname.=%
332     \expandafter\XINT_expr_scandec_a\romannumeral`&&@#1}%
```

### 11.24.1 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.

```
333 \def\XINT_expr_scanint_b #1%
334 {%
335     \ifcat \relax #1\expandafter\XINT_expr_scanint_endbycs\expandafter #1\fi
336     \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanint_c\fi
337     \string#1\XINT_expr_scanint_d
338 }%
339 \def\XINT_expr_scanint_d #1%
340 {%
341     \expandafter\XINT_expr_scanint_b\romannumeral`&&@#1%
342 }%
343 \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.

```
344 \def\XINT_expr_scanint_c\string #1\XINT_expr_scanint_d
345 {%
346     \if     _#1\xint_dothis\XINT_expr_scanint_d\fi
347     \if    e#1\xint_dothis{[\the\numexpr\XINT_expr_scanexp_a +}\fi
348     \if    E#1\xint_dothis{[\the\numexpr\XINT_expr_scanexp_a +}\fi
349     \if    .#1\xint_dothis{\XINT_expr_startdec_a .}\fi
350     \xint_orthat {\endcsname #1}%
351 }%
352 \def\XINT_expr_startdec_a .#1%
353 {%
354     \expandafter\XINT_expr_scandec_a\romannumeral`&&@#1%
355 }%
356 \def\XINT_expr_scandec_a #1%
357 {%
358     \if .#1\xint_dothis{\endcsname.}\fi
359     \xint_orthat {\XINT_expr_scandec_b 0.#1}%
360 }%
361 \def\XINT_expr_gobz_scanint_b #1%
362 {%
363     \ifcat \relax #1\expandafter\XINT_expr_gobz_scanint_endbycs\expandafter #1\fi
364     \ifnum\xint_c_x<1\string#1 \else\expandafter\XINT_expr_gobz_scanint_c\fi
```

```

365 \string#1\XINT_expr_scanint_d
366 }%
367 \def\XINT_expr_gobz_scanint_endbycs#1#2\XINT_expr_scanint_d{0\endcsname #1}%
368 \def\XINT_expr_gobz_scanint_c\string #1\XINT_expr_scanint_d
369 {%
370 \if _#1\xint_dothis\XINT_expr_gobz_scanint_d\fi
371 \if e#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +]\fi
372 \if E#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +]\fi
373 \if .#1\xint_dothis{\XINT_expr_gobz_startdec_a .}\fi
374 \if 0#1\xint_dothis\XINT_expr_gobz_scanint_d\fi
375 \xint_orthat {0\endcsname #1}%
376 }%
377 \def\XINT_expr_gobz_scanint_d #1%
378 {%
379 \expandafter\XINT_expr_gobz_scanint_b\romannumeral`&&@#1%
380 }%
381 \def\XINT_expr_gobz_startdec_a .#1%
382 {%
383 \expandafter\XINT_expr_gobz_scandec_a\romannumeral`&&@#1%
384 }%
385 \def\XINT_expr_gobz_scandec_a #1%
386 {%
387 \if .#1\xint_dothis{0\endcsname .}\fi
388 \xint_orthat {\XINT_expr_gobz_scandec_b 0.#1}%
389 }%

```

### 11.24.2 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 ?

```

390 \def\XINT_expr_scandec_b #1.#2%
391 {%
392 \ifcat \relax #2\expandafter\XINT_expr_scandec_endbycs\expandafter#2\fi
393 \ifnum\xint_c_ix<1\string#2 \else\expandafter\XINT_expr_scandec_c\fi
394 \string#2\expandafter\XINT_expr_scandec_d\the\numexpr #1-\xint_c_i.%
395 }%
396 \def\XINT_expr_scandec_endbycs #1#2\XINT_expr_scandec_d
397 \the\numexpr#3-\xint_c_i.{[#3]\endcsname #1}%
398 \def\XINT_expr_scandec_d #1.#2%
399 {%
400 \expandafter\XINT_expr_scandec_b
401 \the\numexpr #1\expandafter.\romannumeral`&&@#2%
402 }%
403 \def\XINT_expr_scandec_c\string #1#2\the\numexpr#3-\xint_c_i.%
404 {%
405 \if _#1\xint_dothis{\XINT_expr_scandec_d#3.}\fi
406 \if e#1\xint_dothis{[\the\numexpr#3\XINT_expr_scanexp_a +]\fi
407 \if E#1\xint_dothis{[\the\numexpr#3\XINT_expr_scanexp_a +]\fi
408 \xint_orthat {[#3]\endcsname #1}%

```

```

409 }%
410 \def\XINT_expr_gobz_scandec_b #1.#2%
411 {%
412   \ifcat \relax #2\expandafter\XINT_expr_gobz_scandec_endbycs\expandafter#2\fi
413   \ifnum\xint_c_ix<1\string#2 \else\expandafter\XINT_expr_gobz_scandec_c\fi
414   \if0#2\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
415   {\expandafter\XINT_expr_gobz_scandec_b}%
416   {\string#2\expandafter\XINT_expr_scandec_d}\the\numexpr#1-\xint_c_i.%
417 }%
418 \def\XINT_expr_gobz_scandec_endbycs #1#2\xint_c_i.{0[0]\endcsname #1}%
419 \def\XINT_expr_gobz_scandec_c\if0#1#2\fi #3\numexpr#4-\xint_c_i.%
420 {%
421   \if _#1\xint_dothis{\XINT_expr_gobz_scandec_b #4.}\fi
422   \if e#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
423   \if E#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
424   \xint_orthat {0[0]\endcsname #1}%
425 }%

```

### 11.24.3 Scientific notation

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

```

426 \def\XINT_expr_scanexp_a #1#2%
427 {%
428   #1\expandafter\XINT_expr_scanexp_b\romannumeral`&&@#2%
429 }%
430 \def\XINT_expr_scanexp_b #1%
431 {%
432   \ifcat \relax #1\expandafter\XINT_expr_scanexp_endbycs\expandafter #1\fi
433   \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanexp_c\fi
434   \string#1\XINT_expr_scanexp_d
435 }%
436 \def\XINT_expr_scanexpr_endbycs#1#2\XINT_expr_scanexp_d []\endcsname #1}%
437 \def\XINT_expr_scanexp_d #1%
438 {%
439   \expandafter\XINT_expr_scanexp_bb\romannumeral`&&@#1%
440 }%
441 \def\XINT_expr_scanexp_c\string #1\XINT_expr_scanexp_d
442 {%
443   \if _#1\xint_dothis \XINT_expr_scanexp_d \fi
444   \if +#1\xint_dothis {\XINT_expr_scanexp_a +}\fi
445   \if -#1\xint_dothis {\XINT_expr_scanexp_a -}\fi
446   \xint_orthat []\endcsname #1}%
447 }%
448 \def\XINT_expr_scanexp_bb #1%
449 {%
450   \ifcat \relax #1\expandafter\XINT_expr_scanexp_endbycs_b\expandafter #1\fi
451   \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanexp_cb\fi
452   \string#1\XINT_expr_scanexp_db
453 }%
454 \def\XINT_expr_scanexp_endbycs_b#1#2\XINT_expr_scanexp_db []\endcsname #1}%

```

```

455 \def\XINT_expr_scanexp_db #1%
456 {%
457   \expandafter\XINT_expr_scanexp_bb\romannumeral`&&@#1%
458 }%
459 \def\XINT_expr_scanexp_cb\string #1\XINT_expr_scanexp_db
460 {%
461   \if_#1\xint_dothis\XINT_expr_scanexp_d\fi
462   \xint_orthat{]\endcsname #1}%
463 }%

```

#### 11.24.4 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.

```

464 \def\XINT_expr_scanhex_I #1% #1="
465 {%
466   \expandafter\XINT_expr_getop\csname.=\expandafter
467   \XINT_expr_unlock_hex_in\csname.=\XINT_expr_scanhexI_a
468 }%
469 \def\XINT_expr_scanhexI_a #1%
470 {%
471   \ifcat #1\relax\xint_dothis{.>\endcsname\endcsname #1}\fi
472   \ifx !#1\xint_dothis{.>\endcsname\endcsname !}\fi
473   \xint_orthat {\expandafter\XINT_expr_scanhexI_aa\string #1}%
474 }%
475 \def\XINT_expr_scanhexI_aa #1%
476 {%
477   \if\ifnum`#1>`/
478     \ifnum`#1>`9
479     \ifnum`#1>`@
480     \ifnum`#1>`F
481     0\else\fi\else0\fi\else1\fi\else0\fi 1%
482     \expandafter\XINT_expr_scanhexI_b
483   \else
484     \if_#1\xint_dothis{\expandafter\XINT_expr_scanhexI_bgob}\fi
485     \if .#1\xint_dothis{\expandafter\XINT_expr_scanhex_transition}\fi
486     \xint_orthat % gather what we got so far, leave catcode 12 #1 in stream
487     {\xint_afterfi {.>\endcsname\endcsname}}%
488   \fi
489   #1%
490 }%
491 \def\XINT_expr_scanhexI_b #1#2%
492 {%
493   #1\expandafter\XINT_expr_scanhexI_a\romannumeral`&&@#2%
494 }%
495 \def\XINT_expr_scanhexI_bgob #1#2%
496 {%
497   \expandafter\XINT_expr_scanhexI_a\romannumeral`&&@#2%

```

```

498 }%
499 \def\XINT_expr_scanhex_transition .#1%
500 {%
501   \expandafter.\expandafter.\expandafter
502   \XINT_expr_scanhexII_a\romannumeral`&&@#1%
503 }%
504 \def\XINT_expr_scanhexII_a #1%
505 {%
506   \ifcat #1\relax\xint_dothis{\endcsname\endcsname#1}\fi
507   \ifx !#1\xint_dothis{\endcsname\endcsname !}\fi
508   \xint_orthat {\expandafter\XINT_expr_scanhexII_aa\string #1}%
509 }%
510 \def\XINT_expr_scanhexII_aa #1%
511 {%
512   \if\ifnum`#1>`/
513     \ifnum`#1>`9
514     \ifnum`#1>`@
515     \ifnum`#1>`F
516     0\else1\fi\else0\fi\else1\fi\else0\fi 1%
517     \expandafter\XINT_expr_scanhexII_b
518   \else
519     \if_#1\xint_dothis{\expandafter\XINT_expr_scanhexII_bgob}\fi
520     \xint_orthat{\xint_afterfi {\endcsname\endcsname}}%
521   \fi
522   #1%
523 }%
524 \def\XINT_expr_scanhexII_b #1#2%
525 {%
526   #1\expandafter\XINT_expr_scanhexII_a\romannumeral`&&@#2%
527 }%
528 \def\XINT_expr_scanhexII_bgob #1#2%
529 {%
530   \expandafter\XINT_expr_scanhexII_a\romannumeral`&&@#2%
531 }%

```

#### 11.24.5 \XINT\_expr\_scanfunc: parsing names of functions and variables

```

532 \def\XINT_expr_scanfunc
533 {%
534   \expandafter\XINT_expr_func\romannumeral`&&@\XINT_expr_scanfunc_a
535 }%
536 \def\XINT_expr_scanfunc_a #1#2%
537 {%
538   \expandafter #1\romannumeral`&&\expandafter\XINT_expr_scanfunc_b\romannumeral`&&@#2%
539 }%

```

This handles: 1) (indirectly) tacit multiplication by a variable in front a of sub-expression, 2) (indirectly) tacit multiplication in front of a \count etc..., 3) functions which are recognized via an encountered opening parenthesis (but later this must be disambiguated from variables with tacit multiplication) 4) 5) 6) 7) acceptable components of a variable or function names: @, underscore, digits, letters (or chars of category code letter.)

The short lived 1.2d which followed the even shorter lived 1.2c managed to introduce a bug here as it removed the check for catcode 11 !, which must be recognized if ! is not to be taken as part

of a variable name. Don't know what I was thinking, it was the time when I was moving the handling of tacit multiplication entirely to the `\XINT_expr_getop` side. Fixed in 1.2e.

I almost decided to remove the `\ifcat\relax` test whose rôle is to avoid the `\string#1` to do something bad is the escape char is a digit! Perhaps I will remove it at some point ! I truly almost did it, but also the case of no escape char is a problem (`\string\0`, if `\0` is a count ...)

The (indirectly) above means that via `\XINT_expr_func` then `\XINT_expr_op__` one goes back to `\XINT_expr_getop` then `\XINT_expr_getop_b` which is the location where tacit multiplication is now centralized. This makes the treatment of tacit multiplication for situations such as `<variable>\count` or `<variable>xintexpr..relax`, perhaps a bit sub-optimal, but first the variable name must be gathered, second the variable must expand to its value.

```
540 \def\XINT_expr_scanfunc_b #1%
541 {%
542   \ifx !#1\xint_dothis{(_)\fi
543   \ifcat \relax#1\xint_dothis{(_)\fi
544   \if (#1\xint_dothis{\xint_firstoftwo{(`)\fi
545   \if @#1\xint_dothis \XINT_expr_scanfunc_a \fi
546   \if _#1\xint_dothis \XINT_expr_scanfunc_a \fi
547   \ifnum \xint_c_ix<1\string#1 \xint_dothis \XINT_expr_scanfunc_a \fi
548   \ifcat a#1\xint_dothis \XINT_expr_scanfunc_a \fi
549   \xint_orthat {(_)%
550     #1%
551 }%
```

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.

```
552 \def\XINT_expr_func #1(#2{\if _#2\xint_dothis\XINT_expr_op__\fi
553   \xint_orthat{\xint_c_xviii #2}{#1}}%
```

## 11.25 `\XINT_expr_getop`: finding the next operator or closing parenthesis or end of expression

Release 1.1 implements multi-character operators.

1.2d adds tacit multiplication also in front of variable or functions names starting with a letter, not only a @ or a \_ as was already the case. This is for (x+y)z situations. It also applies higher precedence in cases like x/2y or x/2@, or x/2max(3,5), or x/2\xintexpr 3\relax.

In fact, finally I decide that all sorts of tacit multiplication will always use the higher precedence.

Indeed I hesitated somewhat: with the current code one does not know if \XINT\_expr\_getop as invoked after a closing parenthesis or because a number parsing ended, and I felt distinguishing the two was unneeded extra stuff. This means cases like (a+b)/(c+d)(e+f) will first multiply the last two parenthesized terms.

The ! starting a sub-expression must be distinguished from the post-fix ! for factorial, thus we must not do a too early \string. In versions < 1.2c, the catcode 11 ! had to be identified in all branches of the number or function scans. Here it is simply treated as a special case of a letter.

1.2q adds tacit multiplication in cases such as (1+1)3 or 5!7!

```

554 \def\XINT_expr_getop #1#2% this #1 is the current locked computed value
555 {%
556   \expandafter\XINT_expr_getop_a\expandafter #1\romannumeral`&&@#2%
557 }%
558 \catcode`* 11
559 \def\XINT_expr_getop_a #1#2%
560 {%
561   \ifx \relax #2\xint_dothis\xint_firstofthree\fi
562   \ifcat \relax #2\xint_dothis\xint_secondofthree\fi
563   \ifnum\xint_c_ix<1\string#2 \xint_dothis\xint_secondofthree\fi
564   \if _#2\xint_dothis \xint_secondofthree\fi
565   \if @#2\xint_dothis \xint_secondofthree\fi
566   \if (#2\xint_dothis \xint_secondofthree\fi
567   \ifcat a#2\xint_dothis \xint_secondofthree\fi
568   \xint_orthat \xint_thirdofthree
569   {\XINT_expr_foundend #1}%
570   {\XINT_expr_precedence_*** *#1#2}% tacit multiplication with higher precedence
571   {\expandafter\XINT_expr_getop_b \string#2#1}%
572 }%
573 \catcode`* 12
574 \def\XINT_expr_foundend {\xint_c_ \relax }% \relax is a place holder here.

```

? is a very special operator with top precedence which will check if the next token is another ?, while avoiding removing a brace pair from token stream due to its syntax. Pre 1.1 releases used : rather than ??, but we need : for Python like slices of lists.

```

575 \def\XINT_expr_getop_b #1%
576 {%
577   \if '#1\xint_dothis{\XINT_expr_binopwrld }\fi
578   \if ?#1\xint_dothis{\XINT_expr_precedence_? ?}\fi
579   \xint_orthat \XINT_expr_scanop_a #1}%
580 }%
581 \def\XINT_expr_binopwrld #1#2'\{\expandafter\XINT_expr_foundop_a
582   \csname XINT_expr_itself_\xint_zapspace #2 \xint_gobble_i\endcsname #1}%
583 \def\XINT_expr_scanop_a #1#2#3%
584   {\expandafter\XINT_expr_scanop_b\expandafter #1\expandafter #2\romannumeral`&&@#3}%
585 \def\XINT_expr_scanop_b #1#2#3%
586 {%
587   \ifcat#3\relax\xint_dothis{\XINT_expr_foundop_a #1#2#3}\fi

```



```

588 \ifcsname XINT_expr_itself_#1#3\endcsname
589 \xint_dothis
590     {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#3\endcsname #2}\fi
591 \xint_orthat {\XINT_expr_foundop_a #1#2#3}%
592 }%
593 \def\XINT_expr_scanop_c #1#2#3%
594 {%
595     \expandafter\XINT_expr_scanop_d\expandafter #1\expandafter #2\romannumeral`&&@#3%
596 }%
597 \def\XINT_expr_scanop_d #1#2#3%
598 {%
599     \ifcat#3\relax \xint_dothis{\XINT_expr_foundop #1#2#3}\fi
600     \ifcsname XINT_expr_itself_#1#3\endcsname
601     \xint_dothis
602         {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#3\endcsname #2}\fi
603     \xint_orthat {\csname XINT_expr_precedence_#1\endcsname #1#2#3}%
604 }%
605 \def\XINT_expr_foundop_a #1%
606 {%
607     \ifcsname XINT_expr_precedence_#1\endcsname
608         \csname XINT_expr_precedence_#1\endcsname\expandafter\endcsname
609         \expandafter #1%
610     \else
611         \xint_afterfi{\XINT_expr_unknown_operator {#1}\XINT_expr_getop}%
612     \fi
613 }%
614 \def\XINT_expr_unknown_operator #1{\xintError:removed \xint_gobble_i {#1}}%
615 \def\XINT_expr_foundop #1{\csname XINT_expr_precedence_#1\endcsname #1}%

```

## 11.26 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.

```

616 \catcode`) 11
617 \def\XINT_tmpa #1#2#3#4%
618 {%
619     \def##1##1%
620     {%
621         \xint_UDsignfork
622             ##1{\expandafter#1\romannumeral`&&@#3}%
623             -{##1}%
624         \krof
625     }%
626     \def##2##1##2%
627     {%
628         \ifcase ##1\expandafter\XINT_expr_done
629         \or\xint_afterfi{\XINT_expr_extra_)
630             \expandafter #1\romannumeral`&&\XINT_expr_getop }%
631         \else
632         \xint_afterfi{\expandafter#1\romannumeral`&&\csname XINT_#4_op_##2\endcsname }%
633         \fi

```

```

634 }%
635 }%
636 \def\XINT_expr_extra_){\xintError:removed }%
637 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
638   \expandafter\XINT_tmpa
639   \csname XINT_#1_until_end_a\expandafter\endcsname
640   \csname XINT_#1_until_end_b\expandafter\endcsname
641   \csname XINT_#1_op_-vi\endcsname
642   {#1}%
643 }%
644 \def\XINT_tmpa #1#2#3#4#5#6%
645 {%
646   \def #1##1{\expandafter #3\romannumeral`&&\XINT_expr_getnext }%
647   \def #2{\expandafter #3\romannumeral`&&\XINT_expr_getnext }%
648   \def #3##1{\xint_UDsignfork
649     ##1{\expandafter #3\romannumeral`&&#5}%
650     -{#4##1}%
651     \krof }%
652   \def #4##1##2{\ifcase ##1\expandafter\XINT_expr_missing_
653     \or \csname XINT_#6_op_##2\expandafter\endcsname
654     \else
655     \xint_afterfi{\expandafter #3\romannumeral`&&\csname XINT_#6_op_##2\endcsname }%
656     \fi
657   }%
658 }%
659 \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.

```

660 \catcode`)\ 12
661 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
662   \expandafter\XINT_tmpa
663   \csname XINT_#1_op_(\expandafter\endcsname
664   \csname XINT_#1_oparen\expandafter\endcsname
665   \csname XINT_#1_until_)_a\expandafter\endcsname
666   \csname XINT_#1_until_)_b\expandafter\endcsname
667   \csname XINT_#1_op_-vi\endcsname
668   {#1}%
669 }%
670 \expandafter\let\csname XINT_expr_precedence_\endcsname\xint_c_i

```

## 11.27 |, ||, &, &&, <, >, =, ==, <=, >=, !=, +, -, \*, /, ^, \*\*, //, /:, .., ..[, ].., ][, ][: ,:] , and ++ operators

- 11.27.1 Square brackets for lists, the !? for omit and abort, and the ++ postfix construct . . . . . 315
- 11.27.2 The |, &, xor, <, >, =, <=, >=, !=, //, /:, .., +, -, \*, /, ^, ..[, and ].. operators for expr, floatexpr and iiexpr operators . . . . . 316
- 11.27.3 The ]+, ]-, ]\*, ]/, ]^, +[, ]-, ]\*, ]/, and ^[ list operators . . . . . 318
- 11.27.4 The 'and', 'or', 'xor', and 'mod' as infix operator words . . . . . 320
- 11.27.5 The ||, &&, \*\*, \*\*[, ]\*\* operators as synonyms . . . . . 320

### 11.27.1 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.

```
671 \expandafter\let\csname XINT_expr_precedence_]\endcsname\xint_c_i
672 \expandafter\let\csname XINT_expr_precedence_;\endcsname\xint_c_i
673 \let\xint_c_xviii\xint_c_xviii
674 \let\xint_c_xix\xint_c_xix
675 \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 exceptionally 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.)

```
676 \catcode`. 11 \catcode`= 11 \catcode`+ 11
677 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
678   \expandafter\let\csname XINT_#1_op_)\endcsname \XINT_expr_getop
679   \expandafter\let\csname XINT_#1_op_;\endcsname \space
680   \expandafter\def\csname XINT_#1_op_]\endcsname ##1{\XINT_expr_scanop_a a##1}%
681   \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.

```
682   \expandafter\def\csname XINT_#1_op_++)\endcsname ##1##2\relax
683   {\expandafter\XINT_expr_foundend \expandafter
684     {\expandafter\.=+\csname .=\XINT:NEhook:one\xintiCeil{\XINT_expr_unlock ##1}\endcsname }}%
685 }%
686 \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.

```
687 \catcode`& 12
688 \xintFor* #1 in {<=>{<=>{>=}{!={&&}{||}{**}{//}{/:}{..}{.}{.}{.}{.}{.}}%
689     {+[-]{*[/]{**[^]{a+}{a-}{a*}{a/}{a**}{a^}%
690     {][}{:}{:}{!}{?}{++}{++}}%{***}}
691     \do {\expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}}%
692 \catcode`& 7
693 \expandafter\let\csname XINT_expr_precedence_***\endcsname \xint_c_viii
```

### 11.27.2 The |, &, xor, <, >, =, <=, >=, !=, //, /:, .., +, -, \*, /, ^, ..[, and ].. operators for expr, floatexpr and iexpr operators

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

```
694 \def\XINT_expr_defbin_c #1#2#3#4#5#6#7#8#9%
695 {%
696   \def #1#1% \XINT_expr_op_<op> ou flexpr ou iexpr
697   {% keep value, get next number and operator, then do until
698     \expandafter #2\expandafter ##1%
699     \romannumeral`&&\expandafter\XINT_expr_getnext }%
700   \def #2#1#2% \XINT_expr_until_<op>_a ou flexpr ou iexpr
701   {\xint_UDsignfork ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
702   -{#3##1#2}%
703   \krof }%
704   \def #3#1#2#3#4% \XINT_expr_until_<op>_b ou flexpr ou iexpr
705   {% either execute next operation now, or first do next (possibly unary)
706     \ifnum ##2>#7%
707     \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
708     \csname XINT_#8_op_##3\endcsname {##4}}%
709     \else \xint_afterfi {\expandafter ##2\expandafter ##3%
710     \csname .=#9#6{\XINT_expr_unlock #1}{\XINT_expr_unlock ##4}\endcsname }%
711     \fi }%
712   \let #7#5%
713 }%
714 \def\XINT_expr_defbin_b #1#2#3#4#5%
715 {%
716   \expandafter\XINT_expr_defbin_c
717   \csname XINT_#1_op_#2\endcsname
718   \csname XINT_#1_until_#2_a\endcsname
719   \csname XINT_#1_until_#2_b\endcsname
720   \csname XINT_#1_op_#4\endcsname
721   \csname xint_c_#3\endcsname
722   \csname #5\endcsname
723   \csname XINT_expr_precedence_#2\endcsname {#1}\XINT:NEhook:two
724 }%
725 \XINT_expr_defbin_b {expr} | {iii}{vi} {xintOR}%
726 \XINT_expr_defbin_b {flexpr} | {iii}{vi} {xintOR}%
727 \XINT_expr_defbin_b {iexpr} | {iii}{vi} {xintOR}%
728 \XINT_expr_defbin_b {expr} & {iv}{vi} {xintAND}%
729 \XINT_expr_defbin_b {flexpr} & {iv}{vi} {xintAND}%
730 \XINT_expr_defbin_b {iexpr} & {iv}{vi} {xintAND}%
```

```

731 \XINT_expr_defbin_b {expr} {xor}{iii}{vi} {xintXOR}%
732 \XINT_expr_defbin_b {flexpr}{xor}{iii}{vi} {xintXOR}%
733 \XINT_expr_defbin_b {iiexpr}{xor}{iii}{vi} {xintXOR}%
734 \XINT_expr_defbin_b {expr} < {v}{vi} {xintLt}%
735 \XINT_expr_defbin_b {flexpr} < {v}{vi} {xintLt}%
736 \XINT_expr_defbin_b {iiexpr} < {v}{vi} {xintiiLt}%
737 \XINT_expr_defbin_b {expr} > {v}{vi} {xintGt}%
738 \XINT_expr_defbin_b {flexpr} > {v}{vi} {xintGt}%
739 \XINT_expr_defbin_b {iiexpr} > {v}{vi} {xintiiGt}%
740 \XINT_expr_defbin_b {expr} = {v}{vi} {xintEq}%
741 \XINT_expr_defbin_b {flexpr} = {v}{vi} {xintEq}%
742 \XINT_expr_defbin_b {iiexpr} = {v}{vi} {xintiiEq}%
743 \XINT_expr_defbin_b {expr} {<=} {v}{vi} {xintLtorEq}%
744 \XINT_expr_defbin_b {flexpr}{<=} {v}{vi} {xintLtorEq}%
745 \XINT_expr_defbin_b {iiexpr}{<=} {v}{vi} {xintiiLtorEq}%
746 \XINT_expr_defbin_b {expr} {>=} {v}{vi} {xintGtorEq}%
747 \XINT_expr_defbin_b {flexpr}{>=} {v}{vi} {xintGtorEq}%
748 \XINT_expr_defbin_b {iiexpr}{>=} {v}{vi} {xintiiGtorEq}%
749 \XINT_expr_defbin_b {expr} {!=} {v}{vi} {xintNotEq}%
750 \XINT_expr_defbin_b {flexpr}{!=} {v}{vi} {xintNotEq}%
751 \XINT_expr_defbin_b {iiexpr}{!=} {v}{vi} {xintiiNotEq}%
752 \XINT_expr_defbin_b {expr} {//} {vii}{vii}{xintDivFloor}% CHANGED IN 1.2p!
753 \XINT_expr_defbin_b {flexpr}{//} {vii}{vii}{XINTinFloatDivFloor}% "
754 \XINT_expr_defbin_b {iiexpr}{//} {vii}{vii}{xintiiDivFloor}% "
755 \XINT_expr_defbin_b {expr} {/:} {vii}{vii}{xintMod}% "
756 \XINT_expr_defbin_b {flexpr}{/:} {vii}{vii}{XINTinFloatMod}% "
757 \XINT_expr_defbin_b {iiexpr}{/:} {vii}{vii}{xintiiMod}% "
758 \XINT_expr_defbin_b {expr} + {vi}{vi} {xintAdd}%
759 \XINT_expr_defbin_b {flexpr} + {vi}{vi} {XINTinFloatAdd}%
760 \XINT_expr_defbin_b {iiexpr} + {vi}{vi} {xintiiAdd}%
761 \XINT_expr_defbin_b {expr} - {vi}{vi} {xintSub}%
762 \XINT_expr_defbin_b {flexpr} - {vi}{vi} {XINTinFloatSub}%
763 \XINT_expr_defbin_b {iiexpr} - {vi}{vi} {xintiiSub}%
764 \XINT_expr_defbin_b {expr} * {vii}{vii}{xintMul}%
765 \XINT_expr_defbin_b {flexpr} * {vii}{vii}{XINTinFloatMul}%
766 \XINT_expr_defbin_b {iiexpr} * {vii}{vii}{xintiiMul}%
767 \XINT_expr_defbin_b {expr} / {vii}{vii}{xintDiv}%
768 \XINT_expr_defbin_b {flexpr} / {vii}{vii}{XINTinFloatDiv}%
769 \XINT_expr_defbin_b {iiexpr} / {vii}{vii}{xintiiDivRound}% CHANGED IN 1.1!
770 \XINT_expr_defbin_b {expr} ^ {ix}{ix} {xintPow}%
771 \XINT_expr_defbin_b {flexpr} ^ {ix}{ix} {XINTinFloatPowerH}%
772 \XINT_expr_defbin_b {iiexpr} ^ {ix}{ix} {xintiiPow}%
773 \XINT_expr_defbin_b {expr} {..[]}{iii}{vi} {xintSeqA::csv}%
774 \XINT_expr_defbin_b {flexpr}{..[]}{iii}{vi} {XINTinFloatSeqA::csv}%
775 \XINT_expr_defbin_b {iiexpr}{..[]}{iii}{vi} {xintiiSeqA::csv}%
776 \def\XINT_expr_defbin_b #1#2#3#4#5%
777 {%
778 \expandafter\XINT_expr_defbin_c
779 \csname XINT_#1_op_#2\expandafter\endcsname
780 \csname XINT_#1_until_#2_a\expandafter\endcsname
781 \csname XINT_#1_until_#2_b\expandafter\endcsname
782 \csname XINT_#1_op_#4\expandafter\endcsname

```

```

783 \csname xint_c_#3\expandafter\endcsname
784 \csname #5\expandafter\endcsname
785 \csname XINT_expr_precedence_#2\endcsname {#1}{}%
786 }%
787 \XINT_expr_defbin_b {expr} {..} {iii}{vi} {xintSeq::csv}%
788 \XINT_expr_defbin_b {flexpr}{..} {iii}{vi} {xintSeq::csv}%
789 \XINT_expr_defbin_b {iiexpr}{..} {iii}{vi} {xintiiSeq::csv}%
790 \XINT_expr_defbin_b {expr} {[]} {iii}{vi} {xintSeqB::csv}%
791 \XINT_expr_defbin_b {flexpr}{[]} {iii}{vi} {XINTinFloatSeqB::csv}%
792 \XINT_expr_defbin_b {iiexpr}{[]} {iii}{vi} {xintiiSeqB::csv}%

```

### 11.27.3 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`).

```

793 \def\XINT_expr_binop_inline#1%
794   {\XINT_expr_binop_inline_a{\expandafter\XINT:NEhook:two\expandafter#1}}%
795 \def\XINT_expr_binop_inline_a
796   {\expandafter\xint_gobble_i\romannumeral`&&\XINT_expr_binop_inline_b }%
797 \def\XINT_expr_binop_inline_b #1#2,{\XINT_expr_binop_inline_c #2,{#1}}%
798 \def\XINT_expr_binop_inline_c #1{%
799   \if ,#1\xint_dothis\XINT_expr_binop_inline_e\fi
800   \if ^#1\xint_dothis\XINT_expr_binop_inline_end\fi
801   \xint_orthat\XINT_expr_binop_inline_d #1}%
802 \def\XINT_expr_binop_inline_d #1,#2{,#2{#1}\XINT_expr_binop_inline_b {#2}}%
803 \def\XINT_expr_binop_inline_e #1,#2{,\XINT_expr_binop_inline_b {#2}}%
804 \def\XINT_expr_binop_inline_end #1,#2{%
805 \def\XINT_expr_deflistopr_c #1#2#3#4#5#6#7#8%
806 {%
807   \def #1##1% \XINT_expr_op_<op> ou flexpr ou iiexpr
808   {% keep value, get next number and operator, then do until
809     \expandafter #2\expandafter ##1%
810     \romannumeral`&&\expandafter\XINT_expr_getnext }%
811   \def #2##1##2% \XINT_expr_until_<op>_a ou flexpr ou iiexpr
812   {\xint_UDsignfork ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
813     -{#3##1##2}%
814     \krof }%
815   \def #3##1##2##3##4% \XINT_expr_until_<op>_b ou flexpr ou iiexpr
816   {% either execute next operation now, or first do next (possibly unary)
817     \ifnum ##2>#7%
818     \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
819       \csname XINT_#8_op_##3\endcsname {##4}}%
820     \else \xint_afterfi {\expandafter ##2\expandafter ##3%
821       \csname .=\expandafter\XINT_expr_binop_inline\expandafter
822         {\expandafter#6\expandafter\xint_exchangetwo_keepbraces\expandafter
823         {\expandafter\XINT_expr_unlock\expandafter ##4\expandafter}\expandafter}%
824         \romannumeral`&&\XINT_expr_unlock ##1,^,\endcsname }%
825     \fi }%
826   \let #7#5%
827 }%
828 \def\XINT_expr_deflistopr_b #1#2#3#4%
829 {%

```

```

830 \expandafter\XINT_expr_deflistopr_c
831 \csname XINT_#1_op_#2\expandafter\endcsname
832 \csname XINT_#1_until_#2_a\expandafter\endcsname
833 \csname XINT_#1_until_#2_b\expandafter\endcsname
834 \csname XINT_#1_op_#3\expandafter\endcsname
835 \csname xint_c_#3\expandafter\endcsname
836 \csname #4\expandafter\endcsname
837 \csname XINT_expr_precedence_#2\endcsname {#1}%
838 }%

