

# The `xint` source code

JEAN-FRANÇOIS BURNOL

jfbu (at) free (dot) fr

Package version: 1.3d (2019/01/06); documentation date: 2019/01/06.

From source file xint.dtx. Time-stamp: <06-01-2019 at 18:00:05 CET>.

## 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>5</b>
<b>3</b>	<b>Package <code>xinttools</code> implementation</b>	<b>19</b>
<b>4</b>	<b>Package <code>xintcore</code> implementation</b>	<b>55</b>
<b>5</b>	<b>Package <code>xint</code> implementation</b>	<b>113</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>261</b>
<b>10</b>	<b>Package <code>xintcfrc</code> implementation</b>	<b>270</b>
<b>11</b>	<b>Package <code>xintexpr</code> implementation</b>	<b>293</b>
<b>12</b>	<b>Per package indices of control sequences</b>	<b>381</b>
<b>13</b>	<b>Cumulative index</b>	<b>429</b>
<b>14</b>	<b>Cumulative line count</b>	<b>464</b>

## 1 Introduction to the implementation (recent changes)

This is 1.3d of 2019/01/06.

- Release 1.3d of 2019/01/06:
  - `gcd()` and `lcm()` in `\xintexpr...\relax` now handle general arguments, without converting them to integers,
  - it is not needed anymore to load package `xintgcd` to benefit from `gcd()` and `lcm()` in the parsers,
  - `\xintunassignvar` slightly refactored,
  - `\xintifsgnexpr`, `\xintifsgnfloatexpr`, `\xintifsgniexpr`,
  - `\xintunassignexprfunc` and variants for the other parsers,
  - `isone()` and `isint()`,
  - `\xinteval`, `\xintieval`, `\xintiieval`, and `\xintfloateval` (attention that these macros previously existed with different meanings, now available under new names `\xintexprpro`, `\xintiexprpro`, `\xintiievalpro`, and `\xintfloatexprpro`),
  - sadly, in `\xintiexpr...\relax` division with a zero dividend and a one-digit divisor got broken at 1.2p. Fixed. Thanks to KPYM for report. Sorry for long delay in releasing the bugfix, which was done shortly after 1.3c release.
- Release 1.3c of 2018/06/17:
  - `\xintglobaldeftrue` to give global scope to definitions done by `\xintdefvar`, `\xintdeffunc`, `\xintNewExpr`, et al.
  - `qraw()` to allow bypassing the normal mode of functioning and reduce memory footprint from parsing (very many) comma separated numerical items.
  - the colon in the `:=` part of the syntax for `\xintdefvar` and variants is now optional; and if present it may be an active character or have any (reasonable) catcode.
  - `\xintdefvar`, `\xintdeffunc` and their variants try to set the catcode of the semi-colon which delimits their arguments; of course this will not work if that catcode is already frozen.
  - `\xintexprSafeCatcodes` sets a toggle to allow some nested usage.
  - `\xintUniformDeviate` is better documented and [sourcexint.pdf](#) is better hyperlinked and includes indices for the macros defined by each package.
  - since 1.3 release, `xintfrac` (hence `xintexpr`) loaded `xintgcd` in contradiction to what documentation says. Removed for now.
- Release 1.3b of 2018/05/18:
  - `\xintUniformDeviate`, `random()`, `grand()`, `randrange()`,
  - and their support macros `\XINTinRandomFloatSdigits`, `\XINTinRandomFloatSixteen`, `\xintRandomDigits`, `\xintiiRandRange`, `\xintiiRandRangeAtoB`, also `\xintXRandomDigits`,
  - functions defined via `\xintdeffunc` (et al.) to be without argument can be used as `foo():foo(nil)` syntax not anymore required.
- Release 1.3a of 2018/03/07:
  - removes from `xintcore`, `xint` and `xintfrac` the whole deprecation mechanism, as there are no more currently any deprecated macro,
  - adds `ifone()` and `ifint()` conditionals to the expression parsers,
  - has a completely redone `\XINT_factortens`, in the style of 1.2 release (but about 100 digits at least are needed for noticeable speed gain),
  - and last but not least fixes via addition of `\xintExpandArgs` the meaning of user defined functions, which in case of recursivity (as made possible at 1.3) were badly inefficient for lack of expansion of their arguments.
- Release 1.3 of 2018/03/01:

- removed all macros previously deprecated at 1.2o,
  - modified addition, subtraction and modulo operations to use a least common multiple for the denominator of the result,
  - added `\xintPIrr`, `\xintDecToString` and `preduce()`,
  - and last but not least refactored extensively the `\xintNewExpr`/`\xintdeffunc` mechanism. It got both leaner and stronger and makes possible recursive function definitions.
- Release 1.2q of 2018/02/06 was a bugfix release with these changes:
    - fix to an 1.2l `xintcore` subtraction bug causing a breakage under some rare circumstances `:2-(`. It was caused by a refactoring left-over (extra `!` in `XINT_sub_1_Ida` replacement text), and should have been detected by the test suite, but manual testing from early days was not yet entirely included in our automated procedure.
    - new feature: tacit multiplication in front of digits, for example `10!20!30!` or `(10+10)10`.
  - Release 1.2p of 2017/12/05 had some breaking changes:
    - new output for the `\xintBezout` macro (`xintgcd`),
    - the `xintexpr` operators `:` (aka 'mod') and `//`, and the supporting macros from `xintcore` and `xintfrac`, are now associated with the *floored* division. Formerly it was the *truncated* division. This is breaking change for operands of opposite signs.

Improvements and new features:

- `xinttools` macro `\xintListWithSep` is faster (first update since 1.04-2013/04/25...).
  - `divmod()` function added to the `xintexpr` parsers,
  - `\xintdefvar`, `\xintdeffloatvar`, `\xintdefiivar` extended to allow multiple simultaneous assignments.
- Release 1.2o of 2017/08/29 deprecated those macros from `xintcore` and `xint` which filtered their arguments via `\xintNum`. Currently these macros execute as formerly but raise an error message. This deprecation is overruled for most if `xintfrac` is loaded as it provides their proper definitions. Some however (like `\xintiAdd`) remain deprecated even if loading `xintfrac`. All these deprecated macros are destined to be removed at some future release.
- A few macros got renamed (e.g. `\xintNot` became `\xintNOT`.) Former names emit a deprecation error and will get removed at some future release.
- Release 1.2n of 2017/08/06 removed the `xintbinhex` dependencies upon `xintcore`; the package now depends upon, and loads, only `xintkernel`. The allowed maximal size for the inputs of the base conversion macros got increased. The speed got slightly improved.
  - Release 1.2m of 2017/07/31 rewrote entirely the `xintbinhex` module in the style of the techniques from 1.2 `xintcore`. The new macros expand faster but their inputs are now limited to a few thousand characters (the earlier routines, which dated back to 1.08 could handle (slowly) tens of thousands of digits.)
  - Release 1.2l of 2017/07/26 refactored the subtraction and also `\xintiiCmp` got a rewrite. It should certainly use `\pdfstrcmp` for dramatic speed-up but I do not want to have to worry about multi-engine usage.

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

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

- Release 1.2i of 2016/12/13 rewrote some legacy macros like `\xintDSR` or `\xintDecSplit` in the style of the techniques of 1.2. But this means also that they got limited to about 22480 digits for the former and 19970 digits for the latter (this is with the input stack size at 5000 and the maximal expansion depth at 10000.) This is not really an issue from the point of view of calling macros (such as `\xintTrunc`, `\xintRound`), because they usually had since 1.2 their own limitation at about 19950 digits from other code parts (such as division.) The macro `\xintXTrunc` (which is not f-expandable however) can produce tens of thousands of digits and it escapes these limitations. Old macros such as `\xintLength` are not limited either (incidentally it got a lifting in 1.2i.) The macros from `xinttools` (`\xintKeep`, `\xintTrim`, `\xintNthElt`) also are not limited (but slower.)
- Release 1.2 of 2015/10/10 entirely rewrote the core arithmetic routines located in `xintcore`. The parser of `xintexpr` got faster and the limitation at 5000 digits per number was removed.
- Extensive changes in release 1.1 of 2014/10/28 were located in `xintexpr`. Also with that release, packages `xintkernel` and `xintcore` were extracted from `xinttools` and `xint`, and `\xintAdd` was modified to not multiply denominators blindly.

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

Warning: pay attention when looking at the code to the catcode configuration as found in `\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 . . .	5	.10	<code>\xint_zapspaces</code> . . . . .	11
.1.1	<code>\XINT_setcatcodes</code> , <code>\XINT_storecatcodes</code> , <code>\XINT_restorecatcodes_endinput</code> . . . . .	6	.11	<code>\odef</code> , <code>\oodef</code> , <code>\fdef</code> . . . . .	11
.2	Package identification . . . . .	8	.12	<code>\xintReverseOrder</code> . . . . .	11
.3	Constants . . . . .	8	.13	<code>\xintLength</code> . . . . .	12
.4	(WIP) <code>\xint_texuniformdeviate</code> and needed counts . . . . .	8	.14	<code>\xintLastItem</code> . . . . .	12
.5	Token management utilities . . . . .	9	.15	<code>\xintLengthUpTo</code> . . . . .	13
.6	"gob til" macros and UD style fork . . . . .	9	.16	<code>\xintreplicate</code> . . . . .	14
.7	<code>\xint_afterfi</code> . . . . .	10	.17	<code>\xintgobble</code> . . . . .	15
.8	<code>\xint_bye</code> , <code>\xint_Bye</code> . . . . .	10	.18	(WIP) <code>\xintUniformDeviate</code> . . . . .	17
.9	<code>\xintdothis</code> , <code>\xintorthat</code> . . . . .	10	.19	<code>\xintMessage</code> , <code>\ifxintverbose</code> . . . . .	18
			.20	<code>\ifxintglobaldefs</code> , <code>\XINT_global</code> . . . . .	18
			.21	(WIP) Expandable error message . . . . .	18

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. `\xintUniformDeviate`.

### 2.1 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 \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

```

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

```

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 }%
```

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

## 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 H0 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 [2019/01/06 1.3d 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_rord_main #1#2#3#4#5#6#7#8#9%
262 {%
263     \xint_bye #9\XINT_rord_cleanup\xint_bye
264     \XINT_rord_main {#9#8#7#6#5#4#3#2#1}%
265 }%
266 \def\XINT_rord_cleanup #1{%
267 \long\def\XINT_rord_cleanup\xint_bye\XINT_rord_main ##1##2\xint:
268 {%
269     \expandafter#1\xint_gob_til_xint: ##1%
270 }}\XINT_rord_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

```

345 }%

## 2.16 \xintreplicate

**1.2i.**

This is cloned from LaTeX3's `\prg_replicate:nm`, see Joseph's post at

<http://tex.stackexchange.com/questions/16189/repeat-command-n-times>

I posted there an alternative not using the chained `\csname`'s but it is a bit less efficient (except perhaps for thousands of repetitions). The code in Joseph's post does `abs(#1)` replications when input `#1` is negative and then activates an error triggering macro; here we simply do nothing when `#1` is negative.

Usage: `\romannumeral\xintreplicate{N}{stuff}`

When **N** is already explicit digits (even **N=0**, but non-negative) one can call the macro as

```
\romannumeral\XINT rep N\endcsname {foo}
```

to skip the `\numexpr`.

```

346 \def\xintreplicate#1%
347   {\expandafter\XINT_replicate\the\numexpr#1\endcsname}%
348 \def\XINT_replicate #1{\xint_UDsignfork
349     #1\XINT_rep_neg
350     -\XINT_rep
351     \krof #1}%
352 \long\def\XINT_rep_neg #1\endcsname #2{\xint_c_}%
353 \def\XINT_rep #1{\csname XINT_rep_f#1\XINT_rep_a}%
354 \def\XINT_rep_a #1{\csname XINT_rep_#1\XINT_rep_a}%
355 \def\XINT_rep_\XINT_rep_a{\endcsname}%
356 \long\expandafter\def\csname XINT_rep_0\endcsname #1%
357   {\endcsname{#1#1#1#1#1#1#1#1#1#1}}%
358 \long\expandafter\def\csname XINT_rep_1\endcsname #1%
359   {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1}%
360 \long\expandafter\def\csname XINT_rep_2\endcsname #1%
361   {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1}%
362 \long\expandafter\def\csname XINT_rep_3\endcsname #1%
363   {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1}%
364 \long\expandafter\def\csname XINT_rep_4\endcsname #1%
365   {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1}%
366 \long\expandafter\def\csname XINT_rep_5\endcsname #1%
367   {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1#1}%
368 \long\expandafter\def\csname XINT_rep_6\endcsname #1%
369   {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1#1#1}%
370 \long\expandafter\def\csname XINT_rep_7\endcsname #1%
371   {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1#1#1#1}%
372 \long\expandafter\def\csname XINT_rep_8\endcsname #1%
373   {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1#1#1#1#1}%
374 \long\expandafter\def\csname XINT_rep_9\endcsname #1%
375   {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1#1#1#1#1#1}%
376 \long\expandafter\def\csname XINT_rep_f0\endcsname #1%
377   {\xint_c_}%
378 \long\expandafter\def\csname XINT_rep_f1\endcsname #1%
379   {\xint_c_ #1}%
380 \long\expandafter\def\csname XINT_rep_f2\endcsname #1%
381   {\xint_c_ #1#1}%
382 \long\expandafter\def\csname XINT_rep_f3\endcsname #1%

```

```

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%

```

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

```
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.

```
539 \def\xintMessage #1#2#3{%
540     \immediate\write128{Package #1 #2: (on line \the\inputlineno)}}%
541     \immediate\write128{\space\space\space\space#3}%
542 }%
543 \newif\ifxintverbose
```

## 2.20 \ifxintglobaldefs, \XINT\_global

1.3c.

```
544 \newif\ifxintglobaldefs
545 \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.

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

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

.1	Catcodes, $\varepsilon$ -TeX and reload detection . . .	19			
.2	Package identification . . . . .	20	.21	<code>\XINT_xflet</code> . . . . .	36
.3	<code>\xintgodef</code> , <code>\xintgoodef</code> , <code>\xintgdef</code> . . .	20	.22	<code>\xintApplyInline</code> . . . . .	37
.4	<code>\xintRevWithBraces</code> . . . . .	20	.23	<code>\xintFor</code> , <code>\xintFor*</code> , <code>\xintBreakFor</code> , <code>\xintBreakForAndDo</code> . . . . .	37
.5	<code>\xintZapFirstSpaces</code> . . . . .	21	.24	<code>\XINT_forever</code> , <code>\xintintegers</code> , <code>\xintdi-</code> <code>mensions</code> , <code>\xintrationals</code> . . . . .	40
.6	<code>\xintZapLastSpaces</code> . . . . .	22	.25	<code>\xintForpair</code> , <code>\xintForthree</code> , <code>\xintFor-</code> <code>four</code> . . . . .	42
.7	<code>\xintZapSpaces</code> . . . . .	23	.26	<code>\xintAssign</code> , <code>\xintAssignArray</code> , <code>\xint-</code> <code>DigitsOf</code> . . . . .	43
.8	<code>\xintZapSpacesB</code> . . . . .	23	.27	<code>\xintExpandArgs</code> . . . . .	46
.9	<code>\xintCSVtoList</code> , <code>\xintCSVtoListNon-</code> <code>Stripped</code> . . . . .	23	.28	CSV (non user documented) variants of Length, Keep, Trim, NthElt, Reverse . . .	46
.10	<code>\xintListWithSep</code> . . . . .	25	.28.1	<code>\xintLength:f:csv</code> . . . . .	47
.11	<code>\xintNthElt</code> . . . . .	26	.28.2	<code>\xintLengthUpTo:f:csv</code> . . . . .	47
.12	<code>\xintKeep</code> . . . . .	27	.28.3	<code>\xintKeep:f:csv</code> . . . . .	48
.13	<code>\xintKeepUnbraced</code> . . . . .	28	.28.4	<code>\xintTrim:f:csv</code> . . . . .	50
.14	<code>\xintTrim</code> . . . . .	29	.28.5	<code>\xintNthEltPy:f:csv</code> . . . . .	52
.15	<code>\xintTrimUnbraced</code> . . . . .	31	.28.6	<code>\xintReverse:f:csv</code> . . . . .	53
.16	<code>\xintApply</code> . . . . .	32	.28.7	<code>\xintFirstItem:f:csv</code> . . . . .	53
.17	<code>\xintApplyUnbraced</code> . . . . .	32	.28.8	<code>\xintLastItem:f:csv</code> . . . . .	53
.18	<code>\xintSeq</code> . . . . .	33	.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> . . . . .	54
.19	<code>\xintloop</code> , <code>\xintbreakloop</code> , <code>\xintbreak-</code> <code>loopanddo</code> , <code>\xintloopskiptonext</code> . . .	35			
.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.}}%
```

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

```

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 [2019/01/06 1.3d 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

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

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{ }%
```

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

### 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_again\XINT_zapsp_b} {\XINT_zapsp_a }%
124 \xint_firstofone{\def\XINT_zapsp_b} {\XINT_zapsp_c }%
125 \def\XINT_zapsp_c#1{%
126 \long\def\XINT_zapsp_c ##1\xint:##2\xint:%
127     {\XINT_zapsp_a{ }\empty ##1#1#1\xint_bye\xint:}%
128 }\XINT_zapsp_c{ }%
```

### 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.

```
141 \def\xintCSVtoList {\romannumeral0\xintcsvtolist }%
142 \long\def\xintcsvtolist #1{\expandafter\xintApply
143     \expandafter\xintzapspaceb
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_endb`).

```

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

```



```

193 \long\def\XINT_csvtol_finish_dii    #1#2#3#4#5#6#7#8#9%
194                                     { #9{#1}{#2}{#3}{#4}{#5}{#6}}%
195 \long\def\XINT_csvtol_finish_di\Z  #1#2#3#4#5#6#7#8#9%
196                                     { #9{#1}{#2}{#3}{#4}{#5}{#6}{#7}}%

```

### 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}%

```

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

```

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_keeptime
294     0#1\XINT_keep_neg
295     0-{\XINT_keep_pos #1}%
296     \krof
297 }%
298 \long\def\XINT_keep_keeptime .#1{ }%
299 \long\def\XINT_keep_neg #1.#2%
300 {%
301     \expandafter\XINT_keep_neg_a\the\numexpr
302     #1-\numexpr\XINT_length_loop
303     #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
304     \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
305     \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye.#2%
306 }%
307 \def\XINT_keep_neg_a #1%
308 {%
309     \xint_UDsignfork

```

```

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

```

### 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_keepnone
362     0#1\XINT_keep_neg
363     0-{\XINT_keepunbr_pos #1}%
364     \krof
365 }%
366 \long\def\XINT_keepunbr_pos #1.#2%
367 {%
368     \expandafter\XINT_keepunbr_loop
369     \the\numexpr#1-\XINT_lengthupto_loop
370     #1.#2\xint:\xint:\xint:\xint:\xint:\xint:\xint:
371     \xint_c_vii\xint_c_vi\xint_c_v\xint_c_iv
372     \xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye.%
373     -\xint_c_viii.{#2\xint_bye%
374 }%
375 \def\XINT_keepunbr_loop #1#2.%
376 {%
377     \xint_gob_til_minus#1\XINT_keepunbr_loop_end-%
378     \expandafter\XINT_keepunbr_loop
379     \the\numexpr#1#2-\xint_c_viii\expandafter.\XINT_keepunbr_loop_pickeight
380 }%
381 \long\def\XINT_keepunbr_loop_pickeight
382     #1#2#3#4#5#6#7#8#9{{#1#2#3#4#5#6#7#8#9}}%
383 \def\XINT_keepunbr_loop_end-\expandafter\XINT_keepunbr_loop
384     \the\numexpr-#1-\xint_c_viii\expandafter.\XINT_keepunbr_loop_pickeight
385     {\csname XINT_keepunbr_end#1\endcsname}%
386 \long\expandafter\def\csname XINT_keepunbr_end1\endcsname
387     #1#2#3#4#5#6#7#8#9\xint_bye { #1#2#3#4#5#6#7#8}%
388 \long\expandafter\def\csname XINT_keepunbr_end2\endcsname
389     #1#2#3#4#5#6#7#8\xint_bye { #1#2#3#4#5#6#7}%
390 \long\expandafter\def\csname XINT_keepunbr_end3\endcsname
391     #1#2#3#4#5#6#7\xint_bye { #1#2#3#4#5#6}%
392 \long\expandafter\def\csname XINT_keepunbr_end4\endcsname
393     #1#2#3#4#5#6\xint_bye { #1#2#3#4#5}%
394 \long\expandafter\def\csname XINT_keepunbr_end5\endcsname
395     #1#2#3#4#5\xint_bye { #1#2#3#4}%
396 \long\expandafter\def\csname XINT_keepunbr_end6\endcsname
397     #1#2#3#4\xint_bye { #1#2#3}%
398 \long\expandafter\def\csname XINT_keepunbr_end7\endcsname
399     #1#2#3\xint_bye { #1#2}%
400 \long\expandafter\def\csname XINT_keepunbr_end8\endcsname
401     #1#2\xint_bye { #1}%

```

### 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:
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

```

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrc](#), [xintexpr](#), [indices](#)

```
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{z}}` 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_sequo_nb
626      \fi
627      {#1}{#2}{#3}{#1}}%
628 }%
629 \def\XINT_sequo_nb #1#2#3%
630 {%
631      \expandafter\XINT_sequo_nc\expandafter{\the\numexpr #1+#3}{#2}{#3}%
632 }%
633 \def\XINT_sequo_nc #1#2%
634 {%
635      \ifnum #1<#2
636          \expandafter\XINT_sequo_o
637      \else
638          \expandafter\XINT_sequo_nd
639      \fi
640      {#1}{#2}%
641 }%
642 \def\XINT_sequo_nd #1#2#3#4{\XINT_sequo_nb {#1}{#2}{#3}{#4{#1}}}%

```

### 3.19 \xintloop, \xintbreakloop, \xintbreakloopanddo, \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\xintbreakloopanddo #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 \xintilloop, \xintilloopindex, \xintbracedilloopindex, \xintouterilloopindex, \xintbracedouterilloopindex, \xintbreakilloop, \xintbreakilloopanddo, \xintilloopskiptonext, \xintilloopskipandredo

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

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

```

650 \def\xintilloop [#1+#2]{%
651      \expandafter\xintilloop_a\the\numexpr #1\expandafter.\the\numexpr #2.%
652 \long\def\xintilloop_a #1.#2.#3#4\repeat{%
653      #3#4\xintilloop_again\fi\xint_gobble_iii {#1}{#2}{#3#4}}%
654 \def\xintilloop_again\fi\xint_gobble_iii #1#2{%
655      \fi\expandafter\xintilloop_again_b\the\numexpr#1+#2.#2.%
656 \long\def\xintilloop_again_b #1.#2.#3{%
657      #3\xintilloop_again\fi\xint_gobble_iii {#1}{#2}{#3}}%
658 \long\def\xintbreakilloop #1\xintilloop_again\fi\xint_gobble_iii #2#3#4{%
659 \long\def\xintbreakilloopanddo
660      #1.#2\xintilloop_again\fi\xint_gobble_iii #3#4#5{#1}%
661 \long\def\xintilloopindex #1\xintilloop_again\fi\xint_gobble_iii #2%
662      {#2#1\xintilloop_again\fi\xint_gobble_iii {#2}}%
663 \long\def\xintbracedilloopindex #1\xintilloop_again\fi\xint_gobble_iii #2%

```

```

664      {{#2}#1\xintilooop_again\fi\xint_gobble_iii {#2}}%
665 \long\def\xintouterilooopindex #1\xintilooop_again
666      #2\xintilooop_again\fi\xint_gobble_iii #3%
667      {#3#1\xintilooop_again #2\xintilooop_again\fi\xint_gobble_iii {#3}}%
668 \long\def\xintbracedouterilooopindex #1\xintilooop_again
669      #2\xintilooop_again\fi\xint_gobble_iii #3%
670      {{#3}#1\xintilooop_again #2\xintilooop_again\fi\xint_gobble_iii {#3}}%
671 \long\def\xintilooopskiptonext #1\xintilooop_again\fi\xint_gobble_iii #2#3{%
672      \expandafter\xintilooop_again_b \the\numexpr#2+#3.#3.}%
673 \long\def\xintilooopskipandredo #1\xintilooop_again\fi\xint_gobble_iii #2#3#4{%
674      #4\xintilooop_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 to-ken which serves two purposes: preventing brace stripping, and stopping the expansion made by `\xintcsvtolist`. If the `\XINT_forever` branch is taken, the added space will not be a problem there.

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

```

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_forever\fi
799   \ifx\XINT_token V\XINT_to_forever\fi
800   \ifx\XINT_token D\XINT_to_forever\fi
801   \XINT_forx_empty?
802 }%
803 \def\XINT_to_forever\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\the\dimexpr #1\expandafter\relax
865     \expandafter s\expandafter p\expandafter\relax\expandafter}%
866   \expandafter{\the\dimexpr #2}}%
867 \catcode`Z 11
868 \def\XINT_?expr_Va #1#2%
869 {%
870   \expandafter\XINT_?expr_Vb\expandafter
871   {\romannumeral`&&\xintZapSpacesB{#2}}%
872   {\romannumeral`&&\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\xinttiadd {#1}{#2}}{#2}%
901 }%
902 \def\xINT_?expr_Vy #1#2#3#4%
903 {%
904   \expandafter{\romannumeral0\xinttiadd {#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_seconddoftwo
988 \let\xintifForLast\xint_seconddoftwo
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

```

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrc](#), [xintexpr](#), [indices](#)

### 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.

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

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_keeptime

```



```

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 . . .	55	.25	\XINT_zeroes_forviii . . . . .	68
.2	Package identification . . . . .	56	.26	\XINT_sepbyviii_Z . . . . .	68
.3	(WIP!) Error conditions and exceptions . . .	56	.27	\XINT_sepbyviii_andcount . . . . .	69
.4	Counts for holding needed constants . . .	58	.28	\XINT_rsepyviii . . . . .	69
Routines handling integers as lists of token digits			.29	\XINT_sepandrev . . . . .	70
.5	\XINT_cuz_small . . . . .	59	.30	\XINT_sepandrev_andcount . . . . .	70
.6	\xintNum, \xintiNum . . . . .	59	.31	\XINT_rev_nounsep . . . . .	71
.7	\xintiiSgn . . . . .	60	.32	\XINT_unrevbyviii . . . . .	71
.8	\xintiiOpp . . . . .	61	Core arithmetic . . . . .		71
.9	\xintiiAbs . . . . .	61	.33	\xintiiAdd . . . . .	72
.10	\xintFDg . . . . .	61	.34	\xintiiCmp . . . . .	75
.11	\xintLDg . . . . .	62	.35	\xintiiSub . . . . .	77
.12	\xintDouble . . . . .	62	.36	\xintiiMul . . . . .	83
.13	\xintHalf . . . . .	63	.37	\xintiiDivision . . . . .	87
.14	\xintInc . . . . .	63	Derived arithmetic . . . . .		102
.15	\xintDec . . . . .	64	.38	\xintiiQuo, \xintiiRem . . . . .	102
.16	\xintDSL . . . . .	64	.39	\xintiiDivRound . . . . .	102
.17	\xintDSR . . . . .	64	.40	\xintiiDivTrunc . . . . .	103
.18	\xintDSRr . . . . .	65	.41	\xintiiModTrunc . . . . .	103
Blocks of eight digits . . . . .			.42	\xintiiDivMod . . . . .	104
.19	\XINT_cuz . . . . .	65	.43	\xintiiDivFloor . . . . .	105
.20	\XINT_cuz_byviii . . . . .	66	.44	\xintiiMod . . . . .	105
.21	\XINT_unsep_loop . . . . .	66	.45	\xintiiSqr . . . . .	105
.22	\XINT_unsep_cuzsmall . . . . .	67	.46	\xintiiPow . . . . .	106
.23	\XINT_div_unsepQ . . . . .	67	.47	\xintiiFac . . . . .	109
.24	\XINT_div_unsepR . . . . .	68	.48	\XINT_useiimessage . . . . .	112

Got split off from [xint](#) with release 1.1.

The core arithmetic routines have been entirely rewritten for release 1.2. The 1.2i and 1.21 brought again some improvements.

The commenting continues (2019/01/06) to be very sparse: actually it got worse than ever with release 1.2. I will possibly add comments at a later date, but for the time being the new routines are not commented at all.

1.3 removes all macros which were deprecated at 1.20.

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

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

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

```

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

```

12 \let\z\endgroup
13 \expandafter\let\expandafter\x\csname ver@xintcore.sty\endcsname
14 \expandafter\let\expandafter\w\csname ver@xintkernel.sty\endcsname
15 \expandafter
16   \ifx\csname PackageInfo\endcsname\relax
17     \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
18   \else
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 [2019/01/06 1.3d Expandable arithmetic on big integers (JFB)]%

```

## 4.3 (WIP!) Error conditions and exceptions

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

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

and the Python3 implementation in its Decimal module.

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

X3.274 rajoute LostDigits

Python rajoute FloatOperation (et n'inclut pas InsufficientStorage)

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



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     NOT IMPLEMENTED! WORK IN PROGRESS! (ALL SIGNALS TRAPPED, NO HANDLERS USED)
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

```

*TOC, xintkernel, xinttools, [xintcore](#), xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrc, xintexpr, indices*

```

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:\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:\xint:\Z
144 }%
145 \def\XINT_num_loop #1#2#3#4#5#6#7#8#9%

```

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfraction, xintexpr, indices*

```

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

1.21 made \xintiisgn robust against non terminated input.

1.20 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 }%

```

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrc](#), [xintexpr](#), [indices](#)

## 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.

1.21: \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
```



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

```
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}%
```

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

## 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.

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrc, xintexpr, indices*

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 {%

```

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

```
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\R{10}0000001\W

produces a string of k 0's such that k+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 and 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

```

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices*

```
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 #1#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 }%
```

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices*

```

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!\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 1.2l).

```

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.)

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

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_rsepybviii_end_A 2345678%
570     \XINT_rsepybviii_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\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 ;%

```

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices*

```

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 1.2l.

1.2l'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!#2!#3!#4!#5\W #6!#7!#8!#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>!...1<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 `\xintiiexpr` only, but then I need to make it compatible with sub-`\xintiiexpr` 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 <= #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.

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

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<sup>8</sup>) 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_Id\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_Id_a;%
961     1\relax 00000000!1#2#3!#4;!1\XINT_sub_fix_none!%
962 }%
963 \def\XINT_sub_l_Id_a#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 {%

```

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices*

```

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_rsepbyviii_end_A 2345678%
1037   \XINT_rsepbyviii_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!\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 #3!;!\%
1075   {\fi\fi\XINT_mul_start #3!;!\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<8d>!`. Its is generally more efficient if #1 is the shorter one, but a better recipe is implemented in `\XINT_mul_checklengths`. One may call `\XINT_mul_loop` directly (but multiplication by zero will produce many 100000000! blocks on output).

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

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

```

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<8digits carry>.<4 high digits>.<4 low digits!<8digits>.`

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

Multiplication by zero will produce blocks of zeros.

```

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#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#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#1\xint:#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!1#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#1#2\xint:%
1139     {1#2\expandafter!\the\numexpr\XINT_verysmallmul #1\xint:}%
1140 \def\XINT_verysmallmul_bi\the\numexpr\xint_c_x^ix+#1#2#3\xint:%
1141     {1#3\expandafter!\the\numexpr\XINT_verysmallmul #1#2\xint:}%

```

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

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

```

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#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.

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

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

```

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 }%

```

TOC, xintkernel, xinttools, [xintcore](#), xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrc, xintexpr, indices

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 {%

```



*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices*

```

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_ii\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_dosmallldiv
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 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_dosmallldiv
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}}%

```

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices*

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  $\backslash\text{expandafter}\backslash\text{XINT\_sdiv\_out}\backslash\text{the}\backslash\text{numexpr}\backslash\text{XINT\_smalldivx\_a } x.1B!1\langle 8d \rangle! \dots 1\langle 8d \rangle!1$ ! 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_rsepybviii_end_A 2345678%
1285     \XINT_rsepybviii_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\xintcvi\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 }%
```

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

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 {%
```

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices*

```

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

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!#9;!}%
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 main-
tenir 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 }%

```

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices*

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

```
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!\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» -> nouveau 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#51#6!%
1529 {%
1530     \XINT_div_II_m {{#1}{#2}}{{#3}{#4}}{{#5}}{{#5}{#6}1#6!%
1531 }%
```



*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrc, xintexpr, indices*

*{LK{x'y}x},x,a,alpha.B{q}alpha'B -> 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>!\dots1<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\Z R\W` avec un  $R < 10^8$ , et un Q sous forme de blocs `1<8d>!` terminé par `1!` et nécessi-  
tant le nettoyage du premier bloc. Dans cette branche le B n'a pas été multiplié par une puissance  
de 10, il peut avoir moins de huit chiffres.

```

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<sup>8</sup>.

```

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 (ARRRRRRGGGGGHHH) 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\xintiirem }%
1751 \def\xintiiquo
1752   {\expandafter\xint_stop_atfirstoftwo\romannumeral0\xintiidivision }%
1753 \def\xintiirem
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).