```

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

```

839 \XINT_expr_deflistopr_b {expr} {a+}{vi} {xintAdd}%
840 \XINT_expr_deflistopr_b {expr} {a-}{vi} {xintSub}%
841 \XINT_expr_deflistopr_b {expr} {a*}{vii} {xintMul}%
842 \XINT_expr_deflistopr_b {expr} {a/}{vii} {xintDiv}%
843 \XINT_expr_deflistopr_b {expr} {a^}{ix} {xintPow}%
844 \XINT_expr_deflistopr_b {iiexpr}{a+}{vi} {xintiiAdd}%
845 \XINT_expr_deflistopr_b {iiexpr}{a-}{vi} {xintiiSub}%
846 \XINT_expr_deflistopr_b {iiexpr}{a*}{vii} {xintiiMul}%
847 \XINT_expr_deflistopr_b {iiexpr}{a/}{vii} {xintiiDivRound}%
848 \XINT_expr_deflistopr_b {iiexpr}{a^}{ix} {xintiiPow}%
849 \XINT_expr_deflistopr_b {flexpr}{a+}{vi} {XINTinFloatAdd}%
850 \XINT_expr_deflistopr_b {flexpr}{a-}{vi} {XINTinFloatSub}%
851 \XINT_expr_deflistopr_b {flexpr}{a*}{vii} {XINTinFloatMul}%
852 \XINT_expr_deflistopr_b {flexpr}{a/}{vii} {XINTinFloatDiv}%
853 \XINT_expr_deflistopr_b {flexpr}{a^}{ix} {XINTinFloatPowerH}%
854 \def\XINT_expr_deflistopl_c #1#2#3#4#5#6#7%
855 {%
856 \def #1#1{\expandafter#2\expandafter##1\romannumeral`&&@%
857 \expandafter #3\romannumeral`&&@\XINT_expr_getnext }%
858 \def #2##1##2##3##4%
859 {% either execute next operation now, or first do next (possibly unary)
860 \ifnum ##2>#6%
861 \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
862 \csname XINT_#7_op_##3\endcsname {##4}}%
863 \else \xint_afterfi {\expandafter ##2\expandafter ##3%
864 \csname .=\expandafter\XINT_expr_binop_inline\expandafter
865 {\expandafter#5\expandafter
866 {\expandafter\XINT_expr_unlock\expandafter ##1\expandafter}\expandafter}%
867 \romannumeral`&&@\XINT_expr_unlock ##4,^,\endcsname }%
868 \fi }%
869 \let #6#4%
870 }%
871 \def\XINT_expr_deflistopl_b #1#2#3#4%
872 {%
873 \expandafter\XINT_expr_deflistopl_c
874 \csname XINT_#1_op_#2\expandafter\endcsname
875 \csname XINT_#1_until_#2\expandafter\endcsname
876 \csname XINT_#1_until_)_a\expandafter\endcsname
877 \csname xint_c_#3\expandafter\endcsname
878 \csname #4\expandafter\endcsname

```



```
879 \csname XINT_expr_precedence_#2\endcsname {#1}%
880 }%
```

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

```
881 \XINT_expr_deflistopl_b {expr} {+[]}{vi} {xintAdd}%
882 \XINT_expr_deflistopl_b {expr} {-[]}{vi} {xintSub}%
883 \XINT_expr_deflistopl_b {expr} {*[]}{vii} {xintMul}%
884 \XINT_expr_deflistopl_b {expr} {/[]}{vii} {xintDiv}%
885 \XINT_expr_deflistopl_b {expr} {^[]}{ix} {xintPow}%
886 \XINT_expr_deflistopl_b {iiexpr}{+[]}{vi} {xintiiAdd}%
887 \XINT_expr_deflistopl_b {iiexpr}{-[]}{vi} {xintiiSub}%
888 \XINT_expr_deflistopl_b {iiexpr}{*[]}{vii} {xintiiMul}%
889 \XINT_expr_deflistopl_b {iiexpr}{/[]}{vii} {xintiiDivRound}%
890 \XINT_expr_deflistopl_b {iiexpr}{^[]}{ix} {xintiiPow}%
891 \XINT_expr_deflistopl_b {fexpr}{+[]}{vi} {XINTinFloatAdd}%
892 \XINT_expr_deflistopl_b {fexpr}{-[]}{vi} {XINTinFloatSub}%
893 \XINT_expr_deflistopl_b {fexpr}{*[]}{vii} {XINTinFloatMul}%
894 \XINT_expr_deflistopl_b {fexpr}{/[]}{vii} {XINTinFloatDiv}%
895 \XINT_expr_deflistopl_b {fexpr}{^[]}{ix} {XINTinFloatPowerH}%
```

#### 11.27.4 The 'and', 'or', 'xor', and 'mod' as infix operator words

```
896 \xintFor #1 in {and,or,xor,mod} \do {%
897   \expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}}%
898 \expandafter\let\csname XINT_expr_precedence_and\expandafter\endcsname
899   \csname XINT_expr_precedence_&\endcsname
900 \expandafter\let\csname XINT_expr_precedence_or\expandafter\endcsname
901   \csname XINT_expr_precedence_|\endcsname
902 \expandafter\let\csname XINT_expr_precedence_mod\expandafter\endcsname
903   \csname XINT_expr_precedence_/\endcsname
904 \xintFor #1 in {expr, fexpr, iiexpr} \do {%
905   \expandafter\let\csname XINT_#1_op_and\expandafter\endcsname
906     \csname XINT_#1_op_&\endcsname
907   \expandafter\let\csname XINT_#1_op_or\expandafter\endcsname
908     \csname XINT_#1_op_|\endcsname
909   \expandafter\let\csname XINT_#1_op_mod\expandafter\endcsname
910     \csname XINT_#1_op_/\endcsname
911 }%
```

#### 11.27.5 The ||, &&, \*\*, \*\*[, ]\*\* operators as synonyms

```
912 \expandafter\let\csname XINT_expr_precedence_==\expandafter\endcsname
913   \csname XINT_expr_precedence_=\endcsname
914 \expandafter\let\csname XINT_expr_precedence_&\string&\expandafter\endcsname
915   \csname XINT_expr_precedence_&\endcsname
916 \expandafter\let\csname XINT_expr_precedence_||\expandafter\endcsname
917   \csname XINT_expr_precedence_|\endcsname
918 \expandafter\let\csname XINT_expr_precedence_**\expandafter\endcsname
919   \csname XINT_expr_precedence_^\endcsname
920 \expandafter\let\csname XINT_expr_precedence_a**\expandafter\endcsname
921   \csname XINT_expr_precedence_a^\endcsname
922 \expandafter\let\csname XINT_expr_precedence_**[\expandafter\endcsname
923   \csname XINT_expr_precedence_^[ \endcsname
924 \xintFor #1 in {expr, fexpr, iiexpr} \do {%
```



```

925 \expandafter\let\csname XINT_#1_op_==\expandafter\endcsname
926 \csname XINT_#1_op_=\endcsname
927 \expandafter\let\csname XINT_#1_op_&\string&\expandafter\endcsname
928 \csname XINT_#1_op_&\endcsname
929 \expandafter\let\csname XINT_#1_op_||\expandafter\endcsname
930 \csname XINT_#1_op_|\endcsname
931 \expandafter\let\csname XINT_#1_op_**\expandafter\endcsname
932 \csname XINT_#1_op_^\endcsname
933 \expandafter\let\csname XINT_#1_op_a**\expandafter\endcsname
934 \csname XINT_#1_op_a^\endcsname
935 \expandafter\let\csname XINT_#1_op_**[\expandafter\endcsname
936 \csname XINT_#1_op_^[ \endcsname
937 }%

```

## 11.28 Macros for list selectors: [list][N], [list][:b], [list][a:], [list][a:b]

11.28.1	<code>\xintListSel:x:csv</code>	323
11.28.2	<code>\xintListSel:f:csv</code>	324
11.28.3	<code>\xintKeep:x:csv</code>	325
11.28.4	<code>\xintKeep:f:csv</code>	326
11.28.5	<code>\xintTrim:f:csv</code>	326
11.28.6	<code>\xintNthEltPy:f:csv</code>	326
11.28.7	<code>\xintLength:f:csv</code>	326
11.28.8	<code>\xintReverse:f:csv</code>	326

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`.

```

938 \def\xINT_tmpa #1#2#3#4#5#6%
939 {%
940 \def #1##1% \XINT_expr_op_][
941 {%
942 \expandafter #2\expandafter ##1\romannumeral`&&\XINT_expr_getnext
943 }%
944 \def #2##1##2% \XINT_expr_until_][_a
945 {\xint_UDsignfork
946 ##2{\expandafter #2\expandafter ##1\romannumeral`&&#4}%
947 -{#3##1##2}%
948 \krof }%
949 \def #3##1##2##3##4% \XINT_expr_until_][_b
950 {%
951 \ifnum ##2>#5%
952 \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&%
953 \csname XINT_#6_op_##3\endcsname {##4}}%

```

```

954     \else
955     \xint_afterfi
956     {\expandafter ##2\expandafter ##3\csname
957     .=\expandafter\xintListSel:x:csv % will be \xintListSel:f:csv in \xintNewExpr output
958     \romannumeral`&&\XINT_expr_unlock ##4;% selector
959     \XINT_expr_unlock ##1;\endcsname % unlock already pre-positioned for \xintNewExpr
960     }%
961     \fi
962     }%
963     \let #5\xint_c_ii
964 }%
965 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
966 \expandafter\XINT_tmpa
967   \csname XINT_#1_op_][\expandafter\endcsname
968   \csname XINT_#1_until_][_a\expandafter\endcsname
969   \csname XINT_#1_until_][_b\expandafter\endcsname
970   \csname XINT_#1_op_-vi\expandafter\endcsname
971   \csname XINT_expr_precedence_][\endcsname {#1}%
972 }%
973 \def\XINT_tmpa #1#2#3#4#5#6%
974 {%
975   \def #1##1% \XINT_expr_op_:
976   {%
977     \expandafter #2\expandafter ##1\romannumeral`&&\XINT_expr_getnext
978   }%
979   \def #2##1##2% \XINT_expr_until_:_a
980   {\xint_UDsignfork
981     ##2{\expandafter #2\expandafter ##1\romannumeral`&&#4}%
982     -{#3##1##2}%
983   \krof }%
984   \def #3##1##2##3##4% \XINT_expr_until_:_b
985   {%
986     \ifnum ##2>#5%
987     \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&%
988     \csname XINT_#6_op_##3\endcsname {##4}}%
989     \else
990     \xint_afterfi
991     {\expandafter ##2\expandafter ##3\csname
992     .=\XINT:NEhook:one\xintNum{\XINT_expr_unlock ##1};%
993     \XINT:NEhook:one\xintNum{\XINT_expr_unlock ##4}%
994     \endcsname
995     }%
996     \fi
997     }%
998     \let #5\xint_c_iii
999 }%
1000 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1001 \expandafter\XINT_tmpa
1002   \csname XINT_#1_op_:\expandafter\endcsname
1003   \csname XINT_#1_until_:_a\expandafter\endcsname
1004   \csname XINT_#1_until_:_b\expandafter\endcsname
1005   \csname XINT_#1_op_-vi\expandafter\endcsname

```

```

1006 \csname XINT_expr_precedence_:\endcsname {#1}%
1007 }%
1008 \catcode`[ 11 \catcode`] 11
1009 \let\XINT_expr_precedence_:\ \xint_c_iii
1010 \def\XINT_expr_op_:\ #1%
1011 {%
1012 \expandafter\xint_c_i\expandafter )%
1013 \csname .=\XINT:NEhook:one\xintNum{\XINT_expr_unlock #1}\endcsname
1014 }%
1015 \let\XINT_flexpr_op_:\ \XINT_expr_op_:\
1016 \let\XINT_iiexpr_op_:\ \XINT_expr_op_:\
1017 \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).

```

1018 \edef\XINT_expr_op_[: #1{\xint_c_ii\noexpand\XINT_expr_itself_}[#10\string :}%
1019 \let\XINT_flexpr_op_[: \XINT_expr_op_[:
1020 \let\XINT_iiexpr_op_[: \XINT_expr_op_[:
1021 \catcode`[ 12 \catcode`] 12

```

### 11.28.1 `\xintListSel:x:csv`

1.2j. Because there is `\xintKeep:x:csv` which is faster than `\xintKeep:f:csv`.

```

1022 \def\xintListSel:x:csv #1%
1023 {%
1024 \if ]\noexpand#1\xint_dothis\XINT_listsel:_s\fi
1025 \if :\noexpand#1\xint_dothis\XINT_listxsel:_:\fi
1026 \xint_orthat {\XINT_listsel:_nth #1}%
1027 }%
1028 \def\XINT_listsel:_s #1#2;#3;%
1029 {%
1030 \if-#1\expandafter\xintKeep:f:csv\else\expandafter\xintTrim:f:csv\fi
1031 {#1#2}{#3}%
1032 }%
1033 \def\XINT_listsel:_nth #1;#2;{\xintNthEltPy:f:csv {\xintNum{#1}}{#2}}%

```

`\XINT_listsel:_nth` and `\XINT_listsel:_s` located in `\xintListSel:f:csv`.

```

1034 \def\XINT_listxsel:_: #1#2;#3#4;%
1035 {%
1036 \xint_UDsignsfork
1037 #1#3\XINT_listxsel:_N:N
1038 #1-\XINT_listxsel:_N:P
1039 -#3\XINT_listxsel:_P:N
1040 --\XINT_listxsel:_P:P
1041 \krof #1#2;#3#4;%
1042 }%
1043 \def\XINT_listxsel:_P:P #1;#2;#3;%
1044 {%
1045 \unless\ifnum #1<#2 \expandafter\xint_gobble_iii\fi
1046 \xintKeep:x:csv{#2-#1}{\xintTrim:f:csv{#1}{#3}}%

```

```

1047 }%
1048 \def\XINT_listxsel:_N:N #1;#2;#3;%
1049 {%
1050     \expandafter\XINT_listxsel:_N:N_a
1051     \the\numexpr #2-#1\expandafter;\the\numexpr#1+\xintLength:f:csv{#3};#3;%
1052 }%
1053 \def\XINT_listxsel:_N:N_a #1;#2;#3;%
1054 {%
1055     \unless\ifnum #1>\xint_c_ \expandafter\xint_gobble_iii\fi
1056     \xintKeep:x:csv{#1}{\xintTrim:f:csv{\ifnum#2<\xint_c_\xint_c_\else#2\fi}{#3}}%
1057 }%
1058 \def\XINT_listxsel:_N:P #1;#2;#3;{\expandafter\XINT_listxsel:_N:P_a
1059     \the\numexpr #1+\xintLength:f:csv{#3};#2;#3;}%
1060 \def\XINT_listxsel:_N:P_a #1#2;%
1061     {\if -#1\expandafter\XINT_listxsel:_O:P\fi\XINT_listxsel:_P:P #1#2;}%
1062 \def\XINT_listxsel:_O:P\XINT_listxsel:_P:P #1;{\XINT_listxsel:_P:P 0;}%
1063 \def\XINT_listxsel:_P:N #1;#2;#3;{\expandafter\XINT_listxsel:_P:N_a
1064     \the\numexpr #2+\xintLength:f:csv{#3};#1;#3;}%
1065 \def\XINT_listxsel:_P:N_a #1#2;#3;%
1066     {\if -#1\expandafter\XINT_listxsel:_P:O\fi\XINT_listxsel:_P:P #3;#1#2;}%
1067 \def\XINT_listxsel:_P:O\XINT_listxsel:_P:P #1;#2;{\XINT_listxsel:_P:P #1;0;}%

```

### 11.28.2 \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.

```

1068 \def\xintListSel:f:csv #1%
1069 {%
1070     \if ]\noexpand#1\xint_dothis{\expandafter\XINT_listsel:_s\romannumeral`&&@\}\fi
1071     \if :\noexpand#1\xint_dothis{\XINT_listsel:_:}\fi
1072     \xint_orthat {\XINT_listsel:_nth #1}%
1073 }%
1074 \def\XINT_listsel:_: #1;#2;%
1075 {%
1076     \expandafter\XINT_listsel:_:a
1077     \the\numexpr #1\expandafter;\the\numexpr #2\expandafter;\romannumeral`&&@%
1078 }%
1079 \def\XINT_listsel:_:a #1#2;#3#4;%
1080 {%
1081     \xint_UDsignsfork
1082     #1#3\XINT_listsel:_N:N
1083     #1-\XINT_listsel:_N:P
1084     -#3\XINT_listsel:_P:N
1085     --\XINT_listsel:_P:P
1086     \krof #1#2;#3#4;%
1087 }%
1088 \def\XINT_listsel:_P:P #1;#2;#3;%
1089 {%
1090     \unless\ifnum #1<#2 \xint_afterfi{\expandafter\space\xint_gobble_iii}\fi
1091     \xintKeep:f:csv{#2-#1}{\xintTrim:f:csv{#1}{#3}}%
1092 }%

```

```

1093 \def\XINT_listsel:_N:N #1;#2;#3;%
1094 {%
1095     \unless\ifnum #1<#2 \expandafter\XINT_listsel:_N:N_abort\fi
1096     \expandafter\XINT_listsel:_N:N_a
1097     \the\numexpr#1+\xintLength:f:csv{#3}\expandafter;\the\numexpr#2-#1;#3;%
1098 }%
1099 \def\XINT_listsel:_N:N_abort #1;#2;#3;{ }%
1100 \def\XINT_listsel:_N:N_a #1;#2;#3;%
1101 {%
1102     \xintKeep:f:csv{#2}{\xintTrim:f:csv{\ifnum#1<\xint_c_\xint_c_\else#1\fi}{#3}}%
1103 }%
1104 \def\XINT_listsel:_N:P #1;#2;#3;{\expandafter\XINT_listsel:_N:P_a
1105     \the\numexpr #1+\xintLength:f:csv{#3};#2;#3;}%
1106 \def\XINT_listsel:_N:P_a #1#2;%
1107     {\if -#1\expandafter\XINT_listsel:_O:P\fi\XINT_listsel:_P:P #1#2;}%
1108 \def\XINT_listsel:_O:P\XINT_listsel:_P:P #1;{\XINT_listsel:_P:P 0;}%
1109 \def\XINT_listsel:_P:N #1;#2;#3;{\expandafter\XINT_listsel:_P:N_a
1110     \the\numexpr #2+\xintLength:f:csv{#3};#1;#3;}%
1111 \def\XINT_listsel:_P:N_a #1#2;#3;%
1112     {\if -#1\expandafter\XINT_listsel:_P:O\fi\XINT_listsel:_P:P #3;#1#2;}%
1113 \def\XINT_listsel:_P:O\XINT_listsel:_P:P #1;#2;{\XINT_listsel:_P:P #1;0;}%

```

### 11.28.3 \xintKeep:x:csv

1.2j. This macro is used only with positive first argument.

```

1114 \def\xintKeep:x:csv #1#2%
1115 {%
1116     \expandafter\xint_gobble_i
1117     \romannumeral0\expandafter\XINT_keep:x:csv_pos
1118     \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
1119 }%
1120 \def\XINT_keep:x:csv_pos #1.#2%
1121 {%
1122     \expandafter\XINT_keep:x:csv_loop\the\numexpr#1-\xint_c_viii.%
1123     #2\xint_Bye,\xint_Bye,\xint_Bye,\xint_Bye,%
1124     \xint_Bye,\xint_Bye,\xint_Bye,\xint_Bye,\xint_bye
1125 }%
1126 \def\XINT_keep:x:csv_loop #1%
1127 {%
1128     \xint_gob_til_minus#1\XINT_keep:x:csv_finish-%
1129     \XINT_keep:x:csv_loop_pickeight #1%
1130 }%
1131 \def\XINT_keep:x:csv_loop_pickeight #1.#2,#3,#4,#5,#6,#7,#8,#9,%
1132 {%
1133     ,#2,#3,#4,#5,#6,#7,#8,#9%
1134     \expandafter\XINT_keep:x:csv_loop\the\numexpr#1-\xint_c_viii.%
1135 }%
1136 \def\XINT_keep:x:csv_finish-\XINT_keep:x:csv_loop_pickeight -#1.%
1137 {%
1138     \csname XINT_keep:x:csv_finish#1\endcsname
1139 }%
1140 \expandafter\def\csname XINT_keep:x:csv_finish1\endcsname

```

```

1141 #1,#2,#3,#4,#5,#6,#7,{#1,#2,#3,#4,#5,#6,#7\xint_Bye}%
1142 \expandafter\def\csname XINT_keep:x:csv_finish2\endcsname
1143 #1,#2,#3,#4,#5,#6,{#1,#2,#3,#4,#5,#6\xint_Bye}%
1144 \expandafter\def\csname XINT_keep:x:csv_finish3\endcsname
1145 #1,#2,#3,#4,#5,{#1,#2,#3,#4,#5\xint_Bye}%
1146 \expandafter\def\csname XINT_keep:x:csv_finish4\endcsname
1147 #1,#2,#3,#4,{#1,#2,#3,#4\xint_Bye}%
1148 \expandafter\def\csname XINT_keep:x:csv_finish5\endcsname
1149 #1,#2,#3,{#1,#2,#3\xint_Bye}%
1150 \expandafter\def\csname XINT_keep:x:csv_finish6\endcsname
1151 #1,#2,{#1,#2\xint_Bye}%
1152 \expandafter\def\csname XINT_keep:x:csv_finish7\endcsname
1153 #1,{#1\xint_Bye}%
1154 \expandafter\let\csname XINT_keep:x:csv_finish8\endcsname\xint_Bye

```

#### 11.28.4 `\xintKeep:f:csv`

1.2g. moved to [xinttools](#).

#### 11.28.5 `\xintTrim:f:csv`

1.2g. moved to [xinttools](#).

#### 11.28.6 `\xintNthEltPy:f:csv`

1.2g. moved to [xinttools](#).

#### 11.28.7 `\xintLength:f:csv`

1.2g. moved to [xinttools](#).

#### 11.28.8 `\xintReverse:f:csv`

1.2g. moved to [xinttools](#).

### 11.29 Macros for a..b list generation

11.29.1	<code>\xintSeq::csv</code>	326
11.29.2	<code>\xintiiSeq::csv</code>	327

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`.

#### 11.29.1 `\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.

```

1155 \def\xintSeq::csv {\romannumeral0\xintseq::csv}%
1156 \def\xintseq::csv #1#2%

```

```

1157 {%
1158   \expandafter\XINT_seq::csv\expandafter
1159     {\the\numexpr \xintiCeil{#1}\expandafter}\expandafter
1160     {\the\numexpr \xintiFloor{#2}}%
1161 }%
1162 \def\XINT_seq::csv #1#2%
1163 {%
1164   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
1165     \expandafter\XINT_seq::csv_z
1166   \or
1167     \expandafter\XINT_seq::csv_p
1168   \else
1169     \expandafter\XINT_seq::csv_n
1170   \fi
1171   {#2}{#1}%
1172 }%
1173 \def\XINT_seq::csv_z #1#2{ #1/1[0]}%
1174 \def\XINT_seq::csv_p #1#2%
1175 {%
1176   \ifnum #1>#2
1177     \expandafter\expandafter\expandafter\XINT_seq::csv_p
1178   \else
1179     \expandafter\XINT_seq::csv_e
1180   \fi
1181   \expandafter{\the\numexpr #1-\xint_c_i}{#2},#1/1[0]%
1182 }%
1183 \def\XINT_seq::csv_n #1#2%
1184 {%
1185   \ifnum #1<#2
1186     \expandafter\expandafter\expandafter\XINT_seq::csv_n
1187   \else
1188     \expandafter\XINT_seq::csv_e
1189   \fi
1190   \expandafter{\the\numexpr #1+\xint_c_i}{#2},#1/1[0]%
1191 }%
1192 \def\XINT_seq::csv_e #1,{ }%

```

### 11.29.2 \xintiiSeq::csv

```

1193 \def\xintiiSeq::csv {\romannumeral0\xintiiSeq::csv }%
1194 \def\xintiiSeq::csv #1#2%
1195 {%
1196   \expandafter\XINT_iiSeq::csv\expandafter
1197     {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
1198 }%
1199 \def\XINT_iiSeq::csv #1#2%
1200 {%
1201   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
1202     \expandafter\XINT_iiSeq::csv_z
1203   \or
1204     \expandafter\XINT_iiSeq::csv_p
1205   \else
1206     \expandafter\XINT_iiSeq::csv_n

```

```

1207 \fi
1208 {#2}{#1}%
1209 }%
1210 \def\XINT_iiseq::csv_z #1#2{ #1}%
1211 \def\XINT_iiseq::csv_p #1#2%
1212 {%
1213 \ifnum #1>#2
1214 \expandafter\expandafter\expandafter\XINT_iiseq::csv_p
1215 \else
1216 \expandafter\XINT_seq::csv_e
1217 \fi
1218 \expandafter{\the\numexpr #1-\xint_c_i}{#2},#1%
1219 }%
1220 \def\XINT_iiseq::csv_n #1#2%
1221 {%
1222 \ifnum #1<#2
1223 \expandafter\expandafter\expandafter\XINT_iiseq::csv_n
1224 \else
1225 \expandafter\XINT_seq::csv_e
1226 \fi
1227 \expandafter{\the\numexpr #1+\xint_c_i}{#2},#1%
1228 }%
1229 \def\XINT_seq::csv_e #1,{ }%

```

### 11.30 Macros for a..[d].b list generation

11.30.1	\xintSeqA::csv, \xintiiSeqA::csv, \XINTinFloatSeqA::csv	328
11.30.2	\xintSeqB::csv	328
11.30.3	\xintiiSeqB::csv	329
11.30.4	\XINTinFloatSeqB::csv	329

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.

#### 11.30.1 \xintSeqA::csv, \xintiiSeqA::csv, \XINTinFloatSeqA::csv

```

1230 \def\xintSeqA::csv #1%
1231 {\expandafter\XINT_seqa::csv\expandafter{\romannumeral0\xintra {#1}}}%
1232 \def\XINT_seqa::csv #1#2{\expandafter\XINT_seqa::csv_a \romannumeral0\xintra {#2};#1;%
1233 \def\xintiiSeqA::csv #1{\expandafter\XINT_iiseqa::csv\expandafter{\romannumeral`&&@#1}}%
1234 \def\XINT_iiseqa::csv #1#2{\expandafter\XINT_seqa::csv_a\romannumeral`&&@#2;#1;%
1235 \def\XINTinFloatSeqA::csv #1{\expandafter\XINT_flseqa::csv\expandafter
1236 {\romannumeral0\XINTinfloat [\XINTdigits]{#1}}}%
1237 \def\XINT_flseqa::csv #1#2%
1238 {\expandafter\XINT_seqa::csv_a\romannumeral0\XINTinfloat [\XINTdigits]{#2};#1;%
1239 \def\XINT_seqa::csv_a #1{\xint_UDzerominusfork
1240 #1-{z}%
1241 0#1{n}%
1242 0-{p}%
1243 \krof #1}%

```

#### 11.30.2 \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.

```

1244 \def\xintSeqB::csv #1#2%
1245   {\expandafter\XINT_seqb::csv \expandafter{\romannumeral0\xintra{#2}}{#1}}%
1246 \def\XINT_seqb::csv #1#2{\expandafter\XINT_seqb::csv_a\romannumeral`&&@#2#1!}%
1247 \def\XINT_seqb::csv_a #1#2;#3;#4!{\expandafter\XINT_expr_seq_empty?
1248   \romannumeral0\csname XINT_seqb::csv_#1\endcsname {#3}{#4}{#2}}%
1249 \def\XINT_seqb::csv_p #1#2#3%
1250 {%
1251   \xintifCmp {#1}{#2}{, #1\expandafter\XINT_seqb::csv_p\expandafter}%
1252   {, #1\xint_gobble_iii}{\xint_gobble_iii}%

```

\romannumeral0 stopped by \endcsname, XINT\_expr\_seq\_empty? constructs "nil".

```

1253   {\romannumeral0\xintadd {#3}{#1}}{#2}{#3}%
1254 }%
1255 \def\XINT_seqb::csv_n #1#2#3%
1256 {%
1257   \xintifCmp {#1}{#2}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1258   {, #1\expandafter\XINT_seqb::csv_n\expandafter}%
1259   {\romannumeral0\xintadd {#3}{#1}}{#2}{#3}%
1260 }%
1261 \def\XINT_seqb::csv_z #1#2#3{, #1}%

```

### 11.30.3 \xintiiSeqB::csv

```

1262 \def\xintiiSeqB::csv #1#2{\XINT_iiseqb::csv #1#2}%
1263 \def\XINT_iiseqb::csv #1#2#3#4%
1264   {\expandafter\XINT_iiseqb::csv_a
1265     \romannumeral`&&\expandafter \XINT_expr_unlock\expandafter#2%
1266     \romannumeral`&&\XINT_expr_unlock #4!}%
1267 \def\XINT_iiseqb::csv_a #1#2;#3;#4!{\expandafter\XINT_expr_seq_empty?
1268   \romannumeral`&&\csname XINT_iiseqb::csv_#1\endcsname {#3}{#4}{#2}}%
1269 \def\XINT_iiseqb::csv_p #1#2#3%
1270 {%
1271   \xintSgnFork{\XINT_Cmp {#1}{#2}}{, #1\expandafter\XINT_iiseqb::csv_p\expandafter}%
1272   {, #1\xint_gobble_iii}{\xint_gobble_iii}%
1273   {\romannumeral0\xintiiadd {#3}{#1}}{#2}{#3}%
1274 }%
1275 \def\XINT_iiseqb::csv_n #1#2#3%
1276 {%
1277   \xintSgnFork{\XINT_Cmp {#1}{#2}}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1278   {, #1\expandafter\XINT_iiseqb::csv_n\expandafter}%
1279   {\romannumeral0\xintiiadd {#3}{#1}}{#2}{#3}%
1280 }%
1281 \def\XINT_iiseqb::csv_z #1#2#3{, #1}%

```

### 11.30.4 \XINTinFloatSeqB::csv

```

1282 \def\XINTinFloatSeqB::csv #1#2{\expandafter\XINT_flseqb::csv \expandafter
1283   {\romannumeral0\XINTinfloat [\XINTdigits]{#2}}{#1}}%
1284 \def\XINT_flseqb::csv #1#2{\expandafter\XINT_flseqb::csv_a\romannumeral`&&@#2#1!}%
1285 \def\XINT_flseqb::csv_a #1#2;#3;#4!{\expandafter\XINT_expr_seq_empty?
1286   \romannumeral`&&\csname XINT_flseqb::csv_#1\endcsname {#3}{#4}{#2}}%

```

```

1287 \def\XINT_flseqb::csv_p #1#2#3%
1288 {%
1289   \xintifCmp {#1}{#2}{#1\expandafter\XINT_flseqb::csv_p\expandafter}%
1290   {,#1\xint_gobble_iii}{\xint_gobble_iii}%
1291   {\romannumeral0\XINTinfloatadd {#3}{#1}}{#2}{#3}%
1292 }%
1293 \def\XINT_flseqb::csv_n #1#2#3%
1294 {%
1295   \xintifCmp {#1}{#2}{\xint_gobble_iii}{#1\xint_gobble_iii}%
1296   {,#1\expandafter\XINT_flseqb::csv_n\expandafter}%
1297   {\romannumeral0\XINTinfloatadd {#3}{#1}}{#2}{#3}%
1298 }%
1299 \def\XINT_flseqb::csv_z #1#2#3{,#1}%

```

### 11.31 The comma as binary operator

New with 1.09a. Suffices to set its precedence level to two.

```

1300 \def\XINT_tmpa #1#2#3#4#5#6%
1301 {%
1302   \def #1##1% \XINT_expr_op_,
1303   {%
1304     \expandafter #2\expandafter ##1\romannumeral`&&\XINT_expr_getnext
1305   }%
1306   \def #2##1##2% \XINT_expr_until_,_a
1307   {\xint_UDsignfork
1308     ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
1309     -{#3##1##2}%
1310   \krof }%
1311   \def #3##1##2##3##4% \XINT_expr_until_,_b
1312   {%
1313     \ifnum ##2>\xint_c_ii
1314     \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&%
1315                   \csname XINT_#6_op_##3\endcsname {##4}}%
1316   \else
1317     \xint_afterfi
1318     {\expandafter ##2\expandafter ##3%
1319     \csname .=\XINT_expr_unlock ##1,\XINT_expr_unlock ##4\endcsname }%
1320   \fi
1321   }%
1322   \let #5\xint_c_ii
1323 }%
1324 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1325 \expandafter\XINT_tmpa
1326   \csname XINT_#1_op_,\expandafter\endcsname
1327   \csname XINT_#1_until_,_a\expandafter\endcsname
1328   \csname XINT_#1_until_,_b\expandafter\endcsname
1329   \csname XINT_#1_op_-vi\expandafter\endcsname
1330   \csname XINT_expr_precedence_,\endcsname {#1}%
1331 }%

```

### 11.32 The minus as prefix operator of variable precedence level

Inherits the precedence level of the previous infix operator.

```

1332 \def\XINT_tmpa #1#2#3%
1333 {%
1334   \expandafter\XINT_tmpb
1335   \csname XINT_#1_op_#3\expandafter\endcsname
1336   \csname XINT_#1_until_#3_a\expandafter\endcsname
1337   \csname XINT_#1_until_#3_b\expandafter\endcsname
1338   \csname xint_c_#3\endcsname {#1}#2%
1339 }%
1340 \def\XINT_tmpb #1#2#3#4#5#6%
1341 {%
1342   \def #1% \XINT_expr_op_<level>
1343   {% get next number+operator then switch to _until macro
1344     \expandafter #2\romannumeral`&&\XINT_expr_getnext
1345   }%
1346   \def #2##1% \XINT_expr_until_<l>_a
1347   {\xint_UDsignfork
1348     ##1{\expandafter #2\romannumeral`&&@#1}%
1349     -{#3##1}%
1350   \krof }%
1351   \def #3##1##2##3% \XINT_expr_until_<l>_b
1352   {% _until tests precedence level with next op, executes now or postpones
1353     \ifnum ##1>#4%
1354       \xint_afterfi {\expandafter #2\romannumeral`&&%
1355         \csname XINT_#5_op_##2\endcsname {##3}}%
1356     \else
1357       \xint_afterfi {\expandafter ##1\expandafter ##2%
1358         \csname .=%
1359         \XINT:NEhook:one#6{\XINT_expr_unlock ##3}\endcsname }%
1360     \fi
1361   }%
1362 }%

```

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.

```

1363 \xintApplyInline{\XINT_tmpa {expr}\xintOpp}{vi}{vii}{viii}{ix}}%
1364 \xintApplyInline{\XINT_tmpa {flexpr}\xintOpp}{vi}{vii}{viii}{ix}}%
1365 \xintApplyInline{\XINT_tmpa {iiexpr}\xintiiOpp}{vi}{vii}{viii}{ix}}%

```

### 11.33 ? 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.

```

1366 \let\XINT_expr_precedence_? \xint_c_x
1367 \def\XINT_expr_op_? #1#2%
1368   {\XINT_expr_op_?checka #2!\xint_bye\XINT_expr_op_?a #1{#2}}%
1369 \def\XINT_expr_op_?checka #1{\expandafter\XINT_expr_op_?checkb\detokenize{#1}}%
1370 \def\XINT_expr_op_?checkb #1{\if ?#1\expandafter\XINT_expr_op_?checkc
1371   \else\expandafter\xint_bye\fi }%
1372 \def\XINT_expr_op_?checkc #1{\xint_gob_til_! #1\XINT_expr_op_?? !\xint_bye}%
1373 \def\XINT_expr_op_?a #1#2#3%
1374 {%
1375   \xintiiifNotZero{\XINT_expr_unlock #1}{\XINT_expr_getnext #2}{\XINT_expr_getnext #3}%
1376 }%
1377 \let\XINT_flexpr_op_?\XINT_expr_op_?
1378 \let\XINT_iiexpr_op_?\XINT_expr_op_?
1379 \def\XINT_expr_op_?? !\xint_bye\xint_bye\XINT_expr_op_?a #1#2#3#4#5%
1380 {%
1381   \xintiiifSgn {\XINT_expr_unlock #1}%
1382   {\XINT_expr_getnext #3}{\XINT_expr_getnext #4}{\XINT_expr_getnext #5}%
1383 }%

```

### 11.34 ! as postfix factorial operator

```

1384 \let\XINT_expr_precedence_! \xint_c_x
1385 \def\XINT_expr_op_! #1{\expandafter\XINT_expr_getop
1386   \csname .=\XINT:NEhook:one\xintFac{\XINT_expr_unlock #1}\endcsname }%
1387 \def\XINT_flexpr_op_! #1{\expandafter\XINT_expr_getop
1388   \csname .=\XINT:NEhook:one\XINTinFloatFac{\XINT_expr_unlock #1}\endcsname }%
1389 \def\XINT_iiexpr_op_! #1{\expandafter\XINT_expr_getop
1390   \csname .=\XINT:NEhook:one\xintiiFac{\XINT_expr_unlock #1}\endcsname }%

```

### 11.35 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.

```

1391 \catcode`[ 11
1392 \let\XINT_expr_precedence_[ \xint_c_vii
1393 \def\XINT_expr_op_[ #1#2]{\expandafter\XINT_expr_getop
1394   \csname .=\xintE{\XINT_expr_unlock #1}%
1395   {\xint_zapspace #2 \xint_gobble_i}\endcsname}%
1396 \def\XINT_iiexpr_op_[ #1#2]{\expandafter\XINT_expr_getop
1397   \csname .=\xintiiE{\XINT_expr_unlock #1}%
1398   {\xint_zapspace #2 \xint_gobble_i}\endcsname}%
1399 \def\XINT_flexpr_op_[ #1#2]{\expandafter\XINT_expr_getop
1400   \csname .=\XINTinFloatE{\XINT_expr_unlock #1}%
1401   {\xint_zapspace #2 \xint_gobble_i}\endcsname}%
1402 \catcode`[ 12

```

### 11.36 `\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.

```

1403 \def\XINT_tmpa #1#2#3{%
1404   \def #1##1%
1405   {%
1406     \ifcsname XINT_#3_func_##1\endcsname
1407     \xint_dothis{\expandafter\expandafter
1408               \csname XINT_#3_func_##1\endcsname\romannumeral`&&@#2}\fi
1409     \ifcsname XINT_expr_onliteral_##1\endcsname
1410     \xint_dothis{\csname XINT_expr_onliteral_##1\endcsname}\fi
1411     \xint_orthat{\XINT_expr_unknown_function {##1}%
1412               \expandafter\XINT_expr_func_unknown\romannumeral`&&@#2}%
1413   }%
1414 }%
1415 \def\XINT_expr_unknown_function #1{\xintError:removed \xint_gobble_i {#1}}%
1416 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1417   \expandafter\XINT_tmpa
1418     \csname XINT_#1_op_`\expandafter\endcsname
1419     \csname XINT_#1_oparen\endcsname
1420     {#1}%
1421 }%
1422 \def\XINT_expr_func_unknown #1#2#3%
1423   {\expandafter #1\expandafter #2\csname .=0\endcsname }%

```

### 11.37 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".

```

1424 \def\XINT_expr_onliteral_bool #1)%
1425   {\expandafter\XINT_expr_getop\csname .=\xintBool{#1}\endcsname }%
1426 \def\XINT_expr_onliteral_togl #1)%
1427   {\expandafter\XINT_expr_getop\csname .=\xintToggle{#1}\endcsname }%
1428 \def\XINT_expr_onliteral_protect #1)%
1429   {\expandafter\XINT_expr_getop\csname .=\detokenize{#1}\endcsname }%

```

### 11.38 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.

```

1430 \def\XINT_expr_func_break #1#2#3%

```

```

1431     {\expandafter #1\expandafter #2\csname.=?\romannumeral`&&\XINT_expr_unlock #3\endcsname }%
1432 \let\XINT_flexpr_func_break \XINT_expr_func_break
1433 \let\XINT_iiexpr_func_break \XINT_expr_func_break

```

## 11.39 The `\qraw()`, `\qint()`, `\qfrac()`, and `\qfloat()` “functions”

1.2. adds `qint()`, `qfrac()`, `qfloat()`.

1.3c. adds `qraw()`. Useful to limit impact on TeX memory from abuse of `\csname`'s storage when generating many comma separated values from a loop.

1.3e. `qfloat()` keeps a short mantissa if possible.

They allow the user to hand over quickly a big number to the parser, spaces not immediately removed but should be harmless in general. The `qraw()` does no post-processing at all apart complete expansion, useful for comma-separated values, but must be obedient to (non really documented) expected format. Each uses a delimited macro, the closing parenthesis can not emerge from expansion.

```

1434 \def\XINT_expr_onliteral_qint #1)%
1435     {\expandafter\XINT_expr_getop\csname .=\xintiNum{#1}\endcsname }%
1436 \def\XINT_expr_onliteral_qfrac #1)%
1437     {\expandafter\XINT_expr_getop\csname .=\xintRaw{#1}\endcsname }%
1438 \def\XINT_expr_onliteral_qfloat #1)%
1439     {\expandafter\XINT_expr_getop\csname .=\XINTinFloatSdigits{#1}\endcsname }%
1440 \def\XINT_expr_onliteral_qraw #1)%
1441     {\expandafter\XINT_expr_getop\csname .=#1\endcsname }%

```

## 11.40 The `\random()` and `\grand()` “functions”

1.3b. Function-like syntax but with no argument currently, so let's use fast parsing which requires though the closing parenthesis to be explicit.

```

1442 \def\XINT_expr_onliteral_random #1)%
1443     {\expandafter\XINT_expr_getop\csname .=\XINTinRandomFloatSdigits\endcsname }%
1444 \def\XINT_expr_onliteral_grand #1)%
1445     {\expandafter\XINT_expr_getop\csname .=\XINTinRandomFloatSixteen\endcsname }%

```

## 11.41 `\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`.

```

1446 \def\XINT_expr_op__ #1% op__ with two _'s
1447     {%
1448         \ifcsname XINT_expr_var_#1\endcsname
1449             \expandafter\xint_firstoftwo
1450         \else
1451             \expandafter\xint_secondoftwo
1452         \fi
1453         {\expandafter\expandafter\expandafter
1454             \XINT_expr_getop\csname XINT_expr_var_#1\endcsname}%
1455         {\XINT_expr_unknown_variable {#1}%
1456             \expandafter\XINT_expr_getop\csname .=#0\endcsname}%
1457     }%
1458 \def\XINT_expr_unknown_variable #1{\xintError:removed \xint_gobble_i {#1}}%

```

```
1459 \let\XINT_flexpr_op__ \XINT_expr_op__
1460 \let\XINT_iiexpr_op__ \XINT_expr_op__
```

## 11.42 User defined variables: `\xintdefvar`, `\xintdefiivar`, `\xintdeffloatvar`

### 1.1.

**1.2p (2017/12/01).** extends `\xintdefvar` et al. to accept simultaneous assignments to multiple variables.

**1.3c (2018/06/17).** Use `\xintexprSafeCatcodes` (to palliate issue with active semi-colon from Babel+French if in body of a  $\TeX$  document).

And allow usage with both syntaxes `name:=expr`; or `name=expr`; . Also the colon may have catcode 11, 12, or 13 with no issue. Variable names may contain letters, digits, underscores, and must not start with a digit. Names starting with @ or an underscore are reserved.

- currently @, @1, @2, @3, and @4 are reserved because they have special meanings for use in iterations,
- @@, @@@, @@@@ are also reserved but are technically functions, not variables: a user may possibly define @@ as a variable name, but if it is followed by parentheses, the function interpretation will be applied (rather than the variable interpretation followed by a tacit multiplication),
- since 1.21, the underscore \_ may be used as separator of digits in long numbers. Hence a variable whose name starts with \_ will not play well with the mechanism of tacit multiplication of variables by numbers: the underscore will be removed from input stream by the number scanner, thus creating an undefined or wrong variable name, or none at all if the variable name was an initial \_ followed by digits.

```
1461 \catcode`* 11
1462 \def\XINT_expr_defvar_one #1#2%
1463 {%
1464   \XINT_global
1465   \expandafter\edef\csname XINT_expr_var_#1\endcsname
1466     {\expandafter\noexpand#2}%
1467   \XINT_global
1468   \expandafter\edef\csname XINT_expr_onliteral_#1\endcsname
1469     {\XINT_expr_precedence_*** *\expandafter\noexpand#2}%
1470   \ifxintverbose\xintMessage{xintexpr}{Info}
1471     {Variable "#1" \ifxintglobaldefs globally \fi
1472     defined with value \expandafter\XINT_expr_unlock#2.}%
1473   \fi
1474 }%
1475 \catcode`* 12
1476 \catcode`~ 13
1477 \catcode`: 12
1478 \def\XINT_expr_defvar_getname #1:#2~{\endgroup
1479   \def\XINT_defvar_tmpa{#1}\edef\XINT_defvar_tmpr{\xintCSVLength{#1}}%
1480 \def\XINT_expr_defvar #1#2#3;%
1481 {%
1482   \xintexprRestoreCatcodes
```

Maybe `SafeCatcodes` was without effect because the colon and the rest are from some earlier macro definition. Give a safe definition to active colon (even if in math mode with a math active colon..).

```

1483 \begingroup\lccode`~: \lowercase{\let~}\empty
1484 \edef\XINT_defvar_tmpa{#2}%
1485 \edef\XINT_defvar_tmpa{\xint_zapspace_o\XINT_defvar_tmpa}%
1486 \expandafter\XINT_expr_defvar_getname
1487 \detokenize\expandafter{\XINT_defvar_tmpa}:~%
1488 \ifcase\XINT_defvar_tmpc\space
1489 \xintMessage {xintexpr}{Warning}
1490 {Aborting: not allowed to declare variable with empty name.}%
1491 \or
1492 \edef\XINT_defvar_tmpb{\romannumeral0#1#3\relax}%
1493 \XINT_expr_defvar_one\XINT_defvar_tmpa\XINT_defvar_tmpb
1494 \else
1495 \edef\XINT_defvar_tmpb
1496 {\expandafter\XINT_expr_unlock\romannumeral0#1#3\relax}%
1497 \edef\XINT_defvar_tmpd{\xintCSVlength{\XINT_defvar_tmpb}}%
1498 \ifnum\XINT_defvar_tmpc=\XINT_defvar_tmpd\space
1499 \xintAssignArray\xintCSVtoList\XINT_defvar_tmpa\to\XINT_defvar_tmpvar
1500 \xintAssignArray
1501 \xintApply\XINT_expr_lockit{\xintCSVtoList\XINT_defvar_tmpb}%
1502 \to\XINT_defvar_tmpval
1503 \def\XINT_defvar_tmpd{1}%
1504 \xintloop
1505 \expandafter\XINT_expr_defvar_one
1506 \csname XINT_defvar_tmpvar\XINT_defvar_tmpd\expandafter\endcsname
1507 \csname XINT_defvar_tmpval\XINT_defvar_tmpd\endcsname
1508 \ifnum\XINT_defvar_tmpd<\XINT_defvar_tmpc\space
1509 \edef\XINT_defvar_tmpd{\the\numexpr\XINT_defvar_tmpd+1}%
1510 \repeat
1511 \xintRelaxArray\XINT_defvar_tmpvar
1512 \xintRelaxArray\XINT_defvar_tmpval
1513 \else
1514 \xintMessage {xintexpr}{Warning}
1515 {Aborting: mismatch between number of variables (\XINT_defvar_tmpc)
1516 and number of values (\XINT_defvar_tmpd).}%
1517 \fi
1518 \fi
1519 }%
1520 \catcode`~ 3
1521 \catcode`: 11

```

This SafeCatcodes is mainly in the hope that semi-colon ending the expression can still be sanitized.

```

1522 \def\xintdefvar      {\xintexprSafeCatcodes\xintdefvar_a}%
1523 \def\xintdefiivar   {\xintexprSafeCatcodes\xintdefiivar_a}%
1524 \def\xintdeffloatvar {\xintexprSafeCatcodes\xintdeffloatvar_a}%
1525 \def\xintdefvar_a   #1={\XINT_expr_defvar\xintbareeval {#1}}%
1526 \def\xintdefiivar_a #1={\XINT_expr_defvar\xintbareieeval {#1}}%
1527 \def\xintdeffloatvar_a #1={\XINT_expr_defvar\xintbarefloateval {#1}}%

```

## 11.43 \xintunassignvar

### 1.2e.



1.3d. Embarrassingly I had for a long time a misunderstanding of `\ifcsname` (let's blame its documentation) and I was not aware that it chooses FALSE branch if tested control sequence has been `\let` to `\undefined`... So earlier version didn't do the right thing (and had another bug: failure to protect `\.=0` from expansion).

The `\ifcsname` tests are done in `\XINT_expr_op__` and `\XINT_expr_op_``.

```

1528 \def\xintunassignvar #1{%
1529   \edef\XINT_unvar_tmpa{#1}%
1530   \edef\XINT_unvar_tmpa {\xint_zapspace\o\XINT_unvar_tmpa}%
1531   \ifcsname XINT_expr_var_\XINT_unvar_tmpa\endcsname
1532     \ifnum\expandafter\xintLength\expandafter{\XINT_unvar_tmpa}=\@ne
1533       \expandafter\xintnewdummy\XINT_unvar_tmpa
1534     \else
1535       \XINT_global\expandafter
1536       \let\csname XINT_expr_var_\XINT_unvar_tmpa\endcsname\xint_undefined
1537       \XINT_global\expandafter
1538       \let\csname XINT_expr_onliteral_\XINT_unvar_tmpa\endcsname\xint_undefined
1539       \ifxintverbose\xintMessage {xintexpr}{Info}
1540       {Variable \XINT_unvar_tmpa\space has been
1541       \ifxintglobaldefs globally \fi ``unassigned''.}%
1542     \fi
1543   \fi
1544 \else
1545   \xintMessage {xintexpr}{Warning}
1546   {Error: there was no such variable \XINT_unvar_tmpa\space to unassign.}%
1547 \fi
1548 }%

```

## 11.44 seq and the implementation of dummy variables

11.44.1	All letters usable as dummy variables, <code>\xintnewdummy</code> . . . . .	338
11.44.2	<code>xintensuredummy</code> , <code>xintrestorelettervar</code> . . . . .	339
11.44.3	<code>\omit()</code> and <code>\abort()</code> . . . . .	339
11.44.4	The special variables <code>@</code> , <code>@1</code> , <code>@2</code> , <code>@3</code> , <code>@4</code> , <code>@@</code> , <code>@@1</code> , . . . , <code>@@@</code> , <code>@@@1</code> , . . . for recursion	340
11.44.5	<code>\XINT_expr_onliteral_seq</code> . . . . .	341
11.44.6	<code>\XINT_expr_onliteral_seq_a</code> . . . . .	341
11.44.7	<code>\XINT_isbalanced_a</code> for <code>\XINT_expr_onliteral_seq_a</code> . . . . .	342
11.44.8	<code>\XINT_allexpr_func_seqx</code> . . . . .	342
11.44.9	Evaluation over list, <code>\XINT_expr_seq:_a</code> with <code>break</code> , <code>abort</code> , <code>omit</code> . . . . .	343
11.44.10	Evaluation over <code>++</code> generated lists with <code>\XINT_expr_seq:_A</code> . . . . .	343

All of `seq`, `add`, `mul`, `rseq`, etc... (actually all of the extensive changes from `xintexpr` 1.09n to 1.1) was done around June 15-25th 2014, but the problem is that I did not document the code enough, and I had a hard time understanding in October what I had done in June. Despite the lesson, again being short on time, I do not document enough my current understanding of the innards of the beast...

I added `subs`, and `iter` in October (also the `[:n]`, `[n:]` list extractors), proving I did at least understand a bit (or rather could imitate) my earlier code (but don't ask me to explain `\xintNewExpr` !)

The `\XINT_expr_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.

#### 11.44.1 All letters usable as dummy variables, \xintnewdummy

The nil variable was introduced in 1.1 but isn't used under that name. However macros handling a..[d]..b, or for seq with dummy variable where omit has omitted everything may in practice inject a nil value as current number.

1.2c has changed the way variables are disambiguated from functions and for this it has added here the definitions of \XINT\_expr\_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.

1.3e stores earlier meaning for usage by xinttrig and xintlog with \xintensuredummy and \xintrestoredummy as high-level interface.

Do an \xintrestorevar, and patch \xintdefvar to always store previous meaning?

```

1549 \catcode`* 11
1550 \def\xint_expr_makedummy #1%
1551 {%
1552   \ifcsname XINT_expr_var_#1\endcsname
1553     \XINT_global
1554     \expandafter\let\csname XINT_expr_var_#1/old\expandafter\endcsname
1555       \csname XINT_expr_var_#1\expandafter\endcsname
1556   \fi
1557   \ifcsname XINT_expr_onliteral_#1\endcsname
1558     \XINT_global
1559     \expandafter\let\csname XINT_expr_onliteral_#1/old\expandafter\endcsname
1560       \csname XINT_expr_onliteral_#1\expandafter\endcsname
1561   \fi
1562   \XINT_global
1563   \expandafter\def\csname XINT_expr_var_#1\endcsname ##1\relax !#1##2%
1564     {##2##1\relax !#1##2}%
1565   \XINT_global
1566   \expandafter\def\csname XINT_expr_onliteral_#1\endcsname ##1\relax !#1##2%
1567     {\XINT_expr_precedence_*** **2(##1\relax !#1##2)%
1568 }%
1569 \xintApplyUnbraced \XINT_expr_makedummy {abcdefghijklmnopqrstuvwxyz}%

```

```

1570 \xintApplyUnbraced \XINT_expr_makedummy {ABCDEFGHIJKLMNOPQRSTUVWXYZ}%
1571 \def\xintnewdummy #1{%
1572   \XINT_expr_makedummy{#1}%
1573   \ifxintverbose\xintMessage {xintexpr}{Info}%
1574     {#1 (with letter catcode) now
1575     \ifxintglobaldefs globally \fi usable as dummy variable.}%
1576   \fi
1577 }%
1578 \edef\XINT_expr_var_nil {\expandafter\noexpand\csname . = \endcsname}%
1579 \edef\XINT_expr_onliteral_nil
1580   {\XINT_expr_precedence_*** *\expandafter\noexpand\csname . = \endcsname {}}%
1581 \catcode`* 12

```

#### 11.44.2 xintensuredummy, xintrestorelettervar

*\xintensuredummy* differs only in the informational message... Attention that this is not meant to be nested.

```

1582 \def\xintensuredummy #1{%
1583   \XINT_expr_makedummy{#1}%
1584   \ifxintverbose\xintMessage {xintexpr}{Info}%
1585     {#1 (with letter catcode) now
1586     \ifxintglobaldefs globally \fi usable as dummy variable.&&J
1587     Use \string\xintrestoredummy{#1} to restore it to its former meaning.}%
1588   \fi
1589 }%
1590 \def\xintrestorelettervar #1{%
1591   \ifcsname XINT_expr_var_#1/old\endcsname
1592     \XINT_global
1593     \expandafter\let\csname XINT_expr_var_#1\expandafter\endcsname
1594       \csname XINT_expr_var_#1/old\expandafter\endcsname
1595   \fi
1596   \ifcsname XINT_expr_onliteral_#1/old\endcsname
1597     \XINT_global
1598     \expandafter\let\csname XINT_expr_onliteral_#1\expandafter\endcsname
1599       \csname XINT_expr_onliteral_#1/old\expandafter\endcsname
1600   \fi
1601   \ifxintverbose\xintMessage {xintexpr}{Info}%
1602     {Character #1 (with letter catcode)
1603     \ifxintglobaldefs globally \fi restored to its earlier status, if any.}%
1604   \fi
1605 }%

```

#### 11.44.3 \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.

```
1606 \edef\XINT_expr_var_omit #1\relax !{\string !?!\relax !}%
1607 \edef\XINT_expr_var_abort #1\relax !{\string !?^\relax !}%
1608 \def\XINT_expr_op_!? #1#2\relax {\expandafter\XINT_expr_foundend\csname .=#2\endcsname}%
1609 \let\XINT_iiexpr_op_!? \XINT_expr_op_!?
1610 \let\XINT_flexpr_op_!? \XINT_expr_op_!?
```

#### 11.44.4 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.

```
1611 \catcode`? 3 \catcode`* 11
1612 \def\XINT_expr_var_@ #1~#2{#2#1~#2}%
1613 \expandafter\let\csname XINT_expr_var_@1\endcsname \XINT_expr_var_@
1614 \expandafter\def\csname XINT_expr_var_@2\endcsname #1~#2#3{#3#1~#2#3}%
1615 \expandafter\def\csname XINT_expr_var_@3\endcsname #1~#2#3#4{#4#1~#2#3#4}%
1616 \expandafter\def\csname XINT_expr_var_@4\endcsname #1~#2#3#4#5{#5#1~#2#3#4#5}%
1617 \def\XINT_expr_onliteral_@ #1~#2{\XINT_expr_precedence_*** *#2(#1~#2)%
1618 \expandafter\let\csname XINT_expr_onliteral_@1\endcsname \XINT_expr_onliteral_@
1619 \expandafter\def\csname XINT_expr_onliteral_@2\endcsname #1~#2#3%
1620     {\XINT_expr_precedence_*** *#3(#1~#2#3)%
1621 \expandafter\def\csname XINT_expr_onliteral_@3\endcsname #1~#2#3#4%
1622     {\XINT_expr_precedence_*** *#4(#1~#2#3#4)%
1623 \expandafter\def\csname XINT_expr_onliteral_@4\endcsname #1~#2#3#4#5%
1624     {\XINT_expr_precedence_*** *#5(#1~#2#3#4#5)%
1625 \catcode`* 12
1626 \def\XINT_expr_func_@@ #1#2#3#4~#5?%
1627 {%
1628     \expandafter#1\expandafter#2\romannumeral0\XINTntheItnoexpand
```

```

1629             {\xintNum{\XINT_expr_unlock#3}}{#5}#4~#5?%
1630 }%
1631 \def\XINT_expr_func_@@@ #1#2#3#4~#5~#6?%
1632 {%
1633   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1634   {\xintNum{\XINT_expr_unlock#3}}{#6}#4~#5~#6?%
1635 }%
1636 \def\XINT_expr_func_@@@ #1#2#3#4~#5~#6~#7?%
1637 {%
1638   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1639   {\xintNum{\XINT_expr_unlock#3}}{#7}#4~#5~#6~#7?%
1640 }%
1641 \let\XINT_flexpr_func_@@\XINT_expr_func_@@
1642 \let\XINT_flexpr_func_@@@\XINT_expr_func_@@@
1643 \let\XINT_flexpr_func_@@@@\XINT_expr_func_@@@
1644 \def\XINT_iiexpr_func_@@ #1#2#3#4~#5?%
1645 {%
1646   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1647   {\XINT_expr_unlock#3}{#5}#4~#5?%
1648 }%
1649 \def\XINT_iiexpr_func_@@@ #1#2#3#4~#5~#6?%
1650 {%
1651   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1652   {\XINT_expr_unlock#3}{#6}#4~#5~#6?%
1653 }%
1654 \def\XINT_iiexpr_func_@@@ #1#2#3#4~#5~#6~#7?%
1655 {%
1656   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1657   {\XINT_expr_unlock#3}{#7}#4~#5~#6~#7?%
1658 }%
1659 \catcode`? 11

```

#### 11.44.5 \XINT\_expr\_onliteral\_seq

```

1660 \def\XINT_expr_onliteral_seq
1661   {\expandafter\XINT_expr_onliteral_seq_f\romannumeral`&&\XINT_expr_onliteral_seq_a {}}%
1662 \def\XINT_expr_onliteral_seq_f #1#2{\xint_c_xviii `{seqx}#2)\relax #1}%

```

#### 11.44.6 \XINT\_expr\_onliteral\_seq\_a

```

1663 \def\XINT_expr_onliteral_seq_a #1#2,%
1664 {%
1665   \ifcase\XINT_isbalanced_a \relax #1#2(\xint_bye)\xint_bye
1666     \expandafter\XINT_expr_onliteral_seq_c
1667     \or\expandafter\XINT_expr_onliteral_seq_b
1668     \else\expandafter\xintError:we_are_doomed
1669   \fi {#1#2},%
1670 }%
1671 \def\XINT_expr_onliteral_seq_b #1,{\XINT_expr_onliteral_seq_a {#1},}%
1672 \def\XINT_expr_onliteral_seq_c #1,#2#3% #3 pour absorber le =
1673 {%
1674   \XINT_expr_onliteral_seq_d {#2{#1}}{}}%
1675 }%
1676 \def\XINT_expr_onliteral_seq_d #1#2#3)%

```

```

1677 {%
1678   \ifcase\XINT_isbalanced_a \relax #2#3(\xint_bye)\xint_bye
1679     \or\expandafter\XINT_expr_onliteral_seq_e
1680     \else\expandafter\xintError:we_are_doomed
1681   \fi
1682   {#1}{#2#3}%
1683 }%
1684 \def\XINT_expr_onliteral_seq_e #1#2{\XINT_expr_onliteral_seq_d {#1}{#2}}%

```

#### 11.44.7 \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.

```

1685 % use as \XINT_isbalanced_a \relax #1(\xint_bye)\xint_bye
1686 \def\XINT_isbalanced_a #1({\XINT_isbalanced_b #1)\xint_bye }%
1687 \def\XINT_isbalanced_b #1)#2%
1688   {\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
1689 \def\XINT_isbalanced_error #1)\xint_bye {\xint_c_mone}%
    #2 was \xint_bye, was there a ) in original #1?
1690 \def\XINT_isbalanced_c\xint_bye\XINT_isbalanced_error #1%
1691   {\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.
1692 \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.
1693 \def\XINT_isbalanced_d #1)#2%
1694   {\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.
1695 \def\XINT_isbalanced_no\xint_bye #1\xint_bye\xint_bye {\xint_c_i }%

```

#### 11.44.8 \XINT\_allexpr\_func\_seqx

1.2c uses \xintthebareval, ... which strangely were not available at 1.1 time. This spares some tokens from \XINT\_expr\_seq:\_d and cousins. Also now variables have changed their mode of operation they pick only one token which must be an already encapsulated value.

In \XINT\_allexp\_seqx, #2 is the list, evaluated and encapsulated, #3 is the dummy variable, #4 is the expression to evaluate repeatedly.

A special case is a list generated by <variable>+: then #2 is {\.=+\.=<start>}.

```

1696 \def\XINT_expr_func_seqx #1#2{\XINT_allexpr_seqx \xintthebareval }%
1697 \def\XINT_flexpr_func_seqx #1#2{\XINT_allexpr_seqx \xintthebarefloateval}%
1698 \def\XINT_iexpr_func_seqx #1#2{\XINT_allexpr_seqx \xintthebareiieval }%
1699 \def\XINT_allexpr_seqx #1#2#3#4%
1700 {%

```

```

1701 \expandafter \XINT_expr_getop
1702 \csname .=\expandafter\XINT_expr_seq:_aa
1703 \romannumeral`&&\XINT_expr_unlock #2!{#1#4\relax !#3}\endcsname
1704 }%
1705 \def\XINT_expr_seq:_aa #1{\if +#1\expandafter\XINT_expr_seq:_A\else
1706 \expandafter\XINT_expr_seq:_a\fi #1}%

```

#### 11.44.9 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.)

```

1707 \def\XINT_expr_seq:_a #1!#2{\expandafter\XINT_expr_seq_empty?
1708 \romannumeral0\XINT_expr_seq:_b {#2}#1,^,}%
1709 \def\XINT_expr_seq:_b #1#2#3,{%
1710 \if ,#2\xint_dothis\XINT_expr_seq:_noop\fi
1711 \if ^#2\xint_dothis\XINT_expr_seq:_end\fi
1712 \xint_orthat{\expandafter\XINT_expr_seq:_c}\csname.=#2#3\endcsname {#1}%
1713 }%
1714 \def\XINT_expr_seq:_noop\csname.=,#1\endcsname #2{\XINT_expr_seq:_b {#2}#1,}%
1715 \def\XINT_expr_seq:_end \csname.=^\endcsname #1}%
1716 \def\XINT_expr_seq:_c #1#2{\expandafter\XINT_expr_seq:_d\romannumeral`&&@#2#1{#2}}%
1717 \def\XINT_expr_seq:_d #1{\if #1^\xint_dothis\XINT_expr_seq:_abort\fi
1718 \if #1?\xint_dothis\XINT_expr_seq:_break\fi
1719 \if #1!\xint_dothis\XINT_expr_seq:_omit\fi
1720 \xint_orthat{\XINT_expr_seq:_goon #1}}%
1721 \def\XINT_expr_seq:_abort #1!#2#3#4#5^,{}%
1722 \def\XINT_expr_seq:_break #1!#2#3#4#5^,{,#1}%
1723 \def\XINT_expr_seq:_omit #1!#2#3#4{\XINT_expr_seq:_b {#4}}%
1724 \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`.

```

1725 \def\XINT_expr_seq_empty? #1{%
1726 \def\XINT_expr_seq_empty? ##1{\if ,##1\expandafter\xint_gobble_i\fi #1\endcsname }}%
1727 \XINT_expr_seq_empty? { }%

```

#### 11.44.10 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.



```

1728 \def\XINT_expr_seq:_A +#1!%
1729   {\expandafter\XINT_expr_seq_empty?\romannumeral0\XINT_expr_seq:_D #1}%
1730 \def\XINT_expr_seq:_D #1#2{\expandafter\XINT_expr_seq:_E\romannumeral`&&@#2#1{#2}}%
1731 \def\XINT_expr_seq:_E #1{\if #1^\xint_dothis\XINT_expr_seq:_Abort\fi
1732   \if #1?\xint_dothis\XINT_expr_seq:_Break\fi
1733   \if #1!\xint_dothis\XINT_expr_seq:_Omit\fi
1734   \xint_orthat{\XINT_expr_seq:_Goon #1}}%
1735 \def\XINT_expr_seq:_Abort #1!#2#3#4{%
1736 \def\XINT_expr_seq:_Break #1!#2#3#4{,#1}%
1737 \def\XINT_expr_seq:_Omit #1!#2#3%
1738   {\expandafter\XINT_expr_seq:_D
1739     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
1740 \def\XINT_expr_seq:_Goon #1!#2#3%
1741   {,#1\expandafter\XINT_expr_seq:_D
1742     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%

```

## 11.45 \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.

```

1743 \def\XINT_expr_onliteral_add
1744   {\expandafter\XINT_expr_onliteral_add_f\romannumeral`&&\XINT_expr_onliteral_seq_a {}}%
1745 \def\XINT_expr_onliteral_add_f #1#2{\xint_c_xviii `{opxadd}#2)\relax #1}%
1746 \def\XINT_expr_onliteral_mul
1747   {\expandafter\XINT_expr_onliteral_mul_f\romannumeral`&&\XINT_expr_onliteral_seq_a {}}%
1748 \def\XINT_expr_onliteral_mul_f #1#2{\xint_c_xviii `{opxmul}#2)\relax #1}%

```

### 11.45.1 \XINT\_expr\_func\_opxadd, \XINT\_flexpr\_func\_opxadd, \XINT\_iiexpr\_func\_opxadd and same for mul

modified 1.2c.

```

1749 \def\XINT_expr_func_opxadd #1#2{\XINT_allexpr_opx \xintbareeval {\xintAdd 0}}%
1750 \def\XINT_flexpr_func_opxadd #1#2{\XINT_allexpr_opx \xintbarefloateval {\XINTinFloatAdd 0}}%
1751 \def\XINT_iiexpr_func_opxadd #1#2{\XINT_allexpr_opx \xintbareiieval {\xintiiAdd 0}}%
1752 \def\XINT_expr_func_opxmul #1#2{\XINT_allexpr_opx \xintbareeval {\xintMul 1}}%
1753 \def\XINT_flexpr_func_opxmul #1#2{\XINT_allexpr_opx \xintbarefloateval {\XINTinFloatMul 1}}%
1754 \def\XINT_iiexpr_func_opxmul #1#2{\XINT_allexpr_opx \xintbareiieval {\xintiiMul 1}}%

```

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

```

1755 \def\XINT_allexpr_opx #1#2#3#4#5%
1756 {%
1757   \expandafter\XINT_expr_getop
1758   \csname.=\romannumeral`&&\expandafter\XINT_expr_op:_a
1759     \romannumeral`&&\XINT_expr_unlock #3!{#1#5\relax !#4}{#2}\endcsname
1760 }%
1761 \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.