1.2l: \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 de-
    faulted-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{\xintiioopp\XINT_iimodtrunc_pos {}}%
1840 -{\XINT_iimodtrunc_pos #1}%
1841 \krof
1842 }%
1843 \def\XINT_iimodtrunc_bneg #1%
1844 {%
1845 \xint_UDsignfork
1846 #1{\xintiioopp\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 `\xintiiepr` 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\{10}0000001\W
1900   #1\XINT_rsepybviii_end_A 2345678%
1901   \XINT_rsepybviii_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}%

```

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

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#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!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 `\xintiPow` use `\xintNum` on the exponent once `xintfrac.sty` is loaded; 2) also because I noticed it was not very well coded. And it did only a `\numexpr` on the exponent, contradicting the documentation related to the "i" convention in names.

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

```

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:\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<sup>30</sup>, it does bring roughly a 40% time gain though, and 30% gain for 2<sup>60</sup>. 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!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 \xintiFac

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, \xintiFac 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 }%

```

xintcore

 $\{ \%$ 

the `\xintiiFac` here which attempts to be smarter...

Note (2017, 1.21): I found out some code comment of `mi` style of `\xintiiBinomial`, but I left matters untouched.

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

2073 { %

```

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.#1.1000000001!1;!%
2148 }%
2149 \expandafter\def\csname XINT_fac_smallloop_-2\endcsname #1.%
2150 {%
2151      \XINT_fac_smallloop_loop 3.#1.1000000002!1;!%
2152 }%
2153 \expandafter\def\csname XINT_fac_smallloop_-1\endcsname #1.%
2154 {%
2155      \XINT_fac_smallloop_loop 4.#1.1000000006!1;!%
2156 }%
2157 \expandafter\def\csname XINT_fac_smallloop_0\endcsname #1.%
2158 {%
2159      \XINT_fac_smallloop_loop 5.#1.10000000024!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 . . . . .	114	.32	\xintiiifLt . . . . .	126
.2	More token management . . . . .	114	.33	\xintiiifZero . . . . .	126
.3	(WIP) A constant needed by \xintRandomDigits et al. . . . .	114	.34	\xintiiifNotZero . . . . .	126
.4	\xintLen, \xintiLen . . . . .	115	.35	\xintiiifOne . . . . .	126
.5	\xintReverseDigits . . . . .	115	.36	\xintiiifOdd . . . . .	127
.6	\xintiiE . . . . .	116	.37	\xintifTrueAelseB, \xintifFalseAelseB . . . . .	127
.7	\xintDecSplit . . . . .	116	.38	\xintIsTrue, \xintIsFalse . . . . .	127
.8	\xintDecSplitL . . . . .	118	.39	\xintNOT . . . . .	127
.9	\xintDecSplitR . . . . .	118	.40	\xintAND, \xintOR, \xintXOR . . . . .	127
.10	\xintDSHr . . . . .	119	.41	\xintANDof . . . . .	128
.11	\xintDSH . . . . .	119	.42	\xintORof . . . . .	128
.12	\xintDSx . . . . .	120	.43	\xintXORof . . . . .	128
.13	\xintiiEq . . . . .	121	.44	\xintiiMax . . . . .	129
.14	\xintiiNotEq . . . . .	121	.45	\xintiiMin . . . . .	130
.15	\xintiiGeq . . . . .	122	.46	\xintiiMaxof . . . . .	131
.16	\xintiiGt . . . . .	122	.47	\xintiiMinof . . . . .	131
.17	\xintiiLt . . . . .	122	.48	\xintiiSum . . . . .	131
.18	\xintiiGtorEq . . . . .	122	.49	\xintiiPrd . . . . .	132
.19	\xintiiLtorEq . . . . .	123	.50	\xintiiSquareRoot . . . . .	132
.20	\xintiiIsZero . . . . .	123	.51	\xintiiSqrt, \xintiiSqrtR . . . . .	139
.21	\xintiiIsNotZero . . . . .	123	.52	\xintiiBinomial . . . . .	139
.22	\xintiiIsOne . . . . .	123	.53	\xintiiPfactorial . . . . .	145
.23	\xintiiOdd . . . . .	123	.54	\xintBool, \xintToggle . . . . .	148
.24	\xintiiEven . . . . .	124	.55	\xintGCD, \xintiiGCD . . . . .	148
.25	\xintiiMON . . . . .	124	.56	\xintLCM, \xintiiLCM . . . . .	149
.26	\xintiiMMON . . . . .	124	.57	(WIP) \xintRandomDigits . . . . .	150
.27	\xintSgnFork . . . . .	124	.58	(WIP) \XINT_eightrandomdigits . . . . .	150
.28	\xintiiifSgn . . . . .	125	.59	(WIP) \xintXRandomDigits . . . . .	150
.29	\xintiiifCmp . . . . .	125	.60	(WIP) \xintiiRandRangeAtoB . . . . .	151
.30	\xintiiifEq . . . . .	125	.61	(WIP) \xintiiRandRange . . . . .	151
.31	\xintiiifGt . . . . .	125	.62	Adjustments for engines without uniformdeviate primitive . . . . .	153

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

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 [2019/01/06 1.3d 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 len(A)+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:
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 \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.

```

74 \def\xintReverseDigits {\romannumeral0\xintreversedigits }%
75 \def\xintreversedigits #1%
76 {%
77   \expandafter\XINT_revdigits\romannumeral`&&@#1%
78   {\XINT_microrevsep_end\W}\XINT_microrevsep_end
79   \XINT_microrevsep_end\XINT_microrevsep_end
80   \XINT_microrevsep_end\XINT_microrevsep_end
81   \XINT_microrevsep_end\XINT_microrevsep_end\XINT_microrevsep_end\Z
82   1\Z!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
83 }%
84 \def\XINT_revdigits #1%
85 {%
86   \xint_UDsignfork
87   #1{\expandafter-\romannumeral0\XINT_revdigits_a}%
88   -{\XINT_revdigits_a #1}%
89   \krof
90 }%
91 \def\XINT_revdigits_a
92 {%
```

```

93 \expandafter\XINT_revdigits_b\expandafter{\expandafter}%
94 \the\numexpr\XINT_microrevsep
95 }%
96 \def\XINT_microrevsep #1#2#3#4#5#6#7#8#9%
97 {%
98 1#9#8#7#6#5#4#3#2#1\expandafter!\the\numexpr\XINT_microrevsep
99 }%
100 \def\XINT_microrevsep_end #1\W #2\expandafter #3\Z{\relax#2!}%
101 \def\XINT_revdigits_b #11#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
102 {%
103 \xint_gob_til_R #9\XINT_revdigits_end\R
104 \XINT_revdigits_b {#9#8#7#6#5#4#3#2#1}%
105 }%
106 \def\XINT_revdigits_end#1{%
107 \def\XINT_revdigits_end\R\XINT_revdigits_b ##1##2\W
108 {\expandafter#1\xint_gob_til_Z ##1}%
109 }\XINT_revdigits_end{ }%
110 \let\xintRev\xintReverseDigits

```

## 5.6 \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.

```

111 \def\xintiiE {\romannumeral0\xintiiE }%
112 \def\xintiiE #1#2%
113 {\expandafter\XINT_iiE_fork\the\numexpr #2\expandafter.\romannumeral`&&@#1;}%
114 \def\XINT_iiE_fork #1%
115 {%
116 \xint_UDsignfork
117 #1\XINT_iiE_neg
118 -\XINT_iiE_a
119 \krof #1%
120 }%

```

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

```

121 \def\XINT_iiE_a #1.%
122 {\expandafter\XINT_dsx_append\romannumeral\XINT_rep #1\endcsname 0.}%
123 \def\XINT_iiE_neg #1.#2;{ #2}%

```

## 5.7 \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.

```

124 \def\xintDecSplit {\romannumeral0\xintdecsplit }%
125 \def\xintdecsplit #1#2%
126 {%
127   \expandafter\XINT_split_finish
128   \romannumeral0\expandafter\XINT_split_xfork
129   \the\numexpr #1\expandafter.\romannumeral`&&@#2%
130   \xint_bye2345678\xint_bye..%
131 }%
132 \def\XINT_split_finish #1.#2.{{#1}{#2}}%

133 \def\XINT_split_xfork #1%
134 {%
135   \xint_UDzerominusfork
136   #1-\XINT_split_zerosplit
137   0#1\XINT_split_fromleft
138   0-\XINT_split_fromright #1}%
139   \krof
140 }%
141 \def\XINT_split_zerosplit .#1\xint_bye#2\xint_bye..{ #1..}%
142 \def\XINT_split_fromleft
143   {\expandafter\XINT_split_fromleft_a\the\numexpr\xint_c_viii-}%
144 \def\XINT_split_fromleft_a #1%
145 {%
146   \xint_UDsignfork
147   #1\XINT_split_fromleft_b
148   -{\XINT_split_fromleft_end_a #1}%
149   \krof
150 }%
151 \def\XINT_split_fromleft_b #1.#2#3#4#5#6#7#8#9%
152 {%
153   \expandafter\XINT_split_fromleft_clean
154   \the\numexpr1#2#3#4#5#6#7#8#9\expandafter
155   \XINT_split_fromleft_a\the\numexpr\xint_c_viii-#1.%
156 }%
157 \def\XINT_split_fromleft_end_a #1.%
158 {%
159   \expandafter\XINT_split_fromleft_clean
160   \the\numexpr1\csname XINT_split_fromleft_end#1\endcsname
161 }%
162 \def\XINT_split_fromleft_clean 1{ }%
163 \expandafter\def\csname XINT_split_fromleft_end7\endcsname #1%
164   {#1\XINT_split_fromleft_end_b}%
165 \expandafter\def\csname XINT_split_fromleft_end6\endcsname #1#2%
166   {#1#2\XINT_split_fromleft_end_b}%
167 \expandafter\def\csname XINT_split_fromleft_end5\endcsname #1#2#3%
168   {#1#2#3\XINT_split_fromleft_end_b}%
169 \expandafter\def\csname XINT_split_fromleft_end4\endcsname #1#2#3#4%

```

```

170    {#1#2#3#4\XINT_split_fromleft_end_b}%
171 \expandafter\def\csname XINT_split_fromleft_end3\endcsname #1#2#3#4#5%
172    {#1#2#3#4#5\XINT_split_fromleft_end_b}%
173 \expandafter\def\csname XINT_split_fromleft_end2\endcsname #1#2#3#4#5#6%
174    {#1#2#3#4#5#6\XINT_split_fromleft_end_b}%
175 \expandafter\def\csname XINT_split_fromleft_end1\endcsname #1#2#3#4#5#6#7%
176    {#1#2#3#4#5#6#7\XINT_split_fromleft_end_b}%
177 \expandafter\def\csname XINT_split_fromleft_end0\endcsname #1#2#3#4#5#6#7#8%
178    {#1#2#3#4#5#6#7#8\XINT_split_fromleft_end_b}%

179 \def\XINT_split_fromleft_end_b #1\xint_bye#2\xint_bye.{.#1}% puis .
180 \def\XINT_split_fromright #1.#2\xint_bye
181 {%
182     \expandafter\XINT_split_fromright_a
183     \the\numexpr#1-\numexpr\XINT_length_loop
184     #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
185     \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
186     \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
187     .#2\xint_bye
188 }%

189 \def\XINT_split_fromright_a #1%
190 {%
191     \xint_UDsignfork
192     #1\XINT_split_fromleft
193     -\XINT_split_fromright_Lempty
194     \krof
195 }%
196 \def\XINT_split_fromright_Lempty #1.#2\xint_bye#3..{.#2}%

```

## 5.8 \xintDecSplitL

```

197 \def\xintDecSplitL {\romannumeral0\xintdecsplitl }%
198 \def\xintdecsplitl #1#2%
199 {%
200     \expandafter\XINT_splitl_finish
201     \romannumeral0\expandafter\XINT_split_xfork
202     \the\numexpr #1\expandafter.\romannumeral`&&@#2%
203     \xint_bye2345678\xint_bye..%
204 }%
205 \def\XINT_splitl_finish #1.#2.{ #1}%

```

## 5.9 \xintDecSplitR

```

206 \def\xintDecSplitR {\romannumeral0\xintdecsplitr }%
207 \def\xintdecsplitr #1#2%
208 {%
209     \expandafter\XINT_splitr_finish
210     \romannumeral0\expandafter\XINT_split_xfork
211     \the\numexpr #1\expandafter.\romannumeral`&&@#2%
212     \xint_bye2345678\xint_bye..%
213 }%
214 \def\XINT_splitr_finish #1.#2.{ #2}%

```

## 5.10 \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.

```
215 \def\xintDSHr {\romannumeral0\xintdshr }%

216 \def\xintdshr #1#2%
217 {%

218     \expandafter\XINT_dshr_fork\the\numexpr#1\expandafter.\romannumeral`&&@#2;%
219 }%
220 \def\XINT_dshr_fork #1%
221 {%
222     \xint_UDzerominusfork
223     0#1\XINT_dshr_xzeroorneg
224     #1-\XINT_dshr_xzeroorneg
225     0-\XINT_dshr_xpositive
226     \krof #1%
227 }%
228 \def\XINT_dshr_xzeroorneg #1;{ 0}%
229 \def\XINT_dshr_xpositive
230 {%

231     \expandafter\xint_stop_atsecondoftwo\romannumeral0\XINT_dsx_xisPos
232 }%
```

## 5.11 \xintDSH

```
233 \def\xintDSH {\romannumeral0\xintdsh }%
234 \def\xintdsh #1#2%
235 {%

236     \expandafter\XINT_dsh_fork\the\numexpr#1\expandafter.\romannumeral`&&@#2;%
237 }%
238 \def\XINT_dsh_fork #1%
239 {%
240     \xint_UDzerominusfork
241     #1-\XINT_dsh_xiszero

242     0#1\XINT_dsx_xisNeg_checkA
243     0-{ \XINT_dsh_xisPos #1}%
244     \krof
245 }%
246 \def\XINT_dsh_xiszero #1.#2;{ #2}%
247 \def\XINT_dsh_xisPos
```

248 {%

\expandafter\xint\_stop\_atfirstoftwo\romannumeral0\XINT\_dsx\_xisPos

249 }%

## 5.12 \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.

250 \def\xintDSx {\romannumeral0\xintdsx }%

251 \def\xintdsx #1#2%

252 {%

253 \expandafter\XINT\_dsx\_fork\the\numexpr#1\expandafter.\romannumeral`&&@#2;%

254 }%

255 \def\XINT\_dsx\_fork #1%

256 {%

257 \xint\_UDzerominusfork

258 #1-\XINT\_dsx\_xisZero

259 0#1\XINT\_dsx\_xisNeg\_checkA

260 0-\XINT\_dsx\_xisPos #1}%

261 \krof

262 }%

263 \def\XINT\_dsx\_xisZero #1.#2;{{#2}{0}}%

264 \def\XINT\_dsx\_xisNeg\_checkA #1.#2%

265 {%

266 \xint\_gob\_til\_zero #2\XINT\_dsx\_xisNeg\_Azero 0%

267 \expandafter\XINT\_dsx\_append\romannumeral\XINT\_rep #1\endcsname 0.#2%

268 }%

269 \def\XINT\_dsx\_xisNeg\_Azero #1;{ 0}%

270 \def\XINT\_dsx\_addzeros #1%

271 {\expandafter\XINT\_dsx\_append\romannumeral\XINT\_rep#1\endcsname0.}%

272 \def\XINT\_dsx\_addzerosnofuss #1%

273 {\expandafter\XINT\_dsx\_append\romannumeral\xintreplicate{#1}0.}%

274 \def\XINT\_dsx\_append #1.#2;{ #2#1}%

275 \def\XINT\_dsx\_xisPos #1.#2%

276 {%

277 \xint\_UDzerominusfork

278 #2-\XINT\_dsx\_AisZero



```

279      0#2\XINT_dsx_AisNeg
280      0-\XINT_dsx_AisPos
281      \krof #1.#2%
282 }%
283 \def\XINT_dsx_AisZero #1;{{0}{0}}%
284 \def\XINT_dsx_AisNeg #1.-#2;%
285 {%
286      \expandafter\XINT_dsx_AisNeg_checkiffirstempty
287      \romannumeral0\XINT_split_xfork #1.#2\xint_bye2345678\xint_bye..%
288 }%

289 \def\XINT_dsx_AisNeg_checkiffirstempty #1%
290 {%
291      \xint_gob_til_dot #1\XINT_dsx_AisNeg_finish_zero.%
292      \XINT_dsx_AisNeg_finish_notzero #1%
293 }%
294 \def\XINT_dsx_AisNeg_finish_zero.\XINT_dsx_AisNeg_finish_notzero.#1.%
295 {%
296      \expandafter\XINT_dsx_end
297      \expandafter {\romannumeral0\XINT_num {-#1}}{0}%
298 }%
299 \def\XINT_dsx_AisNeg_finish_notzero #1.#2.%
300 {%
301      \expandafter\XINT_dsx_end
302      \expandafter {\romannumeral0\XINT_num {#2}}{-#1}%
303 }%

304 \def\XINT_dsx_AisPos #1.#2;%
305 {%
306      \expandafter\XINT_dsx_AisPos_finish
307      \romannumeral0\XINT_split_xfork #1.#2\xint_bye2345678\xint_bye..%
308 }%

309 \def\XINT_dsx_AisPos_finish #1.#2.%
310 {%
311      \expandafter\XINT_dsx_end
312      \expandafter {\romannumeral0\XINT_num {#2}}%
313      {\romannumeral0\XINT_num {#1}}%
314 }%
315 \def\XINT_dsx_end #1#2{\expandafter{#2}{#1}}%

```

### 5.13 \xintiiEq

*no \xintiieq.*

```

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

```

### 5.14 \xintiiNotEq

*Pour xintexpr. Pas de version en lowercase.*

```

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

```

## 5.15 \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.

```

318 \def\xintiiGeq {\romannumeral0\xintiigeq }%
319 \def\xintiigeq #1{\expandafter\XINT_iigeq\romannumeral`&&#1\xint:}%
320 \def\XINT_iigeq #1#2\xint:#3%
321 {%
322   \expandafter\XINT_geq_fork\expandafter #1\romannumeral`&&#3\xint:#2\xint:
323 }%

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

```

## 5.16 \xintiiGt

```

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

```

## 5.17 \xintiiLt

```

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

```

## 5.18 \xintiiGtorEq

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrc, xintexpr, indices*

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

## 5.19 \xintiiLtorEq

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

## 5.20 \xintiiIsZero

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

```
358 \def\xintiiIsZero {\romannumeral0\xintiiiszero }%
359 \def\xintiiiszero #1{\if0\xintiiSgn{#1}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%
```

## 5.21 \xintiiIsNotZero

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

```
360 \def\xintiiIsNotZero {\romannumeral0\xintiiisnotzero }%
361 \def\xintiiisnotzero
362      #1{\if0\xintiiSgn{#1}\xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi}%
```

## 5.22 \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.

```
363 \def\xintiiIsOne {\romannumeral0\xintiiisone }%
364 \def\xintiiisone #1{\expandafter\XINT_isone\romannumeral`&&@#1XY}%
365 \def\XINT_isone #1#2#3Y%
366 {%
367     \unless\if#2X\xint_dothis{ 0}\fi
368     \unless\if#11\xint_dothis{ 0}\fi
369     \xint_orthat{ 1}%
370 }%
371 \def\XINT_isOne #1{\XINT_is_One#1XY}%
372 \def\XINT_is_One #1#2#3Y%
373 {%
374     \unless\if#2X\xint_dothis0\fi
375     \unless\if#11\xint_dothis0\fi
376     \xint_orthat1%
377 }%
```

## 5.23 \xintiiOdd

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

```
378 \def\xintiiOdd {\romannumeral0\xintiiiodd }%
379 \def\xintiiiodd #1%
380 {%
381     \ifodd\xintLDg{#1} %<- intentional space
382     \xint_afterfi{ 1}%
383     \else
384     \xint_afterfi{ 0}%
385     \fi
386 }%
```

## 5.24 \xintiEven

```

387 \def\xintiEven {\romannumeral0\xintiieven }%
388 \def\xintiieven #1%
389 {%
390   \ifodd\xintLDg{#1} %<- intentional space
391   \xint_afterfi{ 0}%
392   \else
393   \xint_afterfi{ 1}%
394   \fi
395 }%

```

## 5.25 \xintiMON

MINUS ONE TO THE POWER N

```

396 \def\xintiMON {\romannumeral0\xintiimon }%
397 \def\xintiimon #1%
398 {%
399   \ifodd\xintLDg {#1} %<- intentional space
400   \xint_afterfi{ -1}%
401   \else
402   \xint_afterfi{ 1}%
403   \fi
404 }%

```

## 5.26 \xintiMMON

MINUS ONE TO THE POWER N-1

```

405 \def\xintiMMON {\romannumeral0\xintiimmon }%
406 \def\xintiimmon #1%
407 {%
408   \ifodd\xintLDg {#1} %<- intentional space
409   \xint_afterfi{ 1}%
410   \else
411   \xint_afterfi{ -1}%
412   \fi
413 }%

```

## 5.27 \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...`).

```

414 \def\xintSgnFork {\romannumeral0\xintsgnfork }%
415 \def\xintsgnfork #1%
416 {%
417   \ifcase #1 \expandafter\xint_stop_atsecondofthree
418   \or\expandafter\xint_stop_atthirdofthree
419   \else\expandafter\xint_stop_atfirstofthree
420   \fi
421 }%

```

## 5.28 \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)

```
422 \def\xintiiifSgn {\romannumeral0\xintiiifsgn }%
423 \def\xintiiifsgn #1%
424 {%
425     \ifcase \xintiiSgn{#1}
426         \expandafter\xint_stop_atsecondofthree
427         \or\expandafter\xint_stop_atthirdofthree
428         \else\expandafter\xint_stop_atfirstofthree
429     \fi
430 }%
```

## 5.29 \xintiiifCmp

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

```
431 \def\xintiiifCmp {\romannumeral0\xintiiifcmp }%
432 \def\xintiiifcmp #1#2%
433 {%
434     \ifcase\xintiiCmp {#1}{#2}
435         \expandafter\xint_stop_atsecondofthree
436         \or\expandafter\xint_stop_atthirdofthree
437         \else\expandafter\xint_stop_atfirstofthree
438     \fi
439 }%
```

## 5.30 \xintiiifEq

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

```
440 \def\xintiiifEq {\romannumeral0\xintiiifeq }%
441 \def\xintiiifeq #1#2%
442 {%
443     \if0\xintiiCmp{#1}{#2}%
444         \expandafter\xint_stop_atfirstoftwo
445         \else\expandafter\xint_stop_atsecondoftwo
446     \fi
447 }%
```

## 5.31 \xintiiifGt

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

```
448 \def\xintiiifGt {\romannumeral0\xintiiifgt }%
449 \def\xintiiifgt #1#2%
450 {%
```

```

451 \if1\xintiiCmp{#1}{#2}%
452 \expandafter\xint_stop_atfirstoftwo
453 \else\expandafter\xint_stop_atsecondoftwo
454 \fi
455 }%

```

### 5.32 \xintiiifLt

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

```

456 \def\xintiiifLt {\romannumeral0\xintiiiflt }%
457 \def\xintiiiflt #1#2%
458 {%
459 \ifnum\xintiiCmp{#1}{#2}<\xint_c_
460 \expandafter\xint_stop_atfirstoftwo
461 \else \expandafter\xint_stop_atsecondoftwo
462 \fi
463 }%

```

### 5.33 \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.

```

464 \def\xintiiifZero {\romannumeral0\xintiiifzero }%
465 \def\xintiiifzero #1%
466 {%
467 \if0\xintiiSgn{#1}%
468 \expandafter\xint_stop_atfirstoftwo
469 \else
470 \expandafter\xint_stop_atsecondoftwo
471 \fi
472 }%

```

### 5.34 \xintiiifNotZero

```

473 \def\xintiiifNotZero {\romannumeral0\xintiiifnotzero }%
474 \def\xintiiifnotzero #1%
475 {%
476 \if0\xintiiSgn{#1}%
477 \expandafter\xint_stop_atsecondoftwo
478 \else
479 \expandafter\xint_stop_atfirstoftwo
480 \fi
481 }%

```

### 5.35 \xintiiifOne

added in 1.09i. 1.1a adds \xintiiifOne.

```

482 \def\xintiiifOne {\romannumeral0\xintiiifone }%
483 \def\xintiiifone #1%

```

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

```
484 {%
485   \if1\xintiiIsOne{#1}%
486     \expandafter\xint_stop_atfirstoftwo
487   \else
488     \expandafter\xint_stop_atsecondoftwo
489   \fi
490 }%
```

### 5.36 `\xintiiifOdd`

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

```
491 \def\xintiiifOdd {\romannumeral0\xintiiifodd}%
492 \def\xintiiifodd #1%
493 {%
494   \if\xintiiOdd{#1}1%
495     \expandafter\xint_stop_atfirstoftwo
496   \else
497     \expandafter\xint_stop_atsecondoftwo
498   \fi
499 }%
```

### 5.37 `\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.

```
500 \def\xintifTrueAelseB {\romannumeral0\xintiiifnotzero}%
501 \def\xintifFalseAelseB{\romannumeral0\xintiiifzero}%
```

### 5.38 `\xintIsTrue`, `\xintIsFalse`

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

```
502 %\let\xintIsTrue \xintIsNotZero
503 %\let\xintIsFalse\xintIsZero
```

### 5.39 `\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.

```
504 \def\xintNOT{\romannumeral0\xintiiiszero}%
```

### 5.40 `\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.

```

505 \def\xintAND {\romannumeral0\xintand }%
506 \def\xintand #1#2{\if0\xintiiSgn{#1}\expandafter\xint_firstoftwo
507             \else\expandafter\xint_secondoftwo\fi
508             { 0}{\xintiiisnotzero{#2}}}%
509 \def\xintOR {\romannumeral0\xintor }%
510 \def\xintor #1#2{\if0\xintiiSgn{#1}\expandafter\xint_firstoftwo
511             \else\expandafter\xint_secondoftwo\fi
512             {\xintiiisnotzero{#2}}{ 1}}%
513 \def\xintXOR {\romannumeral0\xintxor }%
514 \def\xintxor #1#2{\if\xintiiIsZero{#1}\xintiiIsZero{#2}%
515             \xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi }%

```

## 5.41 \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.

```

516 \def\xintANDof {\romannumeral0\xintandof }%
517 \def\xintandof #1{\expandafter\XINT_andof_a\romannumeral`&&@#1\xint:}%
518 \def\XINT_andof_a #1{\expandafter\XINT_andof_b\romannumeral`&&@#1!}%
519 \def\XINT_andof_b #1%
520     {\xint_gob_til_xint: #1\XINT_andof_e\xint:\XINT_andof_c #1}%
521 \def\XINT_andof_c #1!%
522     {\xintifTrueAelseB {#1}{\XINT_andof_a}{\XINT_andof_no}}%
523 \def\XINT_andof_no #1\xint:{ 0}%
524 \def\XINT_andof_e #1!{ 1}%

```

## 5.42 \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.

```

525 \def\xintORof {\romannumeral0\xintorof }%
526 \def\xintorof #1{\expandafter\XINT_orof_a\romannumeral`&&@#1\xint:}%
527 \def\XINT_orof_a #1{\expandafter\XINT_orof_b\romannumeral`&&@#1!}%
528 \def\XINT_orof_b #1%
529     {\xint_gob_til_xint: #1\XINT_orof_e\xint:\XINT_orof_c #1}%
530 \def\XINT_orof_c #1!%
531     {\xintifTrueAelseB {#1}{\XINT_orof_yes}{\XINT_orof_a}}%
532 \def\XINT_orof_yes #1\xint:{ 1}%
533 \def\XINT_orof_e #1!{ 0}%

```

## 5.43 \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.

```

534 \def\xintXORof {\romannumeral0\xintxorof }%
535 \def\xintxorof #1{\expandafter\XINT_xorof_a\expandafter

```



```

536          0\romannumeral`&&@#1\xint:}%
537 \def\XINT_xorof_a #1#2{\expandafter\XINT_xorof_b\romannumeral`&&@#2!#1}%
538 \def\XINT_xorof_b #1%
539     {\xint_gob_til_xint: #1\XINT_xorof_e\xint:\XINT_xorof_c #1}%
540 \def\XINT_xorof_c #1!#2%
541     {\xintifTrueAelseB {#1}{\if #20\xint_afterfi{\XINT_xorof_a 1}%
542                                     \else\xint_afterfi{\XINT_xorof_a 0}\fi}%
543                                     {\XINT_xorof_a #2}%
544     }%
545 \def\XINT_xorof_e #1!#2{ #2}%

```

## 5.44 \xintiMax

At 1.2m, a long-standing bug was fixed: \xintiMax 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.

```

546 \def\xintiMax {\romannumeral0\xintiimax }%
547 \def\xintiimax #1%
548 {%
549     \expandafter\xint_iimax \romannumeral`&&@#1\xint:
550 }%
551 \def\xint_iimax #1\xint:#2%
552 {%
553     \expandafter\XINT_max_fork\romannumeral`&&@#2\xint:#1\xint:
554 }%

```

#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.

```

555 \def\XINT_max_fork #1#2\xint:#3#4\xint:
556 {%
557     \xint_UDsignsfork
558         #1#3\XINT_max_minusminus % A < 0, B < 0
559         #1-\XINT_max_plusminus % B < 0, A >= 0
560         #3-\XINT_max_minusplus % A < 0, B >= 0
561         --{\xint_UDzerosfork
562             #1#3\XINT_max_zerozero % A = B = 0
563             #10\XINT_max_pluszero % B = 0, A > 0
564             #30\XINT_max_zeropplus % A = 0, B > 0
565             00\XINT_max_plusplus % A, B > 0
566         \krof }%
567     \krof
568     #3#1#2\xint:#4\xint:
569     \expandafter\xint_stop_atfirstoftwo
570     \else
571     \expandafter\xint_stop_atsecondoftwo
572     \fi
573     {#3#4}{#1#2}%
574 }%

```

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

```

575 \def\XINT_max_zerozero #1\fi{\xint_stop_atfirstoftwo }%
576 \def\XINT_max_zeroplus #1\fi{\xint_stop_atsecondoftwo }%
577 \def\XINT_max_pluszero #1\fi{\xint_stop_atfirstoftwo }%
578 \def\XINT_max_minusplus #1\fi{\xint_stop_atsecondoftwo }%
579 \def\XINT_max_plusminus #1\fi{\xint_stop_atfirstoftwo }%
580 \def\XINT_max_plusplus
581 {%
582     \if1\romannumeral0\XINT_geq_plusplus
583 }%

```

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.

```

584 \def\XINT_max_minusminus --%
585 {%
586     \unless\if1\romannumeral0\XINT_geq_plusplus{}\}%
587 }%

```

## 5.45 \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.

```

588 \def\xintiiMin {\romannumeral0\xintiimin }%
589 \def\xintiimin #1%
590 {%
591     \expandafter\xint_iimin \romannumeral`&&@#1\xint:
592 }%
593 \def\xint_iimin #1\xint:#2%
594 {%
595     \expandafter\XINT_min_fork\romannumeral`&&@#2\xint:#1\xint:
596 }%
597 \def\XINT_min_fork #1#2\xint:#3#4\xint:
598 {%
599     \xint_UDsignsfork
600         #1#3\XINT_min_minusminus % A < 0, B < 0
601         #1-\XINT_min_plusminus % B < 0, A >= 0
602         #3-\XINT_min_minusplus % A < 0, B >= 0
603         --{\xint_UDzerosfork
604             #1#3\XINT_min_zerozero % A = B = 0
605             #10\XINT_min_pluszero % B = 0, A > 0
606             #30\XINT_min_zeroplus % A = 0, B > 0
607             00\XINT_min_plusplus % A, B > 0
608             \krof }%
609     \krof
610     #3#1#2\xint:#4\xint:
611     \expandafter\xint_stop_atsecondoftwo
612     \else
613     \expandafter\xint_stop_atfirstoftwo

```

```

614 \fi
615 {#3#4}{#1#2}%
616 }%
617 \def\XINT_min_zerozero #1\fi{\xint_stop_atfirstoftwo }%
618 \def\XINT_min_zeroplus #1\fi{\xint_stop_atfirstoftwo }%
619 \def\XINT_min_pluszero #1\fi{\xint_stop_atsecondoftwo }%
620 \def\XINT_min_minusplus #1\fi{\xint_stop_atfirstoftwo }%
621 \def\XINT_min_plusminus #1\fi{\xint_stop_atsecondoftwo }%
622 \def\XINT_min_plusplus
623 {%
624 \if1\romannumeral0\XINT_geq_plusplus
625 }%
626 \def\XINT_min_minusminus --%
627 {%
628 \unless\if1\romannumeral0\XINT_geq_plusplus{}\}%
629 }%

```

## 5.46 \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.

```

630 \def\xintiiMaxof {\romannumeral0\xintiimaxof }%
631 \def\xintiimaxof #1{\expandafter\XINT_iimaxof_a\romannumeral`&&@#1\xint:}%
632 \def\XINT_iimaxof_a #1{\expandafter\XINT_iimaxof_b\romannumeral`&&@#1!}%
633 \def\XINT_iimaxof_b #1!#2%
634 {\expandafter\XINT_iimaxof_c\romannumeral`&&@#2!{#1}!}%
635 \def\XINT_iimaxof_c #1%
636 {\xint_gob_til_xint: #1\XINT_iimaxof_e\xint:\XINT_iimaxof_d #1}%
637 \def\XINT_iimaxof_d #1!%
638 {\expandafter\XINT_iimaxof_b\romannumeral0\xintiimax {#1}}%
639 \def\XINT_iimaxof_e #1!#2!{ #2}%

```

## 5.47 \xintiiMinof

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

```

640 \def\xintiiMinof {\romannumeral0\xintiiminof }%
641 \def\xintiiminof #1{\expandafter\XINT_iiminof_a\romannumeral`&&@#1\xint:}%
642 \def\XINT_iiminof_a #1{\expandafter\XINT_iiminof_b\romannumeral`&&@#1!}%
643 \def\XINT_iiminof_b #1!#2%
644 {\expandafter\XINT_iiminof_c\romannumeral`&&@#2!{#1}!}%
645 \def\XINT_iiminof_c #1%
646 {\xint_gob_til_xint: #1\XINT_iiminof_e\xint:\XINT_iiminof_d #1}%
647 \def\XINT_iiminof_d #1!%
648 {\expandafter\XINT_iiminof_b\romannumeral0\xintiimin {#1}}%
649 \def\XINT_iiminof_e #1!#2!{ #2}%

```

## 5.48 \xintiiSum

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

```
650 \def\xintiisum {\romannumeral0\xintiisum }%
651 \def\xintiisum #1{\expandafter\XINT_sumexpr\romannumeral`&&#1\xint:}%
652 \def\XINT_sumexpr {\XINT_sum_loop_a 0\Z }%
653 \def\XINT_sum_loop_a #1\Z #2%
654     {\expandafter\XINT_sum_loop_b \romannumeral`&&#2\xint:#1\xint:\Z}%
655 \def\XINT_sum_loop_b #1%
656     {\xint_gob_til_xint: #1\XINT_sum_finished\xint:\XINT_sum_loop_c #1}%
657 \def\XINT_sum_loop_c
658     {\expandafter\XINT_sum_loop_a\romannumeral0\XINT_add_fork }%
659 \def\XINT_sum_finished\xint:\XINT_sum_loop_c\xint:\xint:#1\xint:\Z{ #1}%
```