```

1762 \def\XINT_expr_op:_b #1#2#3#4#5,{%
1763   \if ,#4\xint_dothis\XINT_expr_op:_noop\fi
1764   \if ^#4\xint_dothis\XINT_expr_op:_end\fi
1765   \xint_orthat{\expandafter\XINT_expr_op:_c}\csname.=#4#5\endcsname {#3}#1{#2}%
1766 }%
1767 \def\XINT_expr_op:_c #1#2#3#4%
1768   {\expandafter\XINT_expr_op:_d\romannumeral0#2#1#3{#4}{#2}}%
1769 \def\XINT_expr_op:_d #1!#2#3#4#5%
1770   {\expandafter\XINT_expr_op:_b\expandafter #4\expandafter
1771     {\romannumeral`&&\XINT:NEhook:two#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.

```

1772 \def\XINT_expr_op:_noop\csname.=,#1\endcsname #2#3#4{\XINT_expr_op:_b #3{#4}{#2}#1,%
1773 \def\XINT_expr_op:_end \csname.=^\endcsname #1#2#3{#3}%

```

## 11.46 \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.

```

1774 \def\XINT_expr_onliteral_subs
1775   {\expandafter\XINT_expr_onliteral_subs_f\romannumeral`&&\XINT_expr_onliteral_seq_a {}}%
1776 \def\XINT_expr_onliteral_subs_f #1#2{\xint_c_xviii `{subx}#2)\relax #1}%
1777 \def\XINT_expr_func_subx #1#2{\XINT_allexpr_subx \xintbareeval }%
1778 \def\XINT_flexpr_func_subx #1#2{\XINT_allexpr_subx \xintbarefloateval}%
1779 \def\XINT_iiexpr_func_subx #1#2{\XINT_allexpr_subx \xintbareiieval }%
1780 \def\XINT_allexpr_subx #1#2#3#4% #2 is the value to assign to the dummy variable
1781 {% #3 is the dummy variable, #4 is the expression to evaluate
1782   \expandafter\expandafter\expandafter\XINT_expr_getop
1783   \expandafter\XINT_expr_subx:_end\romannumeral0#1#4\relax !#3#2%
1784 }%
1785 \def\XINT_expr_subx:_end #1!#2#3{#1}%

```

## 11.47 \rseq()

11.47.1	\XINT_expr_rseqx	346
11.47.2	\XINT_expr_rseqy	346
11.47.3	\XINT_expr_rseq:_a etc...	346
11.47.4	\XINT_expr_rseq:_A etc...	347

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...

```

1786 \def\XINT_expr_func_rseq   {\XINT_allexpr_rseq \xintbareeval   \xintthebareeval   }%
1787 \def\XINT_flexpr_func_rseq {\XINT_allexpr_rseq \xintbarefloateval \xintthebarefloateval }%
1788 \def\XINT_iiexpr_func_rseq {\XINT_allexpr_rseq \xintbareiieval   \xintthebareiieval   }%
1789 \def\XINT_allexpr_rseq #1#2#3%

```

```
1790 {%
1791   \expandafter\XINT_expr_rseqx\expandafter #1\expandafter#2\expandafter
1792   #3\romannumeral`&&\XINT_expr_onliteral_seq_a {}%
1793 }%
```

### 11.47.1 \XINT\_expr\_rseqx

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

```
1794 \def\XINT_expr_rseqx #1#2#3#4#5%
1795 {%
1796   \expandafter\XINT_expr_rseqy\romannumeral0#1(#5)\relax #3#4#2%
1797 }%
```

### 11.47.2 \XINT\_expr\_rseqy

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

```
1798 \def\XINT_expr_rseqy #1#2#3#4#5%
1799 {%
1800   \expandafter \XINT_expr_getop
1801   \csname .=\XINT_expr_unlock #2%
1802   \expandafter\XINT_expr_rseq:_aa
1803       \romannumeral`&&\XINT_expr_unlock #1!{#5#4\relax !#3}#2\endcsname
1804 }%
1805 \def\XINT_expr_rseq:_aa #1{\if +#1\expandafter\XINT_expr_rseq:_A\else
1806       \expandafter\XINT_expr_rseq:_a\fi #1}%

```

### 11.47.3 \XINT\_expr\_rseq:\_a etc...

```
1807 \def\XINT_expr_rseq:_a #1!#2#3{\XINT_expr_rseq:_b {#3}{#2}#1,^,%}
1808 \def\XINT_expr_rseq:_b #1#2#3#4,{%
1809   \if ,#3\xint_dothis\XINT_expr_rseq:_noop\fi
1810   \if ^#3\xint_dothis\XINT_expr_rseq:_end\fi
1811   \xint_orthat{\expandafter\XINT_expr_rseq:_c}\csname.=#3#4\endcsname
1812   {#1}{#2}%
1813 }%
1814 \def\XINT_expr_rseq:_noop\csname.=#1\endcsname #2#3{\XINT_expr_rseq:_b {#2}{#3}#1,%}
1815 \def\XINT_expr_rseq:_end \csname.=^\endcsname #1#2{%}
1816 \def\XINT_expr_rseq:_c #1#2#3%
1817   {\expandafter\XINT_expr_rseq:_d\romannumeral`&&#3#1~#2{#3}}%
1818 \def\XINT_expr_rseq:_d #1{%
1819   \if ^#1\xint_dothis\XINT_expr_rseq:_abort\fi
1820   \if ?#1\xint_dothis\XINT_expr_rseq:_break\fi
1821   \if !#1\xint_dothis\XINT_expr_rseq:_omit\fi
1822   \xint_orthat{\XINT_expr_rseq:_goon #1}%}
1823 \def\XINT_expr_rseq:_goon #1!#2#3~#4#5{,#1\expandafter\XINT_expr_rseq:_b
1824   \romannumeral0\XINT_expr_lockit {#1}{#5}}%
1825 \def\XINT_expr_rseq:_omit #1!#2#3~{\XINT_expr_rseq:_b }%
1826 \def\XINT_expr_rseq:_abort #1!#2#3~#4#5#6^,{,%}
1827 \def\XINT_expr_rseq:_break #1!#2#3~#4#5#6^,{,#1}%

```

### 11.47.4 \XINT\_expr\_rseq:\_A etc...

n++ for rseq. With 1.2c dummy variables pick a single token.

```

1828 \def\XINT_expr_rseq:_A +#1!#2#3{\XINT_expr_rseq:_D #1#3{#2}}%
1829 \def\XINT_expr_rseq:_D #1#2#3%
1830   {\expandafter\XINT_expr_rseq:_E\romannumeral`&&#3#1~#2{#3}}%
1831 \def\XINT_expr_rseq:_E #1{\if #1^\xint_dothis\XINT_expr_rseq:_Abort\fi
1832     \if #1?\xint_dothis\XINT_expr_rseq:_Break\fi
1833     \if #1!\xint_dothis\XINT_expr_rseq:_Omit\fi
1834     \xint_orthat{\XINT_expr_rseq:_Goon #1}}%
1835 \def\XINT_expr_rseq:_Goon #1!#2#3~#4#5%
1836   {,#1\expandafter\XINT_expr_rseq:_D
1837     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1838     \romannumeral0\XINT_expr_lockit{#1}{#5}}%
1839 \def\XINT_expr_rseq:_Omit #1!#2#3~#4#5%
1840   {\expandafter\XINT_expr_rseq:_D
1841     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname }%
1842 \def\XINT_expr_rseq:_Abort #1!#2#3~#4#5}%
1843 \def\XINT_expr_rseq:_Break #1!#2#3~#4#5{,#1}%

```

### 11.48 \iter()

11.48.1	\XINT_expr_iterx . . . . .	347
11.48.2	\XINT_expr_itory . . . . .	348
11.48.3	\XINT_expr_iter:_a etc... . . . . .	348
11.48.4	\XINT_expr_iter:_A etc... . . . . .	348

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.

```

1844 \def\XINT_expr_func_iter   {\XINT_allexpr_iter \xintbareeval   \xintthebareeval   }%
1845 \def\XINT_flexpr_func_iter {\XINT_allexpr_iter \xintbarefloateval \xintthebarefloateval }%
1846 \def\XINT_iieexpr_func_iter {\XINT_allexpr_iter \xintbareiieval   \xintthebareiieval   }%
1847 \def\XINT_allexpr_iter #1#2#3%
1848 {%
1849   \expandafter\XINT_expr_iterx\expandafter #1\expandafter#2\expandafter
1850   #3\romannumeral`&&\XINT_expr_onliteral_seq_a {}}%
1851 }%

```

#### 11.48.1 \XINT\_expr\_iterx

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

```

1852 \def\XINT_expr_iterx #1#2#3#4#5%
1853 {%
1854   \expandafter\XINT_expr_itory\romannumeral0#1(#5)\relax #3#4#2%
1855 }%

```

### 11.48.2 \XINT\_expr\_itery

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

```
1856 \def\XINT_expr_itery #1#2#3#4#5%
1857 {%
1858     \expandafter \XINT_expr_getop
1859     \csname .=%
1860     \expandafter\XINT_expr_iter:_aa
1861     \romannumeral`&&\XINT_expr_unlock #1!{#5#4\relax !#3}#2\endcsname
1862 }%
1863 \def\XINT_expr_iter:_aa #1{\if +#1\expandafter\XINT_expr_iter:_A\else
1864     \expandafter\XINT_expr_iter:_a\fi #1}%
```

### 11.48.3 \XINT\_expr\_iter:\_a etc...

```
1865 \def\XINT_expr_iter:_a #1!#2#3{\XINT_expr_iter:_b {#3}{#2}#1,^,%}
1866 \def\XINT_expr_iter:_b #1#2#3#4,%{
1867     \if ,#3\xint_dothis\XINT_expr_iter:_noop\fi
1868     \if ^#3\xint_dothis\XINT_expr_iter:_end\fi
1869     \xint_orthat{\expandafter\XINT_expr_iter:_c}%
1870     \csname.#3#4\endcsname {#1}{#2}%
1871 }%
1872 \def\XINT_expr_iter:_noop\csname.=,#1\endcsname #2#3{\XINT_expr_iter:_b {#2}{#3}#1,%}
1873 \def\XINT_expr_iter:_end \csname.=^\endcsname #1#2{\XINT_expr:_unlock #1}%
1874 \def\XINT_expr_iter:_c #1#2#3%
1875     {\expandafter\XINT_expr_iter:_d\romannumeral`&&@#3#1~#2{#3}}%
1876 \def\XINT_expr_iter:_d #1{%
1877     \if ^#1\xint_dothis\XINT_expr_iter:_abort\fi
1878     \if ?#1\xint_dothis\XINT_expr_iter:_break\fi
1879     \if !#1\xint_dothis\XINT_expr_iter:_omit\fi
1880     \xint_orthat{\XINT_expr_iter:_goon #1}%
1881 \def\XINT_expr_iter:_goon #1!#2#3~#4#5%
1882     {\expandafter\XINT_expr_iter:_b\romannumeral0\XINT_expr_lockit {#1}{#5}}%
1883 \def\XINT_expr_iter:_omit #1!#2#3~{\XINT_expr_iter:_b }%
1884 \def\XINT_expr_iter:_abort #1!#2#3~#4#5#6^,{\XINT_expr_unlock #4}%
1885 \def\XINT_expr_iter:_break #1!#2#3~#4#5#6^,{#1}%
```

### 11.48.4 \XINT\_expr\_iter:\_A etc...

n++ for iter. With 1.2c dummy variables pick a single token.

```
1886 \def\XINT_expr_iter:_A +#1!#2#3{\XINT_expr_iter:_D #1#3{#2}}%
1887 \def\XINT_expr_iter:_D #1#2#3%
1888     {\expandafter\XINT_expr_iter:_E\romannumeral`&&@#3#1~#2{#3}}%
1889 \def\XINT_expr_iter:_E #1{\if #1^\xint_dothis\XINT_expr_iter:_Abort\fi
1890     \if #1?\xint_dothis\XINT_expr_iter:_Break\fi
1891     \if #1!\xint_dothis\XINT_expr_iter:_Omit\fi
1892     \xint_orthat{\XINT_expr_iter:_Goon #1}%
1893 \def\XINT_expr_iter:_Goon #1!#2#3~#4#5%
1894     {\expandafter\XINT_expr_iter:_D
1895     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1896     \romannumeral0\XINT_expr_lockit{#1}{#5}}%
```

```

1897 \def\XINT_expr_iter:_Omit #1!#2#3~%#4#5%
1898     {\expandafter\XINT_expr_iter:_D
1899       \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname }%
1900 \def\XINT_expr_iter:_Abort #1!#2#3~#4#5{\XINT_expr:_unlock #4}%
1901 \def\XINT_expr_iter:_Break #1!#2#3~#4#5{#1}%

```

## 11.49 \rrseq()

11.49.1	\XINT_expr_rrseqx . . . . .	349
11.49.2	\XINT_expr_rrseqy . . . . .	349
11.49.3	\XINT_expr_rrseq:_a etc. . . . .	350
11.49.4	\XINT_expr_rrseq:_A etc. . . . .	350

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.

```

1902 \def\XINT_expr_func_rrseq  {\XINT_allexpr_rrseq \xintbareeval      \xintthebareeval      }%
1903 \def\XINT_flexpr_func_rrseq {\XINT_allexpr_rrseq \xintbarefloateval \xintthebarefloateval }%
1904 \def\XINT_iiepr_func_rrseq  {\XINT_allexpr_rrseq \xintbareiieval   \xintthebareiieval   }%
1905 \def\XINT_allexpr_rrseq #1#2#3%
1906 {%
1907   \expandafter\XINT_expr_rrseqx\expandafter #1\expandafter#2\expandafter
1908   #3\romannumeral`&&\XINT_expr_onliteral_seq_a }%
1909 }%

```

### 11.49.1 \XINT\_expr\_rrseqx

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

```

1910 \def\XINT_expr_rrseqx #1#2#3#4#5%
1911 {%
1912   \expandafter\XINT_expr_rrseqy\romannumeral0#1(#5)\expandafter\relax
1913   \expandafter{\romannumeral0\xintapply \XINT_expr_lockit
1914     {\xintRevWithBraces{\xintCSVtoListNonStripped{\XINT_expr_unlock #3}}}}%
1915   #3#4#2%
1916 }%

```

### 11.49.2 \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

```

1917 \def\XINT_expr_rrseqy #1#2#3#4#5#6%
1918 {%
1919   \expandafter \XINT_expr_getop
1920   \csname .=\XINT_expr_unlock #3%
1921   \expandafter\XINT_expr_rrseq:_aa
1922     \romannumeral`&&\XINT_expr_unlock #1!{#6#5\relax !#4}{#2}\endcsname
1923 }%
1924 \def\XINT_expr_rrseq:_aa #1{\if +#1\expandafter\XINT_expr_rrseq:_A\else
1925   \expandafter\XINT_expr_rrseq:_a\fi #1}%

```

### 11.49.3 \XINT\_expr\_rrseq:\_a etc...

Attention que ? a catcode 3 ici et dans iter.

```

1926 \catcode`? 3
1927 \def\XINT_expr_rrseq:_a #1!#2#3{\XINT_expr_rrseq:_b {#3}{#2}#1,^,%}
1928 \def\XINT_expr_rrseq:_b #1#2#3#4,{%
1929     \if ,#3\xint_dothis\XINT_expr_rrseq:_noop\fi
1930     \if ^#3\xint_dothis\XINT_expr_rrseq:_end\fi
1931     \xint_orthat{\expandafter\XINT_expr_rrseq:_c}\csname.#3#4\endcsname
1932     {#1}{#2}%
1933 }%
1934 \def\XINT_expr_rrseq:_noop\csname.#,\#1\endcsname #2#3{\XINT_expr_rrseq:_b {#2}{#3}#1,%}
1935 \def\XINT_expr_rrseq:_end \csname.#^\endcsname #1#2{%}
1936 \def\XINT_expr_rrseq:_c #1#2#3%
1937     {\expandafter\XINT_expr_rrseq:_d\romannumeral`&&@#3#1~#2?{#3}}%
1938 \def\XINT_expr_rrseq:_d #1{%
1939     \if ^#1\xint_dothis\XINT_expr_rrseq:_abort\fi
1940     \if ?#1\xint_dothis\XINT_expr_rrseq:_break\fi
1941     \if !#1\xint_dothis\XINT_expr_rrseq:_omit\fi
1942     \xint_orthat{\XINT_expr_rrseq:_goon #1}%
1943 }%
1944 \def\XINT_expr_rrseq:_goon #1!#2#3~#4?#5{,#1\expandafter\XINT_expr_rrseq:_b\expandafter
1945     {\romannumeral0\xinttrim{-1}\XINT_expr_lockit{#1}{#4}}{#5}}%
1946 \def\XINT_expr_rrseq:_omit #1!#2#3~{\XINT_expr_rrseq:_b }%
1947 \def\XINT_expr_rrseq:_abort #1!#2#3~#4?#5#6^,{}%
1948 \def\XINT_expr_rrseq:_break #1!#2#3~#4?#5#6^,{,#1}%

```

### 11.49.4 \XINT\_expr\_rrseq:\_A etc...

n++ for rrseq. With 1.2C, the #1 in \XINT\_expr\_rrseq:\_A is a single token.

```

1949 \def\XINT_expr_rrseq:_A +#1!#2#3{\XINT_expr_rrseq:_D #1{#3}{#2}}%
1950 \def\XINT_expr_rrseq:_D #1#2#3%
1951     {\expandafter\XINT_expr_rrseq:_E\romannumeral`&&@#3#1~#2?{#3}}%
1952 \def\XINT_expr_rrseq:_Goon #1!#2#3~#4?#5%
1953     {,#1\expandafter\XINT_expr_rrseq:_D
1954     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1955     \expandafter{\romannumeral0\xinttrim{-1}\XINT_expr_lockit{#1}{#4}}{#5}}%
1956 \def\XINT_expr_rrseq:_Omit #1!#2#3~#4?#5%
1957     {\expandafter\XINT_expr_rrseq:_D
1958     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
1959 \def\XINT_expr_rrseq:_Abort #1!#2#3~#4?#5{%}
1960 \def\XINT_expr_rrseq:_Break #1!#2#3~#4?#5{,#1}%
1961 \def\XINT_expr_rrseq:_E #1{\if #1^\xint_dothis\XINT_expr_rrseq:_Abort\fi
1962     \if #1?\xint_dothis\XINT_expr_rrseq:_Break\fi
1963     \if #1!\xint_dothis\XINT_expr_rrseq:_Omit\fi
1964     \xint_orthat{\XINT_expr_rrseq:_Goon #1}}%

```

## 11.50 \iterr()

11.50.1	\XINT_expriterrx . . . . .	351
11.50.2	\XINT_expriterrry . . . . .	351

11.50.3	\XINT_expr_iterr:_a etc...	351
11.50.4	\XINT_expr_iterr:_A etc...	352

```

1965 \def\XINT_expr_func_iterr  {\XINT_allexpr_iterr \xintbareeval      \xintthebareeval      }%
1966 \def\XINT_flexpr_func_iterr {\XINT_allexpr_iterr \xintbarefloateval \xintthebarefloateval }%
1967 \def\XINT_iiexpr_func_iterr {\XINT_allexpr_iterr \xintbareiieval   \xintthebareiieval   }%
1968 \def\XINT_allexpr_iterr #1#2#3%
1969 {%
1970   \expandafter\XINT_expr_iterrx\expandafter #1\expandafter #2\expandafter
1971   #3\romannumeral`&&\XINT_expr_onliteral_seq_a }%
1972 }%

```

### 11.50.1 \XINT\_expr\_iterrx

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

```

1973 \def\XINT_expr_iterrx #1#2#3#4#5%
1974 {%
1975   \expandafter\XINT_expr_iterry\romannumeral0#1(#5)\expandafter\relax
1976   \expandafter{\romannumeral0\xintapply \XINT_expr_lockit
1977     {\xintRevWithBraces{\xintCSVtoListNonStripped{\XINT_expr_unlock #3}}}%
1978   #3#4#2%
1979 }%

```

### 11.50.2 \XINT\_expr\_iterry

#1=valeurs pour variable (locked), #2=initial values (reversed, one (braced) token each) #3=toutes les valeurs initiales (csv,locked), #4=variable, #5=expr, #6=\xintbareeval ou \xintbarefloateval ou \xintbareiieval

```

1980 \def\XINT_expr_iterry #1#2#3#4#5#6%
1981 {%
1982   \expandafter \XINT_expr_getop
1983   \csname .=%
1984   \expandafter\XINT_expr_iterr:_aa
1985   \romannumeral`&&\XINT_expr_unlock #1!{#6#5\relax !#4}{#2}\endcsname
1986 }%
1987 \def\XINT_expr_iterr:_aa #1{\if +#1\expandafter\XINT_expr_iterr:_A\else
1988   \expandafter\XINT_expr_iterr:_a\fi #1}%

```

### 11.50.3 \XINT\_expr\_iterr:\_a etc...

```

1989 \def\XINT_expr_iterr:_a #1!#2#3{\XINT_expr_iterr:_b {#3}{#2}#1,^,}%
1990 \def\XINT_expr_iterr:_b #1#2#3#4,{%
1991   \if ,#3\xint_dothis\XINT_expr_iterr:_noop\fi
1992   \if ^#3\xint_dothis\XINT_expr_iterr:_end\fi
1993   \xint_orthat{\expandafter\XINT_expr_iterr:_c}%
1994   \csname.=#3#4\endcsname {#1}{#2}%
1995 }%
1996 \def\XINT_expr_iterr:_noop\csname.=#1\endcsname #2#3{\XINT_expr_iterr:_b {#2}{#3}#1,}%
1997 \def\XINT_expr_iterr:_end \csname.=^ \endcsname #1#2%
1998   {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1999   {,\XINT_expr:_unlock}{\xintReverseOrder{#1\space}}}%

```

```

2000 \def\XINT_expr_iterr:_c #1#2#3%
2001   {\expandafter\XINT_expr_iterr:_d\romannumeral`&&@#3#1~#2?{#3}}%
2002 \def\XINT_expr_iterr:_d #1{%
2003   \if ^#1\xint_dothis\XINT_expr_iterr:_abort\fi
2004   \if ?#1\xint_dothis\XINT_expr_iterr:_break\fi
2005   \if !#1\xint_dothis\XINT_expr_iterr:_omit\fi
2006   \xint_orthat{\XINT_expr_iterr:_goon #1}%
2007 }%
2008 \def\XINT_expr_iterr:_goon #1!#2#3~#4?#5{\expandafter\XINT_expr_iterr:_b\expandafter
2009   {\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
2010 \def\XINT_expr_iterr:_omit #1!#2#3~{\XINT_expr_iterr:_b }%
2011 \def\XINT_expr_iterr:_abort #1!#2#3~#4?#5#6^,%
2012   {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
2013     {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}}}%
2014 \def\XINT_expr_iterr:_break #1!#2#3~#4?#5#6^,%
2015   {\expandafter\xint_gobble_iv\romannumeral0\xintapplyunbraced
2016     {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}},#1}%
2017 \def\XINT_expr:_unlock #1{\XINT_expr_unlock #1}%

```

#### 11.50.4 \XINT\_expr\_iterr:\_A etc...

n++ for iterr. ? is of catcode 3 here.

```

2018 \def\XINT_expr_iterr:_A +#1!#2#3{\XINT_expr_iterr:_D #1{#3}{#2}}%
2019 \def\XINT_expr_iterr:_D #1#2#3%
2020   {\expandafter\XINT_expr_iterr:_E\romannumeral`&&@#3#1~#2?{#3}}%
2021 \def\XINT_expr_iterr:_Goon #1!#2#3~#4?#5%
2022   {\expandafter\XINT_expr_iterr:_D
2023     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
2024     \expandafter{\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
2025 \def\XINT_expr_iterr:_Omit #1!#2#3~#4?#5%
2026   {\expandafter\XINT_expr_iterr:_D
2027     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
2028 \def\XINT_expr_iterr:_Abort #1!#2#3~#4?#5%
2029   {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
2030     {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}}}%
2031 \def\XINT_expr_iterr:_Break #1!#2#3~#4?#5%
2032   {\expandafter\xint_gobble_iv\romannumeral0\xintapplyunbraced
2033     {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}},#1}%
2034 \def\XINT_expr_iterr:_E #1{\if #1^\xint_dothis\XINT_expr_iterr:_Abort\fi
2035   \if #1?\xint_dothis\XINT_expr_iterr:_Break\fi
2036   \if #1!\xint_dothis\XINT_expr_iterr:_Omit\fi
2037   \xint_orthat{\XINT_expr_iterr:_Goon #1}}%
2038 \catcode`? 11

```

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

11.51.1	\xintANDof:csv	353
11.51.2	\xintORof:csv	353
11.51.3	\xintXORof:csv	353
11.51.4	Generic csv routine (\XINT_oncsv:_a)	354
11.51.5	\xintMaxof:csv, \xintiiMaxof:csv	354
11.51.6	\xintMinof:csv, \xintiiMinof:csv	354



11.51.7	\xintSum:csv, \xintiiSum:csv . . . . .	354
11.51.8	\xintPrd:csv, \xintiiPrd:csv . . . . .	355
11.51.9	\xintGCDof:csv, \xintLCMof:csv . . . . .	355
11.51.10	\xintiiGCDof:csv, \xintiiLCMof:csv . . . . .	356
11.51.11	\XINTinFloatdigits, \XINTinFloatSqrtdigits, \XINTinFloatFacdigits, \XINTiLog-Tendigits . . . . .	356
11.51.12	\XINTinFloatMaxof:csv, \XINTinFloatMinof:csv . . . . .	357
11.51.13	\XINTinFloatSum:csv, \XINTinFloatPrd:csv . . . . .	357

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)

### 11.51.1 \xintANDof:csv

1.09a. For use by `\xintexpr` inside `\csname. 1.1`, je remplace `ifTrueAelseB` par `iiNotZero` pour des raisons d'optimisations.

```
2039 \def\xintANDof:csv #1{\expandafter\XINT_andof:_a\romannumeral`&&@#1,,^}%
2040 \def\XINT_andof:_a #1{\if ,#1\expandafter\XINT_andof:_e
2041         \else\expandafter\XINT_andof:_c\fi #1}%
2042 \def\XINT_andof:_c #1,{\xintiiifNotZero {#1}{\XINT_andof:_a}{\XINT_andof:_no}}%
2043 \def\XINT_andof:_no #1^{0}%
2044 \def\XINT_andof:_e #1^{1}% works with empty list
```

### 11.51.2 \xintORof:csv

1.09a. For use by `\xintexpr`.

```
2045 \def\xintORof:csv #1{\expandafter\XINT_orof:_a\romannumeral`&&@#1,,^}%
2046 \def\XINT_orof:_a #1{\if ,#1\expandafter\XINT_orof:_e
2047         \else\expandafter\XINT_orof:_c\fi #1}%
2048 \def\XINT_orof:_c #1,{\xintiiifNotZero{#1}{\XINT_orof:_yes}{\XINT_orof:_a}}%
2049 \def\XINT_orof:_yes #1^{1}%
2050 \def\XINT_orof:_e #1^{0}% works with empty list
```

### 11.51.3 \xintXORof:csv

1.09a. For use by `\xintexpr` (inside a `\csname...\endcsname`).

```
2051 \def\xintXORof:csv #1{\expandafter\XINT_xorof:_a\expandafter 0\romannumeral`&&@#1,,^}%
2052 \def\XINT_xorof:_a #1#2,{\XINT_xorof:_b #2,#1}%
2053 \def\XINT_xorof:_b #1{\if ,#1\expandafter\XINT_xorof:_e
2054         \else\expandafter\XINT_xorof:_c\fi #1}%
2055 \def\XINT_xorof:_c #1,#2%
2056         {\xintiiifNotZero {#1}{\if #20\xint_afterfi{\XINT_xorof:_a 1}%
2057                 \else\xint_afterfi{\XINT_xorof:_a 0}\fi}%
2058         {\XINT_xorof:_a #2}%
2059         }%
2060 \def\XINT_xorof:_e ,#1#2^{#1}% allows empty list (then returns 0)
```

#### 11.51.4 Generic csv routine (\XINT\_oncsv:\_a)

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.

FIXME: DOCUMENT BETTER. HOW IS THIS CALLED? WHAT IS MEANING OF ARGUMENTS? IS THERE ANY POST-PROCESSING OF FINAL RESULT?

```

2061 \def\XINT_oncsv:_empty #1,^,#2{#2}%
2062 \def\XINT_oncsv:_end ^,#1#2#3#4{#1}%
2063 \def\XINT_oncsv:_a #1#2#3%
2064 {\if ,#3\expandafter\XINT_oncsv:_empty\else\expandafter\XINT_oncsv:_b\fi #1#2#3}%
2065 \def\XINT_oncsv:_b #1#2#3,%
2066 {\expandafter\XINT_oncsv:_c \expandafter{\romannumeral`&&@#2{#3}}#1#2}%
2067 \def\XINT_oncsv:_c #1#2#3#4,{\expandafter\XINT_oncsv:_d \romannumeral`&&@#4,{#1}#2#3}%
2068 \def\XINT_oncsv:_d #1%
2069 {\if ^#1\expandafter\XINT_oncsv:_end\else\expandafter\XINT_oncsv:_e\fi #1}%
2070 \def\XINT_oncsv:_e #1,#2#3#4%
2071 {\expandafter\XINT_oncsv:_c\expandafter {\romannumeral`&&@#3{#4{#1}}{#2}}#3#4}%

```

#### 11.51.5 \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.

```

2072 \def\xintMaxof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmax
2073 \expandafter\xint_firstofone\romannumeral`&&@#1,^,{0/1[0]}}%
2074 \def\xintiiMaxof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiimax
2075 \expandafter\xint_firstofone\romannumeral`&&@#1,^,0}%

```

#### 11.51.6 \xintMinof:csv, \xintiiMinof:csv

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

```

2076 \def\xintMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmin
2077 \expandafter\xint_firstofone\romannumeral`&&@#1,^,{0/1[0]}}%
2078 \def\xintiiMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiimin
2079 \expandafter\xint_firstofone\romannumeral`&&@#1,^,0}%

```

#### 11.51.7 \xintSum:csv, \xintiiSum:csv

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

```

2080 \def\xintSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintadd
2081 \expandafter\xint_firstofone\romannumeral`&&@#1,^,{0/1[0]}}%
2082 \def\xintiiSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiadd
2083 \expandafter\xint_firstofone\romannumeral`&&@#1,^,0}%

```

### 11.51.8 \xintPrd:csv, \xintiiPrd:csv

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

```
2084 \def\xintPrd:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmul
2085         \expandafter\xint_firstofone\romannumeral`&&@#1,^{1/1[0]}}%
2086 \def\xintiiPrd:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiimul
2087         \expandafter\xint_firstofone\romannumeral`&&@#1,^,1}%
```

### 11.51.9 \xintGCDof:csv, \xintLCMof:csv

1.09a. Non-integer arguments are replaced by integers as \xintGCD and \xintLCM apply \xintNum.

1.1. As with other "csv" macros, the (list) argument needs to be expanded in case it arises within a macro created from \xintNewExpr.

1.3d. No more usage of the integer-only xintgcd macros, replaced by direct coding here, in order to extend scope to fractions (and produce fractions). Hesitation about allowing empty input, and what to return then.

```
2088 \def\xintGCDof:csv #1{\expandafter\XINT_gcdof:_a\romannumeral`&&@#1,^{1/1[0]}}%
2089 \def\XINT_gcdof:_a #1%
2090   {\if ,#1\expandafter\XINT_oncsv:_empty\else\expandafter\XINT_gcdof:_b\fi #1}%
   This abuses the way \xintiiabs works in order to avoid fetching whole argument again.
2091 \def\XINT_gcdof:_b #1,%
2092   {\expandafter\XINT_gcdof:_c\romannumeral0\xintiiabs#1\xint:}%
2093 \def\XINT_gcdof:_c #1\xint:#2,%
2094   {\expandafter\XINT_gcdof:_d\romannumeral0\xintiiabs#2\xint:#1\xint:}%
2095 \def\XINT_gcdof:_d #1%
2096   {\if ^#1\expandafter\XINT_gcdof:_end\else\expandafter\XINT_gcdof:_e\fi #1}%
```

\xintMod will apply \xintRaw on its arguments, and will output in normalized format. But in exceptional case with a one-item or one item and then zeros, the output is (absolute value of) this item, not necessarily in A/B[N] format.

```
2097 \def\XINT_gcdof:_e#1#2\xint:#3\xint:
2098 {%
2099   \if0#1\expandafter\XINT_gcdof:_f\fi
2100   \expandafter\XINT_gcdof:_e\romannumeral0\xintmod{#3}{#1#2}\xint:#1#2\xint:
2101 }%
2102 \def\XINT_gcdof:_f
2103   \expandafter\XINT_gcdof:_e\romannumeral0\xintmod#1#2\xint:#3\xint:#4,%
2104 {%
2105   \expandafter\XINT_gcdof:_d\romannumeral0\xintiiabs#4\xint:#1\xint:
2106 }%
```

As for others :csv macros here expansion in the case of \xintNewExpr crafted macros is triggered by (an equivalent to) \romannumeral-`0. Else it happens inside \cname...\endcname, and there is no triggering \romannumeral, attention to not leave a space upfront.

```
2107 \def\XINT_gcdof:_end ^\xint:#1\xint:#2{#1}%
```

For least common multiple, we will use \xintInv, but this requires to make sure fractional input is in raw format.

```
2108 \def\xintLCMof:csv #1{\expandafter\XINT_lcmof:_a\romannumeral`&&@#1,^{0/1[0]}}%
```

```

2109 \def\XINT_lcmof:_a #1%
2110   {\if ,#1\expandafter\XINT_oncsv:_empty\else\expandafter\XINT_lcmof:_b\fi #1}%
2111 \def\XINT_lcmof:_b #1,%
2112   {\expandafter\XINT_lcmof:_c\romannumeral0\xintiiabs\xintRaw{#1}\xint:}%
2113 \def\XINT_lcmof:_c #1{\if0#1\expandafter\XINT_lcmof:_zero\fi
2114   \expandafter\XINT_lcmof:_d\romannumeral0\XINT_inv #1}%

```

We can do `\xintiiabs^`, but `\xintiiabs\xintRaw{^}` would throw an error. So we need to delay applying `\xintRaw` to new item.

```

2115 \def\XINT_lcmof:_d #1\xint:#2,%
2116   {\expandafter\XINT_lcmof:_e\romannumeral0\xintiiabs#2\xint:#1\xint:}%
2117 \def\XINT_lcmof:_e #1%
2118   {\if ^#1\expandafter\XINT_lcmof:_end\else\expandafter\XINT_lcmof:_f\fi #1}%

```

As soon as we hit against a zero item, the l.c.m is known to be zero itself. Else we need to inverse it, but this requires full A/B[N] raw format, hence the `\xintraw`.

```

2119 \def\XINT_lcmof:_f#1#2\xint:
2120 {%
2121   \if0#1\expandafter\XINT_lcmof:_zero\fi
2122   \expandafter\XINT_lcmof:_g\romannumeral0\expandafter\XINT_inv
2123   \romannumeral0\xintraw{#1#2}\xint:
2124 }%
2125 \def\XINT_lcmof:_g #1#2\xint:#3\xint:
2126 {%
2127   \if0#1\expandafter\XINT_lcmof:_h\fi
2128   \expandafter\XINT_lcmof:_g\romannumeral0\xintmod{#3}{#1#2}\xint:#1#2\xint:
2129 }%
2130 \def\XINT_lcmof:_h
2131   \expandafter\XINT_lcmof:_g\romannumeral0\xintmod#1#2\xint:#3\xint:#4,%
2132 {%
2133   \expandafter\XINT_lcmof:_e\romannumeral0\xintiiabs#4\xint:#1\xint:
2134 }%
2135 \def\XINT_lcmof:_zero #1^,#2{0/1[0]}%

```

We need this `\romannumeral0` to remove the up-front space token which will be left by `\XINT_inv`, in case of `\csname.. \endcsname` expansion.

```

2136 \def\XINT_lcmof:_end ^\xint:#1\xint:#2{\romannumeral0\XINT_inv #1}%

```

### 11.51.10 `\xintiiGCDoF:csv`, `\xintiiLCMoF:csv`

**1.1a.** For `\xintiiexpr`. Requires the `xintgcd` provided macros.

```

2137 \def\xintiiGCDoF:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiigcd
2138   \expandafter\xint_firstofone\romannumeral`&&@#1,^,1}%
2139 \def\xintiiLCMoF:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiLCM
2140   \expandafter\xint_firstofone\romannumeral`&&@#1,^,0}%

```

### 11.51.11 `\XINTinFloatdigits`, `\XINTinFloatSqrtdigits`, `\XINTinFloatFacdigits`, `\XINTiLogTendigits`

For `\xintNewExpr` matters, mainly.

At 1.3e I add `\XINTinFloatSdigits` and use it at various places. I also modified `\XINTinFloatFac` to use S(hort) output format.

Also added `\XINTiLogTendigits`

```
2141 \def\XINTinFloatdigits {\XINTinFloat [\XINTdigits]}%
2142 \def\XINTinFloatSdigits {\XINTinFloatS [\XINTdigits]}%
2143 \def\XINTinFloatSqrtdigits {\XINTinFloatSqrt[\XINTdigits]}%
2144 \def\XINTinFloatFacdigits {\XINTinFloatFac [\XINTdigits]}%
2145 \def\XINTFloatiLogTendigits{\XINTFloatiLogTen[\XINTdigits]}%
```

### 11.51.12 `\XINTinFloatMaxof:csv`, `\XINTinFloatMinof:csv`

1.09a. Rewritten for 1.1. For use by `\xintfloatexpr`. Name changed in 1.09h. Changed at 1.3e to use `\XINTinFloatSdigits`.

```
2146 \def\XINTinFloatMaxof:csv #1{\expandafter\XINT_oncsvg:_a\expandafter\xintmax
2147 \expandafter\XINTinFloatSdigits\romannumeral`&&@#1,^{0[0]}}%
2148 \def\XINTinFloatMinof:csv #1{\expandafter\XINT_oncsvg:_a\expandafter\xintmin
2149 \expandafter\XINTinFloatSdigits\romannumeral`&&@#1,^{0[0]}}%
```

### 11.51.13 `\XINTinFloatSum:csv`, `\XINTinFloatPrd:csv`

1.09a. Rewritten for 1.1. For use by `\xintfloatexpr`. Modified at 1.3e to use `\XINTinFloatSdigits`.

```
2150 \def\XINTinFloatSum:csv #1{\expandafter\XINT_oncsvg:_a\expandafter\XINTinfloatadd
2151 \expandafter\XINTinFloatSdigits\romannumeral`&&@#1,^{0[0]}}%
2152 \def\XINTinFloatPrd:csv #1{\expandafter\XINT_oncsvg:_a\expandafter\XINTinfloatmul
2153 \expandafter\XINTinFloatSdigits\romannumeral`&&@#1,^{1[0]}}%
```

## 11.52 Auxiliary wrappers for function macros

```
2154 \def\XINT:expr:one:and:opt #1,#2,#3!#4#5%
2155 {%
2156 \ifrelax#3\relax\expandafter\xint_firstoftwo\else
2157 \expandafter\xint_secondoftwo\fi
2158 {#4}{#5[\xintNum{#2}]}{#1}%
2159 }%
2160 \def\XINT:expr:tacitzeroifonearg #1,#2,#3!#4#5%
2161 {%
2162 \ifrelax#3\relax\expandafter\xint_firstoftwo\else
2163 \expandafter\xint_secondoftwo\fi
2164 {#4{0}}{#5{\xintNum{#2}}}{#1}%
2165 }%
2166 \def\XINT:iiexpr:tacitzeroifonearg #1,#2,#3!#4%
2167 {%
2168 \ifrelax#3\relax\expandafter\xint_firstoftwo\else
2169 \expandafter\xint_secondoftwo\fi
2170 {#4{0}}{#4{#2}}{#1}%
2171 }%
2172 \def\XINT:expr:totwo #1#2{#1,#2}%
2173 \def\XINT:expr:two:to:two #1,#2,!#3%
2174 {%
2175 \expandafter\XINT:expr:totwo\romannumeral`&&@#3{#1}{#2}%
2176 }%
2177 \let\XINT:flexpr:two:to:two\XINT:expr:two:to:two
```

2178 \let\XINT:iiexpr:two:to:two\XINT:expr:two:to:two

**11.53 The num(), reduce(), preduce(), abs(), sgn(), frac(), floor(), ceil(),  
 sqr(), sqrt(), sqrtr(), float(), sfloat(), ilog10(), inv(), round(),  
 trunc(), mod(), quo(), rem(), divmod(), gcd(), lcm(), max(), min(), `+`(),  
 `\*`(), ?(), !(), not(), all(), any(), xor(), if(), ifsgn(), ifint(), ifone(),  
 even(), odd(), isint(), isone(), first(), last(), len(), reversed(),  
 factorial(), binomial() and randrange() functions**

```

2179 \def\XINT_expr_func_num #1#2#3%
2180 {%
2181     \expandafter #1\expandafter #2\csname.=%
2182     \XINT:NEhook:one\xintNum{\XINT_expr_unlock #3}\endcsname
2183 }%
2184 \let\XINT_flexpr_func_num\XINT_expr_func_num
2185 \let\XINT_iiexpr_func_num\XINT_expr_func_num
2186 \def\XINT_expr_func_reduce #1#2#3%
2187 {%
2188     \expandafter #1\expandafter #2\csname.=%
2189     \XINT:NEhook:one\xintIrr{\XINT_expr_unlock #3}[0]\endcsname
2190 }%
2191 \let\XINT_flexpr_func_reduce\XINT_expr_func_reduce
2192 \def\XINT_expr_func_preduce #1#2#3%
2193 {%
2194     \expandafter #1\expandafter #2\csname.=%
2195     \XINT:NEhook:one\xintPIrr{\XINT_expr_unlock #3}\endcsname
2196 }%
2197 \let\XINT_flexpr_func_preduce\XINT_expr_func_preduce
2198 \def\XINT_expr_func_abs #1#2#3%
2199 {%
2200     \expandafter #1\expandafter #2\csname.=%
2201     \XINT:NEhook:one\xintAbs{\XINT_expr_unlock #3}\endcsname
2202 }%
2203 \let\XINT_flexpr_func_abs\XINT_expr_func_abs
2204 \def\XINT_iiexpr_func_abs #1#2#3%
2205 {%
2206     \expandafter #1\expandafter #2\csname.=%
2207     \XINT:NEhook:one\xintiiAbs{\XINT_expr_unlock #3}\endcsname
2208 }%
2209 \def\XINT_expr_func_sgn #1#2#3%
2210 {%
2211     \expandafter #1\expandafter #2\csname.=%
2212     \XINT:NEhook:one\xintSgn{\XINT_expr_unlock #3}\endcsname
2213 }%
2214 \let\XINT_flexpr_func_sgn\XINT_expr_func_sgn
2215 \def\XINT_iiexpr_func_sgn #1#2#3%
2216 {%
2217     \expandafter #1\expandafter #2\csname.=%
2218     \XINT:NEhook:one\xintiiSgn{\XINT_expr_unlock #3}\endcsname
2219 }%
2220 \def\XINT_expr_func_frac #1#2#3%
2221 {%
2222     \expandafter #1\expandafter #2\csname.=%

```

```

2223 \XINT:NEhook:one\xintTFrac{\XINT_expr_unlock #3}\endcsname
2224 }%
2225 \def\XINT_flexpr_func_frac #1#2#3%
2226 {%
2227 \expandafter #1\expandafter #2\csname.=%
2228 \XINT:NEhook:one\XINTinFloatFracdigits{\XINT_expr_unlock #3}\endcsname
2229 }%

```

no \XINT\_iiexpr\_func\_frac

```

2230 \def\XINT_expr_func_floor #1#2#3%
2231 {%
2232 \expandafter #1\expandafter #2\csname.=%
2233 \XINT:NEhook:one\xintFloor{\XINT_expr_unlock #3}\endcsname
2234 }%
2235 \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.

```

2236 \def\XINT_iiexpr_func_floor #1#2#3%
2237 {%
2238 \expandafter #1\expandafter #2\csname.=%
2239 \XINT:NEhook:one\xintiFloor{\XINT_expr_unlock #3}\endcsname
2240 }%
2241 \def\XINT_expr_func_ceil #1#2#3%
2242 {%
2243 \expandafter #1\expandafter #2\csname.=%
2244 \XINT:NEhook:one\xintCeil{\XINT_expr_unlock #3}\endcsname
2245 }%
2246 \let\XINT_flexpr_func_ceil\XINT_expr_func_ceil
2247 \def\XINT_iiexpr_func_ceil #1#2#3%
2248 {%
2249 \expandafter #1\expandafter #2\csname.=%
2250 \XINT:NEhook:one\xintiCeil{\XINT_expr_unlock #3}\endcsname
2251 }%
2252 \def\XINT_expr_func_sqr #1#2#3%
2253 {%
2254 \expandafter #1\expandafter #2\csname.=%
2255 \XINT:NEhook:one\xintSqr{\XINT_expr_unlock #3}\endcsname
2256 }%
2257 \def\XINTinFloatSqr#1{\XINTinFloatMul{#1}{#1}}% revoir après
2258 \def\XINT_flexpr_func_sqr #1#2#3%
2259 {%
2260 \expandafter #1\expandafter #2\csname.=%
2261 \XINT:NEhook:one\XINTinFloatSqr{\XINT_expr_unlock #3}\endcsname
2262 }%
2263 \def\XINT_iiexpr_func_sqr #1#2#3%
2264 {%
2265 \expandafter #1\expandafter #2\csname.=%
2266 \XINT:NEhook:one\xintiiSqr{\XINT_expr_unlock #3}\endcsname
2267 }%
2268 \def\XINT_expr_func_? #1#2#3%
2269 {%
2270 \expandafter #1\expandafter #2\csname.=%

```

```

2271 \XINT:NEhook:one\xintiiIsNotZero{\XINT_expr_unlock #3}\endcsname
2272 }%
2273 \let\XINT_flexpr_func_? \XINT_expr_func_?
2274 \let\XINT_iiexpr_func_? \XINT_expr_func_?
2275 \def\XINT_expr_func_! #1#2#3%
2276 {%
2277 \expandafter #1\expandafter #2\csname.=%
2278 \XINT:NEhook:one\xintiiIsZero{\XINT_expr_unlock #3}\endcsname
2279 }%
2280 \let\XINT_flexpr_func_! \XINT_expr_func_!
2281 \let\XINT_iiexpr_func_! \XINT_expr_func_!
2282 \def\XINT_expr_func_not #1#2#3%
2283 {%
2284 \expandafter #1\expandafter #2\csname.=%
2285 \XINT:NEhook:one\xintiiIsZero{\XINT_expr_unlock #3}\endcsname
2286 }%
2287 \let\XINT_flexpr_func_not \XINT_expr_func_not
2288 \let\XINT_iiexpr_func_not \XINT_expr_func_not
2289 \def\XINT_expr_func_odd #1#2#3%
2290 {%
2291 \expandafter #1\expandafter #2\csname.=%
2292 \XINT:NEhook:one\xintOdd{\XINT_expr_unlock #3}\endcsname
2293 }%
2294 \let\XINT_flexpr_func_odd\XINT_expr_func_odd
2295 \def\XINT_iiexpr_func_odd #1#2#3%
2296 {%
2297 \expandafter #1\expandafter #2\csname.=%
2298 \XINT:NEhook:one\xintiiOdd{\XINT_expr_unlock #3}\endcsname
2299 }%
2300 \def\XINT_expr_func_even #1#2#3%
2301 {%
2302 \expandafter #1\expandafter #2\csname.=%
2303 \XINT:NEhook:one\xintEven{\XINT_expr_unlock #3}\endcsname
2304 }%
2305 \let\XINT_flexpr_func_even\XINT_expr_func_even
2306 \def\XINT_iiexpr_func_even #1#2#3%
2307 {%
2308 \expandafter #1\expandafter #2\csname.=%
2309 \XINT:NEhook:one\xintiiEven{\XINT_expr_unlock #3}\endcsname
2310 }%
2311 \def\XINT_expr_func_isint #1#2#3%
2312 {%
2313 \expandafter #1\expandafter #2\csname.=%
2314 \XINT:NEhook:one\xintIsInt{\XINT_expr_unlock #3}\endcsname
2315 }%
2316 \def\XINT_flexpr_func_isint #1#2#3%
2317 {%
2318 \expandafter #1\expandafter #2\csname.=%
2319 \XINT:NEhook:one\xintFloatIsInt{\XINT_expr_unlock #3}\endcsname
2320 }%
2321 \let\XINT_iiexpr_func_isint\XINT_expr_func_isint % ? perhaps rather always 1
2322 \def\XINT_expr_func_ison #1#2#3%

```



```

2323 {%
2324   \expandafter #1\expandafter #2\csname.=%
2325   \XINT:NEhook:one\xintIsOne{\XINT_expr_unlock #3}\endcsname
2326 }%
2327 \let\XINT_flexpr_func_isone\XINT_expr_func_isone
2328 \def\XINT_iiexpr_func_isone #1#2#3%
2329 {%
2330   \expandafter #1\expandafter #2\csname.=%
2331   \XINT:NEhook:one\xintiiIsOne{\XINT_expr_unlock #3}\endcsname
2332 }%
2333 % REVOIR nuple
2334 \def\XINT_expr_func_nuple #1#2#3%
2335   {\expandafter #1\expandafter #2\csname.=\XINT_expr_unlock #3\endcsname }%
2336 \let\XINT_flexpr_func_nuple\XINT_expr_func_nuple
2337 \let\XINT_iiexpr_func_nuple\XINT_expr_func_nuple
2338 \def\XINT_expr_func_factorial #1#2#3%
2339 {%
2340   \expandafter #1\expandafter #2\csname.=%
2341   \expandafter\XINT:expr:one:and:opt
2342   \romannumeral`&&\XINT_expr_unlock#3,,!\xintFac\XINTinFloatFac
2343   \endcsname
2344 }%
2345 \def\XINT_flexpr_func_factorial #1#2#3%
2346 {%
2347   \expandafter #1\expandafter #2\csname.=%
2348   \expandafter\XINT:expr:one:and:opt
2349   \romannumeral`&&\XINT_expr_unlock#3,,!\XINTinFloatFacdigits\XINTinFloatFac
2350   \endcsname
2351 }%
2352 \def\XINT_iiexpr_func_factorial #1#2#3%
2353 {%
2354   \expandafter #1\expandafter #2\csname.=%
2355   \XINT:NEhook:one\xintiiFac{\XINT_expr_unlock #3}\endcsname
2356 }%
2357 \def\XINT_expr_func_sqrt #1#2#3%
2358 {%
2359   \expandafter #1\expandafter #2\csname.=%
2360   \expandafter\XINT:expr:one:and:opt
2361   \romannumeral`&&\XINT_expr_unlock#3,,!\XINTinFloatSqrtdigits\XINTinFloatSqrt
2362   \endcsname
2363 }%
2364 \let\XINT_flexpr_func_sqrt\XINT_expr_func_sqrt
2365 \def\XINT_iiexpr_func_sqrt #1#2#3%
2366 {%
2367   \expandafter #1\expandafter #2\csname.=%
2368   \XINT:NEhook:one\xintiiSqrt{\XINT_expr_unlock #3}\endcsname
2369 }%
2370 \def\XINT_iiexpr_func_sqrtr #1#2#3%
2371 {%
2372   \expandafter #1\expandafter #2\csname.=%
2373   \XINT:NEhook:one\xintiiSqrtr{\XINT_expr_unlock #3}\endcsname
2374 }%

```

```

2375 \def\XINT_expr_func_inv #1#2#3%
2376 {%
2377   \expandafter #1\expandafter #2\csname.=%
2378   \XINT:NEhook:one\xintInv{\XINT_expr_unlock #3}\endcsname
2379 }%
2380 \def\XINT_flexpr_func_inv #1#2#3%
2381 {%
2382   \expandafter #1\expandafter #2\csname.=%
2383   \XINT:NEhook:one\XINTinFloatInv{\XINT_expr_unlock #3}\endcsname
2384 }%
2385 \def\XINT_expr_func_round #1#2#3%
2386 {%
2387   \expandafter #1\expandafter #2\csname.=%
2388   \expandafter\XINT:expr:tacitzeroifonearg
2389   \romannumeral`&&\XINT_expr_unlock #3,,!\xintiRound\xintRound
2390   \endcsname
2391 }%
2392 \let\XINT_flexpr_func_round\XINT_expr_func_round
2393 \def\XINT_iiexpr_func_round #1#2#3%
2394 {%
2395   \expandafter #1\expandafter #2\csname.=%
2396   \expandafter\XINT:iiexpr:tacitzeroifonearg
2397   \romannumeral`&&\XINT_expr_unlock #3,,!\xintiRound
2398   \endcsname
2399 }%
2400 \def\XINT_expr_func_trunc #1#2#3%
2401 {%
2402   \expandafter #1\expandafter #2\csname.=%
2403   \expandafter\XINT:expr:tacitzeroifonearg
2404   \romannumeral`&&\XINT_expr_unlock #3,,!\xintiTrunc\xintTrunc
2405   \endcsname
2406 }%
2407 \let\XINT_flexpr_func_trunc\XINT_expr_func_trunc
2408 \def\XINT_iiexpr_func_trunc #1#2#3%
2409 {%
2410   \expandafter #1\expandafter #2\csname.=%
2411   \expandafter\XINT:iiexpr:tacitzeroifonearg
2412   \romannumeral`&&\XINT_expr_unlock #3,,!\xintiTrunc
2413   \endcsname
2414 }%

```

Hesitation at 1.3e about using `\XINTinFloatSdigits` and `\XINTinFloatS`. Finally I add a `sfloat()` function. It helps for `xinttrig.sty`.

```

2415 \def\XINT_expr_func_float #1#2#3%
2416 {%
2417   \expandafter #1\expandafter #2\csname.=%
2418   \expandafter\XINT:expr:one:and:opt
2419   \romannumeral`&&\XINT_expr_unlock #3,,!\XINTinFloatdigits\XINTinFloat
2420   \endcsname
2421 }%
2422 \let\XINT_flexpr_func_float\XINT_expr_func_float
2423 \def\XINT_expr_func_sfloat #1#2#3%
2424 {%

```

```

2425 \expandafter #1\expandafter #2\csname.=%
2426 \expandafter\XINT:expr:one:and:opt
2427 \romannumeral`&&\XINT_expr_unlock #3,,!\XINTinFloatSdigits\XINTinFloatS
2428 \endcsname
2429 }%
2430 \let\XINT_flexpr_func_sfloat\XINT_expr_func_sfloat
2431 % \XINT_iiexpr_func_sfloat not defined
2432 \expandafter\def\csname XINT_expr_func_ilog10\endcsname #1#2#3%
2433 {%
2434 \expandafter #1\expandafter #2\csname.=%
2435 \expandafter\XINT:expr:one:and:opt
2436 \romannumeral`&&\XINT_expr_unlock #3,,!\xintiLogTen\XINTFloatiLogTen
2437 \endcsname
2438 }%
2439 \expandafter\def\csname XINT_flexpr_func_ilog10\endcsname #1#2#3%
2440 {%
2441 \expandafter #1\expandafter #2\csname.=%
2442 \expandafter\XINT:expr:one:and:opt
2443 \romannumeral`&&\XINT_expr_unlock #3,,!\XINTFloatiLogTendigits\XINTFloatiLogTen
2444 \endcsname
2445 }%
2446 \expandafter\def\csname XINT_iiexpr_func_ilog10\endcsname #1#2#3%
2447 {%
2448 \expandafter #1\expandafter #2\csname.=%
2449 \XINT:NEhook:one\xintiLogTen{\XINT_expr_unlock #3}\endcsname
2450 }%
2451 \def\XINT_expr_func_divmod #1#2#3%
2452 {%
2453 \expandafter #1\expandafter #2\csname.=%
2454 \expandafter\XINT:expr:two:to:two
2455 \romannumeral`&&\XINT_expr_unlock #3,!\xintDivMod
2456 \endcsname
2457 }%

```

\XINTinFloatDivMod a un output déjà comma separated

```

2458 \def\XINT_flexpr_func_divmod #1#2#3%
2459 {%
2460 \expandafter #1\expandafter #2\csname.=%
2461 \expandafter\XINT:NEhook:twosp
2462 \romannumeral`&&\XINT_expr_unlock #3,!\XINTinFloatDivMod
2463 \endcsname
2464 }%
2465 \def\XINT_iiexpr_func_divmod #1#2#3%
2466 {%
2467 \expandafter #1\expandafter #2\csname.=%
2468 \expandafter\XINT:expr:two:to:two
2469 \romannumeral`&&\XINT_expr_unlock #3,!\xintiDivMod
2470 \endcsname
2471 }%
2472 \def\XINT_expr_func_mod #1#2#3%
2473 {%
2474 \expandafter #1\expandafter #2\csname.=%
2475 \expandafter\XINT:NEhook:twosp

```

```

2476 \romannumeral`&&\XINT_expr_unlock #3,!\xintMod
2477 \endcsname
2478 }%
2479 \def\XINT_flexpr_func_mod #1#2#3%
2480 {%
2481 \expandafter #1\expandafter #2\csname.=%
2482 \expandafter\XINT:NEhook:tosp
2483 \romannumeral`&&\XINT_expr_unlock #3,!\XINTinFloatMod
2484 \endcsname
2485 }%
2486 \def\XINT_iiexpr_func_mod #1#2#3%
2487 {%
2488 \expandafter #1\expandafter #2\csname.=%
2489 \expandafter\XINT:NEhook:tosp
2490 \romannumeral`&&\XINT_expr_unlock #3,!\xintiiMod
2491 \endcsname
2492 }%
2493 \def\XINT_expr_func_binomial #1#2#3%
2494 {%
2495 \expandafter #1\expandafter #2\csname.=%
2496 \expandafter\XINT:NEhook:tosp
2497 \romannumeral`&&\XINT_expr_unlock #3,!\xintBinomial
2498 \endcsname
2499 }%
2500 \def\XINT_flexpr_func_binomial #1#2#3%
2501 {%
2502 \expandafter #1\expandafter #2\csname.=%
2503 \expandafter\XINT:NEhook:tosp
2504 \romannumeral`&&\XINT_expr_unlock #3,!\XINTinFloatBinomial
2505 \endcsname
2506 }%
2507 \def\XINT_iiexpr_func_binomial #1#2#3%
2508 {%
2509 \expandafter #1\expandafter #2\csname.=%
2510 \expandafter\XINT:NEhook:tosp
2511 \romannumeral`&&\XINT_expr_unlock #3,!\xintiiBinomial
2512 \endcsname
2513 }%
2514 \def\XINT_expr_func_pfactorial #1#2#3%
2515 {%
2516 \expandafter #1\expandafter #2\csname.=%
2517 \expandafter\XINT:NEhook:tosp
2518 \romannumeral`&&\XINT_expr_unlock #3,!\xintPFactorial
2519 \endcsname
2520 }%
2521 \def\XINT_flexpr_func_pfactorial #1#2#3%
2522 {%
2523 \expandafter #1\expandafter #2\csname.=%
2524 \expandafter\XINT:NEhook:tosp
2525 \romannumeral`&&\XINT_expr_unlock #3,!\XINTinFloatPFactorial
2526 \endcsname
2527 }%

```

```

2528 \def\XINT_iiexpr_func_pfactorial #1#2#3%
2529 {%
2530   \expandafter #1\expandafter #2\csname.=%
2531   \expandafter\XINT:NEhook:twosp
2532   \romannumeral`&&\XINT_expr_unlock #3,!\xintiPFactorial
2533   \endcsname
2534 }%
2535 \def\XINT_expr_func_randrange #1#2#3%
2536 {%
2537   \expandafter #1\expandafter #2\csname.=%
2538   \expandafter\XINT:expr:randrange
2539   \romannumeral`&&\XINT_expr_unlock #3,,!%
2540   \endcsname
2541 }%
2542 \let\XINT_flexpr_func_randrange\XINT_expr_func_randrange
2543 \def\XINT_iiexpr_func_randrange #1#2#3%
2544 {%
2545   \expandafter #1\expandafter #2\csname.=%
2546   \expandafter\XINT:iiexpr:randrange
2547   \romannumeral`&&\XINT_expr_unlock #3,,!%
2548   \endcsname
2549 }%
2550 \def\XINT:expr:randrange #1,#2,#3!%
2551 {%
2552   \if\relax#3\relax\expandafter\xint_firstoftwo\else
2553     \expandafter\xint_secondoftwo\fi
2554   {\xintiiRandRange{\XINT:NEhook:one\xintNum{#1}}}%
2555   {\xintiiRandRangeAtoB{\XINT:NEhook:one\xintNum{#1}}%
2556     {\XINT:NEhook:one\xintNum{#2}}}%
2557 }%
2558 \def\XINT:iiexpr:randrange #1,#2,#3!%
2559 {%
2560   \if\relax#3\relax\expandafter\xint_firstoftwo\else
2561     \expandafter\xint_secondoftwo\fi
2562   {\xintiiRandRange{#1}}{\xintiiRandRangeAtoB{#1}{#2}}%
2563 }%
2564 \def\XINT_expr_func_quo #1#2#3%
2565 {%
2566   \expandafter #1\expandafter #2\csname.=%
2567   \expandafter\XINT:NEhook:twosp
2568   \romannumeral`&&\XINT_expr_unlock #3,!\xintiQuo
2569   \endcsname
2570 }%
2571 \let\XINT_flexpr_func_quo\XINT_expr_func_quo
2572 \def\XINT_iiexpr_func_quo #1#2#3%
2573 {%
2574   \expandafter #1\expandafter #2\csname.=%
2575   \expandafter\XINT:NEhook:twosp
2576   \romannumeral`&&\XINT_expr_unlock #3,!\xintiQuo
2577   \endcsname
2578 }%
2579 \def\XINT_expr_func_rem #1#2#3%

```

```

2580 {%
2581   \expandafter #1\expandafter #2\csname.=%
2582   \expandafter\XINT:NEhook:twosp
2583   \romannumeral`&&\XINT_expr_unlock #3,! \xintiRem
2584   \endcsname
2585 }%
2586 \let\XINT_flexpr_func_rem\XINT_expr_func_rem
2587 \def\XINT_iiexpr_func_rem #1#2#3%
2588 {%
2589   \expandafter #1\expandafter #2\csname.=%
2590   \expandafter\XINT:NEhook:twosp
2591   \romannumeral`&&\XINT_expr_unlock #3,! \xintiiRem
2592   \endcsname
2593 }%
2594 \def\XINT_expr_func_gcd #1#2#3%
2595 {%
2596   \expandafter #1\expandafter #2\csname.=%
2597   \XINT:NEhook:csv\xintGCDoF:csv{\XINT_expr_unlock #3}\endcsname
2598 }%
2599 \let\XINT_flexpr_func_gcd\XINT_expr_func_gcd
2600 \def\XINT_iiexpr_func_gcd #1#2#3%
2601 {%
2602   \expandafter #1\expandafter #2\csname.=%
2603   \XINT:NEhook:csv\xintiiGCDoF:csv{\XINT_expr_unlock #3}\endcsname
2604 }%
2605 \def\XINT_expr_func_lcm #1#2#3%
2606 {%
2607   \expandafter #1\expandafter #2\csname.=%
2608   \XINT:NEhook:csv\xintLCMoF:csv{\XINT_expr_unlock #3}\endcsname
2609 }%
2610 \let\XINT_flexpr_func_lcm\XINT_expr_func_lcm
2611 \def\XINT_iiexpr_func_lcm #1#2#3%
2612 {%
2613   \expandafter #1\expandafter #2\csname.=%
2614   \XINT:NEhook:csv\xintiiLCMoF:csv{\XINT_expr_unlock #3}\endcsname
2615 }%
2616 \def\XINT_expr_func_max #1#2#3%
2617 {%
2618   \expandafter #1\expandafter #2\csname.=%
2619   \XINT:NEhook:csv\xintMaxoF:csv{\XINT_expr_unlock #3}\endcsname
2620 }%
2621 \def\XINT_iiexpr_func_max #1#2#3%
2622 {%
2623   \expandafter #1\expandafter #2\csname.=%
2624   \XINT:NEhook:csv\xintiiMaxoF:csv{\XINT_expr_unlock #3}\endcsname
2625 }%
2626 \def\XINT_flexpr_func_max #1#2#3%
2627 {%
2628   \expandafter #1\expandafter #2\csname.=%
2629   \XINT:NEhook:csv\XINTinFloatMaxoF:csv{\XINT_expr_unlock #3}\endcsname
2630 }%
2631 \def\XINT_expr_func_min #1#2#3%

```

```

2632 {%
2633   \expandafter #1\expandafter #2\csname.=%
2634   \XINT:NEhook:csv\xintMinof:csv{\XINT_expr_unlock #3}\endcsname
2635 }%
2636 \def\XINT_iiexpr_func_min #1#2#3%
2637 {%
2638   \expandafter #1\expandafter #2\csname.=%
2639   \XINT:NEhook:csv\xintiMinof:csv{\XINT_expr_unlock #3}\endcsname
2640 }%
2641 \def\XINT_flexpr_func_min #1#2#3%
2642 {%
2643   \expandafter #1\expandafter #2\csname.=%
2644   \XINT:NEhook:csv\XINTinFloatMinof:csv{\XINT_expr_unlock #3}\endcsname
2645 }%
2646 \expandafter
2647 \def\csname XINT_expr_func_+\endcsname #1#2#3%
2648 {%
2649   \expandafter #1\expandafter #2\csname.=%
2650   \XINT:NEhook:csv\xintSum:csv{\XINT_expr_unlock #3}\endcsname
2651 }%
2652 \expandafter
2653 \def\csname XINT_flexpr_func_+\endcsname #1#2#3%
2654 {%
2655   \expandafter #1\expandafter #2\csname.=%
2656   \XINT:NEhook:csv\XINTinFloatSum:csv{\XINT_expr_unlock #3}\endcsname
2657 }%
2658 \expandafter
2659 \def\csname XINT_iiexpr_func_+\endcsname #1#2#3%
2660 {%
2661   \expandafter #1\expandafter #2\csname.=%
2662   \XINT:NEhook:csv\xintiiSum:csv{\XINT_expr_unlock #3}\endcsname
2663 }%
2664 \expandafter
2665 \def\csname XINT_expr_func_*\endcsname #1#2#3%
2666 {%
2667   \expandafter #1\expandafter #2\csname.=%
2668   \XINT:NEhook:csv\xintPrd:csv{\XINT_expr_unlock #3}\endcsname
2669 }%
2670 \expandafter
2671 \def\csname XINT_flexpr_func_*\endcsname #1#2#3%
2672 {%
2673   \expandafter #1\expandafter #2\csname.=%
2674   \XINT:NEhook:csv\XINTinFloatPrd:csv{\XINT_expr_unlock #3}\endcsname
2675 }%
2676 \expandafter
2677 \def\csname XINT_iiexpr_func_*\endcsname #1#2#3%
2678 {%
2679   \expandafter #1\expandafter #2\csname.=%
2680   \XINT:NEhook:csv\xintiiPrd:csv{\XINT_expr_unlock #3}\endcsname
2681 }%
2682 \def\XINT_expr_func_all #1#2#3%
2683 {%

```

```

2684 \expandafter #1\expandafter #2\csname.=%
2685 \XINT:NEhook:csv\xintANDof:csv{\XINT_expr_unlock #3}\endcsname
2686 }%
2687 \let\XINT_flexpr_func_all\XINT_expr_func_all
2688 \let\XINT_iiexpr_func_all\XINT_expr_func_all
2689 \def\XINT_expr_func_any #1#2#3%
2690 {%
2691 \expandafter #1\expandafter #2\csname.=%
2692 \XINT:NEhook:csv\xintORof:csv{\XINT_expr_unlock #3}\endcsname
2693 }%
2694 \let\XINT_flexpr_func_any\XINT_expr_func_any
2695 \let\XINT_iiexpr_func_any\XINT_expr_func_any
2696 \def\XINT_expr_func_xor #1#2#3%
2697 {%
2698 \expandafter #1\expandafter #2\csname.=%
2699 \XINT:NEhook:csv\xintXORof:csv{\XINT_expr_unlock #3}\endcsname
2700 }%
2701 \let\XINT_flexpr_func_xor\XINT_expr_func_xor
2702 \let\XINT_iiexpr_func_xor\XINT_expr_func_xor
2703 \def\XINT_expr_func_len #1#2#3%
2704 {%
2705 \expandafter#1\expandafter#2\csname.=%
2706 \XINT:NEhook:csv\xintLength:f:csv{\XINT_expr_unlock#3}\endcsname
2707 }%
2708 \let\XINT_flexpr_func_len \XINT_expr_func_len
2709 \let\XINT_iiexpr_func_len \XINT_expr_func_len

```

1.2k has `\xintFirstItem:f:csv` for improved `\xintNewExpr` compatibility.

```

2710 \def\XINT_expr_func_first #1#2#3%
2711 {%
2712 \expandafter #1\expandafter #2\csname.=%
2713 \XINT:NEhook:csv\xintFirstItem:f:csv{\XINT_expr_unlock #3}\endcsname
2714 }%
2715 \let\XINT_flexpr_func_first\XINT_expr_func_first
2716 \let\XINT_iiexpr_func_first\XINT_expr_func_first

```

1.2k has `\xintLastItem:f:csv` for efficiency and improved `\xintNewExpr` compatibility.

```

2717 \def\XINT_expr_func_last #1#2#3%
2718 {%
2719 \expandafter #1\expandafter #2\csname.=%
2720 \XINT:NEhook:csv\xintLastItem:f:csv{\XINT_expr_unlock #3}\endcsname
2721 }%
2722 \let\XINT_flexpr_func_last\XINT_expr_func_last
2723 \let\XINT_iiexpr_func_last\XINT_expr_func_last

```

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.