## 5.49 \xintiiprd

`\xintiiprd {{a}...{z}}`

```
660 \def\xintiiprd {\romannumeral0\xintiiprd }%
661 \def\xintiiprd #1{\expandafter\XINT_prdexpr\romannumeral`&&#1\xint:}%
662 \def\XINT_prdexpr {\XINT_prod_loop_a 1\Z }%
663 \def\XINT_prod_loop_a #1\Z #2%
664     {\expandafter\XINT_prod_loop_b\romannumeral`&&#2\xint:#1\xint:\Z}%
665 \def\XINT_prod_loop_b #1%
666     {\xint_gob_til_xint: #1\XINT_prod_finished\xint:\XINT_prod_loop_c #1}%
667 \def\XINT_prod_loop_c
668     {\expandafter\XINT_prod_loop_a\romannumeral0\XINT_mul_fork }%
669 \def\XINT_prod_finished\xint:\XINT_prod_loop_c\xint:\xint:#1\xint:\Z { #1}%
```

## 5.50 \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.)

```

670 \def\xintiisSquareRoot {\romannumeral0\xintiisquareroot }%
671 \def\xintiisquareroot #1{\expandafter\XINT_sqrt_checkin\romannumeral`&&@#1\xint:}%
672 \def\XINT_sqrt_checkin #1%
673 {%
674     \xint_UDzerominusfork
675     #1-\XINT_sqrt_iszero
676     0#1\XINT_sqrt_isneg
677     0-\XINT_sqrt
678     \krof #1%
679 }%
680 \def\XINT_sqrt_iszero #1\xint:{{1}{1}}%
681 \def\XINT_sqrt_isneg #1\xint:{\XINT_signalcondition{InvalidOperation}{square
682     root of negative: #1}{0}{0}}}%
683 \def\XINT_sqrt #1\xint:
684 {%
685     \expandafter\XINT_sqrt_start\romannumeral0\xintlength {#1}.#1.%
686 }%
687 \def\XINT_sqrt_start #1.%
688 {%
689     \ifnum #1<\xint_c_x\xint_dothis\XINT_sqrt_small_a\fi
690     \xint_orthat\XINT_sqrt_big_a #1.%
691 }%
692 \def\XINT_sqrt_small_a #1.{\XINT_sqrt_a #1.\XINT_sqrt_small_d }%
693 \def\XINT_sqrt_big_a #1.{\XINT_sqrt_a #1.\XINT_sqrt_big_d }%
694 \def\XINT_sqrt_a #1.%
695 {%
696     \ifodd #1
697         \expandafter\XINT_sqrt_b0
698     \else
699         \expandafter\XINT_sqrt_bE
700     \fi
701     #1.%
702 }%

703 \def\XINT_sqrt_bE #1.#2#3#4%
704 {%
705     \XINT_sqrt_c {#3#4}#2{#1}#3#4%
706 }%

707 \def\XINT_sqrt_b0 #1.#2#3%
708 {%
709     \XINT_sqrt_c #3#2{#1}#3%
710 }%

711 \def\XINT_sqrt_c #1#2%
712 {%
713     \expandafter #2%
714     \the\numexpr \ifnum #1>\xint_c_ii
715         \ifnum #1>\xint_c_vi
716             \ifnum #1>12 \ifnum #1>20 \ifnum #1>30
717                 \ifnum #1>42 \ifnum #1>56 \ifnum #1>72

```

```

718             \ifnum #1>90
719     10\else 9\fi \else 8\fi \else 7\fi \else 6\fi \else 5\fi
720     \else 4\fi \else 3\fi \else 2\fi \else 1\fi .%
721 }%

722 \def\XINT_sqrt_small_d #1.#2%
723 {%
724     \expandafter\XINT_sqrt_small_e
725     \the\numexpr #1\ifcase \numexpr #2/\xint_c_ii-\xint_c_i\relax
726         \or 0\or 00\or 000\or 0000\fi .%
727 }%

728 \def\XINT_sqrt_small_e #1.#2.%
729 {%
730     \expandafter\XINT_sqrt_small_ea\the\numexpr #1*#1-#2.#1.%
731 }%

732 \def\XINT_sqrt_small_ea #1%
733 {%
734     \if0#1\xint_dothis\XINT_sqrt_small_ez\fi
735     \if-#1\xint_dothis\XINT_sqrt_small_eb\fi
736     \xint_orthat\XINT_sqrt_small_f #1%
737 }%
738 \def\XINT_sqrt_small_ez 0.#1.{\expandafter{\the\numexpr#1+\xint_c_i
739     \expandafter}\expandafter{\the\numexpr #1*\xint_c_ii+\xint_c_i}}%

740 \def\XINT_sqrt_small_eb -#1.#2.%
741 {%
742     \expandafter\XINT_sqrt_small_ec \the\numexpr
743     (#1-\xint_c_i+#2)/(\xint_c_ii*#2).#1.#2.%
744 }%

745 \def\XINT_sqrt_small_ec #1.#2.#3.%
746 {%
747     \expandafter\XINT_sqrt_small_f \the\numexpr
748     -#2+\xint_c_ii*#3*#1+#1*#1\expandafter.\the\numexpr #3+#1.%
749 }%

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

755 \def\XINT_sqrt_small_g #1#2.%
756 {%
757     \if 0#1%
758         \expandafter\XINT_sqrt_small_end
759     \else

```

```

760      \expandafter\XINT_sqrt_small_h
761      \fi
762      #1#2.%
763 }%

764 \def\XINT_sqrt_small_h #1.#2.#3.%
765 {%
766      \expandafter\XINT_sqrt_small_f
767      \the\numexpr #2-\xint_c_ii*#1*#3+#1*#1\expandafter.%
768      \the\numexpr #3-#1.%
769 }%
770 \def\XINT_sqrt_small_end #1.#2.#3.{{#3}{#2}}%

771 \def\XINT_sqrt_big_d #1.#2%
772 {%
773      \ifodd #2 \xint_dothis{\expandafter\XINT_sqrt_big_e0}\fi
774      \xint_orthat{\expandafter\XINT_sqrt_big_eE}%

775      \the\numexpr (#2-\xint_c_i)/\xint_c_ii.#1;%
776 }%

777 \def\XINT_sqrt_big_eE #1;#2#3#4#5#6#7#8#9%
778 {%
779      \XINT_sqrt_big_eE_a #1;{#2#3#4#5#6#7#8#9}%
780 }%

781 \def\XINT_sqrt_big_eE_a #1.#2;#3%
782 {%
783      \expandafter\XINT_sqrt_bigormed_f
784      \romannumeral0\XINT_sqrt_small_e #2000.#3.#1;%
785 }%

786 \def\XINT_sqrt_big_e0 #1;#2#3#4#5#6#7#8#9%
787 {%
788      \XINT_sqrt_big_e0_a #1;{#2#3#4#5#6#7#8#9}%
789 }%
790 \def\XINT_sqrt_big_e0_a #1.#2;#3#4%
791 {%
792      \expandafter\XINT_sqrt_bigormed_f
793      \romannumeral0\XINT_sqrt_small_e #20000.#3#4.#1;%
794 }%

795 \def\XINT_sqrt_bigormed_f #1#2#3;%
796 {%
797      \ifnum#3<\xint_c_ix
798          \xint_dothis {\csname XINT_sqrt_med_f\romannumeral#3\endcsname}%
799      \fi
800      \xint_orthat\XINT_sqrt_big_f #1.#2.#3;%
801 }%

```

```

802 \def\XINT_sqrt_med_fv    {\XINT_sqrt_med_fa .}%
803 \def\XINT_sqrt_med_fvi   {\XINT_sqrt_med_fa 0.}%
804 \def\XINT_sqrt_med_fvii  {\XINT_sqrt_med_fa 00.}%
805 \def\XINT_sqrt_med_fviii {\XINT_sqrt_med_fa 000.}%

806 \def\XINT_sqrt_med_fa #1.#2.#3.#4;%
807 {%
808     \expandafter\XINT_sqrt_med_fb
809     \the\numexpr (#30#1-5#1)/(\xint_c_ii*#2).#1.#2.#3.%
810 }%

811 \def\XINT_sqrt_med_fb #1.#2.#3.#4.#5.%
812 {%
813     \expandafter\XINT_sqrt_small_ea
814     \the\numexpr (#40#2-\xint_c_ii*#3*#1)*10#2+(#1*#1-#5)\expandafter.%
815     \the\numexpr #30#2-#1.%
816 }%

817 \def\XINT_sqrt_big_f #1;#2#3#4#5#6#7#8#9%
818 {%
819     \XINT_sqrt_big_fa #1;{#2#3#4#5#6#7#8#9}%
820 }%

821 \def\XINT_sqrt_big_fa #1.#2.#3;#4%
822 {%
823     \expandafter\XINT_sqrt_big_ga
824     \the\numexpr #3-\xint_c_viii\expandafter.%
825     \romannumeral0\XINT_sqrt_med_fa 000.#1.#2.;#4.%
826 }%

827 \def\XINT_sqrt_big_ga #1.#2#3%
828 {%
829     \ifnum #1>\xint_c_viii
830         \expandafter\XINT_sqrt_big_gb\else
831         \expandafter\XINT_sqrt_big_ka
832     \fi #1.#3.#2.%
833 }%

834 \def\XINT_sqrt_big_gb #1.#2.#3.%
835 {%
836     \expandafter\XINT_sqrt_big_gc
837     \the\numexpr (\xint_c_ii*#2-\xint_c_i)*\xint_c_x^viii/(\xint_c_iv*#3).%
838     #3.#2.#1;%
839 }%

840 \def\XINT_sqrt_big_gc #1.#2.#3.%
841 {%
842     \expandafter\XINT_sqrt_big_gd
843     \romannumeral0\xintiadd

```



```

844      {\xintiisub {#3000000000}\xintDouble{\xintiimul{#2}{#1}}{00000000}%
845      {\xintiisqr {#1}}.%
846      \romannumeral0\xintiisub{#2000000000}{#1}.%
847 }%

848 \def\XINT_sqrt_big_gd #1.#2.%
849 {%
850      \expandafter\XINT_sqrt_big_ge #2.#1.%
851 }%

852 \def\XINT_sqrt_big_ge #1;#2#3#4#5#6#7#8#9%
853      {\XINT_sqrt_big_gf #1.#2#3#4#5#6#7#8#9;}%
854 \def\XINT_sqrt_big_gf #1;#2#3#4#5#6#7#8#9%
855      {\XINT_sqrt_big_gg #1#2#3#4#5#6#7#8#9.}%

856 \def\XINT_sqrt_big_gg #1.#2.#3.#4.%
857 {%
858      \expandafter\XINT_sqrt_big_gloop
859      \expandafter\xint_c_xvi\expandafter.%
860      \the\numexpr #3-\xint_c_viii\expandafter.%
861      \romannumeral0\xintiisub {#2}{\xintiNum{#4}}.#1.%
862 }%

863 \def\XINT_sqrt_big_gloop #1.#2.%
864 {%
865      \unless\ifnum #1<#2 \xint_dothis\XINT_sqrt_big_ka \fi
866      \xint_orthat{\XINT_sqrt_big_gi #1.}#2.%
867 }%

868 \def\XINT_sqrt_big_gi #1.%
869 {%
870      \expandafter\XINT_sqrt_big_gj\romannumeral\xintreplicate{#1}0.#1.%
871 }%

872 \def\XINT_sqrt_big_gj #1.#2.#3.#4.#5.%
873 {%
874      \expandafter\XINT_sqrt_big_gk
875      \romannumeral0\xintiidivision {#4#1}%
876      {\XINT_dbl #5\xint_bye2345678\xint_bye*\xint_c_ii\relax}.%
877      #1.#5.#2.#3.%
878 }%

879 \def\XINT_sqrt_big_gk #1#2.#3.#4.%
880 {%
881      \expandafter\XINT_sqrt_big_gl
882      \romannumeral0\xintiiaadd {#2#3}{\xintiisqr{#1}}.%
883      \romannumeral0\xintiisub {#4#3}{#1}.%
884 }%

```

```

885 \def\XINT_sqrt_big_gl #1.#2.%
886 {%
887     \expandafter\XINT_sqrt_big_gm #2.#1.%
888 }%

889 \def\XINT_sqrt_big_gm #1.#2.#3.#4.#5.%
890 {%
891     \expandafter\XINT_sqrt_big_gn

892     \romannumeral0\XINT_split_fromleft\xint_c_ii*#3.#5\xint_bye2345678\xint_bye..%
893     #1.#2.#3.#4.%
894 }%

895 \def\XINT_sqrt_big_gn #1.#2.#3.#4.#5.#6.%
896 {%
897     \expandafter\XINT_sqrt_big_gloop
898     \the\numexpr \xint_c_ii*#5\expandafter.%
899     \the\numexpr #6-#5\expandafter.%
900     \romannumeral0\xintiisub{#4}{\xintiNum{#1}}.#3.#2.%
901 }%

902 \def\XINT_sqrt_big_ka #1.#2.#3.#4.%
903 {%
904     \expandafter\XINT_sqrt_big_kb

905     \romannumeral0\XINT_dsx_addzeros {#1}#3;.%
906     \romannumeral0\xintiisub
907     {\XINT_dsx_addzerosnofuss {\xint_c_ii*#1}#2;}%
908     {\xintiNum{#4}}}%
909 }%
910 \def\XINT_sqrt_big_kb #1.#2.%
911 {%
912     \expandafter\XINT_sqrt_big_kc #2.#1.%
913 }%

914 \def\XINT_sqrt_big_kc #1%
915 {%
916     \if0#1\xint_dothis\XINT_sqrt_big_kz\fi
917     \xint_orthat\XINT_sqrt_big_kloop #1%
918 }%
919 \def\XINT_sqrt_big_kz 0.#1.%
920 {%
921     \expandafter\XINT_sqrt_big_kend
922     \romannumeral0%
923     \xintinc{\XINT_dbl#1\xint_bye2345678\xint_bye*\xint_c_ii\relax}.#1.%
924 }%
925 \def\XINT_sqrt_big_kend #1.#2.%
926 {%
927     \expandafter{\romannumeral0\xintinc{#2}}{#1}%
928 }%

```

```

929 \def\XINT_sqrt_big_kloop #1.#2.%
930 {%
931     \expandafter\XINT_sqrt_big_ke
932     \romannumeral0\xintidivision{#1}%
933     {\romannumeral0\XINT_dbl #2\xint_bye2345678\xint_bye*\xint_c_ii\relax}{#2}%
934 }%

935 \def\XINT_sqrt_big_ke #1%
936 {%
937     \if0\XINT_Sgn #1\xint:
938         \expandafter \XINT_sqrt_big_end
939     \else \expandafter \XINT_sqrt_big_kf
940     \fi {#1}%
941 }%

942 \def\XINT_sqrt_big_kf #1#2#3%
943 {%
944     \expandafter\XINT_sqrt_big_kg
945     \romannumeral0\xintiisub {#3}{#1}.%
946     \romannumeral0\xintiiaadd {#2}{\xintiiSqr {#1}}.%
947 }%
948 \def\XINT_sqrt_big_kg #1.#2.%
949 {%
950     \expandafter\XINT_sqrt_big_kloop #2.#1.%
951 }%

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

```

### 5.51 \xintiiSqrt, \xintiiSqrR

```

953 \def\xintiiSqrt {\romannumeral0\xintiisqrt }%
954 \def\xintiisqrt {\expandafter\XINT_sqrt_post\romannumeral0\xintiisquareroot }%
955 \def\XINT_sqrt_post #1#2{\XINT_dec #1\XINT_dec_bye234567890\xint_bye}%
956 \def\xintiiSqrR {\romannumeral0\xintiisqrtr }%
957 \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- $\rightarrow \#1$  si  $\#2 \geq \#1$ ,  $(\#1 - .5)^2 \geq N + .25 > N$ , donc rounded- $\rightarrow \#1 - 1$ .

```

958 \def\XINT_sqrtr_post #1#2%
959     {\xintiiiflt {#2}{#1}{ #1}{\XINT_dec #1\XINT_dec_bye234567890\xint_bye}}%

```

### 5.52 \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 \xintiBinomial. (which xintfrac.sty redefined to use \xintNum)

```

960 \def\xintiiBinomial {\romannumeral0\xintiibinomial }%

```

```

961 \def\xintiibinomial #1#2%
962 {%
963   \expandafter\XINT_binom_pre\the\numexpr #1\expandafter.\the\numexpr #2.%
964 }%
965 \def\XINT_binom_pre #1.#2.%
966 {%
967   \expandafter\XINT_binom_fork \the\numexpr#1-#2.#2.#1.%
968 }%

```

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.

```

969 \def\XINT_binom_fork #1#2.#3#4.#5#6.%
970 {%
971   \if-#5\xint_dothis{\XINT_signalcondition{InvalidOperation}{Binomial with
972     negative first arg: #5#6}{0}}\fi
973   \if-#1\xint_dothis{ 0}\fi
974   \if-#3\xint_dothis{ 0}\fi
975   \if0#1\xint_dothis{ 1}\fi
976   \if0#3\xint_dothis{ 1}\fi
977   \ifnum #5#6>\xint_c_x^viii_mone\xint_dothis
978     {\XINT_signalcondition{InvalidOperation}{Binomial with too
979       large argument: 99999999 < #5#6}{0}}\fi
980   \ifnum #1#2>#3#4 \xint_dothis{\XINT_binom_a #1#2.#3#4.}\fi
981   \xint_orthat{\XINT_binom_a #3#4.#1#2.}%
982 }%

```

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

Le \xint\_bye!2!3!4!5!6!7!8!9!\xint\_bye\xint\_c\_i\relax est le terminateur pour le \XINT\_unsep\_cuzsmall final.

```

983 \def\XINT_binom_a #1.#2.%
984 {%
985   \expandafter\XINT_binom_b\the\numexpr \xint_c_i+#1.1.#2.100000001!1!;!0!%
986 }%

```

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

```

987 \def\XINT_binom_b #1.%
988 {%
989   \ifnum #1>9999 \xint_dothis\XINT_binom_vbigloop \fi
990   \ifnum #1>463 \xint_dothis\XINT_binom_bigloop \fi
991   \ifnum #1>98 \xint_dothis\XINT_binom_medloop \fi
992   \ifnum #1>29 \xint_dothis\XINT_binom_smallloop \fi
993   \xint_orthat\XINT_binom_vsmallloop #1.%
994 }%

```

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices*

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.

```

995 \def\XINT_binom_smallloop #1.#2.#3.%
996 {%
997     \ifcase\numexpr #3-#2\relax
998         \expandafter\XINT_binom_end_
999     \or \expandafter\XINT_binom_end_i
1000     \or \expandafter\XINT_binom_end_ii
1001     \or \expandafter\XINT_binom_end_iii
1002     \else\expandafter\XINT_binom_smallloop_a
1003     \fi #1.#2.#3.%
1004 }%
```

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.

```

1005 \def\XINT_binom_smallloop_a #1.#2.#3.%
1006 {%
1007     \expandafter\XINT_binom_smallloop_b
1008     \the\numexpr #1+\xint_c_iv\expandafter.%
1009     \the\numexpr #2+\xint_c_iv\expandafter.%
1010     \the\numexpr #3\expandafter.%
1011     \the\numexpr\expandafter\XINT_binom_div
1012     \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1013     !\romannumeral0\expandafter\XINT_binom_mul
1014     \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1015 }%
1016 \def\XINT_binom_smallloop_b #1.%
1017 {%
1018     \ifnum #1>98 \expandafter\XINT_binom_medloop \else
1019         \expandafter\XINT_binom_smallloop \fi #1.%
1020 }%
```

Ici on prend trois par trois.

```

1021 \def\XINT_binom_medloop #1.#2.#3.%
1022 {%
1023     \ifcase\numexpr #3-#2\relax
1024         \expandafter\XINT_binom_end_
1025     \or \expandafter\XINT_binom_end_i
1026     \or \expandafter\XINT_binom_end_ii
1027     \else\expandafter\XINT_binom_medloop_a
1028     \fi #1.#2.#3.%
1029 }%
1030 \def\XINT_binom_medloop_a #1.#2.#3.%
1031 {%
1032     \expandafter\XINT_binom_medloop_b
1033     \the\numexpr #1+\xint_c_iii\expandafter.%
1034     \the\numexpr #2+\xint_c_iii\expandafter.%
1035     \the\numexpr #3\expandafter.%
1036     \the\numexpr\expandafter\XINT_binom_div
1037     \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
1038     !\romannumeral0\expandafter\XINT_binom_mul
```

```

1039      \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1040 }%
1041 \def\XINT_binom_medloop_b #1.%
1042 {%
1043     \ifnum #1>463 \expandafter\XINT_binom_bigloop \else
1044                 \expandafter\XINT_binom_medloop \fi #1.%
1045 }%

```

Ici on prend deux par deux.

```

1046 \def\XINT_binom_bigloop #1.#2.#3.%
1047 {%
1048     \ifcase\numexpr #3-#2\relax
1049         \expandafter\XINT_binom_end_
1050     \or \expandafter\XINT_binom_end_i
1051     \else\expandafter\XINT_binom_bigloop_a
1052     \fi #1.#2.#3.%
1053 }%
1054 \def\XINT_binom_bigloop_a #1.#2.#3.%
1055 {%
1056     \expandafter\XINT_binom_bigloop_b
1057     \the\numexpr #1+\xint_c_ii\expandafter.%
1058     \the\numexpr #2+\xint_c_ii\expandafter.%
1059     \the\numexpr #3\expandafter.%
1060     \the\numexpr\expandafter\XINT_binom_div
1061         \the\numexpr #2*(#2+\xint_c_i)\expandafter
1062     !\romannumeral0\expandafter\XINT_binom_mul
1063     \the\numexpr #1*(#1+\xint_c_i)!%
1064 }%
1065 \def\XINT_binom_bigloop_b #1.%
1066 {%
1067     \ifnum #1>9999 \expandafter\XINT_binom_vbigloop \else
1068                 \expandafter\XINT_binom_bigloop \fi #1.%
1069 }%

```

Et finalement un par un.

```

1070 \def\XINT_binom_vbigloop #1.#2.#3.%
1071 {%
1072     \ifnum #3=#2
1073         \expandafter\XINT_binom_end_
1074     \else\expandafter\XINT_binom_vbigloop_a
1075     \fi #1.#2.#3.%
1076 }%
1077 \def\XINT_binom_vbigloop_a #1.#2.#3.%
1078 {%
1079     \expandafter\XINT_binom_vbigloop
1080     \the\numexpr #1+\xint_c_i\expandafter.%
1081     \the\numexpr #2+\xint_c_i\expandafter.%
1082     \the\numexpr #3\expandafter.%
1083     \the\numexpr\expandafter\XINT_binom_div\the\numexpr #2\expandafter
1084     !\romannumeral0\XINT_binom_mul #1!%
1085 }%

```

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.

```

1086 \def\XINT_binom_vsmallloop #1.#2.#3.%
1087 {%
1088   \ifcase\numexpr #3-#2\relax
1089     \expandafter\XINT_binom_vsmallend_
1090   \or \expandafter\XINT_binom_vsmallend_i
1091   \or \expandafter\XINT_binom_vsmallend_ii
1092   \or \expandafter\XINT_binom_vsmallend_iii
1093   \else\expandafter\XINT_binom_vsmallloop_a
1094   \fi #1.#2.#3.%
1095 }%
1096 \def\XINT_binom_vsmallloop_a #1.%
1097 {%
1098   \ifnum #1>26 \expandafter\XINT_binom_smallloop_a \else
1099     \expandafter\XINT_binom_vsmallloop_b \fi #1.%
1100 }%
1101 \def\XINT_binom_vsmallloop_b #1.#2.#3.%
1102 {%
1103   \expandafter\XINT_binom_vsmallloop
1104   \the\numexpr #1+\xint_c_iv\expandafter.%
1105   \the\numexpr #2+\xint_c_iv\expandafter.%
1106   \the\numexpr #3\expandafter.%
1107   \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1108   \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1109   !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1110 }%
1111 \def\XINT_binom_mul #1!#21!;!0!%
1112 {%
1113   \expandafter\XINT_rev_nounsep\expandafter{\expandafter}%
1114   \the\numexpr\expandafter\XINT_smallmul
1115   \the\numexpr\xint_c_x^viii+#1\expandafter
1116   !\romannumeral0\XINT_rev_nounsep {}1;!#2%
1117   \R!\R!\R!\R!\R!\R!\R!\R!\W
1118   \R!\R!\R!\R!\R!\R!\R!\R!\W
1119   1;!%
1120 }%
1121 \def\XINT_binom_div #1!1;!%
1122 {%
1123   \expandafter\XINT_smallldivx_a
1124   \the\numexpr #1/\xint_c_ii\expandafter\xint:
1125   \the\numexpr \xint_c_x^viii+#1!%
1126 }%

```

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

```

1127 \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.

```

1128 \def\XINT_binom_end_iii #1.#2.#3.%
1129 {%
1130     \expandafter\XINT_binom_finish
1131     \the\numexpr\expandafter\XINT_binom_div
1132         \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1133         !\romannumeral0\expandafter\XINT_binom_mul
1134         \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1135 }%
1136 \def\XINT_binom_end_ii #1.#2.#3.%
1137 {%
1138     \expandafter\XINT_binom_finish
1139     \the\numexpr\expandafter\XINT_binom_div
1140         \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
1141         !\romannumeral0\expandafter\XINT_binom_mul
1142         \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1143 }%
1144 \def\XINT_binom_end_i #1.#2.#3.%
1145 {%
1146     \expandafter\XINT_binom_finish
1147     \the\numexpr\expandafter\XINT_binom_div
1148         \the\numexpr #2*(#2+\xint_c_i)\expandafter
1149         !\romannumeral0\expandafter\XINT_binom_mul
1150         \the\numexpr #1*(#1+\xint_c_i)!%
1151 }%
1152 \def\XINT_binom_end_ #1.#2.#3.%
1153 {%
1154     \expandafter\XINT_binom_finish
1155     \the\numexpr\expandafter\XINT_binom_div\the\numexpr #2\expandafter
1156     !\romannumeral0\XINT_binom_mul #1!%
1157 }%
1158 \def\XINT_binom_finish #1;!0!%
1159     {\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).

```

1160 \def\XINT_binom_vsmallend_iii #1.%
1161 {%
1162     \ifnum #1>26 \expandafter\XINT_binom_end_iii \else
1163         \expandafter\XINT_binom_vsmallend_iiib \fi #1.%
1164 }%
1165 \def\XINT_binom_vsmallend_iiib #1.#2.#3.%
1166 {%
1167     \expandafter\XINT_binom_vsmallfinish
1168     \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1169     \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1170     !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1171 }%
1172 \def\XINT_binom_vsmallend_ii #1.%

```



```

1173 {%
1174     \ifnum #1>27 \expandafter\XINT_binom_end_ii \else
1175         \expandafter\XINT_binom_vsmallend_iib \fi #1.%
1176 }%
1177 \def\XINT_binom_vsmallend_iib #1.#2.#3.%
1178 {%
1179     \expandafter\XINT_binom_vsmallfinish
1180     \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1181     \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
1182     !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1183 }%
1184 \def\XINT_binom_vsmallend_i #1.%
1185 {%
1186     \ifnum #1>28 \expandafter\XINT_binom_end_i \else
1187         \expandafter\XINT_binom_vsmallend_ib \fi #1.%
1188 }%
1189 \def\XINT_binom_vsmallend_ib #1.#2.#3.%
1190 {%
1191     \expandafter\XINT_binom_vsmallfinish
1192     \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1193     \the\numexpr #2*(#2+\xint_c_i)\expandafter
1194     !\the\numexpr #1*(#1+\xint_c_i)!%
1195 }%
1196 \def\XINT_binom_vsmallend_ #1.%
1197 {%
1198     \ifnum #1>29 \expandafter\XINT_binom_end_ \else
1199         \expandafter\XINT_binom_vsmallend_b \fi #1.%
1200 }%
1201 \def\XINT_binom_vsmallend_b #1.#2.#3.%
1202 {%
1203     \expandafter\XINT_binom_vsmallfinish
1204     \the\numexpr\XINT_binom_vsmallmuldiv #2!#1!%
1205 }%
1206 \def\XINT_binom_vsmallfinish#1{%
1207 \def\XINT_binom_vsmallfinish1##1!1!;!0!{\expandafter#1\the\numexpr##1\relax}%
1208 }\XINT_binom_vsmallfinish{ }%

```

## 5.53 \xintiPFactorial

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

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

```

1209 \def\xintiPFactorial {\romannumeral0\xintiipfactorial }%
1210 \def\xintiipfactorial #1#2%
1211 {%
1212     \expandafter\XINT_pfac_fork\the\numexpr#1\expandafter.\the\numexpr #2.%
1213 }%
1214 \def\xintPFactorial{\romannumeral0\xintpfactorial}%

```

1215 \let\xintpfactorial\xintiipfactorial

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

```

1216 \def\xINT_pfac_fork #1#2.#3#4.%
1217 {%
1218     \unless\ifnum #1#2<#3#4 \xint_dothis\xINT_pfac_one\fi
1219     \if-#3\xint_dothis\xINT_pfac_neg\fi
1220     \if-#1\xint_dothis\xINT_pfac_zero\fi
1221     \ifnum #3#4>\xint_c_x^viii_mone\xint_dothis\xINT_pfac_outofrange\fi
1222     \xint_orthat \xINT_pfac_a #1#2.#3#4.%
1223 }%
1224 \def\xINT_pfac_outofrange #1.#2.%
1225     {\xINT_signalcondition{InvalidOperation}{PFactorial with
1226     too big second arg: 99999999 < #2}{0}}%
1227 \def\xINT_pfac_one #1.#2.{ 1}%
1228 \def\xINT_pfac_zero #1.#2.{ 0}%
1229 \def\xINT_pfac_neg -#1.-#2.%
1230 {%
1231     \ifnum #1>\xint_c_x^viii\xint_dothis\xINT_pfac_outofrange\fi
1232     \xint_orthat
1233     {\ifodd\numexpr#2-#1\relax\xint_afterfi{\expandafter-\romannumeral`&&@\fi
1234     \expandafter\xINT_pfac_a }%
1235     \the\numexpr #2-\xint_c_i\expandafter.\the\numexpr#1-\xint_c_i.%
1236 }%
1237 \def\xINT_pfac_a #1.#2.%
1238 {%
1239     \expandafter\xINT_pfac_b\the\numexpr \xint_c_i+#1.#2.1000000011;!;%
1240     1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1241 }%
1242 \def\xINT_pfac_b #1.%
1243 {%
1244     \ifnum #1>9999 \xint_dothis\xINT_pfac_vbigloop \fi
1245     \ifnum #1>463 \xint_dothis\xINT_pfac_bigloop \fi
1246     \ifnum #1>98 \xint_dothis\xINT_pfac_medloop \fi
1247     \xint_orthat\xINT_pfac_smallloop #1.%
1248 }%
1249 \def\xINT_pfac_smallloop #1.#2.%
1250 {%
1251     \ifcase\numexpr #2-#1\relax
1252         \expandafter\xINT_pfac_end_
1253     \or \expandafter\xINT_pfac_end_i
1254     \or \expandafter\xINT_pfac_end_ii
1255     \or \expandafter\xINT_pfac_end_iii
1256     \else\expandafter\xINT_pfac_smallloop_a
1257     \fi #1.#2.%
1258 }%
1259 \def\xINT_pfac_smallloop_a #1.#2.%
1260 {%
1261     \expandafter\xINT_pfac_smallloop_b
1262     \the\numexpr #1+\xint_c_iv\expandafter.%
1263     \the\numexpr #2\expandafter.%

```

```

1264 \the\numexpr\expandafter\XINT_smallmul
1265 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1266 }%
1267 \def\XINT_pfac_smallloop_b #1.%
1268 {%
1269 \ifnum #1>98 \expandafter\XINT_pfac_medloop \else
1270 \expandafter\XINT_pfac_smallloop \fi #1.%
1271 }%
1272 \def\XINT_pfac_medloop #1.#2.%
1273 {%
1274 \ifcase\numexpr #2-#1\relax
1275 \expandafter\XINT_pfac_end_
1276 \or \expandafter\XINT_pfac_end_i
1277 \or \expandafter\XINT_pfac_end_ii
1278 \else\expandafter\XINT_pfac_medloop_a
1279 \fi #1.#2.%
1280 }%
1281 \def\XINT_pfac_medloop_a #1.#2.%
1282 {%
1283 \expandafter\XINT_pfac_medloop_b
1284 \the\numexpr #1+\xint_c_iii\expandafter.%
1285 \the\numexpr #2\expandafter.%
1286 \the\numexpr\expandafter\XINT_smallmul
1287 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1288 }%
1289 \def\XINT_pfac_medloop_b #1.%
1290 {%
1291 \ifnum #1>463 \expandafter\XINT_pfac_bigloop \else
1292 \expandafter\XINT_pfac_medloop \fi #1.%
1293 }%
1294 \def\XINT_pfac_bigloop #1.#2.%
1295 {%
1296 \ifcase\numexpr #2-#1\relax
1297 \expandafter\XINT_pfac_end_
1298 \or \expandafter\XINT_pfac_end_i
1299 \else\expandafter\XINT_pfac_bigloop_a
1300 \fi #1.#2.%
1301 }%
1302 \def\XINT_pfac_bigloop_a #1.#2.%
1303 {%
1304 \expandafter\XINT_pfac_bigloop_b
1305 \the\numexpr #1+\xint_c_ii\expandafter.%
1306 \the\numexpr #2\expandafter.%
1307 \the\numexpr\expandafter
1308 \XINT_smallmul\the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
1309 }%
1310 \def\XINT_pfac_bigloop_b #1.%
1311 {%
1312 \ifnum #1>9999 \expandafter\XINT_pfac_vbigloop \else
1313 \expandafter\XINT_pfac_bigloop \fi #1.%
1314 }%
1315 \def\XINT_pfac_vbigloop #1.#2.%

```

```

1316 {%
1317     \ifnum #2=#1
1318         \expandafter\XINT_pfac_end_
1319     \else\expandafter\XINT_pfac_vbigloop_a
1320     \fi #1.#2.%
1321 }%
1322 \def\XINT_pfac_vbigloop_a #1.#2.%
1323 {%
1324     \expandafter\XINT_pfac_vbigloop
1325     \the\numexpr #1+\xint_c_i\expandafter.%
1326     \the\numexpr #2\expandafter.%
1327     \the\numexpr\expandafter\XINT_smallmul\the\numexpr\xint_c_x^viii+#1!%
1328 }%
1329 \def\XINT_pfac_end_iii #1.#2.%
1330 {%
1331     \expandafter\XINT_mul_out
1332     \the\numexpr\expandafter\XINT_smallmul
1333     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1334 }%
1335 \def\XINT_pfac_end_ii #1.#2.%
1336 {%
1337     \expandafter\XINT_mul_out
1338     \the\numexpr\expandafter\XINT_smallmul
1339     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1340 }%
1341 \def\XINT_pfac_end_i #1.#2.%
1342 {%
1343     \expandafter\XINT_mul_out
1344     \the\numexpr\expandafter\XINT_smallmul
1345     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
1346 }%
1347 \def\XINT_pfac_end_ #1.#2.%
1348 {%
1349     \expandafter\XINT_mul_out
1350     \the\numexpr\expandafter\XINT_smallmul\the\numexpr \xint_c_x^viii+#1!%
1351 }%

```

## 5.54 \xintBool, \xintToggle

1.09c

```

1352 \def\xintBool #1{\romannumeral`&&@%
1353     \csname if#1\endcsname\expandafter1\else\expandafter0\fi }%
1354 \def\xintToggle #1{\romannumeral`&&@\iftoggle{#1}{1}{0}}%

```

## 5.55 \xintGCD, \xintiiGCD

Copied over from [\xintiiGCD](#) of [xintgcd](#) at 1.3d to support [gcd\(\)](#) function in [\xintiexpr](#).