```

2724 \def\XINT_expr_func_reversed #1#2#3%
2725 {%
2726 \expandafter #1\expandafter #2\csname.=%
2727 \XINT:NEhook:csv\xintReverse:f:csv{\XINT_expr_unlock #3}\endcsname

```



```

2728 }%
2729 \let\XINT_flexpr_func_reversed\XINT_expr_func_reversed
2730 \let\XINT_iiexpr_func_reversed\XINT_expr_func_reversed
2731 \def\xintiiifNotZero: #1,#2,#3,{\xintiiifNotZero{#1}{#2}{#3}}%
2732 \def\XINT_expr_func_if #1#2#3%
2733 {%
2734     \expandafter #1\expandafter #2\csname.=%
2735     \expandafter\xintiiifNotZero:%
2736     \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2737 }%
2738 \let\XINT_flexpr_func_if\XINT_expr_func_if
2739 \let\XINT_iiexpr_func_if\XINT_expr_func_if
2740 \def\xintifInt: #1,#2,#3,{\xintifInt{#1}{#2}{#3}}%
2741 \def\XINT_expr_func_ifint #1#2#3%
2742 {%
2743     \expandafter #1\expandafter #2\csname.=%
2744     \expandafter\xintifInt:%
2745     \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2746 }%
2747 \let\XINT_iiexpr_func_ifint\XINT_expr_func_ifint
2748 \def\xintifFloatInt: #1,#2,#3,{\xintifFloatInt{#1}{#2}{#3}}%
2749 \def\XINT_flexpr_func_ifint #1#2#3%
2750 {%
2751     \expandafter #1\expandafter #2\csname.=%
2752     \expandafter\xintifFloatInt:%
2753     \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2754 }%
2755 \def\xintifOne: #1,#2,#3,{\xintifOne{#1}{#2}{#3}}%
2756 \def\XINT_expr_func_ifone #1#2#3%
2757 {%
2758     \expandafter #1\expandafter #2\csname.=%
2759     \expandafter\xintifOne:%
2760     \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2761 }%
2762 \let\XINT_flexpr_func_ifone\XINT_expr_func_ifone
2763 \def\xintiiifOne: #1,#2,#3,{\xintiiifOne{#1}{#2}{#3}}%
2764 \def\XINT_iiexpr_func_ifone #1#2#3%
2765 {%
2766     \expandafter #1\expandafter #2\csname.=%
2767     \expandafter\xintiiifOne:%
2768     \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2769 }%
2770 \def\xintiiifSgn: #1,#2,#3,#4,{\xintiiifSgn{#1}{#2}{#3}{#4}}%
2771 \def\XINT_expr_func_ifsgn #1#2#3%
2772 {%
2773     \expandafter #1\expandafter #2\csname.=%
2774     \expandafter\xintiiifSgn:%
2775     \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2776 }%
2777 \let\XINT_flexpr_func_ifsgn\XINT_expr_func_ifsgn
2778 \let\XINT_iiexpr_func_ifsgn\XINT_expr_func_ifsgn

```

## 11.54 f-expandable versions of the \xintSeqB:f:csv and alike routines, for \xintNewExpr

11.54.1	\xintSeqB:f:csv . . . . .	370
11.54.2	\xintiiSeqB:f:csv . . . . .	370
11.54.3	\XINTinFloatSeqB:f:csv . . . . .	371

### 11.54.1 \xintSeqB:f:csv

Produces in f-expandable way. If the step is zero, gives empty result except if start and end coincide.

```

2779 \def\xintSeqB:f:csv #1#2%
2780   {\expandafter\XINT_seqb:f:csv \expandafter{\romannumeral0\xintra{#2}}{#1}}%
2781 \def\XINT_seqb:f:csv #1#2{\expandafter\XINT_seqb:f:csv_a\romannumeral`&&@#2#1!}%
2782 \def\XINT_seqb:f:csv_a #1#2;#3;#4!{%
2783   \expandafter\xint_gobble_i\romannumeral`&&@%
2784   \xintifCmp {#3}{#4}\XINT_seqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_seqb:f:csv_bg
2785   #1{#3}{#4}{#2}}%
2786 \def\XINT_seqb:f:csv_be #1#2#3#4#5{,#2}%
2787 \def\XINT_seqb:f:csv_bl #1{\if #1p\expandafter\XINT_seqb:f:csv_pa\else
2788   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2789 \def\XINT_seqb:f:csv_pa #1#2#3#4{\expandafter\XINT_seqb:f:csv_p\expandafter
2790   {\romannumeral0\xintadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2791 \def\XINT_seqb:f:csv_p #1#2%
2792   {%
2793   \xintifCmp {#1}{#2}\XINT_seqb:f:csv_pa\XINT_seqb:f:csv_pb\XINT_seqb:f:csv_pc
2794   {#1}{#2}}%
2795   }%
2796 \def\XINT_seqb:f:csv_pb #1#2#3#4{#3,#1}%
2797 \def\XINT_seqb:f:csv_pc #1#2#3#4{#3}%
2798 \def\XINT_seqb:f:csv_bg #1{\if #1n\expandafter\XINT_seqb:f:csv_na\else
2799   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2800 \def\XINT_seqb:f:csv_na #1#2#3#4{\expandafter\XINT_seqb:f:csv_n\expandafter
2801   {\romannumeral0\xintadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2802 \def\XINT_seqb:f:csv_n #1#2%
2803   {%
2804   \xintifCmp {#1}{#2}\XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_seqb:f:csv_na
2805   {#1}{#2}}%
2806   }%
2807 \def\XINT_seqb:f:csv_nb #1#2#3#4{#3,#1}%
2808 \def\XINT_seqb:f:csv_nc #1#2#3#4{#3}%

```

### 11.54.2 \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).

```

2809 \def\xintiiSeqB:f:csv #1#2%
2810   {\expandafter\XINT_iiSeqb:f:csv \expandafter{\romannumeral`&&@#2}{#1}}%

```

```

2811 \def\XINT_iiseqb:f:csv #1#2{\expandafter\XINT_iiseqb:f:csv_a\romannumeral`&&@#2#1!}%
2812 \def\XINT_iiseqb:f:csv_a #1#2;#3;#4!{%
2813   \expandafter\xint_gobble_i\romannumeral`&&@%
2814   \xintSgnFork{\XINT_Cmp {#3}{#4}}%
2815     \XINT_iiseqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_iiseqb:f:csv_bg
2816     #1{#3}{#4}{}{#2}}%
2817 \def\XINT_iiseqb:f:csv_bl #1{\if #1p\expandafter\XINT_iiseqb:f:csv_pa\else
2818   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2819 \def\XINT_iiseqb:f:csv_pa #1#2#3#4{\expandafter\XINT_iiseqb:f:csv_p\expandafter
2820   {\romannumeral0\xintiiadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2821 \def\XINT_iiseqb:f:csv_p #1#2%
2822 {%
2823   \xintSgnFork{\XINT_Cmp {#1}{#2}}%
2824   \XINT_iiseqb:f:csv_pa\XINT_iiseqb:f:csv_pb\XINT_iiseqb:f:csv_pc {#1}{#2}%
2825 }%
2826 \def\XINT_iiseqb:f:csv_pb #1#2#3#4{#3,#1}%
2827 \def\XINT_iiseqb:f:csv_pc #1#2#3#4{#3}%
2828 \def\XINT_iiseqb:f:csv_bg #1{\if #1n\expandafter\XINT_iiseqb:f:csv_na\else
2829   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2830 \def\XINT_iiseqb:f:csv_na #1#2#3#4{\expandafter\XINT_iiseqb:f:csv_n\expandafter
2831   {\romannumeral0\xintiiadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2832 \def\XINT_iiseqb:f:csv_n #1#2%
2833 {%
2834   \xintSgnFork{\XINT_Cmp {#1}{#2}}%
2835   \XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_iiseqb:f:csv_na {#1}{#2}%
2836 }%

```

### 11.54.3 \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.

```

2837 \def\XINTinFloatSeqB:f:csv #1#2{\expandafter\XINT_flseqb:f:csv \expandafter
2838   {\romannumeral0\XINTinfloat [\XINTdigits]{#2}}{#1}}%
2839 \def\XINT_flseqb:f:csv #1#2{\expandafter\XINT_flseqb:f:csv_a\romannumeral`&&@#2#1!}%
2840 \def\XINT_flseqb:f:csv_a #1#2;#3;#4!{%
2841   \expandafter\xint_gobble_i\romannumeral`&&@%
2842   \xintifCmp {#3}{#4}\XINT_flseqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_flseqb:f:csv_bg
2843   #1{#3}{#4}{}{#2}}%
2844 \def\XINT_flseqb:f:csv_bl #1{\if #1p\expandafter\XINT_flseqb:f:csv_pa\else
2845   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2846 \def\XINT_flseqb:f:csv_pa #1#2#3#4{\expandafter\XINT_flseqb:f:csv_p\expandafter
2847   {\romannumeral0\XINTinfloatadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2848 \def\XINT_flseqb:f:csv_p #1#2%
2849 {%
2850   \xintifCmp {#1}{#2}%
2851   \XINT_flseqb:f:csv_pa\XINT_flseqb:f:csv_pb\XINT_flseqb:f:csv_pc {#1}{#2}%
2852 }%
2853 \def\XINT_flseqb:f:csv_pb #1#2#3#4{#3,#1}%
2854 \def\XINT_flseqb:f:csv_pc #1#2#3#4{#3}%
2855 \def\XINT_flseqb:f:csv_bg #1{\if #1n\expandafter\XINT_flseqb:f:csv_na\else
2856   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2857 \def\XINT_flseqb:f:csv_na #1#2#3#4{\expandafter\XINT_flseqb:f:csv_n\expandafter

```

```

2858             {\romannumeral0\XINTinfloatadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2859 \def\XINT_flseqb:f:csv_n #1#2%
2860 {%
2861     \xintifCmp {#1}{#2}%
2862     \XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_flseqb:f:csv_na {#1}{#2}%
2863 }%

```

## 11.55 \xintdeffunc, \xintdefiifunc, \xintdeffloatfunc

### 1.2c (2015/11/12).

Note: it is possible to have same name assigned both to a variable and a function: things such as add(f(f), f=1..10) are possible.

### 1.2f (2016/03/08).

Comma separated expressions allowed (formerly this required using parenthesis \xintdeffunc foo(x,..):=(., ., .));

### 1.3c (2018/06/17).

Usage of \xintexprSafeCatcodes to be compatible with an active semi-colon at time of use; the colon was not a problem (see ##3) already.

```

2864 \def\XINT_tmpa #1#2#3#4%
2865 {%
2866     \def #1##1(##2)##3=##4;%
2867     \edef\XINT_deffunc_tmpa {##1}%
2868     \edef\XINT_deffunc_tmpa {\xint_zapspaces_o \XINT_deffunc_tmpa}%
2869     \def\XINT_deffunc_tmpb {0}%
2870     \def\XINT_deffunc_tmpc {(##4)}%
2871     \edef\XINT_deffunc_tmpd {##2}%
2872     \ifnum\xintLength:f:csv{\XINT_deffunc_tmpd}>\xint_c_
2873     \xintFor ###1 in {\XINT_deffunc_tmpd}\do
2874     {\edef\XINT_deffunc_tmpb {\the\numexpr\XINT_deffunc_tmpb+\xint_c_i}%
2875     \edef\XINT_deffunc_tmpc {subs(\unexpanded\expandafter{\XINT_deffunc_tmpc},%
2876     ###1=#####\XINT_deffunc_tmpb)}}%
2877     }%
2878     \fi

```

Something like this must be done before the NewFunc, else recursive definitions are impossible as the function will be unknown.

```

2879     \ifnum\XINT_deffunc_tmpb=\xint_c_
2880     \expandafter\XINT_expr_defuserfunc_none\csname
2881     \else
2882     \expandafter\XINT_expr_defuserfunc\csname
2883     \fi
2884     XINT_#2_func_\XINT_deffunc_tmpa\expandafter\endcsname
2885     \expandafter{\XINT_deffunc_tmpa}{#2}%
2886     \expandafter#3\csname XINT_#2_userfunc_\XINT_deffunc_tmpa\endcsname
2887     [\XINT_deffunc_tmpb]{\XINT_deffunc_tmpc}%
2888     \ifxintverbose\xintMessage {xintexpr}{Info}
2889     {Function \XINT_deffunc_tmpa\space for \string\xint #4 parser
2890     associated to \string\XINT_#2_userfunc_\XINT_deffunc_tmpa\space
2891     with \ifxintglobaldefs global \fi meaning \expandafter\meaning

```

```

2892     \csname XINT_#2_userfunc_\XINT_deffunc_tmpa\endcsname}%
2893 \fi
2894 \xintexprRestoreCatcodes
2895 }%
2896 }%
2897 \def\xintdeffunc      {\xintexprSafeCatcodes\xintdeffunc_a}%
2898 \def\xintdefiifunc    {\xintexprSafeCatcodes\xintdefiifunc_a}%
2899 \def\xintdeffloatfunc {\xintexprSafeCatcodes\xintdeffloatfunc_a}%
2900 \XINT_tmpa\xintdeffunc_a {expr} \XINT_NewFunc {expr}%
2901 \XINT_tmpa\xintdefiifunc_a {iiexpr}\XINT_NewIIFunc {iiexpr}%
2902 \XINT_tmpa\xintdeffloatfunc_a{floatexpr}\XINT_NewFloatFunc{floatexpr}%
2903 \def\XINT_expr_defuserfunc #1#2#3%
2904 {%
2905     \XINT_global
2906     \def #1##1##2##3{\expandafter ##1\expandafter ##2%
2907         \csname.\XINT:expr:userfunc{#3}{#2}{\XINT_expr_unlock ##3}\endcsname
2908     }%
2909 }%
2910 \def\XINT:expr:userfunc #1#2#3%
2911 {%
2912     \csname XINT_#1_userfunc_#2\expandafter\endcsname
2913     \romannumeral0\xintcsvtolistnonstripped{#3}%
2914 }%
2915 \def\XINT_expr_defuserfunc_none #1#2#3%
2916 {%
2917     \XINT_global
2918     \def #1##1##2##3{\expandafter ##1\expandafter ##2%
2919         \csname.\XINT:expr:userfunc:none{#3}{#2}\endcsname
2920     }%
2921 }%
2922 \def\XINT:expr:userfunc:none #1#2{\csname XINT_#1_userfunc_#2\endcsname}%

```

### 11.56 \xintdefefunc, \xintdefiiefunc, \xintdeffloatefunc

Added at 1.3e. Please consider the whole business of \xintdefefunc, \xintdefefunc, \xintNewExpr as somewhat like a work in progress, it is complex indeed.

```

2923 \def\XINT_tmpa #1#2#3#4%
2924 {%
2925     \def #1##1(##2)##3=##4;%
2926     \edef\XINT_defefunc_tmpa {##1}%
2927     \edef\XINT_defefunc_tmpa {\xint_zapspaces_o \XINT_defefunc_tmpa}%
2928     \def\XINT_defefunc_tmppb {0}%
2929     \def\XINT_defefunc_tmppc {(##4)}%
2930     \edef\XINT_defefunc_tmppd {##2}%
2931     \ifnum\xintLength:f:csv{\XINT_defefunc_tmppd}>\xint_c_
2932         \xintFor ###1 in {\XINT_defefunc_tmppd}\do
2933             {\edef\XINT_defefunc_tmppb {\the\numexpr\XINT_defefunc_tmppb+\xint_c_i}%
2934                 \edef\XINT_defefunc_tmppc {subs(\unexpanded\expandafter{\XINT_defefunc_tmppc},%
2935                     ###1=#####\XINT_defefunc_tmppb)}}%
2936     }%
2937 \fi

```

No recursivity allowed here with the function to be defined.

```

2938 \expandafter#3\csname XINT_#2_userefunc_\XINT_defefunc_tmpa\endcsname
2939         [\XINT_defefunc_tmpb]{\XINT_defefunc_tmpc}%
2940 \edef\XINT_defefunc_tmpd{\xintLength:f:csv
2941     {\expandafter\meaning\csname
2942         XINT_#2_userefunc_\XINT_defefunc_tmpa\endcsname}}%
```

We try to distinguish wheter the function is supposed to deliver only one value or more than two. And we separate the cases of 0, 1 or 2 variables which can be set-up a bit better for usage in other definitions, in generator environments. But there are many shortcomings. I don't have a very clear view of all the complex situation, in fact.

```

2943 \ifcase\XINT_defefunc_tmpb\space
2944 \expandafter\XINT_expr_defuserefunc_none\csname
2945 \or
2946 % \ifnum\XINT_defefunc_tmpd=\xint_c_i
2947 \expandafter\XINT_expr_defuserefunc_one\csname
2948 % \else
2949 % \expandafter\XINT_expr_defuserefunc_onetocsv\csname
2950 % \fi
2951 \or
2952 % \ifnum\XINT_defefunc_tmpd=\xint_c_i
2953 \expandafter\XINT_expr_defuserefunc_two\csname
2954 % \else
2955 % \expandafter\XINT_expr_defuserefunc_twotocsv\csname
2956 % \fi
2957 \else
2958 % \ifnum\XINT_defefunc_tmpd=\xint_c_i
2959 \expandafter\XINT_expr_defuserefunc_many\csname
2960 % \else
2961 % \expandafter\XINT_expr_defuserefunc_manytocsv\csname
2962 % \fi
2963 \fi
2964 XINT_#2_func_\XINT_defefunc_tmpa\expandafter\endcsname
2965 \expandafter{\XINT_defefunc_tmpa}{#2}%
2966 \ifxintverbose\xintMessage {xintexpr}{Info}
2967     {Function \XINT_defefunc_tmpa\space for \string\xint #4 parser
2968     associated to \string\XINT_#2_userefunc_\XINT_defefunc_tmpa\space
2969     with \ifxintglobaldefs global \fi meaning \expandafter\meaning
2970     \csname XINT_#2_userefunc_\XINT_defefunc_tmpa\endcsname}%
2971 \fi
2972 \xintexprRestoreCatcodes
2973 }%
2974 }%
2975 \def\xintdefefunc     {\xintexprSafeCatcodes\xintdefefunc_a}%
2976 \def\xintdefiiefunc   {\xintexprSafeCatcodes\xintdefiiefunc_a}%
2977 \def\xintdeffloatefunc {\xintexprSafeCatcodes\xintdeffloatefunc_a}%
2978 \XINT_tmpa\xintdefefunc_a {expr} \XINT_NewFunc {expr}%
2979 \XINT_tmpa\xintdefiiefunc_a {iiexpr}\XINT_NewIIFunc {iiexpr}%
2980 \XINT_tmpa\xintdeffloatefunc_a{fexpr}\XINT_NewFloatFunc{floatexpr}%
2981 \def\XINT_expr_defuserefunc_none #1#2#3%
2982 {%
```

```

2983 \expandafter\XINT_expr_defuserefunc_none_a
2984 \csname XINT_#3_userefunc_#2\endcsname
2985 }%
2986 \def\XINT_expr_defuserefunc_none_a #1#2%
2987 {%
2988 \XINT_global
2989 \def #2##1##2##3{\expandafter ##1\expandafter ##2\csname.=#1\endcsname}%
2990 }%

```

Je définis une macro auxiliaire qui fait l'expansion mais tout cela pour éviter le très léger overhead de `\xintExpandArgs...` c'est idiot et le devient encore plus pour deux arguments. Mais c'est aussi dû à `\xintApply::csv` que l'on veut utiliser commodément dans `\XINT:NE:userefunc:one_a`.

```

2991 \def\XINT_expr_defuserefunc_one #1#2#3%
2992 {%
2993 \expandafter\XINT_expr_defuserefunc_one_a
2994 \csname XINT_#3_userefunc_#2\expandafter\endcsname
2995 \csname XINT_#3_userefunc:f_#2\endcsname #1{#2}{#3}%
2996 }%
2997 \def\XINT_expr_defuserefunc_one_a #1#2#3#4#5%
2998 {%
2999 \XINT_global
3000 \def #2##1{\expandafter#1\expandafter{\romannumeral`&&@##1}}%
3001 \XINT_global
3002 \def #3##1##2##3%
3003 {%
3004 \expandafter ##1\expandafter ##2%
3005 \csname.=\XINT:expr:userefunc:one{#5}{#4}{\XINT_expr_unlock##3}\endcsname
3006 }%
3007 }%
3008 \def\XINT:expr:userefunc:one #1#2#3%
3009 {%
3010 \csname XINT_#1_userefunc_#2\expandafter\endcsname\expandafter
3011 {\romannumeral`&&@#3}%
3012 }%
3013 \def\XINT_expr_defuserefunc_two #1#2#3%
3014 {%
3015 \expandafter\XINT_expr_defuserefunc_two_a
3016 \csname XINT_#3_userefunc:f_#2\endcsname #1{#2}{#3}%
3017 }%

```

Le fait que `\xintExpandArgs` demande que les arguments sont regroupés explique pourquoi plus bas j'ai dû faire `\XINT:NE:userefunc:two`.

```
\xintExpandArgs#1{##1}{##2}
```

Mais bon finalement je rajouter encore un helper d'expansion. Que j'ai peut-être d'ailleurs déjà... C'est un peu de l'abus la macro auxiliaire `userefunc:f` pour chaque `userefunc`, mais je dois gérer le problème que les noms de macros ici peuvent contenir des chiffres.

Il y a de la perte dans le grabbing de l'argument qui aura lieu et qu'on pourrait optimiser, mais je commence à sérieusement fatiguer pour 1.3e.

```

3018 \def\XINT_expr_defuserefunc_two_a #1#2#3#4%
3019 {%
3020 \XINT_global
3021 \def #1##1##2{\xintExpandArgs{XINT_#4_userefunc_#3}{##1}{##2}}%

```

```

3022 \XINT_global
3023 \def #2##1##2##3%
3024 {%
3025     \expandafter ##1\expandafter ##2%
3026     \csname.=\XINT:expr:userefunc:two{#4}{#3}{\XINT_expr_unlock##3}\endcsname
3027 }%
3028 }%
3029 \def\XINT:expr:userefunc:two #1#2#3%
3030 {%
3031     \expandafter\XINT:expr:userefunc:two_a
3032     \csname XINT_#1_userefunc_#2\expandafter\endcsname
3033     \romannumeral`&&@#3,%
3034 }%
3035 \def\XINT:expr:userefunc:two_a #1#2,#3,{#1{#2}{#3}}%

```

Paradoxically the general case is code faster. But this is explained because we try to hook into special handlers for one or two variables (see "Mysterious stuff" subsection in NewExpr).

```

3036 \def\XINT_expr_defuserefunc_many #1#2#3%
3037 {%
3038     \XINT_global
3039     \def #1##1##2##3%
3040     {%
3041         \expandafter ##1\expandafter ##2%
3042         \csname.=\XINT:expr:userefunc{#3}{#2}{\XINT_expr_unlock##3}\endcsname
3043     }%
3044 }%
3045 \def\XINT:expr:userefunc #1#2#3%
3046 {%
3047     \csname XINT_#1_userefunc_#2\expandafter\endcsname
3048     \romannumeral0\xintcsvtolistnonstripped{#3}%
3049 }%

```

### 11.57 `\xintunassignexprfunc`, `\xintunassigniiexprfunc`, `\xintunassignfloatexprfunc`

See the [\xintunassignvar](#) for the embarrassing explanations why I had not done that earlier. A bit lazy here, no warning if undefining something not defined, and attention no precaution respective built-in functions.

```

3050 \def\XINT_tmpa #1{\expandafter\def\csname xintunassign#1func\endcsname ##1{%
3051     \edef\XINT_unfunc_tmpa{##1}%
3052     \edef\XINT_unfunc_tmpa {\xint_zapspaces_o\XINT_unfunc_tmpa}%
3053     \XINT_global\expandafter
3054     \let\csname XINT_#1_func_\XINT_unfunc_tmpa\endcsname\xint_undefined
3055     \XINT_global\expandafter
3056     \let\csname XINT_#1_userfunc_\XINT_unfunc_tmpa\endcsname\xint_undefined
3057     \XINT_global\expandafter
3058     \let\csname XINT_#1_userefunc_\XINT_unfunc_tmpa\endcsname\xint_undefined
3059     \ifxintverbose\xintMessage {xintexpr}{Info}
3060     {Function \XINT_unfunc_tmpa\space for \string\xint #1 parser now
3061     \ifxintglobaldefs globally \fi undefined.}%
3062     \fi}}%
3063 \XINT_tmpa{expr}\XINT_tmpa{iiexpr}\XINT_tmpa{floatexpr}%

```



## 11.58 \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`.

```

3064 \def\XINT_expr_wrapit #1{\expandafter\XINT_expr_wrap\csname.=#1\endcsname}%
3065 \def\xintNewFunction #1#2[#3]#4%
3066 {%
3067   \edef\XINT_newfunc_tmpa {#1}%
3068   \edef\XINT_newfunc_tmpa {\xint_zapspace_o \XINT_newfunc_tmpa}%
3069   \def\XINT_newfunc_tmpb ##1##2##3##4##5##6##7##8##9{#4}%
3070   \begingroup
3071     \ifcase #3\relax
3072       \toks0{}%
3073     \or \toks0{##1}%
3074     \or \toks0{##1##2}%
3075     \or \toks0{##1##2##3}%
3076     \or \toks0{##1##2##3##4}%
3077     \or \toks0{##1##2##3##4##5}%
3078     \or \toks0{##1##2##3##4##5##6}%
3079     \or \toks0{##1##2##3##4##5##6##7}%
3080     \or \toks0{##1##2##3##4##5##6##7##8}%
3081     \else \toks0{##1##2##3##4##5##6##7##8##9}%
3082     \fi
3083     \expandafter
3084   \endgroup\expandafter
3085   \XINT_global\expandafter
3086   \def\csname XINT_expr_macrofunc_\XINT_newfunc_tmpa\expandafter\endcsname
3087   \the\toks0\expandafter{\XINT_newfunc_tmpb
3088     {\XINT_expr_wrapit{##1}}{\XINT_expr_wrapit{##2}}{\XINT_expr_wrapit{##3}}%
3089     {\XINT_expr_wrapit{##4}}{\XINT_expr_wrapit{##5}}{\XINT_expr_wrapit{##6}}%
3090     {\XINT_expr_wrapit{##7}}{\XINT_expr_wrapit{##8}}{\XINT_expr_wrapit{##9}}}%
3091   \expandafter\XINT_expr_newfunction
3092     \csname XINT_expr_func_\XINT_newfunc_tmpa\expandafter\endcsname
3093     \expandafter{\XINT_newfunc_tmpa}{eval}\xintbareeval
3094   \expandafter\XINT_expr_newfunction
3095     \csname XINT_iiexpr_func_\XINT_newfunc_tmpa\expandafter\endcsname
3096     \expandafter{\XINT_newfunc_tmpa}{iieval}\xintbareiieval
3097   \expandafter\XINT_expr_newfunction
3098     \csname XINT_flexpr_func_\XINT_newfunc_tmpa\expandafter\endcsname
3099     \expandafter{\XINT_newfunc_tmpa}{floateval}\xintbarefloateval
3100   \ifxintverbose
3101     \xintMessage {xintexpr}{Info}
3102     {Function \XINT_newfunc_tmpa\space for the expression parsers is
3103     associated to \string\XINT_expr_macrofunc_\XINT_newfunc_tmpa\space
3104     with \ifxintglobaldefs global \fi meaning \expandafter\meaning
3105     \csname XINT_expr_macrofunc_\XINT_newfunc_tmpa\endcsname}%
3106   \fi
3107 }%
3108 \def\XINT_expr_newfunction #1#2#3#4%
3109 {%

```

```

3110 \XINT_global
3111 \def#1##1##2##3{\expandafter ##1\expandafter ##2\romannumeral0%
3112 \XINT:expr:macrofunc{#4}{#3}{#2}{\XINT_expr_unlock##3}}%
3113 }%
3114 \def\XINT:expr:macrofunc #1#2#3#4%
3115 {%
3116 #1\csname XINT_expr_macrofunc_#3\expandafter\endcsname
3117 \romannumeral0\xintcsvtolistnonstripped{#4}\relax
3118 }%
3119 \catcode`~ 12
3120 \def\XINT:newexpr:macrofunc #1{%
3121 \def\XINT:newexpr:macrofunc ##1##2##3##4%
3122 {%
3123 \expandafter#1\csname.=~XINT:newexpr:macrofunc:a{##2}{##3}%
3124 {\xintCSVtoListNonStripped{##4}}\endcsname
3125 }%
3126 }\XINT:newexpr:macrofunc { }%
3127 \catcode`~ 3
3128 \def\XINT:newexpr:macrofunc:a #1#2#3%
3129 {%
3130 \expandafter\XINT_expr_unlock\romannumeral0\csname xintbare#1\endcsname
3131 \csname XINT_expr_macrofunc_#2\endcsname#3\relax
3132 }%

```

### 11.59 `\xintNewExpr`, `\xintNewIExpr`, `\xintNewFloatExpr`, `\xintNewIIExpr`

11.59.1	<code>\xintApply::csv</code> and <code>\xintApply:::csv</code> . . . . .	379
11.59.2	Mysterious stuff . . . . .	380
11.59.3	<code>\XINT_expr_redefinemacros</code> . . . . .	384
11.59.4	<code>\XINT_expr_redefineprints</code> . . . . .	385
11.59.5	<code>\xintNewExpr</code> , ..., at last. . . . .	386
11.59.6	<code>\ifxintexprsafecatcodes</code> , <code>\xintexprSafeCatcodes</code> , <code>\xintexprRestoreCatcodes</code> . .	388

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.

Comments added 2018/02/28:

At 1.3 of February 2018, there was important refactoring. Earlier, `\XINT_expr_redefinemacros` was a very big macro which made aliases of the dozens of macros (most from `xintfrac` and some defined especially by `xintexpr` for acting on csv lists primarily) involved in the expression rendering and then redefined them all to expand to their original selves only when applied to purely numeric arguments. At 1.3 only very few such re-definitions are made, as what is redefined are a limited number of core wrapper macros.

Only when the original macros have one or two arguments is it examined if they can expand immediately (this includes case of function having possibly only one, or possibly two arguments). For macros applying to three or more or an undefined number of arguments, we don't complicate matters into checking if expansion is possible, and we delay that expansion automatically (but `if()` and `ifsgn()` do check if first argument is numeric and expand to suitable branch in that case).

In particular any function defined by `\xintdeffunc` or `\xintNewFunction` (it is then basically only a macro abstraction) when used in new function definitions will never be expanded immediately, because the detection of whether they are applied to only numerical data has not yet been added. (this might be added in future).

Some aspects of the 1.3 refactoring have made recursive definition via `\xintdeffunc` possible (of course they always were via `\xintNewFunction` as the latter is but a wrapper of a standard TeX macro definition, where `\xintexpr` parsing is not at all involved).

A somewhat complicated layer (not modified at 1.3) is devoted to making possible the parsing of constructs such as `[#1..[#2]..#3][#4:#5]` or `[#1..#2]*#3` and it seems to work. At 1.3, even esoteric construct such as `[divmod(#1,#2)]*#3` is parsable by `\xintNewExpr`. (In `\xintNewFloatExpr`, don't forget `\empty` token so that square brackets are not mistaken for optional argument of `\xintthefloatexpr`; same for `\xintdeffloatfunc`.)

Side note: I wonder if I really had a good idea to define these list operations `[..]*foo` or `foo^[...]` which do not seem to occur in other languages with the meanings I used. And they caused me lots of efforts for support at `\xintNewExpr` level...

The catcode 12 dollar sign is used to signal when a macro can not be expanded but would produce a csv list. Furthermore some cases require f-expandable macros as the original code expanding in `\xintexpr` is in `\csmname` context and did not need f-expandability.

As mentioned above, currently syntax with dummy variables can not go through where the values iterated over are not explicit; and `omit`, `abort`, `break` mechanisms are not parsable with non purely numerical data, in part because they are not implemented internally via pure f-expansion.

### 11.59.1 `\xintApply::csv` and `\xintApply:::csv`

Serve in particular to support things such as

```
\xintdeffunc foo(x):=seq(sqr(i), i=0..x);
```

... as far as I still understand what is going on here! The most complicated is for list operations; many things involving sequences don't go through `\xintNewExpr`, especially with functions of more than one variable.

```
3133 \def\xintApply::csv #1#2%
3134   {\expandafter\XINT_applyon::_a\expandafter {\romannumeral`&&@#2}{#1}}%
3135 \def\XINT_applyon::_a #1#2{\XINT_applyon::_b {#2}{#1},,%
3136 \def\XINT_applyon::_b #1#2#3,{\expandafter\XINT_applyon::_c \romannumeral`&&@#3,{#1}{#2}}%
3137 \def\XINT_applyon::_c #1{\if #1,\expandafter\XINT_applyon::_end
3138   \else\expandafter\XINT_applyon::_d\fi #1}%
3139 \def\XINT_applyon::_d #1,#2{\expandafter\XINT_applyon::_e\romannumeral`&&@#2{#1},{#2}}%
3140 \def\XINT_applyon::_e #1,#2#3{\XINT_applyon::_b {#2}{#3}, #1}}%
```

```

3141 \def\XINT_applyon::_end #1,#2#3{\xint_secondoftwo #3}%
3142 \def\xintApply:::csv #1#2#3%
3143   {\expandafter\XINT_applyon::_a\expandafter{\romannumeral`&&#2}{#1}{#3}}%
3144 \def\XINT_applyon::_a #1#2#3{\XINT_applyon::_b {#2}{#3}{#1,,}%
3145 \def\XINT_applyon::_b #1#2#3#4,%
3146   {\expandafter\XINT_applyon::_c \romannumeral`&&#4,{#1}{#2}{#3}}%
3147 \def\XINT_applyon::_c #1{\if #1,\expandafter\XINT_applyon::_end
3148   \else\expandafter\XINT_applyon::_d\fi #1}%
3149 \def\XINT_applyon::_d #1,#2#3%
3150   {\expandafter\XINT_applyon::_e\expandafter
3151   {\romannumeral`&&\xintApply:::csv {#2}{#1}{#3}},{#2}{#3}}%
3152 \def\XINT_applyon::_e #1,#2#3#4{\XINT_applyon::_b {#2}{#3}{#4, #1}}%
3153 \def\XINT_applyon::_end #1,#2#3#4{\xint_secondoftwo #4}%

```

### 11.59.2 Mysterious stuff

~ and \$ of catcode 12 in what follows. There was some refactoring at 1.3e, particularly \XINT:NE:userefunc was added.