```

1355 \def\xintiiGCD {\romannumeral0\xintiigcd }%
1356 \def\xintiigcd #1{\expandafter\XINT_iigcd\romannumeral0\xintiiabs#1\xint:}%
1357 \def\XINT_iigcd #1#2\xint:#3%
1358 {%
1359     \expandafter\XINT_gcd_fork\expandafter#1%

```

```

1360          \romannumeral0\xintiiabs#3\xint:#1#2\xint:
1361 }%
1362 \def\xINT_gcd_fork #1#2%
1363 {%
1364   \xint_UDzerofork
1365   #1\xINT_gcd_Aiszero
1366   #2\xINT_gcd_Biszero
1367   0\xINT_gcd_loop
1368   \krof
1369   #2%
1370 }%
1371 \def\xINT_gcd_AisZero #1\xint:#2\xint:{ #1}%
1372 \def\xINT_gcd_BisZero #1\xint:#2\xint:{ #2}%
1373 \def\xINT_gcd_loop #1\xint:#2\xint:
1374 {%
1375   \expandafter\expandafter\expandafter\xINT_gcd_CheckRem
1376   \expandafter\xint_secondoftwo
1377   \romannumeral0\xINT_div_prepare {#1}{#2}\xint:#1\xint:
1378 }%
1379 \def\xINT_gcd_CheckRem #1%
1380 {%
1381   \xint_gob_til_zero #1\xINT_gcd_end0\xINT_gcd_loop #1%
1382 }%
1383 \def\xINT_gcd_end0\xINT_gcd_loop #1\xint:#2\xint:{ #2}%

```

## 5.56 \xintLCM, \xintiilCM

```

1384 \def\xintiilCM {\romannumeral0\xintiilcm}%
1385 \def\xintiilcm #1{\expandafter\xINT_iilcm\romannumeral0\xintiiabs#1\xint:}%
1386 \def\xINT_iilcm #1#2\xint:#3%
1387 {%
1388   \expandafter\xINT_lcm_fork\expandafter#1%
1389   \romannumeral0\xintiiabs#3\xint:#1#2\xint:
1390 }%
1391 \def\xINT_lcm_fork #1#2%
1392 {%
1393   \xint_UDzerofork
1394   #1\xINT_lcm_iszero
1395   #2\xINT_lcm_iszero
1396   0\xINT_lcm_notzero
1397   \krof
1398   #2%
1399 }%
1400 \def\xINT_lcm_iszero #1\xint:#2\xint:{ 0}%
1401 \def\xINT_lcm_notzero #1\xint:#2\xint:
1402 {%
1403   \expandafter\xINT_lcm_end\romannumeral0%
1404   \expandafter\expandafter\expandafter\xINT_gcd_CheckRem
1405   \expandafter\xint_secondoftwo
1406   \romannumeral0\xINT_div_prepare {#1}{#2}\xint:#1\xint:
1407   \xint:#1\xint:#2\xint:
1408 }%
1409 \def\xINT_lcm_end #1\xint:#2\xint:#3\xint:{\xintiimul {#2}{\xintiiQuo{#3}{#1}}}%

```

## 5.57 (WIP) \xintRandomDigits

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

```

1410 \def\xintRandomDigits{\romannumeral0\xintrandomdigits}%
1411 \def\xintrandomdigits#1%
1412 {%
1413   \csname xint_gob_andstop_\expandafter\XINT_randomdigits\the\numexpr#1\xint:
1414 }%
1415 \def\XINT_randomdigits#1\xint:
1416 {%
1417   \expandafter\XINT_randomdigits_a
1418   \the\numexpr(#1+\xint_c_iii)/\xint_c_viii\xint:#1\xint:
1419 }%
1420 \def\XINT_randomdigits_a#1\xint:#2\xint:
1421 {%
1422   \romannumeral\numexpr\xint_c_viii*#1-#2\csname XINT_%
1423   \romannumeral\XINT_replicate #1\endcsname \csname
1424   XINT_rdg\endcsname
1425 }%
1426 \def\XINT_rdg
1427 {%
1428   \expandafter\XINT_rdg_aux\the\numexpr%
1429   \xint_c_nine_x^viii%
1430   -\xint_texuniformdeviate\xint_c_ii^vii%
1431   -\xint_c_ii^vii*\xint_texuniformdeviate\xint_c_ii^vii%
1432   -\xint_c_ii^xiv*\xint_texuniformdeviate\xint_c_ii^vii%
1433   -\xint_c_ii^xxi*\xint_texuniformdeviate\xint_c_ii^vii%
1434   +\xint_texuniformdeviate\xint_c_x^viii%
1435   \relax%
1436 }%
1437 \def\XINT_rdg_aux#1{XINT_rdg\endcsname}%
1438 \let\XINT_XINT_rdg\endcsname

```

## 5.58 (WIP) \XINT\_eightrandomdigits

1.3b.

```

1439 \def\XINT_eightrandomdigits
1440 {%
1441   \expandafter\xint_gobble_i\the\numexpr%
1442   \xint_c_nine_x^viii%
1443   -\xint_texuniformdeviate\xint_c_ii^vii%
1444   -\xint_c_ii^vii*\xint_texuniformdeviate\xint_c_ii^vii%
1445   -\xint_c_ii^xiv*\xint_texuniformdeviate\xint_c_ii^vii%
1446   -\xint_c_ii^xxi*\xint_texuniformdeviate\xint_c_ii^vii%
1447   +\xint_texuniformdeviate\xint_c_x^viii%
1448   \relax%
1449 }%

```

## 5.59 (WIP) \xintXRandomDigits

### 1.3b.

```

1450 \def\xintXRandomDigits#1%
1451 {%
1452     \csname xint_gobble_\expandafter\XINT_xrandomdigits\the\numexpr#1\xint:
1453 }%
1454 \def\XINT_xrandomdigits#1\xint:
1455 {%
1456     \expandafter\XINT_xrandomdigits_a
1457     \the\numexpr(#1+\xint_c_iii)/\xint_c_viii\xint:#1\xint:
1458 }%
1459 \def\XINT_xrandomdigits_a#1\xint:#2\xint:
1460 {%
1461     \romannumeral\numexpr\xint_c_viii*#1-#2\expandafter\endcsname
1462     \romannumeral`&&\romannumeral
1463         \XINT_replicate #1\endcsname\XINT_eightrandomdigits
1464 }%

```

## 5.60 (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 \xintiexpr 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<=a, as in Python.

```

1465 \def\xintiiRandRangeAtoB{\romannumeral`&&\xintiirandrangeAtoB}%
1466 \def\xintiirandrangeAtoB#1%
1467 {%
1468     \expandafter\XINT_randrangeAtoB_a\romannumeral`&&#1\xint:
1469 }%
1470 \def\XINT_randrangeAtoB_a#1\xint:#2%
1471 {%
1472     \xintiiadd{\expandafter\XINT_randrange
1473         \romannumeral0\xintiisub{#2}{#1}\xint:}%
1474     {#1}%
1475 }%

```

## 5.61 (WIP) \xintiiRandRange

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

```

1476 \def\xintiiRandRange{\romannumeral`&&\xintiirandrange}%
1477 \def\xintiirandrange#1%
1478 {%
1479     \expandafter\XINT_randrange\romannumeral`&&#1\xint:
1480 }%
1481 \def\XINT_randrange #1%
1482 {%
1483     \xint_UDzerominusfork
1484     #1-\XINT_randrange_err:empty
1485     0#1\XINT_randrange_err:empty

```

```

1486      0-\XINT_randrange_a
1487      \krof #1%
1488 }%
1489 \def\XINT_randrange_err:empty#1\xint:
1490 {%
1491      \XINT_expandableerror{Empty range for randrange.} 0%
1492 }%
1493 \def\XINT_randrange_a #1\xint:
1494 {%
1495      \expandafter\XINT_randrange_b\romannumeral0\xintlength{#1}.#1\xint:
1496 }%
1497 \def\XINT_randrange_b #1.%
1498 {%
1499      \ifnum#1<\xint_c_x\xint_dothis{\the\numexpr\XINT_uniformdeviate{}}\fi
1500      \xint_orthat{\XINT_randrange_c #1.}%
1501 }%
1502 \def\XINT_randrange_c #1.#2#3#4#5#6#7#8#9%
1503 {%
1504      \expandafter\XINT_randrange_d
1505      \the\numexpr\expandafter\XINT_uniformdeviate\expandafter
1506      {\expandafter}\the\numexpr\xint_c_i+#2#3#4#5#6#7#8#9\xint:\xint:
1507      #2#3#4#5#6#7#8#9\xint:#1\xint:
1508 }%

```

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.

```

1509 \def\XINT_randrange_d #1\xint:#2\xint:
1510 {%
1511      \ifnum#1=\xint_c_\xint_dothis\XINT_randrange_Z\fi
1512      \ifnum#1=#2 \xint_dothis\XINT_randrange_A\fi
1513      \xint_orthat\XINT_randrange_e #1\xint:
1514 }%
1515 \def\XINT_randrange_e #1\xint:#2\xint:#3\xint:
1516 {%
1517      \the\numexpr#1\expandafter\relax
1518      \romannumeral0\xintrandomdigits{#2-\xint_c_viii}%
1519 }%

```

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...

```

1520 \def\XINT_randrange_Z 0\xint:#1\xint:#2\xint:
1521 {%
1522      \xintinum{\xintRandomDigits{#1-\xint_c_viii}}%
1523 }%

```

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.



```

1524 \def\XINT_randrange_A #1\xint:#2\xint:#3\xint:
1525 {%
1526     \expandafter\XINT_randrange_B
1527     \romannumeral0\xintrandomdigits{#2-\xint_c_viii}\xint:
1528     #3\xint:#2.#1\xint:
1529 }%
1530 \def\XINT_randrange_B #1\xint:#2\xint:#3.#4\xint:
1531 {%
1532     \xintiiiflt{#1}{#2}{\XINT_randrange_E}{\XINT_randrange_again}%
1533     #4#1\xint:#3.#4#2\xint:
1534 }%
1535 \def\XINT_randrange_E #1\xint:#2\xint:{ #1}%
1536 \def\XINT_randrange_again #1\xint:{\XINT_randrange_c}%

```

## 5.62 Adjustments for engines without uniformdeviate primitive

### 1.3b.

```

1537 \ifdefined\xint_texuniformdeviate
1538 \else
1539     \def\xintrandomdigits#1%
1540     {%
1541         \XINT_expandableerror
1542         {No uniformdeviate at engine level, returning 0.} 0%
1543     }%
1544     \let\xintXRandomDigits\xintRandomDigits
1545     \def\XINT_randrange#1\xint:
1546     {%
1547         \XINT_expandableerror
1548         {No uniformdeviate at engine level, returning 0.} 0%
1549     }%
1550 \fi
1551 \XINT_restorecatcodes_endinput%

```

## 6 Package [xintbinhex](#) implementation

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

The commenting is currently (2019/01/06) 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\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 [2019/01/06 1.3d 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 {\endcsname 0000}%
81 \expandafter\def\csname XINT_cshtb_1\endcsname {\endcsname 0001}%
82 \expandafter\def\csname XINT_cshtb_2\endcsname {\endcsname 0010}%
83 \expandafter\def\csname XINT_cshtb_3\endcsname {\endcsname 0011}%
84 \expandafter\def\csname XINT_cshtb_4\endcsname {\endcsname 0100}%
85 \expandafter\def\csname XINT_cshtb_5\endcsname {\endcsname 0101}%
86 \expandafter\def\csname XINT_cshtb_6\endcsname {\endcsname 0110}%
87 \expandafter\def\csname XINT_cshtb_7\endcsname {\endcsname 0111}%
88 \expandafter\def\csname XINT_cshtb_8\endcsname {\endcsname 1000}%
89 \expandafter\def\csname XINT_cshtb_9\endcsname {\endcsname 1001}%
90 \def\XINT_cshtb_A {\endcsname 1010}%
91 \def\XINT_cshtb_B {\endcsname 1011}%
92 \def\XINT_cshtb_C {\endcsname 1100}%
93 \def\XINT_cshtb_D {\endcsname 1101}%
94 \def\XINT_cshtb_E {\endcsname 1110}%
95 \def\XINT_cshtb_F {\endcsname 1111}%
96 \let\XINT_cshtb_none \endcsname

```

## 6.4 Helper macros

### 6.4.1 \XINT\_zeroes\_foriv

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

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

```

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

```

### 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+10000000000\relax\xint_bye!%
330   2!3!4!5!6!7!8!9!\xint_bye\XINT_htd_nextfour
331}%

```

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

```

332\def\XINT_htd_update 1#1#2#3#4#5#6!%
333{%
334   *\xint_c_ii^xvi+100000#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\xintbinto hex }%
400 \def\xintbinto hex #1%
401 {%
402     \expandafter\XINT_bth_checkin\romannumeral`&&@#1\xint:
403 }%
404 \def\XINT_bth_checkin #1%
405 {%
406     \xint_UDsignfork
407     #1\XINT_bth_N
408     -{\XINT_bth_main #1}%
409     \krof
410 }%
411 \def\XINT_bth_N {\expandafter-\romannumeral0\XINT_bth_main }%
412 \def\XINT_bth_main #1\xint:
413 {%
414     \csname space\csname\expandafter\XINT_bth_loop
415     \romannumeral0\XINT_zeroes_foriv
416     #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
417     #1\xint_bye2345678\xint_bye none\endcsname
418 }%
419 \def\XINT_bth_loop #1#2#3#4#5#6#7#8%
420 {%
421     XINT_csbth_#1#2#3#4%

```

```
422 \csname XINT_csbth_#5#6#7#8%
423 \csname\XINT_bth_loop
424 }%
```

## 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 }%
```

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices*

```
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 (2019/01/06) very sparse. Release 1.09h has modified a bit the `\xintTypesetEuclideanAlgorithm` and `\xintTypesetBezoutAlgorithm` layout with respect to line indentation in particular. And they use the `xinttools` `\xintloop` rather than the Plain TeX or  $\mu$ 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 [2019/01/06 1.3d Euclide algorithm with xint package (JFB)]%

```

## 7.3 \xintGCD, \xintiigcd

### 1.3d.

Removed some braces in favor of \xint: delimiter at 1.3d (but \xintiigcd was already robust against non-delimited \numexpr inputs thanks to using \xintiabs{...}) and refactored the whole \XINT\_iigcd\_fork.

```

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

```

This abuses the way \xintiabs expands.

```

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

```

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices*

*\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}}%

```

$\{n\}\{r_n\}\{a_n\}\{\{q_n\}\{r_n\}\}\dots\{A\}\{B\}}\}\Z$   
 $a(n) = r(n-1)$ . Pour  $n=0$  on a juste  $\{0\}\{B\}\{A\}\{\{A\}\{B\}\}}\Z$   
 $\backslash\text{XINT\_div\_prepare } \{u\}\{v\}$  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 }%

```

$\{n+1\}\{q(n+1)\}\{r(n+1)\}\{r_n\}\{\{q_n\}\{r_n\}\}\dots$

```

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

```

$r(n+1)\Z \{n+1\}\{r(n+1)\}\{r(n)\}\{\{q(n+1)\}\{r(n+1)\}\}\{\{q_n\}\{r_n\}\}\dots$   
 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}{{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:

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

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

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

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

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

```

363 \def\xintTypesetEuclideanAlgorithm {%
364   \unless\ifdefined\xintAssignArray
365     \errmessage
366     {xintgcd: package xinttools is required for \string\xintTypesetEuclideanAlgorithm}%
367     \expandafter\xint_gobble_iii
368   \fi
369   \XINT_TypesetEuclideanAlgorithm
370 }%
371 \def\XINT_TypesetEuclideanAlgorithm #1#2%
372 {% l'algo remplace #1 et #2 par |#1| et |#2|
373   \par
374   \begingroup
375     \xintAssignArray\xintEuclideanAlgorithm {#1}{#2}\to\U
376     \edef\A{\U2}\edef\B{\U4}\edef\N{\U1}%
377     \setbox 0 \vbox{\halign {\$##$\cr \A\cr \B \cr}}%
378     \count 255 1

```



```

379 \xintloop
380 \indent\hbox to \wd 0 {\hfil$U{\numexpr 2*\count255\relax}$}%
381 ${} = \U{\numexpr 2*\count255 + 3\relax}
382 \times \U{\numexpr 2*\count255 + 2\relax}
383 + \U{\numexpr 2*\count255 + 4\relax}$%
384 \ifnum \count255 < \N
385 \par
386 \advance \count255 1
387 \repeat
388 \endgroup
389 }%

```

## 7.11 \xintTypesetBezoutAlgorithm

Pour Bezout on a:  $\{N\}\{A\}\{0\}\{1\}\{D=r(n)\}\{B\}\{1\}\{0\}\{q_1\}\{r_1\}\{\alpha_1=q_1\}\{\beta_1=1\}$   
 $\{q_2\}\{r_2\}\{\alpha_2\}\{\beta_2\}\dots\{q_N\}\{r_N=0\}\{\alpha_N=A/D\}\{\beta_N=B/D\}$   
 Donc  $4N+8$  termes:  $U_1 = N$ ,  $U_2 = A$ ,  $U_5 = D$ ,  $U_6 = B$ ,  $q_1 = U_9$ ,  $q_n = U_{4n+5}$ ,  $n$  au moins 1  
 $r_n = U_{4n+6}$ ,  $n$  au moins -1  
 $\alpha(n) = U_{4n+7}$ ,  $n$  au moins -1  
 $\beta(n) = U_{4n+8}$ ,  $n$  au moins -1

1.09h uses \xintloop, and \par rather than \endgraf; and no more \parindent0pt

```

390 \def\xintTypesetBezoutAlgorithm {%
391 \unless\ifdefined\xintAssignArray
392 \errmessage
393 {xintgcd: package xinttools is required for \string\xintTypesetBezoutAlgorithm}%
394 \expandafter\xint_gobble_iii
395 \fi
396 \XINT_TypesetBezoutAlgorithm
397 }%
398 \def\XINT_TypesetBezoutAlgorithm #1#2%
399 {%
400 \par
401 \begingroup
402 \xintAssignArray\xintBezoutAlgorithm {#1}{#2}\to\BEZ
403 \edef\A{\BEZ2}\edef\B{\BEZ6}\edef\N{\BEZ1}% A = |#1|, B = |#2|
404 \setbox 0 \vbox{\halign {$##$\cr \A\cr \B \cr}}%
405 \count255 1
406 \xintloop
407 \indent\hbox to \wd 0 {\hfil$\BEZ{4*\count255 - 2}$}%
408 ${} = \BEZ{4*\count255 + 5}
409 \times \BEZ{4*\count255 + 2}
410 + \BEZ{4*\count255 + 6}$\hfill\break
411 \hbox to \wd 0 {\hfil$\BEZ{4*\count255 + 7}$}%
412 ${} = \BEZ{4*\count255 + 5}
413 \times \BEZ{4*\count255 + 3}
414 + \BEZ{4*\count255 - 1}$\hfill\break
415 \hbox to \wd 0 {\hfil$\BEZ{4*\count255 + 8}$}%
416 ${} = \BEZ{4*\count255 + 5}
417 \times \BEZ{4*\count255 + 4}
418 + \BEZ{4*\count255 }$
419 \par
420 \ifnum \count255 < \N

```

```

421   \advance \count255 1
422   \repeat
423   \edef\U{\BEZ{4*\N + 4}}%
424   \edef\V{\BEZ{4*\N + 3}}%
425   \edef\D{\BEZ5}%
426   \ifodd\N
427     $\U\times\A - \V\times \B = -\D$%
428   \else
429     $\U\times\A - \V\times\B = \D$%
430   \fi
431   \par
432   \endgroup
433 }%
434 \XINT_restorecatcodes_endinput%

```

## 8 Package [xintfrac](#) implementation

.1	Catcodes, $\varepsilon$ -TeX and reload detection	180	.43	\xintPFactorial	213
.2	Package identification	180	.44	\xintPrd	213
.3	\XINT_cntSgnFork	181	.45	\xintDiv	213
.4	\xintLen	181	.46	\xintDivFloor	214
.5	\XINT_outfrac	181	.47	\xintDivTrunc	214
.6	\XINT_inFrac	182	.48	\xintDivRound	214
.7	\XINT_frac_gen	184	.49	\xintModTrunc	214
.8	\XINT_factortens	186	.50	\xintDivMod	215
.9	\xintEq, \xintNotEq, \xintGt, \xintLt, \xintGtorEq, \xintLtorEq, \xintIsZero, \xintIsNotZero, \xintOdd, \xintEven, \xintIfSgn, \xintIfCmp, \xintIfEq, \xintIfGt, \xintIfLt, \xintIfZero, \xintIfNotZero, \xintIfOne, \xintIfOdd	187	.51	\xintMod	216
.10	\xintRaw	189	.52	\xintIsOne	217
.11	\xintPRaw	189	.53	\xintGeq	217
.12	\xintRawWithZeros	190	.54	\xintMax	219
.13	\xintDecToString	190	.55	\xintMaxof	219
.14	\xintFloor, \xintiFloor	190	.56	\xintMin	220
.15	\xintCeil, \xintiCeil	191	.57	\xintMinof	220
.16	\xintNumerator	191	.58	\xintCmp	221
.17	\xintDenominator	191	.59	\xintAbs	222
.18	\xintFrac	192	.60	\xintOpp	222
.19	\xintSignedFrac	192	.61	\xintInv	222
.20	\xintFwOver	193	.62	\xintSgn	223
.21	\xintSignedFwOver	193	.63	Floating point macros	223
.22	\xintREZ	194	.64	\xintFloat	223
.23	\xintE	194	.65	\XINTinFloat, \XINTinFloatS	225
.24	\xintIrr, \xintPIrr	195	.66	\xintPFloat	231
.25	\xintIfInt	196	.67	\XINTinFloatFracdigits	233
.26	\xintIsInt	197	.68	\xintFloatAdd, \XINTinFloatAdd	233
.27	\xintJrr	197	.69	\xintFloatSub, \XINTinFloatSub	234
.28	\xintTFrac	198	.70	\xintFloatMul, \XINTinFloatMul	235
.29	\xintTrunc, \xintiTrunc	198	.71	\xintFloatDiv, \XINTinFloatDiv	235
.30	\xintTTrunc	201	.72	\xintFloatPow, \XINTinFloatPow	236
.31	\xintNum	201	.73	\xintFloatPower, \XINTinFloatPower	240
.32	\xintRound, \xintiRound	201	.74	\xintFloatFac, \XINTFloatFac	244
.33	\xintXTrunc	202	.75	\xintFloatPFactorial, \XINTinFloatPFactorial	249
.34	\xintDigits	208	.76	\xintFloatBinomial, \XINTinFloatBinomial	253
.35	\xintAdd	208	.77	\xintFloatSqrt, \XINTinFloatSqrt	254
.36	\xintSub	210	.78	\xintFloatE, \XINTinFloatE	257
.37	\xintSum	210	.79	\XINTinFloatMod	257
.38	\xintMul	210	.80	\XINTinFloatDivFloor	258
.39	\xintSqr	211	.81	\XINTinFloatDivMod	258
.40	\xintPow	211	.82	\xintIfFloatInt	258
.41	\xintFac	212	.83	\xintFloatIsInt	259
.42	\xintBinomial	212	.84	(WIP) \XINTinRandomFloatS, \XINTinRandomFloatSdigits	259
			.85	(WIP) \XINTinRandomFloatSixteen	260

The commenting is currently (2019/01/06) very sparse.

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

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

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

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2 \catcode13=5 % ^^M
3 \endlinechar=13 %
4 \catcode123=1 % {
5 \catcode125=2 % }
6 \catcode64=11 % @
7 \catcode35=6 % #
8 \catcode44=12 % ,
9 \catcode45=12 % -
10 \catcode46=12 % .
11 \catcode58=12 % :
12 \let\z\endgroup
13 \expandafter\let\expandafter\x\csname ver@xintfrac.sty\endcsname
14 \expandafter\let\expandafter\w\csname ver@xint.sty\endcsname
15 \expandafter
16 \ifx\csname PackageInfo\endcsname\relax
17 \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
18 \else
19 \def\y#1#2{\PackageInfo{#1}{#2}}%
20 \fi
21 \expandafter
22 \ifx\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 [2019/01/06 1.3d 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  $\{\text{exponent}\}\{\text{Numerator}\}\{\text{Denominator}\}$  where Denominator is at least 1.

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

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

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

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

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

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

Regarding calling the macro with an argument  $A[\text{<expression>}]$ , 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_infrac_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 an \xintexpr.\relax

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





```

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     {\if90\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     {\if90\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     {\if10\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\xintraw }%
368 \def\xintraw
369 {%
370   \expandafter\XINT_raw\romannumeral0\XINT_infrac
371 }%
372 \def\XINT_raw #1#2#3{ #2/#3[#1]}%

```

## 8.11 \xintPraw

1.09b

```

373 \def\xintPraw {\romannumeral0\xintpraw }%
374 \def\xintpraw
375 {%
376   \expandafter\XINT_praw\romannumeral0\XINT_infrac
377 }%
378 \def\XINT_praw #1%
379 {%
380   \ifnum #1=\xint_c_ \expandafter\XINT_praw_a\fi \XINT_praw_A {#1}%
381 }%
382 \def\XINT_praw_A #1#2#3%
383 {%
384   \if\XINT_isOne{#3}1\expandafter\xint_firstoftwo
385     \else\expandafter\xint_secondoftwo
386   \fi { #2[#1]}{ #2/#3[#1]}%
387 }%
388 \def\XINT_praw_a\XINT_praw_A #1#2#3%
389 {%
390   \if\XINT_isOne{#3}1\expandafter\xint_firstoftwo
391     \else\expandafter\xint_secondoftwo
392   \fi { #2}{ #2/#3}%
393 }%

```

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrc, xintexpr, indices*

## 8.12 \xintRawWithZeros

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

```
394 \def\xintRawWithZeros {\romannumeral0\xinrawwithzeros }%
395 \def\xinrawwithzeros
396 {%
397     \expandafter\XINT_rawz_fork\romannumeral0\XINT_infrac
398 }%

399 \def\XINT_rawz_fork #1%
400 {%
401     \ifnum#1<\xint_c_
402         \expandafter\XINT_rawz_Ba
403     \else
404         \expandafter\XINT_rawz_A
405     \fi
406     #1.%
407 }%
408 \def\XINT_rawz_A  #1.#2#3{\XINT_dsx_addzeros{#1}#2;/#3}%
409 \def\XINT_rawz_Ba -#1.#2#3{\expandafter\XINT_rawz_Bb
410     \expandafter{\romannumeral0\XINT_dsx_addzeros{#1}#3;}{#2}}%
411 \def\XINT_rawz_Bb #1#2{ #2/#1}%

```

## 8.13 \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.*

```
412 \def\xintDecToString{\romannumeral0\xintdectostr}%
413 \def\xintdectostr#1{\expandafter\XINT_dectostr\romannumeral0\xinraw{#1}}%
414 \def\XINT_dectostr #1/#2[#3]{\xintiifZero {#1}%
415     \XINT_dectostr_z
416     {\if1\XINT_isOne{#2}\expandafter\XINT_dectostr_a
417         \else\expandafter\XINT_dectostr_b
418     \fi}%
419     #1/#2[#3]%
420 }%
421 \def\XINT_dectostr_z#1[#2]{ 0}%
422 \def\XINT_dectostr_a#1/#2[#3]{%
423     \ifnum#3<\xint_c_\xint_dothis{\xinttrunc{-#3}{#1[#3]}}\fi
424     \xint_orthat{\xintiie{#1}{#3}}%
425 }%
426 \def\XINT_dectostr_b#1/#2[#3]{% just to handle this somehow
427     \ifnum#3<\xint_c_\xint_dothis{\xinttrunc{-#3}{#1[#3]}/#2}\fi
428     \xint_orthat{\xintiie{#1}{#3}/#2}%
429 }%

```

## 8.14 \xintFloor, \xintiFloor

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

```

430 \def\xintFloor {\romannumeral0\xintfloor }%
431 \def\xintfloor #1% devrais-je faire \xintREZ?
432     {\expandafter\XINT_ifloor \romannumeral0\xintraawithzeros {#1}./1[0]}%
433 \def\xintiFloor {\romannumeral0\xintifloor }%
434 \def\xintifloor #1%
435     {\expandafter\XINT_ifloor \romannumeral0\xintraawithzeros {#1}.}%
436 \def\XINT_ifloor #1/#2.{\xintiigo {#1}{#2}}%

```

## 8.15 \xintCeil, \xintiCeil

1.09a

```

437 \def\xintCeil {\romannumeral0\xintceil }%
438 \def\xintceil #1{\xintiiopp {\xintFloor {\xintOpp{#1}}}}%
439 \def\xintiCeil {\romannumeral0\xinticeil }%
440 \def\xinticeil #1{\xintiiopp {\xintiFloor {\xintOpp{#1}}}}%

```

## 8.16 \xintNumerator

```

441 \def\xintNumerator {\romannumeral0\xintnumerator }%
442 \def\xintnumerator
443 {%
444     \expandafter\XINT_numer\romannumeral0\XINT_infrac
445 }%
446 \def\XINT_numer #1%
447 {%
448     \ifcase\XINT_cntSgn #1\xint:
449         \expandafter\XINT_numer_B
450     \or
451         \expandafter\XINT_numer_A
452     \else
453         \expandafter\XINT_numer_B
454     \fi
455     {#1}%
456 }%
457 \def\XINT_numer_A #1#2#3{\XINT_dsx_addzeros{#1}#2;}%
458 \def\XINT_numer_B #1#2#3{ #2}%

```

## 8.17 \xintDenominator

```

459 \def\xintDenominator {\romannumeral0\xintdenominator }%
460 \def\xintdenominator
461 {%
462     \expandafter\XINT_denom_fork\romannumeral0\XINT_infrac
463 }%
464 \def\XINT_denom_fork #1%
465 {%
466     \ifnum#1<\xint_c_
467         \expandafter\XINT_denom_B
468     \else
469         \expandafter\XINT_denom_A
470     \fi
471     #1.%

```

```

472 }%
473 \def\XINT_denom_A #1.#2#3{ #3}%
474 \def\XINT_denom_B -#1.#2#3{\XINT_dsx_addzeros{#1}#3;}%

```

## 8.18 \xintFrac

Useless typesetting macro.

```

475 \def\xintFrac {\romannumeral0\xintfrac }%
476 \def\xintfrac #1%
477 {%
478   \expandafter\XINT_fracfrac_A\romannumeral0\XINT_infrac {#1}%
479 }%
480 \def\XINT_fracfrac_A #1{\XINT_fracfrac_B #1\Z }%
481 \catcode`^=7
482 \def\XINT_fracfrac_B #1#2\Z
483 {%
484   \xint_gob_til_zero #1\XINT_fracfrac_C 0\XINT_fracfrac_D {10^{#1#2}}%
485 }%
486 \def\XINT_fracfrac_C 0\XINT_fracfrac_D #1#2#3%
487 {%
488   \if1\XINT_isOne {#3}%
489   \xint_afterfi {\expandafter\xint_stop_atfirstoftwo\xint_gobble_ii }%
490   \fi
491   \space
492   \frac {#2}{#3}%
493 }%
494 \def\XINT_fracfrac_D #1#2#3%
495 {%
496   \if1\XINT_isOne {#3}\XINT_fracfrac_E\fi
497   \space
498   \frac {#2}{#3}#1%
499 }%
500 \def\XINT_fracfrac_E \fi\space\frac #1#2{\fi \space #1\cdot }%

```

## 8.19 \xintSignedFrac

```

501 \def\xintSignedFrac {\romannumeral0\xintsignedfrac }%
502 \def\xintsignedfrac #1%
503 {%
504   \expandafter\XINT_sgnfrac_a\romannumeral0\XINT_infrac {#1}%
505 }%
506 \def\XINT_sgnfrac_a #1#2%
507 {%
508   \XINT_sgnfrac_b #2\Z {#1}%
509 }%
510 \def\XINT_sgnfrac_b #1%
511 {%
512   \xint_UDsignfork
513   #1\XINT_sgnfrac_N
514   -{\XINT_sgnfrac_P #1}%
515   \krof
516 }%
517 \def\XINT_sgnfrac_P #1\Z #2%

```



```

518 {%
519   \XINT_fracfrac_A {#2}{#1}%
520 }%
521 \def\XINT_sgnfrac_N
522 {%
523   \expandafter\romannumeral0\XINT_sgnfrac_P
524 }%

```

## 8.20 \xintFwOver

```

525 \def\xintFwOver {\romannumeral0\xintfwover }%
526 \def\xintfwover #1%
527 {%
528   \expandafter\XINT_fwover_A\romannumeral0\XINT_infrac {#1}%
529 }%
530 \def\XINT_fwover_A #1{\XINT_fwover_B #1\Z }%
531 \def\XINT_fwover_B #1#2\Z
532 {%
533   \xint_gob_til_zero #1\XINT_fwover_C 0\XINT_fwover_D {10^{#1#2}}%
534 }%
535 \catcode\^=11
536 \def\XINT_fwover_C #1#2#3#4#5%
537 {%
538   \if0\XINT_isOne {#5}\xint_afterfi { {#4\over #5}}%
539   \else\xint_afterfi { #4}%
540   \fi
541 }%
542 \def\XINT_fwover_D #1#2#3%
543 {%
544   \if0\XINT_isOne {#3}\xint_afterfi { {#2\over #3}}%
545   \else\xint_afterfi { #2\cdot }%
546   \fi
547   #1%
548 }%

```

## 8.21 \xintSignedFwOver

```

549 \def\xintSignedFwOver {\romannumeral0\xintsignedfwover }%
550 \def\xintsignedfwover #1%
551 {%
552   \expandafter\XINT_sgnfwover_a\romannumeral0\XINT_infrac {#1}%
553 }%
554 \def\XINT_sgnfwover_a #1#2%
555 {%
556   \XINT_sgnfwover_b #2\Z {#1}%
557 }%
558 \def\XINT_sgnfwover_b #1%
559 {%
560   \xint_UDsignfork
561   #1\XINT_sgnfwover_N
562   -{\XINT_sgnfwover_P #1}%
563   \krof
564 }%
565 \def\XINT_sgnfwover_P #1\Z #2%
566 {%

```

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfraction](#), [xintexpr](#), [indices](#)

```

567 \XINT_fwover_A {#2}{#1}%
568 }%
569 \def\XINT_sgnfwover_N
570 {%
571 \expandafter-\romannumeral0\XINT_sgnfwover_P
572 }%

```

## 8.22 \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.

```

573 \def\xintREZ {\romannumeral0\xintrez }%
574 \def\xintrez
575 {%
576 \expandafter\XINT_rez_A\romannumeral0\XINT_infrac
577 }%
578 \def\XINT_rez_A #1#2%
579 {%
580 \XINT_rez_AB #2\Z {#1}%
581 }%
582 \def\XINT_rez_AB #1%
583 {%
584 \xint_UDzerominusfork
585 #1-\XINT_rez_zero
586 0#1\XINT_rez_neg
587 0-{\XINT_rez_B #1}%
588 \krof
589 }%
590 \def\XINT_rez_zero #1\Z #2#3{ 0/1[0]}%
591 \def\XINT_rez_neg {\expandafter-\romannumeral0\XINT_rez_B }%
592 \def\XINT_rez_B #1\Z
593 {%
594 \expandafter\XINT_rez_C\romannumeral0\XINT_factortens {#1}%
595 }%
596 \def\XINT_rez_C #1.#2.#3#4%
597 {%
598 \expandafter\XINT_rez_D\romannumeral0\XINT_factortens {#4}#3+#2.#1.%
599 }%
600 \def\XINT_rez_D #1.#2.#3.%
601 {%
602 \expandafter\XINT_rez_E\the\numexpr #3-#2.#1.%
603 }%
604 \def\XINT_rez_E #1.#2.#3.{ #3/#2[#1]}%

```

## 8.23 \xintE

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

1.1 modifies and moves \xintiiE to xint.sty.

```

605 \def\xintE {\romannumeral0\xinte }%
606 \def\xinte #1%

```

```

607 {%
608   \expandafter\XINT_e \romannumeral0\XINT_infrac {#1}%
609 }%
610 \def\XINT_e #1#2#3#4%
611 {%
612   \expandafter\XINT_e_end\the\numexpr #1+#4.{#2}{#3}%
613 }%
614 \def\XINT_e_end #1.#2#3{ #2/#3[#1]}%

```

## 8.24 \xintIrr, \xintPIrr

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

```

615 \def\xintIrr {\romannumeral0\xintirr }%
616 \def\xintPIrr{\romannumeral0\xintpirr }%
617 \def\xintirr #1%
618 {%
619   \expandafter\XINT_irr_start\romannumeral0\xintraawithzeros {#1}\Z
620 }%
621 \def\xintpirr #1%
622 {%
623   \expandafter\XINT_pirr_start\romannumeral0\xintraaw{#1}%
624 }%
625 \def\XINT_irr_start #1#2/#3\Z
626 {%
627   \if0\XINT_isOne {#3}%
628     \xint_afterfi
629     {\xint_UDsignfork
630       #1\XINT_irr_negative
631       -{\XINT_irr_nonneg #1}%
632     \krof}%
633   \else
634     \xint_afterfi{\XINT_irr_denomiseone #1}%
635   \fi
636   #2\Z {#3}%
637 }%
638 \def\XINT_pirr_start #1#2/#3[%
639 {%
640   \if0\XINT_isOne {#3}%
641     \xint_afterfi
642     {\xint_UDsignfork
643       #1\XINT_irr_negative
644       -{\XINT_irr_nonneg #1}%
645     \krof}%
646   \else
647     \xint_afterfi{\XINT_irr_denomiseone #1}%
648   \fi
649   #2\Z {#3}[%
650 }%
651 \def\XINT_irr_denomiseone #1\Z #2{ #1/1}% changed in 1.08
652 \def\XINT_irr_negative #1\Z #2{\XINT_irr_D #1\Z #2\Z -}%
653 \def\XINT_irr_nonneg #1\Z #2{\XINT_irr_D #1\Z #2\Z \space}%
654 \def\XINT_irr_D #1#2\Z #3#4\Z

```

```

655 {%
656   \xint_UDzerosfork
657     #3#1\XINT_irr_indeterminate
658     #30\XINT_irr_divisionbyzero
659     #10\XINT_irr_zero
660     00\XINT_irr_loop_a
661   \krof
662   {#3#4}{#1#2}{#3#4}{#1#2}%
663 }%
664 \def\XINT_irr_indeterminate #1#2#3#4#5%
665 {%
666   \XINT_signalcondition{DivisionUndefined}{indeterminate: 0/0}{{0/1}%
667 }%
668 \def\XINT_irr_divisionbyzero #1#2#3#4#5%
669 {%
670   \XINT_signalcondition{DivisionByZero}{vanishing denominator: #5#2/0}{{0/1}%
671 }%
672 \def\XINT_irr_zero #1#2#3#4#5{ 0/1}% changed in 1.08
673 \def\XINT_irr_loop_a #1#2%
674 {%
675   \expandafter\XINT_irr_loop_d
676   \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
677 }%
678 \def\XINT_irr_loop_d #1#2%
679 {%
680   \XINT_irr_loop_e #2\Z
681 }%
682 \def\XINT_irr_loop_e #1#2\Z
683 {%
684   \xint_gob_til_zero #1\XINT_irr_loop_exit0\XINT_irr_loop_a {#1#2}%
685 }%
686 \def\XINT_irr_loop_exit0\XINT_irr_loop_a #1#2#3#4%
687 {%
688   \expandafter\XINT_irr_loop_exitb\expandafter
689   {\romannumeral0\xintiique {#3}{#2}}%
690   {\romannumeral0\xintiique {#4}{#2}}%
691 }%
692 \def\XINT_irr_loop_exitb #1#2%
693 {%
694   \expandafter\XINT_irr_finish\expandafter {#2}{#1}%
695 }%
696 \def\XINT_irr_finish #1#2#3{#3#1/#2}% changed in 1.08

```

## 8.25 \xintifInt

```

697 \def\xintifInt {\romannumeral0\xintifint }%
698 \def\xintifint #1{\expandafter\XINT_ifint\romannumeral0\xintrawwithzeros {#1}.}%
699 \def\XINT_ifint #1/#2.%
700 {%
701   \if 0\xintiiRem {#1}{#2}%
702   \expandafter\xint_stop_atfirstoftwo
703   \else
704   \expandafter\xint_stop_atsecondoftwo

```

```

705 \fi
706 }%
```

## 8.26 \xintIsInt

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

```

707 \def\xintIsInt {\romannumeral0\xintisint }%
708 \def\xintisint #1%
709 {\expandafter\XINT_ifint\romannumeral0\xintraewithzeros {#1}.10}%
```

## 8.27 \xintJrr

```

710 \def\xintJrr {\romannumeral0\xintjrr }%
711 \def\xintjrr #1%
712 {%
713 \expandafter\XINT_jrr_start\romannumeral0\xintraewithzeros {#1}\Z
714 }%
715 \def\XINT_jrr_start #1#2/#3\Z
716 {%
717 \if0\XINT_isOne {#3}\xint_afterfi
718 {\xint_UDsignfork
719 #1\XINT_jrr_negative
720 -{\XINT_jrr_nonneg #1}%
721 \krof}%
722 \else
723 \xint_afterfi{\XINT_jrr_denomisine #1}%
724 \fi
725 #2\Z {#3}%
726 }%
727 \def\XINT_jrr_denomisine #1\Z #2{ #1/1}% changed in 1.08
728 \def\XINT_jrr_negative #1\Z #2{\XINT_jrr_D #1\Z #2\Z -}%
729 \def\XINT_jrr_nonneg #1\Z #2{\XINT_jrr_D #1\Z #2\Z \space}%
730 \def\XINT_jrr_D #1#2\Z #3#4\Z
731 {%
732 \xint_UDzerosfork
733 #3#1\XINT_jrr_indeterminate
734 #30\XINT_jrr_divisionbyzero
735 #10\XINT_jrr_zero
736 00\XINT_jrr_loop_a
737 \krof
738 {#3#4}{#1#2}1001%
739 }%
740 \def\XINT_jrr_indeterminate #1#2#3#4#5#6#7%
741 {%
742 \XINT_signalcondition{DivisionUndefined}{indeterminate: 0/0}{0/1}%
743 }%
744 \def\XINT_jrr_divisionbyzero #1#2#3#4#5#6#7%
745 {%
746 \XINT_signalcondition{DivisionByZero}{Vanishing denominator: #7#2/0}{0/1}%
747 }%
748 \def\XINT_jrr_zero #1#2#3#4#5#6#7{ 0/1}% changed in 1.08
749 \def\XINT_jrr_loop_a #1#2%
750 {%
```

```

751 \expandafter\XINT_jrr_loop_b
752 \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
753 }%
754 \def\XINT_jrr_loop_b #1#2#3#4#5#6#7%
755 {%
756 \expandafter \XINT_jrr_loop_c \expandafter
757 {\romannumeral0\xintiadd{\XINT_mul_fork #4\xint:#1\xint:}{#6}}%
758 {\romannumeral0\xintiadd{\XINT_mul_fork #5\xint:#1\xint:}{#7}}%
759 {#2}{#3}{#4}{#5}%
760 }%
761 \def\XINT_jrr_loop_c #1#2%
762 {%
763 \expandafter \XINT_jrr_loop_d \expandafter{#2}{#1}%
764 }%
765 \def\XINT_jrr_loop_d #1#2#3#4%
766 {%
767 \XINT_jrr_loop_e #3\Z {#4}{#2}{#1}%
768 }%
769 \def\XINT_jrr_loop_e #1#2\Z
770 {%
771 \xint_gob_til_zero #1\XINT_jrr_loop_exit0\XINT_jrr_loop_a {#1#2}%
772 }%
773 \def\XINT_jrr_loop_exit0\XINT_jrr_loop_a #1#2#3#4#5#6%
774 {%
775 \XINT_irr_finish {#3}{#4}%
776 }%

```

## 8.28 \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.

```

777 \def\xintTFrac {\romannumeral0\xinttfrac }%
778 \def\xinttfrac #1{\expandafter\XINT_tfrac_fork\romannumeral0\xintrawwithzeros {#1}\Z }%
779 \def\XINT_tfrac_fork #1%
780 {%
781 \xint_UDzerominusfork
782 #1-\XINT_tfrac_zero
783 0#1{\xintiiopp\XINT_tfrac_P }%
784 0-{\XINT_tfrac_P #1}%
785 \krof
786 }%
787 \def\XINT_tfrac_zero #1\Z { 0/1[0]}%
788 \def\XINT_tfrac_P #1/#2\Z {\expandafter\XINT_rez_AB
789 \romannumeral0\xintiirem{#1}{#2}\Z {0}{#2}}%

```

## 8.29 \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.

```

790 \def\xintTrunc {\romannumeral0\xinttrunc}%
791 \def\xintiTrunc {\romannumeral0\xintitrunc}%
792 \def\xinttrunc #1{\expandafter\XINT_trunc\the\numexpr#1.\XINT_trunc_G}%
793 \def\xintitrunc #1{\expandafter\XINT_trunc\the\numexpr#1.\XINT_itrunc_G}%
794 \def\XINT_trunc #1.#2#3%
795 {%
796   \expandafter\XINT_trunc_a\romannumeral0\XINT_infrac{#3}#1.#2%
797 }%
798 \def\XINT_trunc_a #1#2#3#4.#5%
799 {%
800   \if0\XINT_Sgn#2\xint:\xint_dothis\XINT_trunc_zero\fi
801   \if1\XINT_is_One#3XY\xint_dothis\XINT_trunc_sp_b\fi
802   \xint_orthat\XINT_trunc_b #1+#4.{#2}{#3}#5#4.%
803 }%
804 \def\XINT_trunc_zero #1.#2.{ 0}%

805 \def\XINT_trunc_b {\expandafter\XINT_trunc_B\the\numexpr}%
806 \def\XINT_trunc_sp_b {\expandafter\XINT_trunc_sp_B\the\numexpr}%

807 \def\XINT_trunc_B #1%
808 {%
809   \xint_UDsignfork
810   #1\XINT_trunc_C
811   -\XINT_trunc_D
812   \krof #1%
813 }%

814 \def\XINT_trunc_sp_B #1%
815 {%
816   \xint_UDsignfork
817   #1\XINT_trunc_sp_C
818   -\XINT_trunc_sp_D
819   \krof #1%
820 }%

821 \def\XINT_trunc_C -#1.#2#3%
822 {%
823   \expandafter\XINT_trunc_CE
824   \romannumeral0\XINT_dsx_addzeros{#1}#3;.{#2}%
825 }%
826 \def\XINT_trunc_CE #1.#2{\XINT_trunc_E #2.{#1}}%

```

```

827 \def\XINT_trunc_sp_C -#1.#2#3{\XINT_trunc_sp_Ca #2.#1.}%
828 \def\XINT_trunc_sp_Ca #1%
829 {%
830     \xint_UDsignfork
831     #1{\XINT_trunc_sp_Cb -}%
832     -{\XINT_trunc_sp_Cb \space#1}%
833     \krof
834 }%
835 \def\XINT_trunc_sp_Cb #1#2.#3.%
836 {%
837     \expandafter\XINT_trunc_sp_Cc

838     \romannumeral0\expandafter\XINT_split_fromright_a
839     \the\numexpr#3-\numexpr\XINT_length_loop
840     #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
841     \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
842     \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
843     .#2\xint_bye2345678\xint_bye..#1%
844 }%

845 \def\XINT_trunc_sp_Cc #1%
846 {%
847     \if.#1\xint_dothis{\XINT_trunc_sp_Cd 0.}\fi
848     \xint_orthat {\XINT_trunc_sp_Cd #1}%
849 }%
850 \def\XINT_trunc_sp_Cd #1.#2.#3%
851 {%
852     \XINT_trunc_sp_F #3#1.%
853 }%
854 \def\XINT_trunc_D #1.#2%
855 {%
856     \expandafter\XINT_trunc_E
857     \romannumeral0\XINT_dsx_addzeros {#1}#2;.%
858 }%
859 \def\XINT_trunc_sp_D #1.#2#3%
860 {%
861     \expandafter\XINT_trunc_sp_E
862     \romannumeral0\XINT_dsx_addzeros {#1}#2;.%
863 }%
864 \def\XINT_trunc_E #1%
865 {%
866     \xint_UDsignfork
867     #1{\XINT_trunc_F -}%
868     -{\XINT_trunc_F \space#1}%
869     \krof
870 }%
871 \def\XINT_trunc_sp_E #1%
872 {%
873     \xint_UDsignfork
874     #1{\XINT_trunc_sp_F -}%
875     -{\XINT_trunc_sp_F\space#1}%
876     \krof

```



```

877 }%
878 \def\XINT_trunc_F #1#2.#3#4%
879   {\expandafter#4\romannumeral`&&\expandafter\xint_firstoftwo
880     \romannumeral0\XINT_div_prepare {#3}{#2}.#1}%
881 \def\XINT_trunc_sp_F #1#2.#3{#3#2.#1}%
882 \def\XINT_itrunc_G #1#2.#3#4.{\if10\xint_dothis{ 0}\fi\xint_orthat{#3#1}#2}%
883 \def\XINT_trunc_G #1.#2#3.%
884 {%
885   \expandafter\XINT_trunc_H
886   \the\numexpr\romannumeral0\xintlength {#1}-#3.#3.{#1}#2%
887 }%
888 \def\XINT_trunc_H #1.#2.%
889 {%
890   \ifnum #1 > \xint_c_
891     \xint_afterfi {\XINT_trunc_Ha {#2}}%
892   \else
893     \xint_afterfi {\XINT_trunc_Hb {-#1}}% -0,--1,--2, ....
894   \fi
895 }%
896 \def\XINT_trunc_Ha{\expandafter\XINT_trunc_Haa\romannumeral0\xintdecsplit}%
897 \def\XINT_trunc_Haa #1#2#3{#3#1.#2}%
898 \def\XINT_trunc_Hb #1#2#3%
899 {%
900   \expandafter #3\expandafter0\expandafter.%

901   \romannumeral\xintreplicate{#1}0#2%
902 }%

```

## 8.30 \xintTTrunc

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

```

903 \def\xintTTrunc {\romannumeral0\xintttrunc }%
904 \def\xintttrunc {\xintitrunc\xint_c_}%

```

## 8.31 \xintNum

```

905 \let\xintnum \xintttrunc

```

## 8.32 \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.

```

906 \def\xintRound {\romannumeral0\xintround }%
907 \def\xintiRound {\romannumeral0\xintiround }%
908 \def\xintround #1{\expandafter\XINT_round\the\numexpr #1.\XINT_round_A}%
909 \def\xintiround #1{\expandafter\XINT_round\the\numexpr #1.\XINT_iround_A}%
910 \def\XINT_round #1.{\expandafter\XINT_round_aa\the\numexpr #1+\xint_c_i.#1.}%

```

```

911 \def\XINT_round_aa #1.#2.#3#4%
912 {%
913     \expandafter\XINT_round_a\romannumeral0\XINT_infrac{#4}#1.#3#2.%
914 }%
915 \def\XINT_round_a #1#2#3#4.%
916 {%
917     \if0\XINT_Sgn#2\xint:\xint_dothis\XINT_trunc_zero\fi
918     \if1\XINT_is_One#3XY\xint_dothis\XINT_trunc_sp_b\fi
919     \xint_orthat\XINT_trunc_b #1+#4.{#2}{#3}%
920 }%
921 \def\XINT_round_A{\expandafter\XINT_trunc_G\romannumeral0\XINT_round_B}%
922 \def\XINT_iround_A{\expandafter\XINT_itrunc_G\romannumeral0\XINT_round_B}%
923 \def\XINT_round_B #1.%
924     {\XINT_dsrr #1\xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax.}%

```

### 8.33 \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.

```

925 \def\xintXTrunc #1%#2%
926 {%
927     \expandafter\XINT_xtrunc_a
928     \the\numexpr #1\expandafter.\romannumeral0\xintraw
929 }%
930 \def\XINT_xtrunc_a #1.% ?? faire autre chose
931 {%
932     \expandafter\XINT_xtrunc_b\the\numexpr\ifnum#1<\xint_c_i \xint_c_i-\fi #1.%
933 }%

934 \def\XINT_xtrunc_b #1.#2{\XINT_xtrunc_c #2{#1}}%

935 \def\XINT_xtrunc_c #1%
936 {%
937     \xint_UDzerominusfork
938     #1-\XINT_xtrunc_zero
939     0#1{-\XINT_xtrunc_d {}}%
940     0-{\XINT_xtrunc_d #1}%
941     \krof
942 }%[
943 \def\XINT_xtrunc_zero #1#2]{0.\romannumeral\xintreplicate{#1}0}%

944 \def\XINT_xtrunc_d #1#2#3/#4[#5]%
945 {%
946     \XINT_xtrunc_prepare_a#4\R\R\R\R\R\R\R {10}0000001\W
947     !{#4};{#5}{#2}{#1#3}%

```

```

948 }%
949 \def\XINT_xtrunc_prepare_a #1#2#3#4#5#6#7#8#9%
950 {%
951     \xint_gob_til_R #9\XINT_xtrunc_prepare_small\R
952     \XINT_xtrunc_prepare_b #9%
953 }%
954 \def\XINT_xtrunc_prepare_small\R #1!#2;%
955 {%
956     \ifcase #2
957     \or\expandafter\XINT_xtrunc_BisOne
958     \or\expandafter\XINT_xtrunc_BisTwo
959     \or
960     \or\expandafter\XINT_xtrunc_BisFour
961     \or\expandafter\XINT_xtrunc_BisFive
962     \or
963     \or
964     \or\expandafter\XINT_xtrunc_BisEight
965     \fi\XINT_xtrunc_BisSmall {#2}%
966 }%

967 \def\XINT_xtrunc_BisOne\XINT_xtrunc_BisSmall #1#2#3#4%
968     {\XINT_xtrunc_sp_e {#2}{#4}{#3}}%
969 \def\XINT_xtrunc_BisTwo\XINT_xtrunc_BisSmall #1#2#3#4%
970 {%
971     \expandafter\XINT_xtrunc_sp_e\expandafter
972     {\the\numexpr #2-\xint_c_i\expandafter}\expandafter
973     {\romannumeral0\xintiimul 5{#4}{#3}}%
974 }%
975 \def\XINT_xtrunc_BisFour\XINT_xtrunc_BisSmall #1#2#3#4%
976 {%
977     \expandafter\XINT_xtrunc_sp_e\expandafter
978     {\the\numexpr #2-\xint_c_ii\expandafter}\expandafter
979     {\romannumeral0\xintiimul {25}{#4}{#3}}%
980 }%
981 \def\XINT_xtrunc_BisFive\XINT_xtrunc_BisSmall #1#2#3#4%
982 {%
983     \expandafter\XINT_xtrunc_sp_e\expandafter
984     {\the\numexpr #2-\xint_c_i\expandafter}\expandafter
985     {\romannumeral0\xintdouble {#4}{#3}}%
986 }%
987 \def\XINT_xtrunc_BisEight\XINT_xtrunc_BisSmall #1#2#3#4%
988 {%
989     \expandafter\XINT_xtrunc_sp_e\expandafter
990     {\the\numexpr #2-\xint_c_iii\expandafter}\expandafter
991     {\romannumeral0\xintiimul {125}{#4}{#3}}%
992 }%
993 \def\XINT_xtrunc_BisSmall #1%
994 {%
995     \expandafter\XINT_xtrunc_e\expandafter
996     {\expandafter\XINT_xtrunc_small_a
997     \the\numexpr #1\-\xint_c_ii\expandafter
998     .\the\numexpr \xint_c_x^viii+1!}%

```

999 }%

1000 \def\XINT\_xtrunc\_small\_a #1.#2!#3%

1001 {%

1002 \expandafter\XINT\_div\_small\_b\the\numexpr #1\expandafter

1003 \xint:\the\numexpr #2\expandafter!%

1004 \romannumeral0\XINT\_div\_small\_ba #3\R\R\R\R\R\R\R\{10}0000001\W

1005 #3\XINT\_sepbyviii\_Z\_end 2345678\relax

1006 }%

1007 \def\XINT\_xtrunc\_prepare\_b

1008 {\expandafter\XINT\_xtrunc\_prepare\_c\romannumeral0\XINT\_zeroes\_forviii }%

1009 \def\XINT\_xtrunc\_prepare\_c #1!%

1010 {%

1011 \XINT\_xtrunc\_prepare\_d #1.00000000!{#1}%

1012 }%

1013 \def\XINT\_xtrunc\_prepare\_d #1#2#3#4#5#6#7#8#9%

1014 {%

1015 \expandafter\XINT\_xtrunc\_prepare\_e

1016 \xint\_gob\_til\_dot #1#2#3#4#5#6#7#8#9!%

1017 }%

1018 \def\XINT\_xtrunc\_prepare\_e #1!#2!#3#4%

1019 {%

1020 \XINT\_xtrunc\_prepare\_f #4#3\X {#1}{#3}%

1021 }%

1022 \def\XINT\_xtrunc\_prepare\_f #1#2#3#4#5#6#7#8#9\X

1023 {%

1024 \expandafter\XINT\_xtrunc\_prepare\_g\expandafter

1025 \XINT\_div\_prepare\_g

1026 \the\numexpr #1#2#3#4#5#6#7#8+\xint\_c\_i\expandafter

1027 \xint:\the\numexpr (#1#2#3#4#5#6#7#8+\xint\_c\_i)/\xint\_c\_ii\expandafter

1028 \xint:\the\numexpr #1#2#3#4#5#6#7#8\expandafter

1029 \xint:\romannumeral0\XINT\_sepandrev\_andcount

1030 #1#2#3#4#5#6#7#8#9\XINT\_rsepbyviii\_end\_A 2345678%

1031 \XINT\_rsepbyviii\_end\_B 2345678\relax\xint\_c\_ii\xint\_c\_i

1032 \R\xint:\xint\_c\_xii \R\xint:\xint\_c\_x \R\xint:\xint\_c\_viii \R\xint:\xint\_c\_vi

1033 \R\xint:\xint\_c\_iv \R\xint:\xint\_c\_ii \R\xint:\xint\_c\_W

1034 \X

1035 }%

1036 \def\XINT\_xtrunc\_prepare\_g #1;{\XINT\_xtrunc\_e {#1}}%

1037 \def\XINT\_xtrunc\_e #1#2%

1038 {%

1039 \ifnum #2<\xint\_c\_

1040 \expandafter\XINT\_xtrunc\_I

1041 \else

1042 \expandafter\XINT\_xtrunc\_II

1043 \fi #2\xint:{#1}%

1044 }%

```

1045 \def\XINT_xtrunc_I -#1\xint:#2#3#4%
1046 {%
1047     \expandafter\XINT_xtrunc_I_a\romannumeral0#2{#4}{#2}{#1}{#3}%
1048 }%

1049 \def\XINT_xtrunc_I_a #1#2#3#4#5%
1050 {%
1051     \expandafter\XINT_xtrunc_I_b\the\numexpr #4-#5\xint:#4\xint:{#5}{#2}{#3}{#1}%
1052 }%

1053 \def\XINT_xtrunc_I_b #1%
1054 {%
1055     \xint_UDsignfork
1056     #1\XINT_xtrunc_IA_c
1057     -\XINT_xtrunc_IB_c
1058     \krof #1%
1059 }%

1060 \def\XINT_xtrunc_IA_c -#1\xint:#2\xint:#3#4#5#6%
1061 {%
1062     \expandafter\XINT_xtrunc_IA_d
1063     \the\numexpr#2-\xintlength{#6}\xint:{#6}%
1064     \expandafter\XINT_xtrunc_IA_xd
1065     \the\numexpr (#1+\xint_c_ii^v)/\xint_c_ii^vi-\xint_c_i\xint:#1\xint:{#5}{#4}%
1066 }%

1067 \def\XINT_xtrunc_IA_d #1%
1068 {%
1069     \xint_UDsignfork
1070     #1\XINT_xtrunc_IAA_e
1071     -\XINT_xtrunc_IAB_e
1072     \krof #1%
1073 }%

1074 \def\XINT_xtrunc_IAA_e -#1\xint:#2%
1075 {%
1076     \romannumeral0\XINT_split_fromleft
1077     #1.#2\xint_gobble_i\xint_bye2345678\xint_bye..%
1078 }%

1079 \def\XINT_xtrunc_IAB_e #1\xint:#2%
1080 {%
1081     0.\romannumeral\XINT_rep#1\endcsname0#2%
1082 }%

1083 \def\XINT_xtrunc_IA_xd #1\xint:#2\xint:%
1084 {%
1085     \expandafter\XINT_xtrunc_IA_xe\the\numexpr #2-\xint_c_ii^vi*#1\xint:#1\xint:%
1086 }%

```

xintfrac

```

1134 \def\XINT_xtrunc_transition
1135   \expandafter\XINT_xtrunc_loop_a\the\numexpr #1\xint:#2#3#4%
1136 {%
1137   \ifnum #4=\xint_c_ \expandafter\xint_gobble_vi\fi
1138   \expandafter\XINT_xtrunc_finish\expandafter
1139   {\romannumeral0\XINT_dsx_addzeros{#4}#2;}{#3}{#4}%
1140 }%
1141 \def\XINT_xtrunc_finish #1#2%
1142 {%
1143   \expandafter\XINT_xtrunc_finish_a\romannumeral0#2{#1}%
1144 }%
1145 \def\XINT_xtrunc_finish_a #1#2#3%
1146 {%
1147   \romannumeral\xintreplicate{#3-\xintLength{#1}}0#1%
1148 }%

1149 \def\XINT_xtrunc_sp_e #1%
1150 {%
1151   \ifnum #1<\xint_c_
1152     \expandafter\XINT_xtrunc_sp_I
1153   \else
1154     \expandafter\XINT_xtrunc_sp_II
1155   \fi #1\xint:%
1156 }%

1157 \def\XINT_xtrunc_sp_I -#1\xint:#2#3%
1158 {%
1159   \expandafter\XINT_xtrunc_sp_I_a\the\numexpr #1-#3\xint:#1\xint:{#3}{#2}%
1160 }%

1161 \def\XINT_xtrunc_sp_I_a #1%
1162 {%
1163   \xint_UDsignfork
1164   #1\XINT_xtrunc_sp_IA_b
1165   -\XINT_xtrunc_sp_IB_b
1166   \krof #1%
1167 }%

1168 \def\XINT_xtrunc_sp_IA_b -#1\xint:#2\xint:#3#4%
1169 {%
1170   \expandafter\XINT_xtrunc_sp_IA_c
1171   \the\numexpr#2-\xintLength{#4}\xint:{#4}\romannumeral\XINT_rep#1\endcsname0%
1172 }%

1173 \def\XINT_xtrunc_sp_IA_c #1%
1174 {%
1175   \xint_UDsignfork
1176   #1\XINT_xtrunc_sp_IAA
1177   -\XINT_xtrunc_sp_IAB
1178   \krof #1%
1179 }%

```

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices*

```

1180 \def\XINT_xtrunc_sp_IAA -#1\xint:#2%
1181 {%
1182     \romannumeral0\XINT_split_fromleft
1183     #1.#2\xint_gobble_i\xint_bye2345678\xint_bye.>%
1184 }%

1185 \def\XINT_xtrunc_sp_IAB #1\xint:#2%
1186 {%
1187     0.\romannumeral\XINT_rep#1\endcsname0#2%
1188 }%

1189 \def\XINT_xtrunc_sp_IB_b #1\xint:#2\xint:#3#4%
1190 {%
1191     \expandafter\XINT_xtrunc_sp_IB_c
1192     \romannumeral0\XINT_split_xfork #1.#4\xint_bye2345678\xint_bye..{#3}%
1193 }%

1194 \def\XINT_xtrunc_sp_IB_c #1.#2.#3%
1195 {%
1196     \expandafter\XINT_xtrunc_sp_IA_c\the\numexpr#3-\xintLength {#1}\xint:{#1}%
1197 }%

1198 \def\XINT_xtrunc_sp_II #1\xint:#2#3%
1199 {%
1200     #2\romannumeral\XINT_rep#1\endcsname0.\romannumeral\XINT_rep#3\endcsname0%
1201 }%

```

### 8.34 \xintDigits

The `\mathchardef` used to be called `\XINT_digits`, but for reasons originating in `\xintNewExpr` (and now obsolete), release 1.09a uses `\XINTdigits` without underscore.

```

1202 \mathchardef\XINTdigits 16
1203 \def\xintDigits #1#2%
1204     {\afterassignment \xint_gobble_i \mathchardef\XINTdigits=}%
1205 \def\xinttheDigits {\number\XINTdigits }%

```

### 8.35 \xintAdd

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

```

1206 \def\xintAdd {\romannumeral0\xintadd }%
1207 \def\xintadd #1{\expandafter\XINT_fadd\romannumeral0\xintra #1}%
1208 \def\XINT_fadd #1{\xint_gob_til_zero #1\XINT_fadd_Azero 0\XINT_fadd_a #1}%
1209 \def\XINT_fadd_Azero #1]{\xintra }%
1210 \def\XINT_fadd_a #1/#2[#3]#4%
1211     {\expandafter\XINT_fadd_b\romannumeral0\xintra {#4}{#3}{#1}{#2}}%
1212 \def\XINT_fadd_b #1{\xint_gob_til_zero #1\XINT_fadd_Bzero 0\XINT_fadd_c #1}%
1213 \def\XINT_fadd_Bzero #1]#2#3#4{ #3/#4[#2]}%
1214 \def\XINT_fadd_c #1/#2[#3]#4%
1215 {%

```



```

1216 \expandafter\XINT_fadd_Aa\the\numexpr #4-#3.{#3}{#4}{#1}{#2}%
1217 }%
1218 \def\XINT_fadd_Aa #1%
1219 {%
1220 \xint_UDzerominusfork
1221 #1-\XINT_fadd_B
1222 0#1\XINT_fadd_Bb
1223 0-\XINT_fadd_Ba
1224 \krof #1%
1225 }%
1226 \def\XINT_fadd_B #1.#2#3#4#5#6#7{\XINT_fadd_C {#4}{#5}{#7}{#6}[#3]}%
1227 \def\XINT_fadd_Ba #1.#2#3#4#5#6#7%
1228 {%
1229 \expandafter\XINT_fadd_C\expandafter
1230 {\romannumeral0\XINT_dsx_addzeros {#1}#6;}%
1231 {#7}{#5}{#4}[#2]}%
1232 }%
1233 \def\XINT_fadd_Bb -#1.#2#3#4#5#6#7%
1234 {%
1235 \expandafter\XINT_fadd_C\expandafter
1236 {\romannumeral0\XINT_dsx_addzeros {#1}#4;}%
1237 {#5}{#7}{#6}[#3]}%
1238 }%
1239 \def\XINT_fadd_iszero #1[#2]{ 0/1[0]}% ou [#2] origine1?
1240 \def\XINT_fadd_C #1#2#3%
1241 {%
1242 \expandafter\XINT_fadd_D_b
1243 \romannumeral0\XINT_div_prepare{#2}{#3}{#2}{#2}{#3}{#1}%
1244 }%

```

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.

```

1245 \def\XINT_fadd_D_a #1#2%
1246 {%
1247 \expandafter\XINT_fadd_D_b
1248 \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
1249 }%
1250 \def\XINT_fadd_D_b #1#2{\XINT_fadd_D_c #2\Z}%
1251 \def\XINT_fadd_D_c #1#2\Z
1252 {%
1253 \xint_gob_til_zero #1\XINT_fadd_D_exit0\XINT_fadd_D_a {#1#2}%
1254 }%
1255 \def\XINT_fadd_D_exit0\XINT_fadd_D_a #1#2#3%
1256 {%
1257 \expandafter\XINT_fadd_E
1258 \romannumeral0\xintiipro {#3}{#2}.{#2}%
1259 }%
1260 \def\XINT_fadd_E #1.#2#3%
1261 {%
1262 \expandafter\XINT_fadd_F
1263 \romannumeral0\xintiimul{#1}{#3}.{\xintiipro{#3}{#2}}{#1}%
1264 }%

```

```

1265 \def\XINT_fadd_F #1.#2#3#4#5%
1266 {%
1267     \expandafter\XINT_fadd_G
1268     \romannumeral0\xintiiadd{\xintiiMul{#2}{#4}}{\xintiiMul{#3}{#5}}/#1%
1269 }%
1270 \def\XINT_fadd_G #1{%
1271 \def\XINT_fadd_G ##1{\if0##1\expandafter\XINT_fadd_iszero\fi#1##1}%
1272 }\XINT_fadd_G{ }%

```

## 8.36 \xintSub

Since 1.3 will use least common multiple of denominators.

```

1273 \def\xintSub {\romannumeral0\xintsub }%
1274 \def\xintsub #1{\expandafter\XINT_fsub\romannumeral0\xintra {#1}}%
1275 \def\XINT_fsub #1{\xint_gob_til_zero #1\XINT_fsub_Azero 0\XINT_fsub_a #1}%
1276 \def\XINT_fsub_Azero #1]{\xintopp }%
1277 \def\XINT_fsub_a #1/#2[#3]#4%
1278     {\expandafter\XINT_fsub_b\romannumeral0\xintra {#4}{#3}{#1}{#2}}%
1279 \def\XINT_fsub_b #1{\xint_UDzerominusfork
1280     #1-\XINT_fadd_Bzero
1281     0#1\XINT_fadd_c
1282     0-\XINT_fadd_c -#1}%
1283 \krof }%

```

## 8.37 \xintSum

There was (not documented anymore since 1.09d, 2013/10/22) a macro \xintSumExpr, but it has been deleted at 1.2l.