```

3154 \catcode`~ 12
3155 \catcode`$ 12 % $
3156 \def\xint_dfork #1$#2#3\krof {#2}% $
3157 \def\xint_ddfork #1$$#2#3\krof {#2}% $$
3158 \def\XINT:NE:RApply:::csv #1#2#3#4%
3159   {\xintApply:::csv{\~expandafter #2~\xint_exchangetwo_keepbraces{#4}}{#3}}%
3160 \def\XINT:NE:LApply:::csv #1#2#3{\xintApply:::csv{#2}{#3}}%
3161 \def\XINT:NE:RLApply:::csv #1{\xintApply:::csv}%
3162 \def\XINT:NE:two#1{\XINT:NE:two_{#1}{\detokenize{#1}}}%
3163 \def\XINT:NE:two_#1#2#3#4%
3164   {\expandafter\XINT:NE:two_a\romannumeral`&&#4!{#3}{#1}{#2}}%
3165 \def\XINT:NE:two_a#1#2!#3#4#5%
3166   {\expandafter\XINT:NE:two_b\romannumeral`&&#3!#1{#4}{#5}{#1#2}}%
3167 \def\XINT:NE:two_b#1#2!#3#4#5{\XINT:NE:two_fork_dd#1#3{#4}{#5}{#1#2}}%
3168 \def\XINT:NE:two_fork_dd #1#2{%
3169   \xint_ddfork
3170   #1#2\XINT:NE:RLApply:::csv
3171   #1$\XINT:NE:RApply:::csv% $
3172   $#2\XINT:NE:LApply:::csv% $
3173   $$\XINT:NE:two_fork_nn #1#2}% $$
3174   \krof
3175 }%
3176 \def\XINT:NE:two_fork_nn #1#2#3#4{%
3177   \if #1#\xint_dothis{#4}\fi
3178   \if #1~\xint_dothis{#4}\fi
3179   \if #2#\xint_dothis{#4}\fi
3180   \if #2~\xint_dothis{#4}\fi
3181   \xint_orthat{#3}%
3182 }%

```

Problème pour autoriser ici qu'une liste arrive comme argument qui potentiellement en donnerait deux mais pour le moment on ne sait pas. Donc j'impose le prérequis que \XINT:NE:twosp soit toujours suivi de deux items exactement.

Il y a eu des modifications à 1.3e car j'ai supprimé des choses que je ne comprenais pas puis je les ai remises lorsque je comprenais mais sans doute pas pareil (certains tests passent et pas avant) et ensuite je ne comprenais plus à nouveau.

```

3183 \def\XINT:NE:tosp#1#2,#3#4,!#5%
3184 {%
3185   \XINT:NE:two_fork_dd#1#3{#5}{\detokenize{#5}}{#1#2}{#3#4}%
3186 }%
3187 \def\XINT:NE:one#1#2{\expandafter\XINT:NE:one_a\romannumeral`&&@#2!#1}%
3188 \def\XINT:NE:one_a#1#2!#3%
3189 {%
3190   \if ###1\xint_dothis {\detokenize{#3}}\fi
3191   \if ~#1\xint_dothis {\detokenize{#3}}\fi
3192   \if $#1\xint_dothis {\xintApply::csv{\detokenize{#3}}}\fi %$
3193   \xint_orthat #3{#1#2}%
3194 }%

```

`\xintExpandArgs` is defined in `xinttools.sty` (I don't recall why; not for reasons internal to `xint` I guess). Attention here that user function names may contain digits, so we don't use a `\detokenize` or `~` approach.

```

3195 \def\XINT:NE:userfunc #1#2#3%
3196   {\xintExpandArgs{XINT_#1_userfunc_#2}{\xintCSVtoListNonStripped{#3}}}%
3197 \def\XINT:NE:userfunc:none #1#2{~!{XINT_#1_userfunc_#2}}%

```

`\XINT:NE:userefunc` et al. added at 1.3e. For one and two I can not use `\XINT:NE:one` due to possible digits in names. For more than two nothing special done with mysterious "Apply" macros above.

Should they ever use in output? Je crois que j'ai des problèmes en particulier car j'utilise le `most` liste dans plusieurs sens il y a en particulier la confusion possible pour liste dans le sens restreint devant être géré par les opérations genre `]*` ou celui avec les `seq()` ou finalement la liste des arguments d'une fonction.

When expansion of the user func can not happend on the spot, the version which will be expanded later one must first expand its argument for efficiency because the functions from `\XINT_NewFunc` do not do that and we must thus have an auxiliary variant expanding its argument.

```

3198 \def\XINT:NE:userefunc:one#1#2#3%
3199   {\expandafter\XINT:NE:userefunc:one_a\romannumeral`&&@#3!{#1}{#2}}%
3200 \def\XINT:NE:userefunc:one_a#1#2!#3#4%
3201 {%
3202   \if ###1\xint_dothis {~!{XINT_#3_userefunc:f_#4}}\fi
3203   \if ~#1\xint_dothis {~!{XINT_#3_userefunc:f_#4}}\fi

```

Quickly checked this `\csname` presentation ok for `\xintApply::csv`.

```

3204   \if $#1\xint_dothis {\xintApply::csv~!{XINT_#3_userefunc:f_#4}}\fi %$
3205   \xint_orthat {\csname XINT_#3_userefunc_#4\endcsname}%
3206   {#1#2}%
3207 }%
3208 \def\XINT:NE:tosp#1#2,#3#4,!#5%
3209 {%
3210   \XINT:NE:two_fork_dd#1#3{#5}{\detokenize{#5}}{#1#2}{#3#4}%
3211 }%
3212 \def\XINT:NE:userefunc:two#1#2#3%
3213   {\expandafter\XINT:NE:userefunc:two_a\romannumeral`&&@#3,!{#1}{#2}}%

```

Je ne peux pas faire `~xintExpandArgs{XINT_#5_userefunc_#6}` à cause du fait que j'ai `{#1#2}{#3#4}` pas `{{#1#2}{#3#4}}` à cause de la première branche, celle qui s'étend.

```

3214 \def\XINT:NE:userefunc:two_a#1#2,#3#4,!#5#6%
3215 {%
3216   \XINT:NE:two_fork_dd#1#3{\csname XINT_#5_userefunc_#6\endcsname}%
3217   {~!\XINT_#5_userefunc:f_#6}}%
3218   {#1#2}{#3#4}%
3219 }%
3220 \def\XINT:NE:userefunc#1#2#3%
3221 {%
3222   \expandafter\XINT:NE:userefunc_a\romannumeral`&&@#3,2,3,4,5,6,7,8,9,!%
3223   {#1}{#2}{#3}%
3224 }%
3225 \def\XINT:NE:userefunc_a#1#2,#3#4,#5#6,#7#8,#9%
3226 {%
3227   \XINT:NE:userefunc_b{#1#3#5#7#9}%
3228 }%
3229 \def\XINT:NE:userefunc_b#1#2,#3#4,#5#6,#7#8,#9%
3230 {%
3231   \XINT:NE:userefunc_c{#1#3#5#7#9}%
3232 }%
3233 \def\XINT:NE:iftilde #1~#2#3\relax{\unless\if !#21\fi}%
3234 \def\XINT:NE:ifdollar #1$#2#3\relax{\unless\if !#21\fi}%$
3235 \def\XINT:NE:ifhash#1{%
3236 \def\XINT:NE:ifhash##1#1##2##3\relax{\unless\if !##21\fi}%
3237 }\expandafter\XINT:NE:ifhash\string#%
3238 \def\XINT:NE:userefunc_c#1#2!%
3239 {%
3240   \if0\XINT:NE:iftilde #1~!\relax\XINT:NE:ifdollar #1$!\relax%$
3241   \XINT:NE:ifhash #1##!\relax 0%
3242   \expandafter\XINT:NE:userefunc_x
3243   \else
3244   \expandafter\XINT:NE:userefunc_p
3245   \fi
3246 }%
3247 \def\XINT:NE:userefunc_x#1#2%
3248   {\csname XINT_#1_userefunc_#2\expandafter\endcsname
3249   \romannumeral0\xintcsvtolistnonstripped}%
3250 \def\XINT:NE:userefunc_p #1#2#3%
3251   {\xintExpandArgs{XINT_#1_userefunc_#2}{\xintCSVtoListNonStripped{#3}}}%

```

Back to older stuff.

```

3252 \def\XINT:NE:oneopt#1[#2]#3%
3253   {\expandafter\XINT:NE:oneopt_a\romannumeral`&&@#3!{#2}#1}%
3254 \def\XINT:NE:oneopt_a#1#2!#3#4%
3255   {\expandafter\XINT:NE:oneopt_b\romannumeral`&&@#3!#1#4{#1#2}}%
3256 \def\XINT:NE:oneopt_b#1#2!#3#4%
3257   {\expandafter\XINT:NE:oneopt_fork#1#3#4{#1#2}}%
3258 \def\XINT:NE:oneopt_fork#1#2#3#4{%
3259   \if1\if##11\else\if~#11\else\if###21\else\if~#21\else0\fi\fi\fi\fi
3260   \xint_dothis {\detokenize{#3}[#4]}\fi

```

```

3261 \if $#2\xint_dothis {~xintApply::csv{\detokenize{#3}[#4]}\fi %$
3262 \xint_orthat{#3[#4]}%
3263 }% pas complètement général, mais bon
3264 \def\xINT:NE:csv #1{\detokenize{#1}}% radicalement fainéant
3265 \def\xINT:newexpr:one:and:opt #1,#2,#3!#4#5%
3266 {%
3267 \if\relax#3\relax\expandafter\xint_firstoftwo\else
3268 \expandafter\xint_secondoftwo\fi
3269 {\XINT:NE:one#4}{\XINT:NE:oneopt#5[\XINT:NE:one\xintNum{#2}]}{#1}%
3270 }%
3271 \def\xINT:newexpr:tacitzeroifonearg #1,#2,#3!#4#5%
3272 {%
3273 \if\relax#3\relax\expandafter\xint_firstoftwo\else
3274 \expandafter\xint_secondoftwo\fi
3275 {\XINT:NE:two#4{0}}{\XINT:NE:two#5[\XINT:NE:one\xintNum{#2}]}{#1}%
3276 }%
3277 \def\xINT:newiexpr:tacitzeroifonearg #1,#2,#3!#4%
3278 {%
3279 \if\relax#3\relax\expandafter\xint_firstoftwo\else
3280 \expandafter\xint_secondoftwo\fi
3281 {\XINT:NE:two#4{0}}{\XINT:NE:two#4{#2}}{#1}%
3282 }%
3283 \def\xINT:newexpr:insertdollar~{\$noexpand$}%
3284 \def\xINT:newexpr:two:to:two #1,#2,!#3%
3285 {%
3286 \XINT:NE:two_
3287 {\expandafter\xINT:expr:totwo\romannumeral`&&@#3}%
3288 {\$noexpand$\expandafter~XINT:expr:totwo~romannumeral-`0\detokenize{#3}}%
3289 {#1}{#2}%
3290 }%
3291 \def\xINT:newfexpr:two:to:two #1,#2,!#3%
3292 {%
3293 \XINT:NE:two_
3294 {#3}%
3295 {\expandafter\xINT:newexpr:insertdollar\detokenize{#3}}%
3296 {#1}{#2}%
3297 }%
3298 \def\xintiiifNotZeroNE:#1#2,#3,#4,%
3299 {%
3300 \if1\if###11\else\if~#11\else\if$#11\else0%$
3301 \fi\fi\fi
3302 \xint_dothis{~xintiiifNotZero}\fi
3303 \xint_orthat\xintiiifNotZero
3304 {#1#2}{#3}{#4}%
3305 }%
3306 \def\xintifIntNE:#1#2,#3,#4,%
3307 {%
3308 \if1\if###11\else\if~#11\else\if$#11\else0%$
3309 \fi\fi\fi
3310 \xint_dothis{~xintifInt}\fi
3311 \xint_orthat\xintifInt
3312 {#1#2}{#3}{#4}%

```



```

3313 }%
3314 \def\xintifFloatIntNE:#1#2,#3,#4,%
3315 {%
3316   \if1\if###11\else\if~#11\else\if$#11\else0%$
3317     \fi\fi\fi
3318   \xint_dothis{~xintifFloatInt}\fi
3319   \xint_orthat\xintifFloatInt
3320   {#1#2}{#3}{#4}%
3321 }%
3322 \def\xintiifOneNE:#1#2,#3,#4,%
3323 {%
3324   \if1\if###11\else\if~#11\else\if$#11\else0%$
3325     \fi\fi\fi
3326   \xint_dothis{~xintiifOne}\fi
3327   \xint_orthat\xintiifOne
3328   {#1#2}{#3}{#4}%
3329 }%
3330 \def\xintifOneNE:#1#2,#3,#4,%
3331 {%
3332   \if1\if###11\else\if~#11\else\if$#11\else0%$
3333     \fi\fi\fi
3334   \xint_dothis{~xintifOne}\fi
3335   \xint_orthat\xintifOne
3336   {#1#2}{#3}{#4}%
3337 }%
3338 \def\xintiifSgnNE:#1#2,#3,#4,#5,%
3339 {%
3340   \if1\if###11\else\if~#11\else\if$#11\else0%$
3341     \fi\fi\fi
3342   \xint_dothis{~xintiifSgn}\fi
3343   \xint_orthat\xintiifSgn
3344   {#1#2}{#3}{#4}{#5}%
3345 }%

```

### 11.59.3 \XINT\_expr\_redefinmacros

Completely refactored at 1.3.

```

3346 \def\XINT_expr_redefinmacros {%
3347   \let\XINT:NEhook:one \XINT:NE:one
3348   \let\XINT:NEhook:two \XINT:NE:two
3349   \let\XINT:NEhook:csv \XINT:NE:csv
3350   \let\XINT:NEhook:twosp\XINT:NE:twosp
3351   \let\XINT:expr:userfunc \XINT:NE:userfunc
3352   \let\XINT:expr:userfunc:none \XINT:NE:userfunc:none
3353   \let\XINT:expr:userefunc \XINT:NE:userefunc
3354   \let\XINT:expr:userefunc:one \XINT:NE:userefunc:one
3355   \let\XINT:expr:userefunc:two \XINT:NE:userefunc:two
3356   \let\XINT:expr:macrofunc \XINT:newexpr:macrofunc
3357   \let\XINT:expr:one:and:opt \XINT:newexpr:one:and:opt
3358   \let\XINT:expr:one:or:two:nums \XINT:newexpr:one:or:two:nums
3359   \let\XINT:iiexpr:one:or:two: \XINT:newiiexpr:one:or:two:
3360   \let\XINT:expr:tacitzeroifonearg \XINT:newexpr:tacitzeroifonearg

```



```

3361 \let\XINT:iiexpr:tacitzeroifonearg \XINT:newiiexpr:tacitzeroifonearg
3362 \let\XINT:expr:two:to:two \XINT:newexpr:two:to:two
3363 \let\XINT:flexpr:two:to:two \XINT:newflexpr:two:to:two
3364 \let\xintiiifNotZero: \xintiiifNotZeroNE:
3365 \let\xintifInt: \xintifIntNE:
3366 \let\xintifFloatInt: \xintifFloatIntNE:
3367 \let\xintiiifOne: \xintiiifOneNE:
3368 \let\xintifOne: \xintifOneNE:
3369 \let\xintiiifSgn: \xintiiifSgnNE:
3370 \let\xintSeqNumeric::csv \xintSeq::csv
3371 \let\xintiiSeqNumeric::csv \xintiiSeq::csv
3372 \let\XINTinFloatSeqNumeric::csv \XINTinFloatSeq::csv
3373 \let\xintSeqBNumeric::csv \xintSeqB::csv
3374 \let\xintiiSeqBNumeric::csv \xintiiSeqB::csv
3375 \let\XINTinFloatSeqBNumeric::csv\XINTinFloatSeqB::csv
3376 \def\xintSeq::csv
3377   {\XINT:NE:two_\xintSeqNumeric::csv{$noexpand\xintSeq::csv}}%
3378 \def\xintiiSeq::csv
3379   {\XINT:NE:two_\xintiiSeqNumeric::csv{$noexpand\xintiiSeq::csv}}%
3380 \def\XINTinFloatSeq::csv
3381   {\XINT:NE:two_\XINTinFloatSeqNumeric::csv{$noexpand\XINTinFloatSeq::csv}}%
3382 \def\xintSeqB::csv
3383   {\XINT:NE:two_\xintSeqBNumeric::csv{$noexpand\xintSeqB:f:csv}}%
3384 \def\xintiiSeqB::csv
3385   {\XINT:NE:two_\xintiiSeqBNumeric::csv{$noexpand\xintiiSeqB:f:csv}}%
3386 \def\XINTinFloatSeqB::csv
3387   {\XINT:NE:two_\XINTinFloatSeqBNumeric::csv{$noexpand\XINTinFloatSeqB:f:csv}}%
3388 \def\xintListSel:x:csv {~xintListSel:f:csv }%
3389 \def\XINTinRandomFloatSdigits{~XINTinRandomFloatSdigits }%
3390 \def\XINTinRandomFloatSixteen{~XINTinRandomFloatSixteen }%
3391 \def\xintiiRandRange{~xintiiRandRange }%
3392 \def\xintiiRandRangeAtoB{~xintiiRandRangeAtoB }%
3393 }%

```

#### 11.59.4 \XINT\_expr\_redefineprints

This is used by `\xintNewExpr` and prior to 1.3e not by `\xintdeffunc`, presumably to avoid some supposedly unneeded overhead. Now also used by `\xintdeffunc` see some comment further below for why.

```

3394 \def\XINT_expr_redefineprints
3395 {%
3396   \def\XINT_flexpr_noopt
3397   {%
3398     \expandafter
3399     \XINT_flexpr_preprint\expandafter-\romannumeral0\xintbarefloateval
3400   }%
3401   \def\XINT_flexpr_preprint ##1##2%
3402   {%
3403     \expandafter\XINT_flexpr_wrap
3404     \csname .;##1.=\XINT_expr_unlock##2\endcsname
3405   }%
3406   \def\XINT_expr_unlock_sp ##1.;##2##3.==##4!%
3407   {%

```

```

3408     \if -##2\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
3409     \XINTdigits{##2##3}{##4}%
3410 }%
3411 \def\xint_expr_print ##1{\expandafter
3412   \xintSPRaw::csv\expandafter{\romannumeral`&&\XINT_expr_unlock ##1}}%
3413 \def\xint_iiexpr_print ##1{\expandafter
3414   \xintCSV::csv\expandafter{\romannumeral`&&\XINT_expr_unlock ##1}}%
3415 \def\xint_boolexpr_print ##1{\expandafter
3416   \xintIsTrue::csv\expandafter{\romannumeral`&&\XINT_expr_unlock ##1}}%
3417 \def\xintCSV::csv   {\xintCSV::csv   }%
3418 \def\xintSPRaw::csv {\xintSPRaw::csv }%
3419 \def\xintPFloat::csv {\xintPFloat::csv }%
3420 \def\xintIsTrue::csv {\xintIsTrue::csv }%
3421 \def\xintRound::csv  {\xintRound::csv  }%
3422 }%

```

### 11.59.5 \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.

```

3423 \def\xintNewExpr      {\XINT_NewExpr\xint_expr_redefineprints\xint_firstofone
3424                       \xinttheexpr\xint_newexpr_clean}%
3425 \def\xintNewFloatExpr{\XINT_NewExpr\xint_expr_redefineprints\xint_firstofone
3426                       \xintthefloatexpr\xint_newexpr_clean}%
3427 \def\xintNewIExpr     {\XINT_NewExpr\xint_expr_redefineprints\xint_firstofone
3428                       \xinttheiexpr\xint_newexpr_clean}%
3429 \def\xintNewIIExpr    {\XINT_NewExpr\xint_expr_redefineprints\xint_firstofone
3430                       \xinttheiiexpr\xint_newexpr_clean}%
3431 \def\xintNewBoolExpr  {\XINT_NewExpr\xint_expr_redefineprints\xint_firstofone
3432                       \xinttheboolexpr\xint_newexpr_clean}%
3433 \def\xint_newexpr_clean #1>{\noexpand\romannumeral`&&@}%

```

1.2c for \xintdeffunc, \xintdefiifunc, \xintdeffloatfunc.

At 1.3, NewFunc does not use a comma delimited pattern anymore.

```

3434 \def\xint_NewFunc
3435   {\XINT_NewExpr\xint_expr_redefineprints\xint_gobble_i\xintthebareeval\xint_newfunc_clean}%
3436 \def\xint_NewFloatFunc
3437   {\XINT_NewExpr\xint_expr_redefineprints\xint_gobble_i\xintthebarefloateval\xint_newfunc_clean}%
3438 \def\xint_NewIIFunc
3439   {\XINT_NewExpr\xint_expr_redefineprints\xint_gobble_i\xintthebareiieval\xint_newfunc_clean}%
3440 \def\xint_newfunc_clean #1>{}%

```

1.2c adds optional logging. For this needed to pass to \_NewExpr\_a the macro name as parameter.

Up to and including 1.2c the definition was global. Starting with 1.2d it is done locally.

Modified at 1.3c so that \XINT\_NewFunc et al. do not execute the \xintexprSafeCatcodes, as it is now already done earlier by \xintdeffunc: and as already #2 was either \xint\_firstofone (for \xintNewExpr et al.) or \xint\_gobble\_i (for \XINT\_NewFunc et al.) we can use that #2. This is only to avoid doing twice the catcodes, as anyhow there is an \endgroup coming later, so external \xintexprRestoreCatcodes would not have been compromised.

Modified at 1.3e: \XINT\_NewFunc et al. do issue \XINT\_expr\_redefineprints. I suppose I did not use it formely as I considered it unneeded overhead, but this meant that \xintdeffunc

foo(x):=\xintfloatexpr bar(x)\relax; was impossible. And in fact this is convenient for xint-trig.sty to transfer float functions to normal functions.

```

3441 \def\xINT_NewExpr #1#2#3#4#5#6[#7]%
3442 {%
3443 \begingroup
3444 \ifcase #7\relax
3445 \toks0 {\endgroup\xINT_global\def#5}%
3446 \or \toks0 {\endgroup\xINT_global\def#5##1}%
3447 \or \toks0 {\endgroup\xINT_global\def#5##1##2}%
3448 \or \toks0 {\endgroup\xINT_global\def#5##1##2##3}%
3449 \or \toks0 {\endgroup\xINT_global\def#5##1##2##3##4}%
3450 \or \toks0 {\endgroup\xINT_global\def#5##1##2##3##4##5}%
3451 \or \toks0 {\endgroup\xINT_global\def#5##1##2##3##4##5##6}%
3452 \or \toks0 {\endgroup\xINT_global\def#5##1##2##3##4##5##6##7}%
3453 \or \toks0 {\endgroup\xINT_global\def#5##1##2##3##4##5##6##7##8}%
3454 \or \toks0 {\endgroup\xINT_global\def#5##1##2##3##4##5##6##7##8##9}%
3455 \fi
3456 #2\xintexprSafeCatcodes
3457 \XINT_expr_redefinmacros
3458 #1%
3459 \XINT_NewExpr_a #2#3#4#5%
3460 }%

```

1.2d's \xintNewExpr makes a local definition. In earlier releases, the definition was global. \the\toks0 inserts the \endgroup, but this will happen after \XINT\_tmpa has already been expanded...

The %1 is \xint\_firstofone for \xintNewExpr, \xint\_gobble\_i for \xintdeffunc.

The ~ action was modified at 1.3e for ~! constructs (userefunc:f macros).

```

3461 \catcode`~ 13 \catcode`@ 14 \catcode`\ 6 \catcode`# 12 \catcode`$ 11 @ $
3462 \def\xINT_NewExpr_a %1%2%3%4%5@
3463 {@
3464 \def\xINT_tmpa %1%2%3%4%5%6%7%8%9{%5}@
3465 \def~%1{\if !%1\noexpand~\else $noexpand$%1\fi}@
3466 \catcode` : 11 \catcode`_ 11
3467 \catcode`# 12 \catcode`~ 13 \escapechar 126
3468 \endlinechar -1 \everyeof {\noexpand }@
3469 \edef\xINT_tmpb
3470 {\scantokens\expandafter{\romannumeral`&&\expandafter
3471 %2\xINT_tmpa{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}\relax}@
3472 }@
3473 \escapechar 92 \catcode`# 6 \catcode`$ 0 @ $
3474 \def~%1{\expandafter\noexpand\csname %1\endcsname}@
3475 \edef\xINT_tmpa %1%2%3%4%5%6%7%8%9@
3476 {\scantokens\expandafter{\expandafter%3\meaning\xINT_tmpb}}@
3477 \the\toks0\expandafter
3478 {\XINT_tmpa{%1}{%2}{%3}{%4}{%5}{%6}{%7}{%8}{%9}}@
3479 %1{\ifxintverbose
3480 \xintMessage{xintexpr}{Info}@
3481 {\string%4\space now with @
3482 \ifxintglobaldefs global \fi meaning \meaning%4}@
3483 \fi}@

```

```
3484 }@
3485 \catcode`% 14
3486 \XINT_setcatcodes % clean up to avoid surprises if something changes
```

### 11.59.6 `\ifxintexprsafecatcodes`, `\xintexprSafeCatcodes`, `\xintexprRestoreCatcodes`

#### 1.3c (2018/06/17).

Added `\ifxintexprsafecatcodes` to allow nesting

```
3487 \newif\ifxintexprsafecatcodes
3488 \let\xintexprRestoreCatcodes\empty
3489 \def\xintexprSafeCatcodes
3490 {%
3491   \unless\ifxintexprsafecatcodes
3492     \edef\xintexprRestoreCatcodes {%
3493       \catcode59=\the\catcode59 % ;
3494       \catcode34=\the\catcode34 % "
3495       \catcode63=\the\catcode63 % ?
3496       \catcode124=\the\catcode124 % |
3497       \catcode38=\the\catcode38 % &
3498       \catcode33=\the\catcode33 % !
3499       \catcode93=\the\catcode93 % ]
3500       \catcode91=\the\catcode91 % [
3501       \catcode94=\the\catcode94 % ^
3502       \catcode95=\the\catcode95 % _
3503       \catcode47=\the\catcode47 % /
3504       \catcode41=\the\catcode41 % )
3505       \catcode40=\the\catcode40 % (
3506       \catcode42=\the\catcode42 % *
3507       \catcode43=\the\catcode43 % +
3508       \catcode62=\the\catcode62 % >
3509       \catcode60=\the\catcode60 % <
3510       \catcode58=\the\catcode58 % :
3511       \catcode46=\the\catcode46 % .
3512       \catcode45=\the\catcode45 % -
3513       \catcode44=\the\catcode44 % ,
3514       \catcode61=\the\catcode61 % =
3515       \catcode96=\the\catcode96 % `
3516       \catcode32=\the\catcode32\relax % space
3517       \noexpand\xintexprsafecatcodesfalse
3518     }%
3519   \fi
3520   \xintexprsafecatcodestrue
3521     \catcode59=12 % ;
3522     \catcode34=12 % "
3523     \catcode63=12 % ?
3524     \catcode124=12 % |
3525     \catcode38=4 % &
3526     \catcode33=12 % !
3527     \catcode93=12 % ]
3528     \catcode91=12 % [
3529     \catcode94=7 % ^
3530     \catcode95=8 % _
```

```
3531      \catcode47=12 % /
3532      \catcode41=12 % )
3533      \catcode40=12 % (
3534      \catcode42=12 % *
3535      \catcode43=12 % +
3536      \catcode62=12 % >
3537      \catcode60=12 % <
3538      \catcode58=12 % :
3539      \catcode46=12 % .
3540      \catcode45=12 % -
3541      \catcode44=12 % ,
3542      \catcode61=12 % =
3543      \catcode96=12 % `
3544      \catcode32=10 % space
3545 }%
3546 \let\XINT_tmpa\undefined \let\XINT_tmpb\undefined \let\XINT_tmpc\undefined
3547 \ifdefined\RequirePackage\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
3548 {\RequirePackage{xinttrig}}%
3549 \RequirePackage{xintlog}}%
3550 {\input xinttrig.sty
3551 \input xintlog.sty
3552 }%
3553 \XINT_restorecatcodes_endinput%
```

## 12 Package [xinttrig](#) implementation

### Contents

12.1	Catcodes, $\epsilon$ -TeX and reload detection	390
12.2	Library identification	391
12.3	Ensure used letters are dummy letters	391
12.4	<code>\xintreloadxinttrig</code>	392
12.5	Auxiliary variables (only temporarily needed, but left free to re-use)	392
12.5.1	<code>twoPi</code> , <code>threePiOver2</code> , <code>Pi</code> , <code>PiOver2</code>	392
12.5.2	<code>oneDegree</code> , <code>oneRadian</code>	392
12.5.3	Inverse factorial coefficients: <code>invfact2</code> , ..., <code>invfact44</code>	392
12.6	The sine and cosine series	393
12.6.1	<code>sin_aux()</code> , <code>cos_aux()</code>	393
12.6.2	Make <code>sin_aux()</code> and <code>cos_aux()</code> known to <code>\xintexpr</code>	395
12.6.3	<code>sin_()</code> , <code>cos_()</code>	395
12.7	Range reduction for sine and cosine using degrees	395
12.7.1	Core level macro <code>\XINT_mod_ccclx_i</code>	395
12.7.2	<code>sind_()</code> , <code>cosd_()</code> , and support macros <code>\xintSind</code> , <code>\xintCosd</code>	396
12.8	<code>sind()</code> , <code>cosd()</code>	400
12.9	<code>sin()</code> , <code>cos()</code>	400
12.10	<code>sinc()</code>	400
12.11	<code>tan()</code> , <code>tand()</code> , <code>cot()</code> , <code>cotd()</code>	401
12.12	<code>sec()</code> , <code>secd()</code> , <code>csc()</code> , <code>cscd()</code>	401
12.13	Core routine for inverse trigonometry	401
12.14	<code>asin()</code> , <code>asind()</code>	403
12.15	<code>acos()</code> , <code>acosd()</code>	403
12.16	<code>atan()</code> , <code>atand()</code>	403
12.17	<code>Arg()</code> , <code>atan2()</code> , <code>Argd()</code> , <code>atan2d()</code> , <code>pArg()</code> , <code>pArgd()</code>	404
12.18	Synonyms: <code>tg()</code> , <code>cotg()</code>	405
12.19	Let the functions be known to the <code>\xintexpr</code> parser	405

The original was done in January 15 and 16, 2019. It provided `asin()` and `acos()` based on a Newton algorithm approach. Then during March 25-31 I revisited the code, adding more inverse trigonometrical functions (with a modified algorithm, quintically convergent), extending the precision range (so that the package reacts to the `\xintDigits` value at time of load, or reload), and replaced high level range reduction by some optimized lower level coding.

This led me next to improve upon the innards of `\xintdeffunc` and `\xintNewExpr`, and to add to `xintexpr` the `\xintdefefunc` macro (see user documentation).

Finally on April 5, 2019 I pushed the idea of the algorithm for the arcsine function to its logical limit obtaining the method finally in use here.

Almost all of the code remains written at high level, and in particular it is not easily feasible from this interface to execute computations with guard digits. Expect the last one or two digits to be systematically off.

Also, small floating-point inputs are handled quite sub-optimally both for the direct and inverse functions; substantial gains are possible. I added the `ilog10()` function too late to consider using it here with the high level interface.

### 12.1 Catcodes, $\epsilon$ -TeX and reload detection

```
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 \catcode94=7 % ^
13 \def\z{\endgroup}%
14 \def\empty{}\def\space{ }\newlinechar10
15 \expandafter\let\expandafter\w\csname ver@xintexpr.sty\endcsname
16 \expandafter
17 \ifx\csname PackageInfo\endcsname\relax
18 \def\y#1#2{\immediate\write-1{Package #1 Info:^^}%
19 \space\space\space\space#2.}}%
20 \else
21 \def\y#1#2{\PackageInfo{#1}{#2}}%
22 \fi
23 \expandafter
24 \ifx\csname numexpr\endcsname\relax
25 \y{xinttrig}{numexpr not available, aborting input}%
26 \aftergroup\endinput
27 \else
28 \ifx\w\relax % xintexpr.sty not yet loaded.
29 \y{xinttrig}%
30 {Loading should be via \ifx\x\empty\string\usepackage{xintexpr.sty}
31 \else\string\input\space xintexpr.sty \fi
32 rather, aborting}%
33 \aftergroup\endinput
34 \fi
35 \fi
36 \z%
37 \catcode`_ 11 \XINT_setcatcodes \catcode`? 12

```

## 12.2 Library identification

```

38 \ifcsname xintlibver@trig\endcsname
39 \expandafter\xint_firstoftwo
40 \else
41 \expandafter\xint_secondoftwo
42 \fi
43 {\immediate\write-1{Reloading xinttrig library using Digits=\xinttheDigits.}}%
44 {\expandafter\gdef\csname xintlibver@trig\endcsname{2019/04/05 1.3e}%
45 \XINT_providespackage
46 \ProvidesPackage{xinttrig}%
47 [2019/04/05 1.3e Trigonometrical functions for xintexpr (JFB)]%
48 }%

```

## 12.3 Ensure used letters are dummy letters

```

49 \xintFor* #1 in {iDTVtuwxyzX}\do{\xintensuredummy{#1}}%

```

## 12.4 `\xintreloadxinttrig`

```
50 \def\xintreloadxinttrig
51   {\edef\XINT_restorecatcodes_now{\XINT_restorecatcodes}%
52    \XINT_setcatcodes\catcode`? 12
53    \input xinttrig.sty
54    \XINT_restorecatcodes_now}%
```

## 12.5 Auxiliary variables (only temporarily needed, but left free to re-use)

These variables don't have really private names but this does not matter because only their actual values will be stored in the functions defined next. Nevertheless they are not unassigned, and are left free to use as is.

### 12.5.1 `twoPi`, `threePiover2`, `Pi`, `Piover2`

We take them with 60 digits and force conversion to `\xintDigits` setting via "0 + " syntax.

```
55 \xintdeffloatvar twoPi      := 0 +
56   6.28318530717958647692528676655900576839433879875021164194989;%
57 \xintdeffloatvar threePiover2 := 0 +
58   4.71238898038468985769396507491925432629575409906265873146242;%
59 \xintdeffloatvar Pi        := 0 +
60   3.14159265358979323846264338327950288419716939937510582097494;%
61 \xintdeffloatvar Piover2   := 0 +
62   1.57079632679489661923132169163975144209858469968755291048747;%
```

### 12.5.2 `oneDegree`, `oneRadian`

```
63 \xintdeffloatvar oneDegree := 0 +
64   0.0174532925199432957692369076848861271344287188854172545609719;% Pi/180
65 \xintdeffloatvar oneRadian := 0 +
66   57.2957795130823208767981548141051703324054724665643215491602;% 180/Pi
```

### 12.5.3 Inverse factorial coefficients: `invfact2`, ..., `invfact44`

Pre-compute  $1/n!$  for  $n = 2, \dots, 44$

The following example (among many, see below) shows that we must be careful when pre-computing the  $1/i!$ .

Consider  $35!=10333147966386144929666651337523200000000$ .  
 With `\xintDigit:=26; \xintfloateval{35!}` obtains  $1.0333147966386144929666651e40$  which is the correct rounding to 26 digits. But `\xintfloateval{1/35!}` obtains  $9.6775929586318909920898167e-41$  which differs by 3ulps from the correct rounding of  $1/35!$  to 26 places which is  $9.6775929586318909920898164e-41$ . The problem isn't in the factorial computations, but in the fact that the rounding of the inverse of a quantity which is itself a rounding is not necessarily the rounding of the exact inverse of the original.

Here is a little program to explore this phenomenon systematically:

```
\xintDigits:=55;%
\edef\tempNlist{\xintSeq{2}{39}}
\xintFor*#1in\tempNlist\do{% we precompute some rounding here to
% speed up things in the next double loop.
\expandafter\edef\cname invfact#1\endcname {\xintfloatexpr 1/#1!\relax}%
}%
```



```

\intFor*#1in{\intSeq{4}{50}}\do{%
  \intDigits:=#1;%
  \intFor*#2in{\tempNlist}\do{%
    (D=#1, N=#2)
    % attention to != which is parsed as negation operator != followed by = (sigh...)
    \intifboolfloatexpr{(1/#2!)==0+\csname invfact#2\endcsname}%
    {ok}
    {mismatch: \intfloateval{1/#2!} vs (exact)
      \intfloateval{0+\csname invfact#2\endcsname}}%
  \par
}%
}

```

We can see that for D=16, the problem is there with N=22, 25, 26, 27, 28...and more. If we were to use 1/i! directly in the `\intdeffloatefunc` of `sin_aux(X)` and `cos_aux(X)` we would have this problem.

If we use `\intexpr1/i!\relax` encapsulation in the function declaration the rounding will be delayed to actual use of the function... which is bad, so we need it to happen now. We could use `(0+\intexpr1/i!\relax)` inside the declaration of the sine and cosine series, which will give the expected result but for readability we use some temporary variables. We could use `seq(0+\intexpr1/i!\relax, i = 2..44)` but opt for an `rseq`. The semi-colon must be braced to hide it from `\intdeffloatvar` grabbing of the delimited argument.

```

67 \intdeffloatvar invfact\intListWithSep{, invfact}{\intSeq{2}{44}}%
68 := seq(0+x, x=\intexpr rseq(1/2{;}@/i, i=3..44)\relax);% need to hide inner ;

```

## 12.6 The sine and cosine series

### 12.6.1 `sin_aux()`, `cos_aux()`

Should I rather use successive divisions by  $(2n+1)(2n)$ , or rather multiplication by their pre-computed inverses, in a modified Horner scheme? The `\ifnum` tests are executed at time of definition.

Criteria for truncated series using  $\pi/4$ , actually 0.79.

Small values of the variable X are very badly handled here because a much shorter truncation of the sine series should be used.

```

69 \intdeffloatefunc sin_aux(X) := 1 - X(invfact3 - X(invfact5
70 \ifnum\XINTdigits>4
71 - X(invfact7
72 \ifnum\XINTdigits>6
73 - X(invfact9
74 \ifnum\XINTdigits>8
75 - X(invfact11
76 \ifnum\XINTdigits>10
77 - X(invfact13
78 \ifnum\XINTdigits>13
79 - X(invfact15
80 \ifnum\XINTdigits>15
81 - X(invfact17
82 \ifnum\XINTdigits>18
83 - X(invfact19
84 \ifnum\XINTdigits>21
85 - X(invfact21
86 \ifnum\XINTdigits>24

```





360 divides 9000 hence  $10^k$  is 280 for  $k$  at least 3 and the additive group generated by it modulo 360 is the set of multiples of 40.

```

157 \def\XINT_mod_ccclx_i #1.% input <k>.<N>. k is a non-negative exponent
158 {%
159   \expandafter\XINT_mod_ccclx_e\the\numexpr
160   \expandafter\XINT_mod_ccclx_j\the\numexpr1\ifcase#1 \or0\or00\else000\fi.%
161 }%
162 \def\XINT_mod_ccclx_j 1#1.#2.% #2=N is a non-negative mantissa
163 {%
164   (\XINT_mod_ccclx_ja {++}#2#1\XINT_mod_ccclx_jb 0000000\relax
165   }%
166   1      2345678
167 \def\XINT_mod_ccclx_ja #1#2#3#4#5#6#7#8#9%
168 {%
169   #9+#8+#7+#6+#5+#4+#3+#2\xint_firstoftwo{\XINT_mod_ccclx_ja{+#9+#8+#7}}{#1}%
170 }%
171 \def\XINT_mod_ccclx_jb #1\xint_firstoftwo#2#3{#1+0)*280\XINT_mod_ccclx_jc #1#3}%

```

Attention that `\XINT_cclcx_e` wants non negative input because `\numexpr` division is not periodical ...

```

171 \def\XINT_mod_ccclx_jc  +#1+#2+#3#4\relax{+80*(#3+#2+#1)+#3#2#1.}%
172 \def\XINT_mod_ccclx_e#1.{\expandafter\XINT_mod_ccclx_z\the\numexpr(#1+180)/360-1.#1.}%
173 \def\XINT_mod_ccclx_z#1.#2.{#2-360*#1.}%

```

### 12.7.2 `sind_()`, `cosd_()`, and support macros `\xintSind`, `\xintCosd`

`sind_()` coded directly at macro level with a macro `\xintSind` (ATTENTION! it requires a positive argument) which will suitably use `\XINT_flexpr_func_sin_` defined from `\xintdeffloatefunc`

```

174 \def\XINT_flexpr_func_sind_ #1#2#3%
175 {%
176   \expandafter #1\expandafter #2\csname.=%
177   \XINT:NEhook:one\xintSind{\XINT_expr_unlock#3}\endcsname
178 }%

```

Must be f-expandable for nesting macros from `\xintNewExpr`  
**ATTENTION ONLY FOR POSITIVE ARGUMENTS**

```

179 \def\xintSind#1{\romannumeral`&&\expandafter\xintsind
180   \romannumeral0\XINTinfloatS[\XINTdigits]{#1}}%
181 \def\xintsind #1[#2#3]%
182 {%
183   \xint_UDsignfork
184   #2\XINT_sind
185   -\XINT_sind_int
186   \kroft#2#3.#1..%<< attention extra dot
187 }%
188 \def\XINT_sind #1.#2.% NOT TO BE USED WITH VANISHING (OR NEGATIVE) #2.
189 {%
190   \expandafter\XINT_sind_a
191   \romannumeral0\xinttrunc{\XINTdigits}{#2[#1]}%
192 }%
193 \def\XINT_sind_a{\expandafter\XINT_sind_i\the\numexpr\XINT_mod_ccclx_i0.}%

```

```

194 \def\XINT_sind_int
195 {%
196   \expandafter\XINT_sind_i\the\numexpr\expandafter\XINT_mod_ccclx_i
197 }%
198 \def\XINT_sind_i #1.% range reduction inside [0, 360[
199 {%
200   \ifcase\numexpr#1/90\relax
201     \expandafter\XINT_sind_A
202   \or\expandafter\XINT_sind_B\the\numexpr-90+%
203   \or\expandafter\XINT_sind_C\the\numexpr-180+%
204   \or\expandafter\XINT_sind_D\the\numexpr-270+%
205   \else\expandafter\XINT_sind_E\the\numexpr-360+%
206   \fi#1.%
207 }%

```

#2 will be empty in the "integer branch". Notice that a single dot "." is valid as input to the xintfrac macros. During developing phase I did many silly mistakes due to wanting to use too low-level interface, e.g. I would use something like #2[-\XINTdigits] with #2 the fractional digits, but there maybe some leading zero and then xintfrac.sty will think the whole thing is zero due to the requirements of my own core format A[N]...

The "userefunc" auxiliary macros do not pre-expand their arguments (but the macros which end up used in other ones defined from \csbxintdeffunc do).

Multiplication is done exactly but anyway currently float multiplication goes via exact multiplication after rounding arguments ; as here integer part has at most three digits, doing exact multiplication will prove not only more accurate but probably faster.

```

208 \def\XINT_sind_A#1{%
209 \def\XINT_sind_A##1.##2.%
210 {%
211   \expandafter\XINT_flexpr_userefunc_sin_\expandafter
212     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{##1.##2}{#1}}}%
213 }%
214 }\expandafter
215 \XINT_sind_A\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%
216 \def\XINT_sind_B#1{\xint_UDsignfork#1\XINT_sind_B_n-\XINT_sind_B_p\krof #1}%
217 \def\XINT_tmpa#1{%
218 \def\XINT_sind_B_n-##1.##2.%
219 {%
220   \expandafter\XINT_flexpr_userefunc_cos_\expandafter
221     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{\xintSub{##1[0]}{.##2}}{#1}}}%
222 }%
223 \def\XINT_sind_B_p##1.##2.%
224 {%
225   \expandafter\XINT_flexpr_userefunc_cos_\expandafter
226     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{##1.##2}{#1}}}%
227 }%
228 }\expandafter
229 \XINT_tmpa\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%
230 \def\XINT_sind_C#1{\xint_UDsignfork#1\XINT_sind_C_n-\XINT_sind_C_p\krof #1}%
231 \def\XINT_tmpa#1{%
232 \def\XINT_sind_C_n-##1.##2.%
233 {%
234   \expandafter\XINT_flexpr_userefunc_sin_\expandafter

```

```

235      {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{\xintSub{##1[0]}.{##2}}{#1}}}%
236 }%
237 \def\XINT_sind_C_p##1.##2.%
238 {%
239   \xintiopp\expandafter\XINT_flexpr_userefunc_sin_\expandafter
240     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{##1.##2}{#1}}}%
241 }%
242 }\expandafter
243 \XINT_tmpa\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%
244 \def\XINT_sind_D#1{\xint_UDsignfork#1\XINT_sind_D_n-\XINT_sind_D_p\krof #1}%
245 \def\XINT_tmpa#1{%
246 \def\XINT_sind_D_n-##1.##2.%
247 {%
248   \xintiopp\expandafter\XINT_flexpr_userefunc_cos_\expandafter
249     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{\xintSub{##1[0]}.{##2}}{#1}}}%
250 }%
251 \def\XINT_sind_D_p##1.##2.%
252 {%
253   \xintiopp\expandafter\XINT_flexpr_userefunc_cos_\expandafter
254     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{##1.##2}{#1}}}%
255 }%
256 }\expandafter
257 \XINT_tmpa\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%
258 \def\XINT_sind_E#1{%
259 \def\XINT_sind_E-##1.##2.%
260 {%
261   \xintiopp\expandafter\XINT_flexpr_userefunc_sin_\expandafter
262     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{\xintSub{##1[0]}.{##2}}{#1}}}%
263 }%
264 }\expandafter
265 \XINT_sind_E\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%

```

#### The cosd\_ auxiliary function

```

266 \def\XINT_flexpr_func_cosd_ #1#2#3%
267 {%
268   \expandafter #1\expandafter #2\csname.=%
269   \XINT:NEhook:one\xintCosd{\XINT_expr_unlock#3}\endcsname
270 }%

```

#### ATTENTION ONLY FOR POSITIVE ARGUMENTS

```

271 \def\xintCosd#1{\romannumeral`&&\expandafter\xintcosd
272   \romannumeral0\XINTinfloatS[\XINTdigits]{#1}}%
273 \def\xintcosd #1[#2#3]%
274 {%
275   \xint_UDsignfork
276     #2\XINT_cosd
277     -\XINT_cosd_int
278   \krof#2#3.#1..%<< attention extra dot
279 }%
280 \def\XINT_cosd #1.#2.% NOT TO BE USED WITH VANISHING (OR NEGATIVE) #2.
281 {%
282   \expandafter\XINT_cosd_a

```

```

283 \romannumeral0\xinttrunc{\XINTdigits}{#2[#1]}%
284 }%
285 \def\XINT_cosd_a{\expandafter\XINT_cosd_i\the\numexpr\XINT_mod_ccclx_i0.}%
286 \def\XINT_cosd_int
287 {%
288 \expandafter\XINT_cosd_i\the\numexpr\expandafter\XINT_mod_ccclx_i
289 }%
290 \def\XINT_cosd_i #1.%
291 {%
292 \ifcase\numexpr#1/90\relax
293 \expandafter\XINT_cosd_A
294 \or\expandafter\XINT_cosd_B\the\numexpr-90+%
295 \or\expandafter\XINT_cosd_C\the\numexpr-180+%
296 \or\expandafter\XINT_cosd_D\the\numexpr-270+%
297 \else\expandafter\XINT_cosd_E\the\numexpr-360+%
298 \fi#1.%
299 }%

```

#2 will be empty in the "integer" branch, but attention in general branch to handling of negative integer part after the subtraction of 90, 180, 270, or 360, and avoid abusing A[N] notation which yes speeds up xintfrac parsing but has its pitfalls.

```

300 \def\XINT_cosd_A#1{%
301 \def\XINT_cosd_A##1.##2.%
302 {%
303 \expandafter\XINT_flexpr_userefunc_cos_\expandafter
304 {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{##1.##2}{#1}}}%
305 }%
306 }\expandafter
307 \XINT_cosd_A\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%
308 \def\XINT_cosd_B#1{\xint_UDsignfork#1\XINT_cosd_B_n-\XINT_cosd_B_p\krof #1}%
309 \def\XINT_tmpa#1{%
310 \def\XINT_cosd_B_n-##1.##2.%
311 {%
312 \expandafter\XINT_flexpr_userefunc_sin_\expandafter
313 {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{\xintSub{##1[0]}{.##2}}{#1}}}%
314 }%
315 \def\XINT_cosd_B_p##1.##2.%
316 {%
317 \xintiopp\expandafter\XINT_flexpr_userefunc_sin_\expandafter
318 {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{##1.##2}{#1}}}%
319 }%
320 }\expandafter
321 \XINT_tmpa\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%
322 \def\XINT_cosd_C#1{\xint_UDsignfork#1\XINT_cosd_C_n-\XINT_cosd_C_p\krof #1}%
323 \def\XINT_tmpa#1{%
324 \def\XINT_cosd_C_n-##1.##2.%
325 {%
326 \xintiopp\expandafter\XINT_flexpr_userefunc_cos_\expandafter
327 {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{\xintSub{##1[0]}{.##2}}{#1}}}%
328 }%
329 \def\XINT_cosd_C_p##1.##2.%
330 {%

```

```

331 \xintiopp\expandafter\XINT_flexpr_userefunc_cos_\expandafter
332     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{##1.##2}{#1}}}%
333 }%
334 }\expandafter
335 \XINT_tmpa\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%
336 \def\XINT_cosd_D#1{\xint_UDsignfork#1\XINT_cosd_D_n-\XINT_cosd_D_p\krof #1}%
337 \def\XINT_tmpa#1{%
338 \def\XINT_cosd_D_n-##1.##2.%
339 {%
340 \xintiopp\expandafter\XINT_flexpr_userefunc_sin_\expandafter
341     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{\xintSub{##1[0]}{.##2}}{#1}}}%
342 }%
343 \def\XINT_cosd_D_p##1.##2.%
344 {%
345 \expandafter\XINT_flexpr_userefunc_sin_\expandafter
346     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{##1.##2}{#1}}}%
347 }%
348 }\expandafter
349 \XINT_tmpa\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%
350 \def\XINT_cosd_E#1{%
351 \def\XINT_cosd_E-##1.##2.%
352 {%
353 \expandafter\XINT_flexpr_userefunc_cos_\expandafter
354     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{\xintSub{##1[0]}{.##2}}{#1}}}%
355 }%
356 }\expandafter
357 \XINT_cosd_E\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%

```

## 12.8 sind(), cosd()

```

358 \xintdeffloatefunc sind(x) := ifsgn(x, if(x>=-45, sin_(x*oneDegree), -sind_(-x)),
359     0,
360     if(x<=45, sin_(x*oneDegree), sind_(x)));%
361 \xintdeffloatefunc cosd(x) := ifsgn(x, if(x>=-45, cos_(x*oneDegree), cosd_(-x)),
362     1,
363     if(x<=45, cos_(x*oneDegree), cosd_(x)));%

```

## 12.9 sin(), cos()

For some reason I did not define sin() and cos() in January 2019 ??

```

364 \xintdeffloatefunc sin(x):= if(abs(x)<0.79, sin_(x),%
365     ifsgn(x, -sind_(-x*oneRadian),
366     0,
367     sind_(x*oneRadian))
368     );%
369 \xintdeffloatefunc cos(x):= if(abs(x)<0.79, cos_(x), cosd_(abs(x*oneRadian)));%

```

## 12.10 sinc()

Should I also consider adding  $(1-\cos(x))/x^2$  ?

```

370 \xintdeffloatefunc sinc(x):=
371     if(abs(x)<0.79, sin_aux(sqrt(x)), sind_(abs(x)*oneRadian)/abs(x));%

```



## 12.11 tan(), tand(), cot(), cotd()

The 0 in cot(x) is a dummy place holder, 1/0 would raise an error at time of definition...

```

372 \xintdeffloatfunc tand(x):= sind(x)/cosd(x);%
373 \xintdeffloatfunc cotd(x):= cosd(x)/sind(x);%
374 \xintdeffloatfunc tan(x) := ifsgn(x, if(x>-0.79, sin(x)/cos(x), -cotd(90+x*oneRadian)),
375                               0,
376                               if(x<0.79, sin(x)/cos(x), cotd(90-x*oneRadian))
377                               );%
378 \xintdeffloatfunc cot(x) := if(abs(x)<0.79, cos(x)/sin(x),
379                               ifsgn(x, -tand(90+x*oneRadian),
380                               0,
381                               tand(90-x*oneRadian))
382                               );%

```

## 12.12 sec(), secd(), csc(), cscd()

```

383 \xintdeffloatfunc sec(x) := inv(cos(x));%
384 \xintdeffloatfunc csc(x) := inv(sin(x));%
385 \xintdeffloatfunc secd(x):= inv(cosd(x));%
386 \xintdeffloatfunc cscd(x):= inv(sind(x));%

```

## 12.13 Core routine for inverse trigonometry

Compute asin(x)

The approach I shall first describe (which is only a first step towards our final approach) converges quintically but requires an initial square root computation. However, when used for atan(x), we then do not have to do any such square root extraction. See code next.

The algorithm (for this first approach): we have  $0 \leq t < 0.72$ , let  $t_1 = t*(1+t^2/6)$ . We also have  $u = \sqrt{1 - t^2}$ . We seek  $a = \text{Arcsin } t$  with  $t = \sin(a)$ .

Then  $t_1 < \text{Arcsin } t$  and the difference (we don't know it!)  $\delta_1$  is  $< 0.02$ . We compute  $D = t*\cos(t_1) - u*\sin(t_1)$ . This computation is done "exactly" via the `\xintexpr` encapsulation. In other terms we use doubled precision. Anyhow, currently (1.3e) the Float macros of `xintfrac.sty` for multiplication do go via such exact multiplication when the mantissas have the expected sizes. So we can't gain but only lose due to catastrophic subtraction in using float operations here.

Thus  $D$  is  $\sin(a-t_1) = \sin(\delta_1)$ . And  $\delta_1 = \text{Arcsin } D$ , but  $D$  is small! We then use again two terms of the Arcsin series and define  $t_2 = t_1 + D * (1 + D^2/6)$ . Let  $\delta_2 = a - t_2$ . Then  $\delta_2$  is of the order of the neglected term  $3*(\delta_1)^5/40$ .

©copyright J.F. Burnol, March 30, 2019. This surely has a name.

The algorithm is quintically convergent! I must have thought about this some many years ago, but I like it a lot and I found it again on March 30, 2019. One can do the same to go from exp to log. Basically the idea is that we can improve the Newton Method for any function  $f$  for which knowing target value of  $f$  implies one also knows target value of its derivative. In fact I obtained the quintic algorithm by combining the Newton formula with the one from using  $f(x)/f'(a)$  and not  $f(x)/f'(x)$  in the update to cancel the two quadratic errors.

One iteration ( $t_2$ ) gives about 9 digits, two iterations ( $t_3$ ) 49 digits ! And if we want hepta-convergence we only need to use one more term of the Arcsin series in the update of the  $t_n$ ... really this is very nice.

And actually ( $t_2$ ) already gives 30 digits of floating point precision for input  $t < 0.1$ . Let's confirm this:

```

> Digits := 60:
> t0 := 0.1; t1 := t0*(1+t0^2/6); u0 := sqrt(1-t0^2); D1 :=t0*cos(t1)-u0*sin(t1);
t0 := 0.1

```



```

393             + X(5/112
394 \ifnum\XINTdigits>25
395             + X(35/1152
396 \ifnum\XINTdigits>35
397             + X(63/2816
398 \ifnum\XINTdigits>46
399             + X(231/13312
400             )\fi)\fi)\fi)\fi)\fi)\fi;%
401 \xintdeffloatefunc asin_o(D, T) := T + D*asin_aux(sqrt(D));%
402 \xintdeffloatefunc asin_n(V, T, t, u) :=% V is square of T
403             asin_o (\xintexpr t*cos_aux(V) - u*T*sin_aux(V)\relax, T);%
404 \xintdeffloatefunc asin_m(T, t, u) := asin_n(sqrt(T), T, t, u);%
405 \xintdeffloatefunc asin_l(t, u) := asin_m(t*asin_aux(sqrt(t)), t, u);%

```

## 12.14 asin(), asind()

Only non-negative arguments  $t$  and  $u$  for  $\text{asin}_a(t,u)$ , and  $\text{asind}_a(t,u)$ .

```

406 \xintdeffloatefunc asin_a(t, u) :=
407     if(t<u, asin_l(t, u), Pi/2 - asin_l(u, t));%
408 \xintdeffloatefunc asind_a(t, u):=
409     if(t<u, asin_l(t, u) * oneRadian, 90 - asin_l(u, t) * oneRadian);%
410 \xintdeffloatefunc asin(t) := ifsgn(t, -asin_a(-t, sqrt(1-sqr(t))),
411                               0,
412                               asin_a(t, sqrt(1-sqr(t))));%
413 \xintdeffloatefunc asind(t) := ifsgn(t, -asind_a(-t, sqrt(1-sqr(t))),
414                               0,
415                               asind_a(t, sqrt(1-sqr(t))));%

```

## 12.15 acos(), acosd()

```

416 \xintdeffloatefunc acos(t) := Pi/2 - asin(t);%
417 \xintdeffloatefunc acosd(t):= 90 - asind(t);%

```

## 12.16 atan(), atand()

This involves no square root!

TeX hackers note 1:

The `subs( , x = ..)` mechanism has no utility in a function definition, there is no parallel mechanism at the underlying macros, so in fact the substituted things will remain unevaluated if they involve indeterminates, so this is exactly like not trying to make things more efficient at all.

Currently, the only way is thus to employ auxiliary functions like is done next. Contrarily to TeX macros, we must define the functions one after the other in the correct order, so the auxiliaries come first.

TeX hackers note 2:

The `if(,,)` and `ifsgn(,,)` tests when used numerically compute all ; but when used into a `\xintdeffloatefunc`, they are converted to macros with basically `\firstofthree`, `\secondofthree`, `\thirdofthree` behaviour so then only the actually executed branch will do computations.

For numeric computations the `?` and `??` operators are used for this effect, but they can not be used in `\xintdeffloatefunc` if the test involves unknown variables; as explained above fortunately then `if(,,)` and `ifsgn(,,)` work.

radians

```

418 \xintdeffloatefunc atan_b(t, w, z):=%
419     0.5 * if(w< 0, Pi - asin_a(2z * t, -w*z), asin_a(2z * t, w*z));%
420 \xintdeffloatefunc atan_a(t, T) := atan_b(t, 1-T, inv(1+T));%
421 \xintdeffloatefunc atan(t):= ifsgn(t,-atan_a(-t, sqr(t)), 0, atan_a(t, sqr(t)));%

```

degrees

```

422 \xintdeffloatefunc atand_b(t, w, z) :=
423     0.5 * if(w< 0, 180 - asind_a(2z * t, -w*z), asind_a(2z * t, w*z));%
424 \xintdeffloatefunc atand_a(t, T) := atand_b(t, 1-T, inv(1+T));%
425 \xintdeffloatefunc atand(t):= ifsgn(t,-atand_a(-t, sqr(t)), 0, atand_a(t, sqr(t)));%

```

## 12.17 Arg(), atan2(), Argd(), atan2d(), pArg(), pArgd()

Arg(x,y) function from  $-\pi$  (excluded) to  $+\pi$  (included)

```

426 \xintdeffloatefunc Arg(x, y):=
427     if(y>x,
428         if(y>-x, Piover2 - atan(x/y),
429             if(y<0, -Pi + atan(y/x), Pi + atan(y/x))),
430         if(y>-x, atan(y/x), -Piover2 + atan(x/-y))
431     );%

```

atan2(y,x) = Arg(x,y) ... (some people have atan2 with arguments reversed but the convention here seems the most often encountered)

```

432 \xintdeffloatefunc atan2(y,x) := Arg(x, y);%

```

Argd(x,y) function from -180 (excluded) to +180 (included)

```

433 \xintdeffloatefunc Argd(x, y):=
434     if(y>x,
435         if(y>-x, 90 - atand(x/y),
436             if(y<0, -180 + atand(y/x), 180 + atand(y/x))),
437         if(y>-x, atand(y/x), -90 + atand(x/-y))
438     );%

```

atan2d(y,x) = Argd(x,y)

```

439 \xintdeffloatefunc atan2d(y,x) := Argd(x, y);%

```

pArg(x,y) function from 0 (included) to  $2\pi$  (excluded) I hesitated between pArg, Argpos, and Argplus. Opting for pArg in the end.

```

440 \xintdeffloatefunc pArg(x, y):=
441     if(y>x,
442         if(y>-x, Piover2 - atan(x/y), Pi + atan(y/x)),
443         if(y>-x, if(y<0, twoPi + atan(y/x), atan(y/x)),
444             threePiover2 + atan(x/-y))
445     );%

```

pArgd(x,y) function from 0 (included) to 360 (excluded)

```

446 \xintdeffloatefunc pArgd(x, y):=
447     if(y>x,
448         if(y>-x, 90 - atan(x/y)*oneRadian, 180 + atan(y/x)*oneRadian),
449         if(y>-x, if(y<0, 360 + atan(y/x)*oneRadian, atan(y/x)*oneRadian),
450             270 + atan(x/-y)*oneRadian)
451     );%

```

## 12.18 Synonyms: `tg()`, `cotg()`

These are my childhood notations and I am attached to them. In radians only. We skip some overhead here by using a `\let` at core level.

```
452 \expandafter\let\csname XINT_flexpr_func_tg\expandafter\endcsname
453           \csname XINT_flexpr_func_tan\endcsname
454 \expandafter\let\csname XINT_flexpr_func_cotg\expandafter\endcsname
455           \csname XINT_flexpr_func_cot\endcsname
```

## 12.19 Let the functions be known to the `\xintexpr` parser

See `xint.pdf` for some explanations (as well as code comments in `xintexpr.sty`). In fact it is this context which led to my addition at 1.3e of `\xintdefefunc` to the `\xintexpr` syntax.

```
456 \xintFor #1 in {sin, cos, tan, sec, csc, cot,
457           asin, acos, atan}\do
458 {%
459   \xintdefefunc #1(x) := \xintfloatexpr #1(sfloat(x))\relax;%
460   \xintdefefunc #1d(x) := \xintfloatexpr #1d(sfloat(x))\relax;%
461 }%
462 \xintFor #1 in {Arg, pArg, atan2}\do
463 {%
464   \xintdefefunc #1(x, y) := \xintfloatexpr #1(sfloat(x), sfloat(y))\relax;%
465   \xintdefefunc #1d(x, y) := \xintfloatexpr #1d(sfloat(x), sfloat(y))\relax;%
466 }%
467 \xintdefefunc tg(x) := \xintfloatexpr tg(sfloat(x))\relax;%
468 \xintdefefunc cotg(x) := \xintfloatexpr cotg(sfloat(x))\relax;%
469 \xintdefefunc sinc(x) := \xintfloatexpr sinc(sfloat(x))\relax;%
```

Restore used dummy variables to their status prior to the package reloading. On first loading this is not needed naturally, because this is done immediately at end of `xintexpr.sty`.

```
470 \xintFor* #1 in {iDTVtuwxyzX}\do{\xintrestorelettervar{#1}}%
```

## 13 Package [xintlog](#) implementation

### Contents

13.1	Catcodes, $\varepsilon$ -TeX and reload detection	406
13.2	Library identification	407
13.3	Loading of poormanlog package	407
13.4	Support macros for natural logarithm and exponential	407
13.5	The log(), exp(), and pow() function	407

I almost included extended precision implementation for 1.3e but was a bit short on time; besides I hesitated between using poormanlog at starting point or not. For up to 50 digits, it would help reduce considerably the needed series for the logarithm. For more digits I should rather apply my copyrighted method of the arcsine (it must be in literature).

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

```

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 \catcode94=7  % ^
13 \def\z{\endgroup}%
14 \def\empty{\def\space{ }\newlinechar10
15 \expandafter\let\expandafter\w\csname ver@xintexpr.sty\endcsname
16 \expandafter\let\expandafter\x\csname ver@xintlog.sty\endcsname
17 \expandafter
18   \ifx\csname PackageInfo\endcsname\relax
19     \def\y#1#2{\immediate\write-1{Package #1 Info:^^J%
20       \space\space\space\space#2.}}%
21   \else
22     \def\y#1#2{\PackageInfo{#1}{#2}}%
23   \fi
24 \expandafter
25 \ifx\csname numexpr\endcsname\relax
26   \y{xintlog}{\numexpr not available, aborting input}%
27   \aftergroup\endinput
28 \else
29   \ifx\w\relax % xintexpr.sty not yet loaded.
30     \y{xintlog}%
31     {Loading should be via \ifx\x\empty\string\usepackage{xintexpr.sty}
32       \else\string\input\space xintexpr.sty \fi
33     rather, aborting}%
34   \aftergroup\endinput
35 \else

```

```

36     \ifx\x\relax % first loading (initiated from xintexpr.sty)
37     \else
38     \ifx\x\empty % LaTeX first loading, \ProvidesPackage not yet seen
39     \else
40     \y{xintlog}{Already loaded, aborting}%
41     \aftergroup\endinput
42     \fi
43     \fi
44     \fi
45     \fi
46 \z%
```

Attention to catcode regime when loading below `poormanlog`. It (v0.04) uses `^` with its normal catcode but `\XINT_setcatcodes` would set it to letter.

This file can only be loaded from `xintexpr.sty` and it restores catcodes near its end. To play it safe and be hopefully immune to whatever is done in `poormanlog` or in `xinttrig.sty` which is loaded before, we will switch to standard catcode regime here.

As I learned the hard way (I never use my user macros), at the worst moment when wrapping up the final things for 1.3e release, `\xintexprSafeCatcodes` MUST be followed by some `\xintexprRestoreCatcodes` quickly, else next time it is used (for example by `\xintdefvar`) the `\xintexprRestoreCatcodes` will restore an obsolete catcode regime...

## 13.2 Library identification

```

47 \xintexprSafeCatcodes\catcode`_ 11
48 \XINT_providespackage
49 \ProvidesPackage{xintlog}%
50 [2019/04/05 1.3e Logarithms and exponentials for xintexpr (JFB)]%
```

## 13.3 Loading of `poormanlog` package

Attention to catcode regime when loading `poormanlog`.

```

51 \ifdefined\RequirePackage
52 \RequirePackage{poormanlog}%
53 \else
54 \input poormanlog.tex
55 \fi
```

`\XINT_setcatcodes` switches to the standard catcode regime of `xint*.sty` files. And we need the `xintexpr` catcode for ! too.

See remark above about `\xintexprRestoreCatcodes`.

```

56 \xintexprRestoreCatcodes\csname XINT_setcatcodes\endcsname\catcode`\! 11
```

## 13.4 Support macros for natural logarithm and exponential

```

57 \def\xintLog#1{\xintMul{\PoorManLogBaseTen{#1}}{23025850923[-10]}}%
58 \def\XINTinFloatLog#1{\XINTinFloatMul{\PoorManLogBaseTen{#1}}{23025850923[-10]}}%
59 \def\xintExp#1{\PoorManPowerOfTen{\xintMul{#1}{434294481903[-12]}}}%
60 \def\XINTinFloatExp#1{\PoorManPowerOfTen{\XINTinFloatMul{#1}{434294481903[-12]}}}%

```

## 13.5 The `log()`, `exp()`, and `pow()` function

The `log10()` and `pow10()` are already defined by `poormanlog`.

```

61 \def\XINT_expr_func_log #1#2#3%
62 {%
63   \expandafter #1\expandafter #2\csname.=%
64   \XINT:NEhook:one\xintLog{\XINT_expr_unlock #3}\endcsname
65 }%
66 \def\XINT_flexpr_func_log #1#2#3%
67 {%
68   \expandafter #1\expandafter #2\csname.=%
69   \XINT:NEhook:one\XINTinFloatLog{\XINT_expr_unlock #3}\endcsname
70 }%
71 \def\XINT_expr_func_exp #1#2#3%
72 {%
73   \expandafter #1\expandafter #2\csname.=%
74   \XINT:NEhook:one\xintExp{\XINT_expr_unlock #3}\endcsname
75 }%
76 \def\XINT_flexpr_func_exp #1#2#3%
77 {%
78   \expandafter #1\expandafter #2\csname.=%
79   \XINT:NEhook:one\XINTinFloatExp{\XINT_expr_unlock #3}\endcsname
80 }%
81 \def\XINT_expr_func_pow #1#2#3%
82 {%
83   \expandafter #1\expandafter #2\csname.=%
84   \expandafter\XINT:NEhook:twosp
85   \romannumeral`&&\XINT_expr_unlock #3,!\PoorManPower
86   \endcsname
87 }%
88 \let\XINT_flexpr_func_pow\XINT_expr_func_pow

```

We don't worry about setting catcodes as this file is theoretically only loadable from `xintexpr.sty` itself.



## 14 Cumulative line count

`xintkernel`: 557. Total number of code lines: 15482. (but 3507 lines among them  
`xinttools`:1454. start either with `{%` or with `}%`.)  
`xintcore`:2182. Each package starts with circa 50 lines dealing with cat-  
`xint`:1562. codes, package identification and reloading management,  
`xintbinhex`: 472. also for Plain TeX. Version 1.3e of 2019/04/05.  
`xintgcd`: 434.  
`xintfrac`:3295.  
`xintseries`: 386.  
`xintcfrac`:1029.  
`xintexpr`:3553.  
`xinttrig`: 470.  
`xintlog`: 88.