Empty items are not accepted by this macro.

```

1284 \def\xintSum {\romannumeral0\xintsum }%
1285 \def\xintsum #1{\expandafter\XINT_fsumexpr\romannumeral`&&@#1\xint:}%
1286 \def\XINT_fsumexpr {\XINT_fsum_loop_a {0/1[0]}}%
1287 \def\XINT_fsum_loop_a #1#2%
1288 {%
1289     \expandafter\XINT_fsum_loop_b \romannumeral`&&@#2\xint:{#1}%
1290 }%
1291 \def\XINT_fsum_loop_b #1%
1292 {%
1293     \xint_gob_til_xint: #1\XINT_fsum_finished\xint:\XINT_fsum_loop_c #1%
1294 }%
1295 \def\XINT_fsum_loop_c #1\xint:#2%
1296 {%
1297     \expandafter\XINT_fsum_loop_a\expandafter{\romannumeral0\xintadd {#2}{#1}}%
1298 }%
1299 \def\XINT_fsum_finished #1\xint:\xint:#2{ #2}%

```

## 8.38 \xintMul

```

1300 \def\xintMul {\romannumeral0\xintmul }%
1301 \def\xintmul #1{\expandafter\XINT_fmula\romannumeral0\xintra {#1}.}%
1302 \def\XINT_fmula #1{\xint_gob_til_zero #1\XINT_fmula_zero 0\XINT_fmula_a #1}%

```

```

1303 \def\XINT_fmula #1[#2].#3%
1304   {\expandafter\XINT_fmula_b\romannumeral0\xintra {#3}#1[#2.]}%
1305 \def\XINT_fmula_b #1{\xint_gob_til_zero #1\XINT_fmula_zero 0\XINT_fmula_c #1}%
1306 \def\XINT_fmula_c #1/#2[#3]#4/#5[#6.]%
1307 {%
1308   \expandafter\XINT_fmula_d
1309   \expandafter{\the\numexpr #3+#6\expandafter}%
1310   \expandafter{\romannumeral0\xintiimul {#5}{#2}}%
1311   {\romannumeral0\xintiimul {#4}{#1}}%
1312 }%
1313 \def\XINT_fmula_d #1#2#3%
1314 {%
1315   \expandafter \XINT_fmula_e \expandafter{#3}{#1}{#2}%
1316 }%
1317 \def\XINT_fmula_e #1#2{\XINT_outfrac {#2}{#1}}%
1318 \def\XINT_fmula_zero #1.#2{ 0/1[0]}%

```

## 8.39 \xintSqr

### 1.1 modifs comme xintMul.

```

1319 \def\xintSqr {\romannumeral0\xintsqr}%
1320 \def\xintsqr #1{\expandafter\XINT_fsqr\romannumeral0\xintra {#1}}%
1321 \def\XINT_fsqr #1{\xint_gob_til_zero #1\XINT_fsqr_zero 0\XINT_fsqr_a #1}%
1322 \def\XINT_fsqr_a #1/#2[#3]%
1323 {%
1324   \expandafter\XINT_fsqr_b
1325   \expandafter{\the\numexpr #3+#3\expandafter}%
1326   \expandafter{\romannumeral0\xintiisqr {#2}}%
1327   {\romannumeral0\xintiisqr {#1}}%
1328 }%
1329 \def\XINT_fsqr_b #1#2#3{\expandafter \XINT_fmula_e \expandafter{#3}{#1}{#2}}%
1330 \def\XINT_fsqr_zero #1{\ 0/1[0]}%

```

## 8.40 \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.

```

1331 \def\xintipow #1#2%
1332 {%
1333   \expandafter\xint_pow\the\numexpr \xintNum{#2}\expandafter
1334   .\romannumeral0\xintnum{#1}\xint:
1335 }%
1336 \def\xintPow {\romannumeral0\xintpow}%
1337 \def\xintpow #1%
1338 {%
1339   \expandafter\XINT_fpow\expandafter {\romannumeral0\XINT_infrac {#1}}%
1340 }%
1341 \def\XINT_fpow #1#2%
1342 {%

```

```

1343 \expandafter\XINT_fpow_fork\the\numexpr \xintNum{#2}\relax\Z #1%
1344 }%
1345 \def\XINT_fpow_fork #1#2\Z
1346 {%
1347 \xint_UDzerominusfork
1348 #1-\XINT_fpow_zero
1349 0#1\XINT_fpow_neg
1350 0-\XINT_fpow_pos #1}%
1351 \krof
1352 {#2}%
1353 }%
1354 \def\XINT_fpow_zero #1#2#3#4{ 1/1[0]}%
1355 \def\XINT_fpow_pos #1#2#3#4#5%
1356 {%
1357 \expandafter\XINT_fpow_pos_A\expandafter
1358 {the\numexpr #1#2*#3\expandafter}\expandafter
1359 {\romannumeral0\xintiipow {#5}{#1#2}}%
1360 {\romannumeral0\xintiipow {#4}{#1#2}}%
1361 }%
1362 \def\XINT_fpow_neg #1#2#3#4%
1363 {%
1364 \expandafter\XINT_fpow_pos_A\expandafter
1365 {the\numexpr -#1*#2\expandafter}\expandafter
1366 {\romannumeral0\xintiipow {#3}{#1}}%
1367 {\romannumeral0\xintiipow {#4}{#1}}%
1368 }%
1369 \def\XINT_fpow_pos_A #1#2#3%
1370 {%
1371 \expandafter\XINT_fpow_pos_B\expandafter {#3}{#1}{#2}%
1372 }%
1373 \def\XINT_fpow_pos_B #1#2{\XINT_outfrac {#2}{#1}}%

```

## 8.41 \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.

```

1374 \def\xintFac {\romannumeral0\xintfac}%
1375 \def\xintfac #1{\expandafter\XINT_fac_fork\the\numexpr\xintNum{#1}.[0]}%

```

## 8.42 \xintBinomial

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

```

1376 \def\xintBinomial {\romannumeral0\xintbinomial}%
1377 \def\xintbinomial #1#2%
1378 {%
1379 \expandafter\XINT_binom_pre
1380 the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.[0]%
1381 }%

```

## 8.43 \xintPfactorial

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

```

1382 \def\xintpfactorial #1#2%
1383 {%
1384     \expandafter\XINT_pfac_fork
1385     \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.%
1386 }%
1387 \def\xintPfactorial {\romannumeral0\xintpfactorial}%
1388 \def\xintpfactorial #1#2%
1389 {%
1390     \expandafter\XINT_pfac_fork
1391     \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.[0]%
1392 }%

```

## 8.44 \xintPrd

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

```

1393 \def\xintPrd {\romannumeral0\xintprd}%
1394 \def\xintprd #1{\expandafter\XINT_fprdexpr \romannumeral`&&@#1\xint:}%
1395 \def\XINT_fprdexpr {\XINT_fprod_loop_a {1/1[0]}}%
1396 \def\XINT_fprod_loop_a #1#2%
1397 {%
1398     \expandafter\XINT_fprod_loop_b \romannumeral`&&@#2\xint:{#1}%
1399 }%
1400 \def\XINT_fprod_loop_b #1%
1401 {%
1402     \xint_gob_til_xint: #1\XINT_fprod_finished\xint:\XINT_fprod_loop_c #1%
1403 }%
1404 \def\XINT_fprod_loop_c #1\xint:#2%
1405 {%
1406     \expandafter\XINT_fprod_loop_a\expandafter{\romannumeral0\xintmul {#1}{#2}}%
1407 }%
1408 \def\XINT_fprod_finished#1\xint:\xint:#2{ #2}%

```

## 8.45 \xintDiv

```

1409 \def\xintDiv {\romannumeral0\xintdiv}%
1410 \def\xintdiv #1%
1411 {%
1412     \expandafter\XINT_fdiv\expandafter {\romannumeral0\XINT_infrac {#1}}%
1413 }%
1414 \def\XINT_fdiv #1#2%
1415     {\expandafter\XINT_fdiv_A\romannumeral0\XINT_infrac {#2}#1}%
1416 \def\XINT_fdiv_A #1#2#3#4#5#6%
1417 {%
1418     \expandafter\XINT_fdiv_B
1419     \expandafter{\the\numexpr #4-#1\expandafter}%
1420     \expandafter{\romannumeral0\xintiimul {#2}{#6}}%
1421     {\romannumeral0\xintiimul {#3}{#5}}%

```

```

1422 }%
1423 \def\XINT_fdiv_B #1#2#3%
1424 {%
1425     \expandafter\XINT_fdiv_C
1426     \expandafter{#3}{#1}{#2}%
1427 }%
1428 \def\XINT_fdiv_C #1#2{\XINT_outfrac {#2}{#1}}%

```

## 8.46 \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.

```

1429 \def\xintDivFloor    {\romannumeral0\xintdivfloor }%
1430 \def\xintdivfloor #1#2{\xintifloor{\xintDiv {#1}{#2}}}%

```

## 8.47 \xintDivTrunc

1.1. \xintttrunc rather than \xintitrunc0 in 1.1a

```

1431 \def\xintDivTrunc    {\romannumeral0\xintdivtrunc }%
1432 \def\xintdivtrunc #1#2{\xintttrunc {\xintDiv {#1}{#2}}}%

```

## 8.48 \xintDivRound

1.1

```

1433 \def\xintDivRound    {\romannumeral0\xintdivround }%
1434 \def\xintdivround #1#2{\xintiround 0{\xintDiv {#1}{#2}}}%

```

## 8.49 \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).

```

1435 \def\xintModTrunc {\romannumeral0\xintmodtrunc }%
1436 \def\xintmodtrunc #1{\expandafter\XINT_modtrunc_a\romannumeral0\xintra{#1}.}%
1437 \def\XINT_modtrunc_a #1#2.#3%
1438     {\expandafter\XINT_modtrunc_b\expandafter #1\romannumeral0\xintra{#3}#2.}%
1439 \def\XINT_modtrunc_b #1#2% #1 de A, #2 de B.
1440 {%
1441     \if0#2\xint_dothis{\XINT_modtrunc_divbyzero #1#2}\fi
1442     \if0#1\xint_dothis\XINT_modtrunc_aiszero\fi
1443     \if-#2\xint_dothis{\XINT_modtrunc_bneg #1}\fi
1444     \xint_orthat{\XINT_modtrunc_bpos #1#2}%
1445 }%
1446 \def\XINT_modtrunc_divbyzero #1#2[#3]#4.%
1447 {%
1448     \XINT_signalcondition{DivisionByZero}{Division by #2[#3] of #1#4}{0/1[0]}%
1449 }%
1450 \def\XINT_modtrunc_aiszero #1.{ 0/1[0]}%

```

```

1451 \def\XINT_modtrunc_bneg #1%
1452 {%
1453   \xint_UDsignfork
1454     #1{\xintiiopp\XINT_modtrunc_pos {}}%
1455     -{\XINT_modtrunc_pos #1}%
1456   \krof
1457 }%
1458 \def\XINT_modtrunc_bpos #1%
1459 {%
1460   \xint_UDsignfork
1461     #1{\xintiiopp\XINT_modtrunc_pos {}}%
1462     -{\XINT_modtrunc_pos #1}%
1463   \krof
1464 }%

1465 \def\XINT_modtrunc_pos #1#2/#3[#4]#5/#6[#7].%
1466 {%
1467   \expandafter\XINT_modtrunc_pos_a
1468   \the\numexprifnum#7>#4 #4\else #7\fi\expandafter.%
1469   \romannumeral0\expandafter\XINT_mod_D_b
1470   \romannumeral0\XINT_div_prepare{#3}{#6}{#3}{#3}{#6}%
1471   {#1#5}{#7-#4}{#2}{#4-#7}%
1472 }%
1473 \def\XINT_modtrunc_pos_a #1.#2#3#4{\xintiirem {#3}{#4}/#2[#1]}%

```

## 8.50 \xintDivMod

1.2p. `\xintDivMod{q1}{q2}` outputs `{floor(q1/q2)}{q1 - q2*floor(q1/q2)}`. Attention that it relies on `\xintiiE{x}{e}` returning `x` if `e < 0`.

Modified (like `\xintAdd` and `\xintSub`) at 1.3 to use a l.c.m for final denominator of the "mod" part.

```

1474 \def\xintDivMod {\romannumeral0\xintdivmod }%
1475 \def\xintdivmod #1{\expandafter\XINT_divmod_a\romannumeral0\xintra{#1}.}%
1476 \def\XINT_divmod_a #1#2.#3%
1477   {\expandafter\XINT_divmod_b\expandafter #1\romannumeral0\xintra{#3}#2.}%
1478 \def\XINT_divmod_b #1#2% #1 de A, #2 de B.
1479 {%
1480   \if0#2\xint_dothis{\XINT_divmod_divbyzero #1#2}\fi
1481   \if0#1\xint_dothis\XINT_divmod_aiszero\fi
1482   \if-#2\xint_dothis{\XINT_divmod_bneg #1}\fi
1483   \xint_orthat{\XINT_divmod_bpos #1#2}%
1484 }%
1485 \def\XINT_divmod_divbyzero #1#2[#3]#4.%
1486 {%
1487   \XINT_signalcondition{DivisionByZero}{Division by #2[#3] of #1#4}{}%
1488   {{0}{0/1[0]}}% à revoir...
1489 }%
1490 \def\XINT_divmod_aiszero #1.{{0}{0/1[0]}}%
1491 \def\XINT_divmod_bneg #1% f // -g = (-f) // g, f % -g = -((-f) % g)

```

```

1492 {%
1493   \expandafter\XINT_divmod_bneg_finish
1494   \romannumeral0\xint_UDsignfork
1495     #1{\XINT_divmod_bpos {}}%
1496     -{\XINT_divmod_bpos {-#1}}%
1497   \krof
1498 }%
1499 \def\XINT_divmod_bneg_finish#1#2%
1500 {%
1501   \expandafter\xint_exchangetwo_keepbraces\expandafter
1502   {\romannumeral0\xintiiopt#2}{#1}%
1503 }%
1504 \def\XINT_divmod_bpos #1#2/#3[#4]#5/#6[#7].%
1505 {%
1506   \expandafter\XINT_divmod_bpos_a
1507   \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.%
1508   \romannumeral0\expandafter\XINT_mod_D_b
1509   \romannumeral0\XINT_div_prepare{#3}{#6}{#3}{#3}{#6}%
1510   {#1#5}{#7-#4}{#2}{#4-#7}%
1511 }%
1512 \def\XINT_divmod_bpos_a #1.#2#3#4%
1513 {%
1514   \expandafter\XINT_divmod_bpos_finish
1515   \romannumeral0\xintiidivision{#3}{#4}{/#2[#1]}%
1516 }%
1517 \def\XINT_divmod_bpos_finish #1#2#3{{#1}{#2#3}}%

```

## 8.51 \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.

```

1518 \def\xintMod {\romannumeral0\xintmod}%
1519 \def\xintmod #1{\expandafter\XINT_mod_a\romannumeral0\xintraw{#1}.}%
1520 \def\XINT_mod_a #1#2.#3%
1521   {\expandafter\XINT_mod_b\expandafter #1\romannumeral0\xintraw{#3}#2.}%
1522 \def\XINT_mod_b #1#2% #1 de A, #2 de B.
1523 {%
1524   \if0#2\xint_dothis{\XINT_mod_divbyzero #1#2}\fi
1525   \if0#1\xint_dothis\XINT_mod_aiszero\fi
1526   \if-#2\xint_dothis{\XINT_mod_bneg #1}\fi
1527   \xint_orthat{\XINT_mod_bpos #1#2}%
1528 }%

```

Attention to not move `ModTrunc` code beyond that point.

```

1529 \let\XINT_mod_divbyzero\XINT_modtrunc_divbyzero
1530 \let\XINT_mod_aiszero \XINT_modtrunc_aiszero
1531 \def\XINT_mod_bneg #1% f % -g = -((-f) % g), for g > 0
1532 {%
1533   \xintiiopt\xint_UDsignfork

```



```

1534      #1{\XINT_mod_bpos {}}%
1535      -{\XINT_mod_bpos {-#1}}%
1536      \krof
1537 }%
1538 \def\XINT_mod_bpos #1#2/#3[#4]#5/#6[#7].%
1539 {%
1540     \expandafter\XINT_mod_bpos_a
1541     \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.%
1542     \romannumeral0\expandafter\XINT_mod_D_b
1543     \romannumeral0\XINT_div_prepare{#3}{#6}{#3}{#3}{#6}%
1544     {#1#5}{#7-#4}{#2}{#4-#7}%
1545 }%
1546 \def\XINT_mod_D_a #1#2%
1547 {%
1548     \expandafter\XINT_mod_D_b
1549     \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
1550 }%
1551 \def\XINT_mod_D_b #1#2{\XINT_mod_D_c #2\Z}%
1552 \def\XINT_mod_D_c #1#2\Z
1553 {%
1554     \xint_gob_til_zero #1\XINT_mod_D_exit0\XINT_mod_D_a {#1#2}%
1555 }%
1556 \def\XINT_mod_D_exit0\XINT_mod_D_a #1#2#3%
1557 {%
1558     \expandafter\XINT_mod_E
1559     \romannumeral0\xintiipro {#3}{#2}.{#2}%
1560 }%
1561 \def\XINT_mod_E #1.#2#3%
1562 {%
1563     \expandafter\XINT_mod_F
1564     \romannumeral0\xintiimul{#1}{#3}.{\xintiiQuo{#3}{#2}}{#1}%
1565 }%
1566 \def\XINT_mod_F #1.#2#3#4#5#6#7%
1567 {%
1568     {#1}{\xintiiE{\xintiiMul{#4}{#3}}{#5}}%
1569     {\xintiiE{\xintiiMul{#6}{#2}}{#7}}%
1570 }%
1571 \def\XINT_mod_bpos_a #1.#2#3#4{\xintiiirem {#3}{#4}/#2[#1]}%

```

## 8.52 \xintIsOne

New with 1.09a. Could be more efficient. For fractions with big powers of tens, it is better to use `\xintCmp{f}{1}`. Restyled in 1.09i.

```

1572 \def\xintIsOne {\romannumeral0\xintisone }%
1573 \def\xintisone #1{\expandafter\XINT_fracisone
1574     \romannumeral0\xintrawwithzeros{#1}\Z }%
1575 \def\XINT_fracisone #1/#2\Z
1576     {\if0\xintiiCmp {#1}{#2}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%

```

## 8.53 \xintGeq

```

1577 \def\xintGeq {\romannumeral0\xintgeq }%

```

```

1578 \def\xintgeq #1%
1579 {%
1580   \expandafter\XINT_fgeq\expandafter {\romannumeral0\xintabs {#1}}%
1581 }%
1582 \def\XINT_fgeq #1#2%
1583 {%
1584   \expandafter\XINT_fgeq_A \romannumeral0\xintabs {#2}#1%
1585 }%
1586 \def\XINT_fgeq_A #1%
1587 {%
1588   \xint_gob_til_zero #1\XINT_fgeq_Zii 0%
1589   \XINT_fgeq_B #1%
1590 }%
1591 \def\XINT_fgeq_Zii 0\XINT_fgeq_B #1[#2]#3[#4]{ 1}%
1592 \def\XINT_fgeq_B #1/#2[#3]#4#5/#6[#7]%
1593 {%
1594   \xint_gob_til_zero #4\XINT_fgeq_Zi 0%
1595   \expandafter\XINT_fgeq_C\expandafter
1596   {\the\numexpr #7-#3\expandafter}\expandafter
1597   {\romannumeral0\xintiimul {#4#5}{#2}}%
1598   {\romannumeral0\xintiimul {#6}{#1}}%
1599 }%
1600 \def\XINT_fgeq_Zi 0#1#2#3#4#5#6#7{ 0}%
1601 \def\XINT_fgeq_C #1#2#3%
1602 {%
1603   \expandafter\XINT_fgeq_D\expandafter
1604   {#3}{#1}{#2}%
1605 }%
1606 \def\XINT_fgeq_D #1#2#3%
1607 {%
1608   \expandafter\XINT_cntSgnFork\romannumeral`&&\expandafter\XINT_cntSgn
1609   \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\xint:
1610   { 0}{\XINT_fgeq_E #2\Z {#3}{#1}}{ 1}%
1611 }%
1612 \def\XINT_fgeq_E #1%
1613 {%
1614   \xint_UDsignfork
1615   #1\XINT_fgeq_Fd
1616   -{\XINT_fgeq_Fn #1}%
1617   \krof
1618 }%

1619 \def\XINT_fgeq_Fd #1\Z #2#3%
1620 {%
1621   \expandafter\XINT_fgeq_Fe
1622   \romannumeral0\XINT_dsx_addzeros {#1}#3;\xint:#2\xint:
1623 }%
1624 \def\XINT_fgeq_Fe #1\xint:#2#3\xint:{\XINT_geq_plusplus #2#1\xint:#3\xint:}%
1625 \def\XINT_fgeq_Fn #1\Z #2#3%
1626 {%
1627   \expandafter\XINT_fgeq_Fo
1628   \romannumeral0\XINT_dsx_addzeros {#1}#2;\xint:#3\xint:
1629 }%

```

```
1630 \def\XINT_fgeq_Fo #1#2\xint:#3\xint:{\XINT_geq_plusplus #1#3\xint:#2\xint:}%
```

## 8.54 \xintMax

```
1631 \def\xintMax {\romannumeral0\xintmax }%
1632 \def\xintmax #1%
1633 {%
1634     \expandafter\XINT_fmax\expandafter {\romannumeral0\xintraw {#1}}%
1635 }%
1636 \def\XINT_fmax #1#2%
1637 {%
1638     \expandafter\XINT_fmax_A\romannumeral0\xintraw {#2}#1%
1639 }%
1640 \def\XINT_fmax_A #1#2/#3[#4]#5#6/#7[#8]%
1641 {%
1642     \xint_UDsignsfork
1643     #1#5\XINT_fmax_minusminus
1644     -#5\XINT_fmax_firstneg
1645     #1-\XINT_fmax_secondneg
1646     --\XINT_fmax_nonneg_a
1647     \krof
1648     #1#5{#2/#3[#4]}{#6/#7[#8]}%
1649 }%
1650 \def\XINT_fmax_minusminus --%
1651     {\expandafter-\romannumeral0\XINT_fmin_nonneg_b }%
1652 \def\XINT_fmax_firstneg #1-#2#3{ #1#2}%
1653 \def\XINT_fmax_secondneg -#1#2#3{ #1#3}%
1654 \def\XINT_fmax_nonneg_a #1#2#3#4%
1655 {%
1656     \XINT_fmax_nonneg_b {#1#3}{#2#4}%
1657 }%
1658 \def\XINT_fmax_nonneg_b #1#2%
1659 {%
1660     \if0\romannumeral0\XINT_fgeq_A #1#2%
1661         \xint_afterfi{ #1}%
1662     \else \xint_afterfi{ #2}%
1663     \fi
1664 }%
```

## 8.55 \xintMaxof

1.21 protects \xintMaxof against items with non terminated \the\numexpr expressions.  
The macro is not compatible with an empty list.

```
1665 \def\xintMaxof {\romannumeral0\xintmaxof }%
1666 \def\xintmaxof #1{\expandafter\XINT_maxof_a\romannumeral`&&@#1\xint:}%
1667 \def\XINT_maxof_a #1{\expandafter\XINT_maxof_b\romannumeral0\xintraw{#1}!}%
1668 \def\XINT_maxof_b #1!#2%
1669     {\expandafter\XINT_maxof_c\romannumeral`&&@#2!{#1}!}%
1670 \def\XINT_maxof_c #1%
1671     {\xint_gob_til_xint: #1\XINT_maxof_e\xint:\XINT_maxof_d #1}%
1672 \def\XINT_maxof_d #1!%
1673     {\expandafter\XINT_maxof_b\romannumeral0\xintmax {#1}}%
1674 \def\XINT_maxof_e #1!#2!{ #2}%

```

## 8.56 \xintMin

```

1675 \def\xintMin {\romannumeral0\xintmin }%
1676 \def\xintmin #1%
1677 {%
1678   \expandafter\XINT_fmin\expandafter {\romannumeral0\xintra {#1}}%
1679 }%
1680 \def\XINT_fmin #1#2%
1681 {%
1682   \expandafter\XINT_fmin_A\romannumeral0\xintra {#2}#1%
1683 }%
1684 \def\XINT_fmin_A #1#2/#3[#4]#5#6/#7[#8]%
1685 {%
1686   \xint_UDsignsfork
1687     #1#5\XINT_fmin_minusminus
1688     -#5\XINT_fmin_firstneg
1689     #1-\XINT_fmin_secondneg
1690     --\XINT_fmin_nonneg_a
1691   \krof
1692   #1#5{#2/#3[#4]}{#6/#7[#8]}%
1693 }%
1694 \def\XINT_fmin_minusminus --%
1695   {\expandafter-\romannumeral0\XINT_fmax_nonneg_b }%
1696 \def\XINT_fmin_firstneg #1-#2#3{ -#3}%
1697 \def\XINT_fmin_secondneg -#1#2#3{ -#2}%
1698 \def\XINT_fmin_nonneg_a #1#2#3#4%
1699 {%
1700   \XINT_fmin_nonneg_b {#1#3}{#2#4}%
1701 }%
1702 \def\XINT_fmin_nonneg_b #1#2%
1703 {%
1704   \if0\romannumeral0\XINT_fgeq_A #1#2%
1705     \xint_afterfi{ #2}%
1706   \else \xint_afterfi{ #1}%
1707   \fi
1708 }%

```

## 8.57 \xintMinof

1.2l protects \xintMinof against items with non terminated \the\numexpr expressions.  
 The macro is not compatible with an empty list.

```

1709 \def\xintMinof {\romannumeral0\xintminof }%
1710 \def\xintminof #1{\expandafter\XINT_minof_a\romannumeral`&&@#1\xint:}%
1711 \def\XINT_minof_a #1{\expandafter\XINT_minof_b\romannumeral0\xintra{#1}!}%
1712 \def\XINT_minof_b #1!#2%
1713   {\expandafter\XINT_minof_c\romannumeral`&&@#2!{#1}!}%
1714 \def\XINT_minof_c #1%
1715   {\xint_gob_til_xint: #1\XINT_minof_e\xint:\XINT_minof_d #1}%
1716 \def\XINT_minof_d #1!%
1717   {\expandafter\XINT_minof_b\romannumeral0\xintmin {#1}}%
1718 \def\XINT_minof_e #1!#2!{ #2}%

```

## 8.58 \xintCmp

```

1719 \def\xintCmp {\romannumeral0\xintcmp }%
1720 \def\xintcmp #1%
1721 {%
1722   \expandafter\XINT_fcmp\expandafter {\romannumeral0\xintra {#1}}%
1723 }%
1724 \def\XINT_fcmp #1#2%
1725 {%
1726   \expandafter\XINT_fcmp_A\romannumeral0\xintra {#2}#1%
1727 }%
1728 \def\XINT_fcmp_A #1#2/#3[#4]#5#6/#7[#8]%
1729 {%
1730   \xint_UDsignsfork
1731     #1#5\XINT_fcmp_minusminus
1732     -#5\XINT_fcmp_firstneg
1733     #1-\XINT_fcmp_secondneg
1734     --\XINT_fcmp_nonneg_a
1735   \krof
1736   #1#5{#2/#3[#4]}{#6/#7[#8]}%
1737 }%
1738 \def\XINT_fcmp_minusminus --#1#2{\XINT_fcmp_B #2#1}%
1739 \def\XINT_fcmp_firstneg #1-#2#3{ -1}%
1740 \def\XINT_fcmp_secondneg -#1#2#3{ 1}%
1741 \def\XINT_fcmp_nonneg_a #1#2%
1742 {%
1743   \xint_UDzerosfork
1744     #1#2\XINT_fcmp_zerozero
1745     0#2\XINT_fcmp_firstzero
1746     #10\XINT_fcmp_secondzero
1747     00\XINT_fcmp_pos
1748   \krof
1749   #1#2%
1750 }%
1751 \def\XINT_fcmp_zerozero #1#2#3#4{ 0}%
1752 \def\XINT_fcmp_firstzero #1#2#3#4{ -1}%
1753 \def\XINT_fcmp_secondzero #1#2#3#4{ 1}%
1754 \def\XINT_fcmp_pos #1#2#3#4%
1755 {%
1756   \XINT_fcmp_B #1#3#2#4%
1757 }%
1758 \def\XINT_fcmp_B #1/#2[#3]#4/#5[#6]%
1759 {%
1760   \expandafter\XINT_fcmp_C\expandafter
1761   {\the\numexpr #6-#3\expandafter}\expandafter
1762   {\romannumeral0\xintiimul {#4}{#2}}%
1763   {\romannumeral0\xintiimul {#5}{#1}}%
1764 }%
1765 \def\XINT_fcmp_C #1#2#3%
1766 {%
1767   \expandafter\XINT_fcmp_D\expandafter
1768   {#3}{#1}{#2}%
1769 }%

```

```

1770 \def\XINT_fcmp_D #1#2#3%
1771 {%
1772     \expandafter\XINT_cntSgnFork\romannumeral`&&\expandafter\XINT_cntSgn
1773     \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\xint:
1774     { -1}{\XINT_fcmp_E #2\Z {#3}{#1}}{ 1}%
1775 }%
1776 \def\XINT_fcmp_E #1%
1777 {%
1778     \xint_UDsignfork
1779     #1\XINT_fcmp_Fd
1780     -{\XINT_fcmp_Fn #1}%
1781     \krof
1782 }%

1783 \def\XINT_fcmp_Fd #1\Z #2#3%
1784 {%
1785     \expandafter\XINT_fcmp_Fe
1786     \romannumeral0\XINT_dsx_addzeros {#1}#3;\xint:#2\xint:
1787 }%
1788 \def\XINT_fcmp_Fe #1\xint:#2#3\xint:{\XINT_cmp_plusplus #2#1\xint:#3\xint:}%
1789 \def\XINT_fcmp_Fn #1\Z #2#3%
1790 {%
1791     \expandafter\XINT_fcmp_Fo
1792     \romannumeral0\XINT_dsx_addzeros {#1}#2;\xint:#3\xint:
1793 }%
1794 \def\XINT_fcmp_Fo #1#2\xint:#3\xint:{\XINT_cmp_plusplus #1#3\xint:#2\xint:}%

```

## 8.59 \xintAbs

```

1795 \def\xintAbs {\romannumeral0\xintabs }%
1796 \def\xintabs #1{\expandafter\XINT_abs\romannumeral0\xintraw {#1}}%

```

## 8.60 \xintOpp

```

1797 \def\xintOpp {\romannumeral0\xintopp }%
1798 \def\xintopp #1{\expandafter\XINT_opp\romannumeral0\xintraw {#1}}%

```

## 8.61 \xintInv

### 1.3d.

```

1799 \def\xintInv {\romannumeral0\xintinv }%
1800 \def\xintinv #1{\expandafter\XINT_inv\romannumeral0\xintraw {#1}}%
1801 \def\XINT_inv #1%
1802 {%
1803     \xint_UDzerominusfork
1804     #1-\XINT_inv_iszero
1805     0#1\XINT_inv_a
1806     0-{\XINT_inv_a {}}%
1807     \krof #1%
1808 }%
1809 \def\XINT_inv_iszero #1]%
1810     {\XINT_signalcondition{DivisionByZero}{Division of 1 by zero (#1)}}{{0/1[0]}}%
1811 \def\XINT_inv_a #1#2/#3[#4#5]%
1812 {%
1813     \xint_UDzerominusfork

```

```

1814      #4-\XINT_inv_expiszero
1815      0#4\XINT_inv_b
1816      0-{\XINT_inv_b -#4}%
1817      \krof #5.{#1#3/#2}%
1818 }%
1819 \def\XINT_inv_expiszero #1.#2{ #2[0]}%
1820 \def\XINT_inv_b #1.#2{ #2[#1]}%

```

## 8.62 \xintSgn

```

1821 \def\xintSgn {\romannumeral0\xintsfn}%
1822 \def\xintsfn #1{\expandafter\XINT_sgn\romannumeral0\xintra {#1}\xint:}%

```

## 8.63 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.

## 8.64 \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`.

```

1823 \def\xintFloat    {\romannumeral0\xintfloat }%
1824 \def\xintfloat #1{\XINT_float_chkopt #1\xint:}%
1825 \def\XINT_float_chkopt #1%
1826 {%
1827     \ifx [#1\expandafter\XINT_float_opt
1828         \else\expandafter\XINT_float_noopt
1829     \fi #1%
1830 }%
1831 \def\XINT_float_noopt #1\xint:%
1832 {%
1833     \expandafter\XINT_float_post
1834     \romannumeral0\XINTinfloat[\XINTdigits]{#1}\XINTdigits.%
1835 }%
1836 \def\XINT_float_opt [\xint:#1]%
1837 {%
1838     \expandafter\XINT_float_opt_a\the\numexpr #1.%
1839 }%
1840 \def\XINT_float_opt_a #1.#2%
1841 {%
1842     \expandafter\XINT_float_post

```



```

1843 \romannumeral0\XINTinfloat[#1]{#2}#1.%
1844 }%
1845 \def\XINT_float_post #1%
1846 {%
1847 \xint_UDzerominusfork
1848 #1-\XINT_float_zero
1849 0#1\XINT_float_neg
1850 0-\XINT_float_pos
1851 \krof #1%
1852 }%[
1853 \def\XINT_float_zero #1]#2.{ 0.e0}%
1854 \def\XINT_float_neg-{\expandafter-\romannumeral0\XINT_float_pos}%
1855 \def\XINT_float_pos #1#2[#3]#4.%
1856 {%
1857 \expandafter\XINT_float_pos_done\the\numexpr#3+#4-\xint_c_i.#1.#2;%
1858 }%
1859 \def\XINT_float_pos_done #1.#2;{ #2e#1}%

```

## 8.65 \XINTinFloat, \XINTinFloatS

This routine is like `\xintFloat` but produces an output of the shape `A[N]` which is then parsed faster as input to other float macros. Float operations in `\xintfloatexpr...` relax use internally this format.

It must be used in form `\XINTinFloat[P]{f}`: the optional `[P]` is mandatory.

Since 1.2f, the mantissa always has exactly `P` digits even in case of rounding up to next power of ten. This simplifies other routines.

1.2g added a variant `\XINTinFloatS` which, in case of decimal input with less than the asked for precision `P` will not add extra zeros to the mantissa. For example it may output `2[0]` even if `P=500`, rather than the canonical representation `200...000[-499]`. This is how `\xintFloatMul` and `\xintFloatDiv` parse their inputs, which speeds-up follow-up processing. But `\xintFloatAdd` and `\xintFloatSub` still use `\XINTinFloat` for parsing their inputs; anyway this will have to be changed again when inner structure will carry upfront at least the length of mantissa as data.

Each time `\XINTinFloat` is called it at least computes a length. Naturally if we had some format for floats that would be dispensed of...

something like `<letterP><length of mantissa>.mantissa.exponent`, etc... not yet.

Since 1.2k, `\XINTinFloat` always correctly rounds its argument, even if it is a fraction with very big numerator and denominator. See the discussion of `\xintFloat`.

```

1860 \def\XINTinFloat {\romannumeral0\XINTinfloat }%
1861 \def\XINTinfloat
1862 {\expandafter\XINT_infloat_clean\romannumeral0\XINT_infloat}%
1863 \def\XINT_infloat_clean #1%
1864 {\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`

```

1865 \def\XINT_infloat_clean_a !#1.#2[#3]%
1866 {%
1867 \expandafter\XINT_infloat_done
1868 \the\numexpr #3-#1\expandafter.%
1869 \romannumeral0\XINT_dsx_addzeros {#1}#2;%
1870 }%
1871 \def\XINT_infloat_done #1.#2;{ #2[#1]}%

```

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrc, xintexpr, indices*

variant which allows output with shorter mantissas.

```
1872 \def\XINTinFloatS {\romannumeral0\XINTinfloatS}%
1873 \def\XINTinfloatS
1874   {\expandafter\XINT_infloatS_clean\romannumeral0\XINT_infloat}%
1875 \def\XINT_infloatS_clean #1%
1876   {\if #1!\xint_dothis\XINT_infloatS_clean_a\fi\xint_orthat{ }#1}%
1877 \def\XINT_infloatS_clean_a !#1.{ }%
```

début de la routine proprement dite, l'argument optionnel est obligatoire.

```
1878 \def\XINT_infloat [#1]#2%
1879 {%
1880   \expandafter\XINT_infloat_a\the\numexpr #1\expandafter.%
1881   \romannumeral0\XINT_infrac {#2}%
1882 }%
```

#1=P, #2=n, #3=A, #4=B.

```
1883 \def\XINT_infloat_a #1.#2#3#4%
1884 {%
```

micro boost au lieu d'utiliser \XINT\_isOne{#4}, mais pas bon style.

```
1885   \if1\XINT_is_One#4XY%
1886     \expandafter\XINT_infloat_sp
1887   \else\expandafter\XINT_infloat_fork
1888   \fi #3.{#1}{#2}{#4}%
1889 }%
```

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.

```
1890 \def\XINT_infloat_sp #1%
1891 {%
1892   \xint_UDzerominusfork
1893   #1-\XINT_infloat_spzero
1894   0#1\XINT_infloat_spneg
1895   0-\XINT_infloat_sppos
1896   \krof #1%
1897 }%
```

Attention surtout pas 0/1[0] ici.

```
1898 \def\XINT_infloat_spzero 0.#1#2#3{ 0[0]}%
1899 \def\XINT_infloat_spneg-%
1900   {\expandafter\XINT_infloat_spnegend\romannumeral0\XINT_infloat_sppos}%
1901 \def\XINT_infloat_spnegend #1%
1902   {\if#1!\expandafter\XINT_infloat_spneg_needzeros\fi -#1}%
1903 \def\XINT_infloat_spneg_needzeros -!#1.{!#1.-}%

```

in: A.{P}{N}{1}

out: P-L.A.P.N.

```
1904 \def\XINT_infloat_sppos #1.#2#3#4%
1905 {%
1906   \expandafter\XINT_infloat_sp_b\the\numexpr#2-\xintLength{#1}.#1.#2.#3.%
1907 }%
```

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices*

#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.

```
1908 \def\XINT_infloat_sp_b #1%
1909 {%
1910     \xint_UDzerominusfork
1911     #1-\XINT_infloat_sp_quick
1912     0#1\XINT_infloat_sp_c
1913     0-\XINT_infloat_sp_needzeros
1914     \krof #1%
1915 }%
```

Ici P=L. Le cas usuel dans \xintfloatexpr.

```
1916 \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.

```
1917 \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.

```
1918 \def\XINT_infloat_sp_c -#1.#2#3.#4.#5.%
1919 {%
1920     \expandafter\XINT_infloat_Y
1921     \the\numexpr #5+#1\expandafter.%
1922     \romannumeral0\expandafter\XINT_infloat_sp_round
1923     \romannumeral0\XINT_split_fromleft
1924     (\xint_c_i+#4).#2#3\xint_bye2345678\xint_bye..#2%
1925 }%
1926 \def\XINT_infloat_sp_round #1.#2.%
1927 {%
1928     \XINT_dsrr#1\xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax.%
1929 }%
```

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.

```
1930 \def\XINT_infloat_fork #1%
1931 {%
1932     \xint_UDsignfork
1933     #1\XINT_infloat_J
1934     -\XINT_infloat_K
1935     \krof #1%
1936 }%
1937 \def\XINT_infloat_J-{\expandafter-\romannumeral0\XINT_infloat_K }%
```

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrc, xintexpr, indices*

$A.\{P\}_{n\{B\}}$  avec  $B>1$ .

```
1938 \def\XINT_infloat_K #1.#2%
1939 {%
1940     \expandafter\XINT_infloat_L
1941     \the\numexpr\xintLength{#1}\expandafter.\the\numexpr #2+\xint_c_iv.{#1}{#2}%
1942 }%
```

$|A|.P+4.\{A\}_{P\{n\{B\}}$ . We check if A already has length  $\leq P+4$ .

```
1943 \def\XINT_infloat_L #1.#2.%
1944 {%
1945     \ifnum #1>#2
1946         \expandafter\XINT_infloat_Ma
1947     \else
1948         \expandafter\XINT_infloat_Mb
1949     \fi #1.#2.%
1950 }%
```

$|A|.P+4.\{A\}_{P\{n\{B\}}$ . We will keep only the first  $P+4$  digits of A, denoted  $A'$  in what follows.  
output:  $u=-0.A'.junk.P+4.|A|. \{A\}_{P\{n\{B\}}$

```
1951 \def\XINT_infloat_Ma #1.#2.#3%
1952 {%
1953     \expandafter\XINT_infloat_MtoN\expandafter-\expandafter0\expandafter.%
1954     \romannumeral0\XINT_split_fromleft#2.#3\xint_bye2345678\xint_bye..%
1955     #2.#1.{#3}%
1956 }%
```

$|A|.P+4.\{A\}_{P\{n\{B\}}$ .

Here A is short. We set  $u = P+4 - |A|$ , and  $A' = A$  ( $A' = 10^u A$ )

output:  $u.A'..P+4.|A|. \{A\}_{P\{n\{B\}}$

```
1957 \def\XINT_infloat_Mb #1.#2.#3%
1958 {%
1959     \expandafter\XINT_infloat_MtoN\the\numexpr#2-#1.%
1960     #3..#2.#1.{#3}%
1961 }%
```

input  $u.A'..junk.P+4.|A|. \{A\}_{P\{n\{B\}}$

output  $|B|.P+4.\{B\}u.A'.P.|A|.n.\{A\}_{B\}$

```
1962 \def\XINT_infloat_MtoN #1.#2.#3.#4.#5.#6#7#8#9%
1963 {%
1964     \expandafter\XINT_infloat_N
1965     \the\numexpr\xintLength{#9}.#4.{#9}#1.#2.#7.#5.#8.{#6}{#9}%
1966 }%
1967 \def\XINT_infloat_N #1.#2.%
1968 {%
1969     \ifnum #1>#2
1970         \expandafter\XINT_infloat_Oa
1971     \else
1972         \expandafter\XINT_infloat_Ob
1973     \fi #1.#2.%
1974 }%
```

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices*

```
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}
```

```
1975 \def\XINT_infloat_0a #1.#2.#3%
1976 {%
1977     \expandafter\XINT_infloat_P\expandafter-\expandafter0\expandafter.%
1978     \romannumeral0\XINT_split_fromleft#2.#3\xint_bye2345678\xint_bye..%
1979     #1.%
1980 }%
```

```
output v=P+4-|B|>=0.B''.junk.|B|.u.A''.P.|A|.n.{A}{B}
```

```
1981 \def\XINT_infloat_0b #1.#2.#3%
1982 {%
1983     \expandafter\XINT_infloat_P\the\numexpr#2-#1.#3..#1.%
1984 }%
```

```
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.

```
1985 \def\XINT_infloat_P #1#2.#3.#4.#5.#6#7.#8.#9.%
1986 {%
1987     \csname XINT_infloat_Q\if-#1\else\if-#6\else q\fi\fi\expandafter\endcsname
1988     \romannumeral0\xintiiquo
1989     {\romannumeral0\XINT_dsx_addzerosnofuss
1990         {#6#7-#1#2+#9+\xint_c_iii\if-#1\else\if-#6\else-\xint_c_ii\fi\fi}#8;}%
1991     {#3}.#9.#5.%
1992 }%
```

«quick» branch.

```
1993 \def\XINT_infloat_Qq #1.#2.%
1994 {%
1995     \expandafter\XINT_infloat_Rq
1996     \romannumeral0\XINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..#2.%
1997 }%
1998 \def\XINT_infloat_Rq #1.#2#3.%
1999 {%
2000     \ifnum#2<\xint_c_v
2001         \expandafter\XINT_infloat_SEq
2002     \else\expandafter\XINT_infloat_SUP
2003     \fi
2004     {\if.#3.\xint_c_\else\xint_c_i\fi}#1.%
2005 }%
```

standard branch which will have to handle undecided rounding, if too close to a mid-value.

```
2006 \def\XINT_infloat_Q #1.#2.%
2007 {%
2008     \expandafter\XINT_infloat_R
2009     \romannumeral0\XINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..#2.%
```

```

2010 }%
2011 \def\XINT_infloat_R #1.#2#3#4#5.%
2012 {%
2013     \if.#5.\expandafter\XINT_infloat_Sa\else\expandafter\XINT_infloat_Sb\fi
2014     #2#3#4#5.#1.%
2015 }%

```

trailing digits.Q.P.|B|.|A|.n.{A}{B}  
 #1=trailing digits (they may have leading zeros.)

```

2016 \def\XINT_infloat_Sa #1.%
2017 {%
2018     \ifnum#1>500 \xint_dothis\XINT_infloat_SUP\fi
2019     \ifnum#1<499 \xint_dothis\XINT_infloat_SEq\fi
2020     \xint_orthat\XINT_infloat_X\xint_c_
2021 }%
2022 \def\XINT_infloat_Sb #1.%
2023 {%
2024     \ifnum#1>5009 \xint_dothis\XINT_infloat_SUP\fi
2025     \ifnum#1<4990 \xint_dothis\XINT_infloat_SEq\fi
2026     \xint_orthat\XINT_infloat_X\xint_c_i
2027 }%

```

epsilon #2=Q.#3=P.#4=|B|. #5=|A|. #6=n.{A}{B}  
 exposant final est n+|A|-|B|-P+epsilon

```

2028 \def\XINT_infloat_SEq #1#2.#3.#4.#5.#6.#7#8%
2029 {%
2030     \expandafter\XINT_infloat_SY
2031     \the\numexpr #6+#5-#4-#3+#1.#2.%
2032 }%
2033 \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.

```

2034 \def\XINT_infloat_SUP #1#2#3.#4.#5.#6.#7.#8#9%
2035 {%
2036     \expandafter\XINT_infloat_Y
2037     \the\numexpr#7+#6-#5-#4+#1\expandafter.%
2038     \romannumeral0\xintinc{#2#3}.#2%
2039 }%

```

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

```

2040 \def\XINT_infloat_X #1#2#3.#4.#5.#6.#7.#8#9%
2041 {%
2042     \expandafter\XINT_infloat_Y
2043     \the\numexpr #7+#6-#5-#4+#1\expandafter.%

```

```

2044 \romannumeral`&&@\romannumeral0\xintiiiflt
2045 {\xintDSH{#6-#5-#4+#1}{\xintDouble{#8}}}%
2046 {\xintiiMul{\xintInc{\xintDouble{#2#3}}}{#9}}%
2047 \xint_firstofone
2048 \xintinc{#2#3}.#2%
2049 }%

```

check for rounding up to next power of ten.

```

2050 \def\xINT_infloat_Y #1{%
2051 \def\xINT_infloat_Y ##1.##2##3.##4%
2052 {%
2053 \if##49\if##21\expandafter\expandafter\expandafter\xINT_infloat_Z\fi\fi
2054 #1##2##3[##1]%
2055 }}\xINT_infloat_Y{ }%

```

#1=1, #2=0.

```

2056 \def\xINT_infloat_Z #1#2#3[#4]%
2057 {%
2058 \expandafter\xINT_infloat_ZZ\the\numexpr#4+\xint_c_i.#3.%
2059 }%
2060 \def\xINT_infloat_ZZ #1.#2.{ 1#2[#1]}%

```

## 8.66 \xintPFloat

1.1. This is a prettifying printing macro for floats.

The macro applies one simple rule:  $x.yz...eN$  will drop scientific notation in favor of pure decimal notation if  $-5 \leq N \leq 5$ . This is the default behaviour of Maple. The  $N$  here is as produced on output by `\xintFloat`.

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  $10.0...0eN$  annoying exception).

```

2061 \def\xintPFloat {\romannumeral0\xintpfloat }%
2062 \def\xintpfloat #1{\XINT_pfloat_chkopt #1\xint:}%
2063 \def\xINT_pfloat_chkopt #1%
2064 {%
2065 \ifx [#1\expandafter\xINT_pfloat_opt
2066 \else\expandafter\xINT_pfloat_noopt
2067 \fi #1%
2068 }%
2069 \def\xINT_pfloat_noopt #1\xint:%
2070 {%
2071 \expandafter\xINT_pfloat_a
2072 \romannumeral0\xintfloat [\XINTdigits]{#1};\XINTdigits.%
2073 }%

```

```

2074 \def\xINT_pfloat_opt [\xint:#1]%
2075 {%
2076 \expandafter\xINT_pfloat_opt_a \the\numexpr #1.%
2077 }%
2078 \def\xINT_pfloat_opt_a #1.#2%

```

```

2079 {%
2080     \expandafter\XINT_pfloat_a\romannumeral0\xintfloat [#1]{#2};#1.%
2081 }%
2082 \def\XINT_pfloat_a #1%
2083 {%
2084     \xint_UDzerominusfork
2085         #1-\XINT_pfloat_zero
2086         0#1\XINT_pfloat_neg
2087         0-\XINT_pfloat_pos
2088     \krof #1%
2089 }%

2090 \def\XINT_pfloat_zero #1;#2.{ 0.}%
2091 \def\XINT_pfloat_neg-{\expandafter-\romannumeral0\XINT_pfloat_pos }%

2092 \def\XINT_pfloat_pos #1.#2e#3;#4.%
2093 {%
2094     \ifnum #3>\xint_c_v \xint_dothis\XINT_pfloat_no\fi
2095     \ifnum #3<-\xint_c_v \xint_dothis\XINT_pfloat_no\fi
2096     \ifnum #3<\xint_c_ \xint_dothis\XINT_pfloat_N\fi
2097     \ifnum #3>\numexpr #4-\xint_c_i\relax \xint_dothis\XINT_pfloat_Ps\fi
2098     \xint_orthat\XINT_pfloat_P #1#2e#3;%
2099 }%
2100 \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.

2101 \def\XINT_pfloat_N #1e-#2;%
2102 {%
2103     \csname XINT_pfloat_N_\romannumeral#2\endcsname #1%
2104 }%
2105 \def\XINT_pfloat_N_i { 0.}%
2106 \def\XINT_pfloat_N_ii { 0.0}%
2107 \def\XINT_pfloat_N_iii{ 0.00}%
2108 \def\XINT_pfloat_N_iv { 0.000}%
2109 \def\XINT_pfloat_N_v { 0.0000}%

2110 \def\XINT_pfloat_P #1e#2;%
2111 {%
2112     \csname XINT_pfloat_P_\romannumeral#2\endcsname #1%
2113 }%
2114 \def\XINT_pfloat_P_ #1{ #1.}%
2115 \def\XINT_pfloat_P_i #1#2{ #1#2.}%
2116 \def\XINT_pfloat_P_ii #1#2#3{ #1#2#3.}%
2117 \def\XINT_pfloat_P_iii#1#2#3#4{ #1#2#3#4.}%
2118 \def\XINT_pfloat_P_iv #1#2#3#4#5{ #1#2#3#4#5.}%
2119 \def\XINT_pfloat_P_v #1#2#3#4#5#6{ #1#2#3#4#5#6.}%

2120 \def\XINT_pfloat_Ps #1e#2;%
2121 {%

```



```

2122 \csname XINT_pfloat_Ps\romannumeral#2\endcsname #100000;%
2123 }%
2124 \def\XINT_pfloat_Psi #1#2#3;{ #1#2.%}%
2125 \def\XINT_pfloat_Psii #1#2#3#4;{ #1#2#3.%}%
2126 \def\XINT_pfloat_Psiii#1#2#3#4#5;{ #1#2#3#4.%}%
2127 \def\XINT_pfloat_Psiv #1#2#3#4#5#6;{ #1#2#3#4#5.%}%
2128 \def\XINT_pfloat_Psv #1#2#3#4#5#6#7;{ #1#2#3#4#5#6.%}%

```

## 8.67 \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 \xinttfrc was just a first implementation.

```

2129 \def\XINTinFloatFracdigits {\romannumeral0\XINTinfloatfracdigits }%
2130 \def\XINTinfloatfracdigits #1%
2131 {%
2132 \expandafter\XINT_infloatfracdg_a\expandafter {\romannumeral0\xinttfrc{#1}}%
2133 }%
2134 \def\XINT_infloatfracdg_a {\XINTinfloat [\XINTdigits]}%

```

## 8.68 \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.

```

2135 \def\xintFloatAdd {\romannumeral0\xintfloatadd }%
2136 \def\xintfloatadd #1{\XINT_fladd_chkopt \xintfloat #1\xint:}%
2137 \def\XINTinFloatAdd {\romannumeral0\XINTinfloatadd }%
2138 \def\XINTinfloatadd #1{\XINT_fladd_chkopt \XINTinfloatS #1\xint:}%
2139 \def\XINT_fladd_chkopt #1#2%
2140 {%
2141 \ifx [#2\expandafter\XINT_fladd_opt
2142 \else\expandafter\XINT_fladd_noopt
2143 \fi #1#2%
2144 }%
2145 \def\XINT_fladd_noopt #1#2\xint:#3%
2146 {%
2147 #1[\XINTdigits]%
2148 {\expandafter\XINT_FL_add_a
2149 \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{#3}}%
2150 }%
2151 \def\XINT_fladd_opt #1[\xint:#2]%#3#4%
2152 {%
2153 \expandafter\XINT_fladd_opt_a\the\numexpr #2.#1%

```

```

2154 }%
2155 \def\XINT_fladd_opt_a #1.#2#3#4%
2156 {%
2157     #2[#1]{\expandafter\XINT_FL_add_a\romannumeral0\XINTinfloat[#1]{#3}#1.{#4}}%
2158 }%
2159 \def\XINT_FL_add_a #1%
2160 {%
2161     \xint_gob_til_zero #1\XINT_FL_add_zero 0\XINT_FL_add_b #1%
2162 }%

2163 \def\XINT_FL_add_zero #1.#2{#2}%[
2164 \def\XINT_FL_add_b #1]#2.#3%
2165 {%
2166     \expandafter\XINT_FL_add_c\romannumeral0\XINTinfloat[#2]{#3}#2.#1)%
2167 }%

2168 \def\XINT_FL_add_c #1%
2169 {%
2170     \xint_gob_til_zero #1\XINT_FL_add_zero 0\XINT_FL_add_d #1%
2171 }%

2172 \def\XINT_FL_add_d #1[#2]#3.#4[#5]%
2173 {%
2174     \ifnum\numexpr #2-#3-#5>\xint_c_\xint_dothis\xint_firstoftwo\fi
2175     \ifnum\numexpr #5-#3-#2>\xint_c_\xint_dothis\xint_secondoftwo\fi
2176     \xint_orthat\xintAdd {#1[#2]}{#4[#5]}%
2177 }%

```

## 8.69 \xintFloatSub, \XINTinFloatSub

First done 1.07.

Starting with 1.2f the arguments undergo an initial rounding to the target precision P not P+2.

```

2178 \def\xintFloatSub {\romannumeral0\xintfloatsub }%
2179 \def\xintfloatsub #1{\XINT_fsub_chkopt \xintfloat #1\xint:}%
2180 \def\XINTinFloatSub {\romannumeral0\XINTinfloatsub }%
2181 \def\XINTinfloatsub #1{\XINT_fsub_chkopt \XINTinfloatS #1\xint:}%
2182 \def\XINT_fsub_chkopt #1#2%
2183 {%
2184     \ifx [#2\expandafter\XINT_fsub_opt
2185     \else\expandafter\XINT_fsub_noopt
2186     \fi #1#2%
2187 }%
2188 \def\XINT_fsub_noopt #1#2\xint:#3%
2189 {%
2190     #1[\XINTdigits]%
2191     {\expandafter\XINT_FL_add_a
2192     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{\xintOpp{#3}}}%
2193 }%
2194 \def\XINT_fsub_opt #1[\xint:#2]%#3#4%
2195 {%

```

```

2196 \expandafter\XINT_flsb_opt_a\the\numexpr #2.#1%
2197 }%
2198 \def\XINT_flsb_opt_a #1.#2#3#4%
2199 {%
2200 #2[#1]{\expandafter\XINT_FL_add_a\romannumeral0\XINTinfloat[#1]{#3}#1.{\xintOpp{#4}}}%
2201 }%

```

## 8.70 \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).

```

2202 \def\xintFloatMul {\romannumeral0\xintfloatmul }%
2203 \def\xintfloatmul #1{\XINT_flmul_chkopt \xintfloat #1\xint:}%
2204 \def\XINTinFloatMul {\romannumeral0\XINTinfloatmul }%
2205 \def\XINTinfloatmul #1{\XINT_flmul_chkopt \XINTinfloatS #1\xint:}%
2206 \def\XINT_flmul_chkopt #1#2%
2207 {%
2208 \ifx [#2\expandafter\XINT_flmul_opt
2209 \else\expandafter\XINT_flmul_noopt
2210 \fi #1#2%
2211 }%
2212 \def\XINT_flmul_noopt #1#2\xint:#3%
2213 {%
2214 #1[\XINTdigits]%
2215 {\expandafter\XINT_FL_mul_a
2216 \romannumeral0\XINTinfloatS[\XINTdigits]{#2}\XINTdigits.{#3}}%
2217 }%
2218 \def\XINT_flmul_opt #1[\xint:#2]#3#4%
2219 {%
2220 \expandafter\XINT_flmul_opt_a\the\numexpr #2.#1%
2221 }%
2222 \def\XINT_flmul_opt_a #1.#2#3#4%
2223 {%
2224 #2[#1]{\expandafter\XINT_FL_mul_a\romannumeral0\XINTinfloatS[#1]{#3}#1.{#4}}%
2225 }%
2226 \def\XINT_FL_mul_a #1[#2]#3.#4%
2227 {%
2228 \expandafter\XINT_FL_mul_b\romannumeral0\XINTinfloatS[#3]{#4}#1[#2]%
2229 }%
2230 \def\XINT_FL_mul_b #1[#2]#3[#4]{\xintiiMul{#3}{#1}/1[#4+#2]}%

```

## 8.71 \xintFloatDiv, \XINTinFloatDiv

1.07.

Starting with 1.2f the arguments are rounded to the target precision P not P+2.

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrc, xintexpr, indices*

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.

```

2231 \def\xintFloatDiv    {\romannumeral0\xintfloatdiv    }%
2232 \def\xintfloatdiv    #1{\XINT_fldiv_chkopt \xintfloat #1\xint:}%
2233 \def\XINTinFloatDiv  {\romannumeral0\XINTinfloatdiv }%
2234 \def\XINTinfloatdiv  #1{\XINT_fldiv_chkopt \XINTinfloatS #1\xint:}%
2235 \def\XINT_fldiv_chkopt #1#2%
2236 {%
2237     \ifx [#2\expandafter\XINT_fldiv_opt
2238         \else\expandafter\XINT_fldiv_noopt
2239     \fi  #1#2%
2240 }%

2241 \def\XINT_fldiv_noopt #1#2\xint:#3%
2242 {%
2243     #1[\XINTdigits]%
2244     {\expandafter\XINT_FL_div_a
2245       \romannumeral0\XINTinfloatS[\XINTdigits]{#3}\XINTdigits.{#2}}%
2246 }%
2247 \def\XINT_fldiv_opt #1[\xint:#2]%#3#4%
2248 {%
2249     \expandafter\XINT_fldiv_opt_a\the\numexpr #2.#1%
2250 }%

2251 \def\XINT_fldiv_opt_a #1.#2#3#4%
2252 {%
2253     #2[#1]{\expandafter\XINT_FL_div_a\romannumeral0\XINTinfloatS[#1]{#4}#1.#3}}%
2254 }%
2255 \def\XINT_FL_div_a #1[#2]#3.#4%
2256 {%
2257     \expandafter\XINT_FL_div_b\romannumeral0\XINTinfloatS[#3]{#4}/#1e#2%
2258 }%

2259 \def\XINT_FL_div_b #1[#2]{#1e#2}%

```

## 8.72 `\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  $x$ ).

```

2260 \def\xintFloatPow    {\romannumeral0\xintfloatpow}%
2261 \def\xintfloatpow    #1{\XINT_flpow_chkopt \xintfloat #1\xint:}%

```

```

2262 \def\XINTinFloatPow {\romannumeral0\XINTinfloatpow}%
2263 \def\XINTinfloatpow #1{\XINT_flpow_chkopt \XINTinfloatS #1\xint:}%
2264 \def\XINT_flpow_chkopt #1#2%
2265 {%
2266   \ifx [#2\expandafter\XINT_flpow_opt
2267   \else\expandafter\XINT_flpow_noopt
2268   \fi
2269   #1#2%
2270}%
2271 \def\XINT_flpow_noopt #1#2\xint:#3%
2272 {%
2273   \expandafter\XINT_flpow_checkB_a
2274   \the\numexpr #3.\XINTdigits.{#2}{#1[\XINTdigits]]}%
2275}%
2276 \def\XINT_flpow_opt #1[\xint:#2]%
2277 {%
2278   \expandafter\XINT_flpow_opt_a\the\numexpr #2.#1%
2279}%
2280 \def\XINT_flpow_opt_a #1.#2#3#4%
2281 {%
2282   \expandafter\XINT_flpow_checkB_a\the\numexpr #4.#1.{#3}{#2[#1]]}%
2283}%
2284 \def\XINT_flpow_checkB_a #1%
2285 {%
2286   \xint_UDzerominusfork
2287   #1-\XINT_flpow_BisZero
2288   0#1{\XINT_flpow_checkB_b -}%
2289   0-{\XINT_flpow_checkB_b {}#1}%
2290   \krof
2291}%
2292 \def\XINT_flpow_BisZero .#1.#2#3{#3{1[0]}}%

2293 \def\XINT_flpow_checkB_b #1#2.#3.%
2294 {%
2295   \expandafter\XINT_flpow_checkB_c
2296   \the\numexpr\xintLength{#2}+\xint_c_iii.#3.#2.{#1}%
2297}%

2298 \def\XINT_flpow_checkB_c #1.#2.%
2299 {%
2300   \expandafter\XINT_flpow_checkB_d\the\numexpr#1+#2.#1.#2.%
2301}%

1.2f rounds input to P digits, first.

2302 \def\XINT_flpow_checkB_d #1.#2.#3.#4.#5#6%
2303 {%
2304   \expandafter \XINT_flpow_aa
2305   \romannumeral0\XINTinfloat [#3]{#6}{#2}{#1}{#4}{#5}%
2306}%

2307 \def\XINT_flpow_aa #1[#2]#3%
2308 {%

```

```

2309 \expandafter\XINT_flpow_ab\the\numexpr #2-#3\expandafter.%
2310 \romannumeral\XINT_rep #3\endcsname0.#1.%
2311 }%

2312 \def\XINT_flpow_ab #1.#2.#3.{\XINT_flpow_a #3#2[#1]}%

2313 \def\XINT_flpow_a #1%
2314 {%
2315 \xint_UDzerominusfork
2316 #1-\XINT_flpow_zero
2317 0#1{\XINT_flpow_b \iftrue}%
2318 0-{\XINT_flpow_b \iffalse#1}%
2319 \krof
2320 }%
2321 \def\XINT_flpow_zero #1[#2]#3#4#5#6%
2322 {%
2323 #6{\if 1#51\xint_dothis {0[0]}\fi
2324 \xint_orthat
2325 {\XINT_signalcondition{DivisionByZero}{0 to the power #4}{0[0]}}%
2326 }%
2327 }%

2328 \def\XINT_flpow_b #1#2[#3]#4#5%
2329 {%
2330 \XINT_flpow_loopI #5.#3.#2.#4.{#1\ifodd #5 \xint_c_i\fi\fi}%
2331 }%

2332 \def\XINT_flpow_truncate #1.#2.#3.%
2333 {%
2334 \expandafter\XINT_flpow_truncate_a
2335 \romannumeral0\XINT_split_fromleft
2336 #3.#2\xint_bye2345678\xint_bye..#1.#3.%
2337 }%

2338 \def\XINT_flpow_truncate_a #1.#2.#3.{#3+\xintLength{#2}.#1.%
2339 \def\XINT_flpow_loopI #1.%
2340 {%
2341 \ifnum #1=\xint_c_i\expandafter\XINT_flpow_ItoIII\fi
2342 \ifodd #1
2343 \expandafter\XINT_flpow_loopI_odd
2344 \else
2345 \expandafter\XINT_flpow_loopI_even
2346 \fi
2347 #1.%
2348 }%

2349 \def\XINT_flpow_ItoIII\ifodd #1\fi #2.#3.#4.#5.#6%
2350 {%
2351 \expandafter\XINT_flpow_III\the\numexpr #6+\xint_c_.#3.#4.#5.%
2352 }%

```

```

2353 \def\XINT_flpow_loopI_even #1.#2.#3.%#4.%
2354 {%
2355     \expandafter\XINT_flpow_loopI
2356     \the\numexpr #1/\xint_c_ii\expandafter.%
2357     \the\numexpr\expandafter\XINT_flpow_truncate
2358     \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.%
2359 }%
2360 \def\XINT_flpow_loopI_odd #1.#2.#3.#4.%
2361 {%
2362     \expandafter\XINT_flpow_loopII
2363     \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter.%
2364     \the\numexpr\expandafter\XINT_flpow_truncate
2365     \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#2.#3.%
2366 }%
2367 \def\XINT_flpow_loopII #1.%
2368 {%
2369     \ifnum #1 = \xint_c_i\expandafter\XINT_flpow_IItoIII\fi
2370     \ifodd #1
2371         \expandafter\XINT_flpow_loopII_odd
2372     \else
2373         \expandafter\XINT_flpow_loopII_even
2374     \fi
2375     #1.%
2376 }%
2377 \def\XINT_flpow_loopII_even #1.#2.#3.%#4.%
2378 {%
2379     \expandafter\XINT_flpow_loopII
2380     \the\numexpr #1/\xint_c_ii\expandafter.%
2381     \the\numexpr\expandafter\XINT_flpow_truncate
2382     \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.%
2383 }%
2384 \def\XINT_flpow_loopII_odd #1.#2.#3.#4.#5.#6.%
2385 {%
2386     \expandafter\XINT_flpow_loopII_odda
2387     \the\numexpr\expandafter\XINT_flpow_truncate
2388     \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2389     #1.#2.#3.%
2390 }%
2391 \def\XINT_flpow_loopII_odda #1.#2.#3.#4.#5.#6.%
2392 {%
2393     \expandafter\XINT_flpow_loopII
2394     \the\numexpr #4/\xint_c_ii-\xint_c_i\expandafter.%
2395     \the\numexpr\expandafter\XINT_flpow_truncate
2396     \the\numexpr\xint_c_ii*#5\expandafter.\romannumeral0\xintiisqr{#6}.#3.%
2397     #1.#2.%
2398 }%

2399 \def\XINT_flpow_IItoIII\ifodd #1\fi #2.#3.#4.#5.#6.#7.#8%
2400 {%
2401     \expandafter\XINT_flpow_III\the\numexpr #8+\xint_c_\expandafter.%
2402     \the\numexpr\expandafter\XINT_flpow_truncate
2403     \the\numexpr#3+#6\expandafter.\romannumeral0\xintiimul{#4}{#7}.#5.%

```

2404 }%

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).

2405 `\def\xINT_flpow_III #1.#2.#3.#4.#5%`

2406 `{%`

2407 `\expandafter\xINT_flpow_IIIend`

2408 `\xint_UDsignfork`

2409 `#5{{1/#3[-#2]}}%`

2410 `-{{#3[#2]}}%`

2411 `\krof #1%`

2412 `}%`

2413 `\def\xINT_flpow_IIIend #1#2#3%`

2414 `{#3{\if#21\xint_afterfi{\expandafter-\romannumeral`&&}\fi#1}}%`

## 8.73 `\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.

2415 `\def\xintFloatPower {\romannumeral0\xintfloatpower}%`

2416 `\def\xintfloatpower #1{\XINT_flpower_chkopt \xintfloat #1\xint:}%`

2417 `\def\XINTinFloatPower {\romannumeral0\XINTinfloatpower }%`

2418 `\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



*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices*

recognize quickly the case of integer exponent, for speed, but there is overhead of going through `\xintiTrunc1`.

```

2419 \def\xINTinFloatPowerH {\romannumeral0\xINTinfloatpowerh }%

2420 \def\xINTinfloatpowerh #1#2%
2421 {%
2422   \expandafter\xINT_flpowerh_a\romannumeral0\xintiTrunc1{#2};%
2423   \XINTdigits.{#1}\XINTinfloatS[\XINTdigits]}%
2424 }%

2425 \def\xINT_flpowerh_a #1;%
2426 {%
2427   \if0\xintLDg{#1}\expandafter\xINT_flpowerh_int
2428   \else\expandafter\xINT_flpowerh_b
2429   \fi #1.%
2430 }%
2431 \def\xINT_flpowerh_int #1%
2432 {%
2433   \if0#1\expandafter\xINT_flpower_BisZero
2434   \else\expandafter\xINT_flpowerh_i
2435   \fi #1%
2436 }%
2437 \def\xINT_flpowerh_i #10.{\expandafter\xINT_flpower_checkB_a#1.}%
2438 \def\xINT_flpowerh_b #1.%
2439 {%
2440   \expandafter\xINT_flpowerh_c\romannumeral0\xintdsrr{\xintDouble{#1}}.%
2441 }%
2442 \def\xINT_flpowerh_c #1.%
2443 {%
2444   \ifodd\xintLDg{#1} %<- intentional space
2445   \expandafter\xINT_flpowerh_d\else\expandafter\xINT_flpowerh_e
2446   \fi #1.%
2447 }%
2448 \def\xINT_flpowerh_d #1.\XINTdigits.#2#3%
2449 {%
2450   \XINT_flpower_checkB_a #1.\XINTdigits.{#2}\XINT_flpowerh_finish
2451 }%
2452 \def\xINT_flpowerh_finish #1%
2453   {\XINTinfloatS[\XINTdigits]{\XINTinFloatSqrt[\XINTdigits+\xint_c_iii]{#1}}}%
2454 \def\xINT_flpowerh_e #1.%
2455   {\expandafter\xINT_flpower_checkB_a\romannumeral0\xinthalff{#1}.}%

```

Start of macro. Check for optional argument.

```

2456 \def\xINT_flpower_chkopt #1#2%
2457 {%
2458   \ifx [#2\expandafter\xINT_flpower_opt
2459   \else\expandafter\xINT_flpower_noopt
2460   \fi
2461   #1#2%
2462 }%

```

```

2463 \def\XINT_flpower_noopt #1#2\xint:#3%
2464 {%
2465   \expandafter\XINT_flpower_checkB_a
2466   \romannumeral0\xintnum{#3}.\XINTdigits.{#2}{#1[\XINTdigits]]}%
2467 }%
2468 \def\XINT_flpower_opt #1[\xint:#2]%
2469 {%
2470   \expandafter\XINT_flpower_opt_a\the\numexpr #2.#1%
2471 }%
2472 \def\XINT_flpower_opt_a #1.#2#3#4%
2473 {%
2474   \expandafter\XINT_flpower_checkB_a
2475   \romannumeral0\xintnum{#4}.#1.{#3}{#2[#1]]}%
2476 }%
2477 \def\XINT_flpower_checkB_a #1%
2478 {%
2479   \xint_UDzerominusfork
2480   #1-{\XINT_flpower_BisZero 0}%
2481   0#1{\XINT_flpower_checkB_b -}%
2482   0-{\XINT_flpower_checkB_b }{#1}%
2483   \krof
2484 }%
2485 \def\XINT_flpower_BisZero 0.#1.#2#3{#3{1[0]}}%
2486 \def\XINT_flpower_checkB_b #1#2.#3.%
2487 {%
2488   \expandafter\XINT_flpower_checkB_c
2489   \the\numexpr\xintLength{#2}+\xint_c_iii.#3.#2.{#1}%
2490 }%

2491 \def\XINT_flpower_checkB_c #1.#2.%
2492 {%
2493   \expandafter\XINT_flpower_checkB_d\the\numexpr#1+#2.#1.#2.%
2494 }%

2495 \def\XINT_flpower_checkB_d #1.#2.#3.#4.#5#6%
2496 {%
2497   \expandafter \XINT_flpower_aa
2498   \romannumeral0\XINTinfloat [#3]{#6}{#2}{#1}{#4}{#5}%
2499 }%

2500 \def\XINT_flpower_aa #1[#2]#3%
2501 {%
2502   \expandafter\XINT_flpower_ab\the\numexpr #2-#3\expandafter.%
2503   \romannumeral\XINT_rep #3\endcsname0.#1.%
2504 }%
2505 \def\XINT_flpower_ab #1.#2.#3.{\XINT_flpower_a #3#2[#1]}%
2506 \def\XINT_flpower_a #1%
2507 {%
2508   \xint_UDzerominusfork
2509   #1-\XINT_flpow_zero
2510   0#1{\XINT_flpower_b \iftrue}%

```

```

2511      0-{\XINT_flpower_b \iffalse#1}%
2512      \krof
2513 }%
2514 \def\XINT_flpower_b #1#2[#3]#4#5%
2515 {%
2516      \XINT_flpower_loopI #5.#3.#2.#4.{#1\xintiiOdd{#5}\fi}%
2517 }%
2518 \def\XINT_flpower_loopI #1.%
2519 {%
2520      \if1\XINT_isOne {#1}\xint_dothis\XINT_flpower_ItoIII\fi
2521      \ifodd\xintLDg{#1} %<- intentional space
2522      \xint_dothis{\expandafter\XINT_flpower_loopI_odd}\fi
2523      \xint_orthat{\expandafter\XINT_flpower_loopI_even}%

2524      \romannumeral0\XINT_half
2525      #1\xint_bye\xint_Bye345678\xint_bye
2526      *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax.%
2527 }%
2528 \def\XINT_flpower_ItoIII #1.#2.#3.#4.#5%
2529 {%
2530      \expandafter\XINT_flpow_III\the\numexpr #5+\xint_c_.#2.#3.#4.%
2531 }%
2532 \def\XINT_flpower_loopI_even #1.#2.#3.#4.%
2533 {%
2534      \expandafter\XINT_flpower_toloopI
2535      \the\numexpr\expandafter\XINT_flpow_truncate
2536      \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#1.%
2537 }%
2538 \def\XINT_flpower_toloopI #1.#2.#3.#4.{\XINT_flpower_loopI #4.#1.#2.#3.}%
2539 \def\XINT_flpower_loopI_odd #1.#2.#3.#4.%
2540 {%
2541      \expandafter\XINT_flpower_toloopII
2542      \the\numexpr\expandafter\XINT_flpow_truncate
2543      \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.%
2544      #1.#2.#3.%
2545 }%
2546 \def\XINT_flpower_toloopII #1.#2.#3.#4.{\XINT_flpower_loopII #4.#1.#2.#3.}%
2547 \def\XINT_flpower_loopII #1.%
2548 {%
2549      \if1\XINT_isOne{#1}\xint_dothis\XINT_flpower_ItoIII\fi
2550      \ifodd\xintLDg{#1} %<- intentional space
2551      \xint_dothis{\expandafter\XINT_flpower_loopII_odd}\fi
2552      \xint_orthat{\expandafter\XINT_flpower_loopII_even}%

2553      \romannumeral0\XINT_half#1\xint_bye\xint_Bye345678\xint_bye
2554      *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax.%
2555 }%
2556 \def\XINT_flpower_loopII_even #1.#2.#3.#4.%
2557 {%
2558      \expandafter\XINT_flpower_toloopII
2559      \the\numexpr\expandafter\XINT_flpow_truncate
2560      \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#1.%

```

```

2561 }%
2562 \def\XINT_flpower_loopII_odd #1.#2.#3.#4.#5.#6.%
2563 {%
2564     \expandafter\XINT_flpower_loopII_odda
2565     \the\numexpr\expandafter\XINT_flpow_truncate
2566     \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2567     #1.#2.#3.%
2568 }%
2569 \def\XINT_flpower_loopII_odda #1.#2.#3.#4.#5.#6.%
2570 {%
2571     \expandafter\XINT_flpower_toloopII
2572     \the\numexpr\expandafter\XINT_flpow_truncate
2573     \the\numexpr\xint_c_ii*#5\expandafter.\romannumeral0\xintiisqr{#6}.#3.%
2574     #4.#1.#2.%
2575 }%
2576 \def\XINT_flpower_IItoIII #1.#2.#3.#4.#5.#6.#7%
2577 {%
2578     \expandafter\XINT_flpow_III\the\numexpr #7+\xint_c_\expandafter.%
2579     \the\numexpr\expandafter\XINT_flpow_truncate
2580     \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2581 }%

```

## 8.74 \xintFloatFac, \XINTFloatFac

```

2582 \def\xintFloatFac    {\romannumeral0\xintfloatfac}%
2583 \def\xintfloatfac    #1{\XINT_flfac_chkopt \xintfloat #1\xint:}%
2584 \def\XINTinFloatFac  {\romannumeral0\XINTinfloatfac }%
2585 \def\XINTinfloatfac #1{\XINT_flfac_chkopt \XINTinfloat #1\xint:}%
2586 \def\XINT_flfac_chkopt #1#2%
2587 {%
2588     \ifx [#2\expandafter\XINT_flfac_opt
2589     \else\expandafter\XINT_flfac_noopt
2590     \fi
2591     #1#2%
2592 }%
2593 \def\XINT_flfac_noopt #1#2\xint:
2594 {%
2595     \expandafter\XINT_FL_fac_fork_a
2596     \the\numexpr \xintNum{#2}.\xint_c_i \XINTdigits\XINT_FL_fac_out{#1[\XINTdigits]]}%
2597 }%
2598 \def\XINT_flfac_opt #1[\xint:#2]%
2599 {%
2600     \expandafter\XINT_flfac_opt_a\the\numexpr #2.#1%
2601 }%
2602 \def\XINT_flfac_opt_a #1.#2#3%
2603 {%
2604     \expandafter\XINT_FL_fac_fork_a\the\numexpr \xintNum{#3}.\xint_c_i {#1}\XINT_FL_fac_out{#2[#1]]}%
2605 }%
2606 \def\XINT_FL_fac_fork_a #1%
2607 {%
2608     \xint_UDzerominusfork
2609     #1-\XINT_FL_fac_iszero
2610     0#1\XINT_FL_fac_isneg

```

```

2611      0-{\XINT_FL_fac_fork_b #1}%
2612      \krof
2613 }%
2614 \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.

2615 \def\XINT_FL_fac_isneg #1.#2#3#4#5%
2616 {%
2617      #5{\XINT_signalcondition{InvalidOperation}
2618              {Factorial of negative: (-#1)!}{0[0]}}%
2619 }%
2620 \def\XINT_FL_fac_fork_b #1.%
2621 {%
2622      \ifnum #1>\xint_c_x^viii_mone\xint_dothis\XINT_FL_fac_toobig\fi
2623      \ifnum #1>\xint_c_x^iv\xint_dothis\XINT_FL_fac_vbig \fi
2624      \ifnum #1>465 \xint_dothis\XINT_FL_fac_big\fi
2625      \ifnum #1>101 \xint_dothis\XINT_FL_fac_med\fi
2626              \xint_orthat\XINT_FL_fac_small
2627      #1.%
2628 }%
2629 \def\XINT_FL_fac_toobig #1.#2#3#4#5%
2630 {%
2631      #5{\XINT_signalcondition{InvalidOperation}
2632              {Factorial of too big: (#1)!}{0[0]}}%
2633 }%

```

Computations are done with  $Q$  blocks of eight digits. When a multiplication has a carry, hence creates  $Q+1$  blocks, the least significant one is dropped. The goal is to compute an approximate value  $X'$  to the exact value  $X$ , such that the final relative error  $(X-X')/X$  will be at most  $10^{-P}$  with  $P$  the desired precision. Then, when we round  $X'$  to  $X''$  with  $P$  significant digits, we can prove that the absolute error  $|X-X''|$  is bounded (strictly) by  $0.6 \text{ ulp}(X'')$ . (ulp= unit in the last (significant) place). Let  $N$  be the number of such operations, the formula for  $Q$  deduces from the previous explanations is that  $8Q$  should be at least  $P+9+k$ , with  $k$  the number of digits of  $N$  (in base 10). Note that 1.2 version used  $P+10+k$ , for 1.2f I reduced to  $P+9+k$ . Also,  $k$  should be the number of digits of the number  $N$  of multiplications done, hence for  $n \leq 10000$  we can take  $N=n/2$ , or  $N/3$ , or  $N/4$ . This is rounded above by numexpr and always an overestimate of the actual number of approximate multiplications done (the first ones are exact). (vérifier ce que je raconte, j'ai la flemme là).

We then want  $\text{ceil}((P+k+n)/8)$ . Using \numexpr rounding division (ARRRRRGGGHHHH), if  $m$  is a positive integer,  $\text{ceil}(m/8)$  can be computed as  $(m+3)/8$ . Thus with  $m=P+10+k$ , this gives  $Q < -(P+13+k)/8$ . 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\_increasP 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.

```

2634 \def\XINT_FL_fac_vbig
2635   {\expandafter\XINT_FL_fac_vbigloop_a
2636    \the\numexpr \XINT_FL_fac_increasP \xint_c_i }%
2637 \def\XINT_FL_fac_big
2638   {\expandafter\XINT_FL_fac_bigloop_a
2639    \the\numexpr \XINT_FL_fac_increasP \xint_c_ii }%
2640 \def\XINT_FL_fac_med
2641   {\expandafter\XINT_FL_fac_medloop_a
2642    \the\numexpr \XINT_FL_fac_increasP \xint_c_iii }%
2643 \def\XINT_FL_fac_small
2644   {\expandafter\XINT_FL_fac_smallloop_a
2645    \the\numexpr \XINT_FL_fac_increasP \xint_c_iv }%
2646 \def\XINT_FL_fac_increasP #1#2.#3#4%
2647 {%
2648   #2\expandafter.\the\numexpr\xint_c_viii*%
2649   ((\xint_c_iv+#4+\expandafter\XINT_FL_fac_countdigits
2650    \the\numexpr #2/(#1*#3)\relax 87654321\Z)/\xint_c_viii).%
2651 }%
2652 \def\XINT_FL_fac_countdigits #1#2#3#4#5#6#7#8{\XINT_FL_fac_countdone }%
2653 \def\XINT_FL_fac_countdone #1#2\Z {#1}%
2654 \def\XINT_FL_fac_out #1;![#2]#3%
2655   {#3{\romannumeral0\XINT_mul_out
2656    #1;!1\R!1\R!1\R!1\R!%
2657    1\R!1\R!1\R!1\R!\W [#2]}}%
2658 \def\XINT_FL_fac_vbigloop_a #1.#2.%
2659 {%
2660   \XINT_FL_fac_bigloop_a \xint_c_x^iv.#2.%
2661   {\expandafter\XINT_FL_fac_vbigloop_loop\the\numexpr 100010001\expandafter.%
2662    \the\numexpr \xint_c_x^viii+#1.}%
2663 }%
2664 \def\XINT_FL_fac_vbigloop_loop #1.#2.%
2665 {%
2666   \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2667   \expandafter\XINT_FL_fac_vbigloop_loop
2668   \the\numexpr #1+\xint_c_i\expandafter.%
2669   \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_mul #1!%
2670 }%
2671 \def\XINT_FL_fac_bigloop_a #1.%
2672 {%
2673   \expandafter\XINT_FL_fac_bigloop_b \the\numexpr
2674   #1+\xint_c_i-\xint_c_ii*((#1-464)/\xint_c_ii).#1.%
2675 }%
2676 \def\XINT_FL_fac_bigloop_b #1.#2.#3.%
2677 {%
2678   \expandafter\XINT_FL_fac_medloop_a
2679   \the\numexpr #1-\xint_c_i.#3.{\XINT_FL_fac_bigloop_loop #1.#2.}%
2680 }%
2681 \def\XINT_FL_fac_bigloop_loop #1.#2.%
2682 {%
2683   \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi

```

```

2684 \expandafter\XINT_FL_fac_bigloop_loop
2685 \the\numexpr #1+\xint_c_ii\expandafter.%
2686 \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_bigloop_mul #1!%
2687 }%
2688 \def\XINT_FL_fac_bigloop_mul #1!%
2689 {%
2690 \expandafter\XINT_FL_fac_mul
2691 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2692 }%
2693 \def\XINT_FL_fac_medloop_a #1.%
2694 {%
2695 \expandafter\XINT_FL_fac_medloop_b
2696 \the\numexpr #1+\xint_c_i-\xint_c_iii*((#1-100)/\xint_c_iii).#1.%
2697 }%
2698 \def\XINT_FL_fac_medloop_b #1.#2.#3.%
2699 {%
2700 \expandafter\XINT_FL_fac_smallloop_a
2701 \the\numexpr #1-\xint_c_i.#3.{\XINT_FL_fac_medloop_loop #1.#2.}%
2702 }%
2703 \def\XINT_FL_fac_medloop_loop #1.#2.%
2704 {%
2705 \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2706 \expandafter\XINT_FL_fac_medloop_loop
2707 \the\numexpr #1+\xint_c_iii\expandafter.%
2708 \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_medloop_mul #1!%
2709 }%
2710 \def\XINT_FL_fac_medloop_mul #1!%
2711 {%
2712 \expandafter\XINT_FL_fac_mul
2713 \the\numexpr
2714 \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2715 }%
2716 \def\XINT_FL_fac_smallloop_a #1.%
2717 {%
2718 \csname
2719 XINT_FL_fac_smallloop_\the\numexpr #1-\xint_c_iv*(#1/\xint_c_iv)\relax
2720 \endcsname #1.%
2721 }%
2722 \expandafter\def\csname XINT_FL_fac_smallloop_1\endcsname #1.#2.%
2723 {%
2724 \XINT_FL_fac_addzeros #2.100000001!.{2.#1.}{#2}%
2725 }%
2726 \expandafter\def\csname XINT_FL_fac_smallloop_2\endcsname #1.#2.%
2727 {%
2728 \XINT_FL_fac_addzeros #2.100000002!.{3.#1.}{#2}%
2729 }%
2730 \expandafter\def\csname XINT_FL_fac_smallloop_3\endcsname #1.#2.%
2731 {%
2732 \XINT_FL_fac_addzeros #2.100000003!.{4.#1.}{#2}%
2733 }%
2734 \expandafter\def\csname XINT_FL_fac_smallloop_4\endcsname #1.#2.%
2735 {%

```

```

2736 \XINT_FL_fac_addzeros #2.1000000024!.{5.#1.}{#2}%
2737 }%
2738 \def\XINT_FL_fac_addzeros #1.%
2739 {%
2740 \ifnum #1=\xint_c_viii \expandafter\XINT_FL_fac_addzeros_exit\fi
2741 \expandafter\XINT_FL_fac_addzeros
2742 \the\numexpr #1-\xint_c_viii.1000000000!%
2743 }%

```

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.

```

2744 \def\XINT_FL_fac_addzeros_exit #1.#2.#3#4{\XINT_FL_fac_smallloop_loop #3#21;![ -#4]}%
2745 \def\XINT_FL_fac_smallloop_loop #1.#2.%
2746 {%
2747 \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2748 \expandafter\XINT_FL_fac_smallloop_loop
2749 \the\numexpr #1+\xint_c_iv\expandafter.%
2750 \the\numexpr #2\expandafter.\romannumeral0\XINT_FL_fac_smallloop_mul #1!%
2751 }%
2752 \def\XINT_FL_fac_smallloop_mul #1!%
2753 {%
2754 \expandafter\XINT_FL_fac_mul
2755 \the\numexpr
2756 \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2757 }%[[
2758 \def\XINT_FL_fac_loop_exit #1!#2]#3{#3#2]}%
2759 \def\XINT_FL_fac_mul 1#1!%
2760 {\expandafter\XINT_FL_fac_mul_a\the\numexpr\XINT_FL_fac_smallmul 10!{#1}}%
2761 \def\XINT_FL_fac_mul_a #1-#2%
2762 {%
2763 \if#21\xint_afterfi{\expandafter\space\xint_gob_til_exclam}\else
2764 \expandafter\space\fi #11;!%
2765 }%
2766 \def\XINT_FL_fac_minimulwc_a #1#2#3#4#5!#6#7#8#9%
2767 {%
2768 \XINT_FL_fac_minimulwc_b {#1#2#3#4}{#5}{#6#7#8#9}%
2769 }%
2770 \def\XINT_FL_fac_minimulwc_b #1#2#3#4!#5%
2771 {%
2772 \expandafter\XINT_FL_fac_minimulwc_c
2773 \the\numexpr \xint_c_x^ix+#5+#2*#4!{{#1}{#2}{#3}{#4}}%
2774 }%
2775 \def\XINT_FL_fac_minimulwc_c 1#1#2#3#4#5#6!#7%
2776 {%
2777 \expandafter\XINT_FL_fac_minimulwc_d {#1#2#3#4#5}#7{#6}%
2778 }%
2779 \def\XINT_FL_fac_minimulwc_d #1#2#3#4#5%
2780 {%
2781 \expandafter\XINT_FL_fac_minimulwc_e
2782 \the\numexpr \xint_c_x^ix+#1+#2*#5+#3*#4!{{#2}{#4}}%
2783 }%
2784 \def\XINT_FL_fac_minimulwc_e 1#1#2#3#4#5#6!#7#8#9%

```



```

2785 {%
2786     1#6#9\expandafter!%
2787     \the\numexpr\expandafter\XINT_FL_fac_smallmul
2788     \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#7*#8!%
2789 }%
2790 \def\XINT_FL_fac_smallmul 1#1!#21#3!%
2791 {%
2792     \xint_gob_til_sc #3\XINT_FL_fac_smallmul_end;%
2793     \XINT_FL_fac_minimulwc_a #2!#3!{#1}{#2}%
2794 }%

```

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.)

```

2795 \def\XINT_FL_fac_smallmul_end;\XINT_FL_fac_minimulwc_a #1!;!#2#3[#4]%
2796 {%
2797     \ifnum #2=\xint_c_
2798         \expandafter\xint_firstoftwo\else
2799         \expandafter\xint_secondoftwo
2800     \fi
2801     {-2\relax[#4]}%
2802     {1#2\expandafter!\expandafter-\expandafter1\expandafter
2803         [\the\numexpr #4+\xint_c_viii]}%
2804 }%

```

## 8.75 `\xintFloatPFactorial`, `\XINTinFloatPFactorial`

2015/11/29 for 1.2f. Partial factorial `pfactorial(a,b)=(a+1)...`, only for non-negative integers with `a<=b<10^8`.

1.2h (2016/11/20) now avoids raising `\xintError:OutOfRangePFac` if the condition `0<=a<=b<10^8` is violated. Same as for `\xintiPFactorial`.

```

2805 \def\xintFloatPFactorial {\romannumeral0\xintfloatpfactorial}%
2806 \def\xintfloatpfactorial #1{\XINT_flpfac_chkopt \xintfloat #1\xint:}%
2807 \def\XINTinFloatPFactorial {\romannumeral0\XINTinfloatpfactorial}%
2808 \def\XINTinfloatpfactorial #1{\XINT_flpfac_chkopt \XINTinfloat #1\xint:}%
2809 \def\XINT_flpfac_chkopt #1#2%
2810 {%
2811     \ifx [#2\expandafter\XINT_flpfac_opt
2812         \else\expandafter\XINT_flpfac_noopt
2813     \fi
2814     #1#2%
2815 }%
2816 \def\XINT_flpfac_noopt #1#2\xint:#3%
2817 {%
2818     \expandafter\XINT_FL_pfac_fork
2819     \the\numexpr \xintNum{#2}\expandafter.%
2820     \the\numexpr \xintNum{#3}.\xint_c_i{\XINTdigits}{#1[\XINTdigits]}%
2821 }%
2822 \def\XINT_flpfac_opt #1[\xint:#2]%
2823 {%
2824     \expandafter\XINT_flpfac_opt_b\the\numexpr #2.#1%

```

```

2825 }%
2826 \def\XINT_flgfac_opt_b #1.#2#3#4%
2827 {%
2828   \expandafter\XINT_FL_pfac_fork
2829   \the\numexpr \xintNum{#3}\expandafter.%
2830   \the\numexpr \xintNum{#4}.\xint_c_i{#1}{#2[#1]}%
2831 }%
2832 \def\XINT_FL_pfac_fork #1#2.#3#4.%
2833 {%
2834   \unless\ifnum #1#2<#3#4 \xint_dothis\XINT_FL_pfac_one\fi
2835   \if-#3\xint_dothis\XINT_FL_pfac_neg \fi
2836   \if-#1\xint_dothis\XINT_FL_pfac_zero\fi
2837   \ifnum #3#4>\xint_c_x^viii_mone\xint_dothis\XINT_FL_pfac_outofrange\fi
2838   \xint_orthat \XINT_FL_pfac_increaseP #1#2.#3#4.%
2839 }%
2840 \def\XINT_FL_pfac_outofrange #1.#2.#3#4#5%
2841 {%
2842   #5{\XINT_signalcondition{InvalidOperation}
2843     {pfactorial second arg too big: 99999999 < #2}}{0[0]}}%
2844 }%
2845 \def\XINT_FL_pfac_one #1.#2.#3#4#5{#5{1[0]}}%
2846 \def\XINT_FL_pfac_zero #1.#2.#3#4#5{#5{0[0]}}%
2847 \def\XINT_FL_pfac_neg -#1.-#2.%
2848 {%
2849   \ifnum #1>\xint_c_x^viii\xint_dothis\XINT_FL_pfac_outofrange\fi
2850   \xint_orthat {%
2851     \ifodd\numexpr#2-#1\relax\xint_afterfi{\expandafter-\romannumeral`&&@}\fi
2852     \expandafter\XINT_FL_pfac_increaseP}%
2853   \the\numexpr #2-\xint_c_i\expandafter.\the\numexpr#1-\xint_c_i.%
2854 }%

```

See the comments for `\XINT_FL_pfac_increaseP`. Case of  $b=a+1$  should be filtered out perhaps. We only needed here to copy the `\xintPFactorial` macros and re-use `\XINT_FL_fac_mul/\XINT_FL_fac_out`. Had to modify a bit `\XINT_FL_pfac_addzeroes`. We can enter here directly with #3 equal to specify the precision (the calculated value before final rounding has a relative error less than  $3 \cdot 10^{-(\#4-1)}$ ), and #5 would hold the macro doing the final rounding (or truncating, if I make a `FloatTrunc` available) to a given number of digits, possibly not #4. By default the #3 is 1, but `FloatBinomial` calls it with #3=4.

```

2855 \def\XINT_FL_pfac_increaseP #1.#2.#3#4%
2856 {%
2857   \expandafter\XINT_FL_pfac_a
2858   \the\numexpr \xint_c_viii*((\xint_c_iv+#4)\expandafter
2859     \XINT_FL_fac_countdigits\the\numexpr (#2-#1-\xint_c_i)%
2860     /\ifnum #2>\xint_c_x^iv #3\else(#3*\xint_c_ii)\fi\relax
2861     87654321\Z)/\xint_c_viii).#1.#2.%
2862 }%
2863 \def\XINT_FL_pfac_a #1.#2.#3.%
2864 {%
2865   \expandafter\XINT_FL_pfac_b\the\numexpr \xint_c_i+#2\expandafter.%
2866   \the\numexpr#3\expandafter.%
2867   \romannumeral0\XINT_FL_pfac_addzeroes #1.100000001!1;![ -#1]%
2868 }%

```

```

2869 \def\XINT_FL_pfac_addzeroes #1.%
2870 {%
2871     \ifnum #1=\xint_c_viii \expandafter\XINT_FL_pfac_addzeroes_exit\fi
2872     \expandafter\XINT_FL_pfac_addzeroes\the\numexpr #1-\xint_c_viii.1000000000!%
2873 }%
2874 \def\XINT_FL_pfac_addzeroes_exit #1.{ }%
2875 \def\XINT_FL_pfac_b #1.%
2876 {%
2877     \ifnum #1>9999 \xint_dothis\XINT_FL_pfac_vbigloop \fi
2878     \ifnum #1>463 \xint_dothis\XINT_FL_pfac_bigloop \fi
2879     \ifnum #1>98 \xint_dothis\XINT_FL_pfac_medloop \fi
2880     \xint_orthat\XINT_FL_pfac_smallloop #1.%
2881 }%
2882 \def\XINT_FL_pfac_smallloop #1.#2.%
2883 {%
2884     \ifcase\numexpr #2-#1\relax
2885         \expandafter\XINT_FL_pfac_end_
2886     \or \expandafter\XINT_FL_pfac_end_i
2887     \or \expandafter\XINT_FL_pfac_end_ii
2888     \or \expandafter\XINT_FL_pfac_end_iii
2889     \else\expandafter\XINT_FL_pfac_smallloop_a
2890     \fi #1.#2.%
2891 }%
2892 \def\XINT_FL_pfac_smallloop_a #1.#2.%
2893 {%
2894     \expandafter\XINT_FL_pfac_smallloop_b
2895     \the\numexpr #1+\xint_c_iv\expandafter.%
2896     \the\numexpr #2\expandafter.%
2897     \romannumeral0\expandafter\XINT_FL_fac_mul
2898     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2899 }%
2900 \def\XINT_FL_pfac_smallloop_b #1.%
2901 {%
2902     \ifnum #1>98 \expandafter\XINT_FL_pfac_medloop \else
2903         \expandafter\XINT_FL_pfac_smallloop \fi #1.%
2904 }%
2905 \def\XINT_FL_pfac_medloop #1.#2.%
2906 {%
2907     \ifcase\numexpr #2-#1\relax
2908         \expandafter\XINT_FL_pfac_end_
2909     \or \expandafter\XINT_FL_pfac_end_i
2910     \or \expandafter\XINT_FL_pfac_end_ii
2911     \else\expandafter\XINT_FL_pfac_medloop_a
2912     \fi #1.#2.%
2913 }%
2914 \def\XINT_FL_pfac_medloop_a #1.#2.%
2915 {%
2916     \expandafter\XINT_FL_pfac_medloop_b
2917     \the\numexpr #1+\xint_c_iii\expandafter.%
2918     \the\numexpr #2\expandafter.%
2919     \romannumeral0\expandafter\XINT_FL_fac_mul
2920     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%

```

```

2921 }%
2922 \def\XINT_FL_pfac_medloop_b #1.%
2923 {%
2924     \ifnum #1>463 \expandafter\XINT_FL_pfac_bigloop \else
2925         \expandafter\XINT_FL_pfac_medloop \fi #1.%
2926 }%
2927 \def\XINT_FL_pfac_bigloop #1.#2.%
2928 {%
2929     \ifcase\numexpr #2-#1\relax
2930         \expandafter\XINT_FL_pfac_end_
2931     \or \expandafter\XINT_FL_pfac_end_i
2932     \else\expandafter\XINT_FL_pfac_bigloop_a
2933     \fi #1.#2.%
2934 }%
2935 \def\XINT_FL_pfac_bigloop_a #1.#2.%
2936 {%
2937     \expandafter\XINT_FL_pfac_bigloop_b
2938     \the\numexpr #1+\xint_c_ii\expandafter.%
2939     \the\numexpr #2\expandafter.%
2940     \romannumeral0\expandafter\XINT_FL_fac_mul
2941     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!\%
2942 }%
2943 \def\XINT_FL_pfac_bigloop_b #1.%
2944 {%
2945     \ifnum #1>9999 \expandafter\XINT_FL_pfac_vbigloop \else
2946         \expandafter\XINT_FL_pfac_bigloop \fi #1.%
2947 }%
2948 \def\XINT_FL_pfac_vbigloop #1.#2.%
2949 {%
2950     \ifnum #2=#1
2951         \expandafter\XINT_FL_pfac_end_
2952     \else\expandafter\XINT_FL_pfac_vbigloop_a
2953     \fi #1.#2.%
2954 }%
2955 \def\XINT_FL_pfac_vbigloop_a #1.#2.%
2956 {%
2957     \expandafter\XINT_FL_pfac_vbigloop
2958     \the\numexpr #1+\xint_c_i\expandafter.%
2959     \the\numexpr #2\expandafter.%
2960     \romannumeral0\expandafter\XINT_FL_fac_mul
2961     \the\numexpr\xint_c_x^viii+#1!\%
2962 }%
2963 \def\XINT_FL_pfac_end_iii #1.#2.%
2964 {%
2965     \expandafter\XINT_FL_fac_out
2966     \romannumeral0\expandafter\XINT_FL_fac_mul
2967     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!\%
2968 }%
2969 \def\XINT_FL_pfac_end_ii #1.#2.%
2970 {%
2971     \expandafter\XINT_FL_fac_out
2972     \romannumeral0\expandafter\XINT_FL_fac_mul

```

```

2973 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2974 }%
2975 \def\XINT_FL_pfac_end_i #1.#2.%
2976 {%
2977 \expandafter\XINT_FL_fac_out
2978 \romannumeral0\expandafter\XINT_FL_fac_mul
2979 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2980 }%
2981 \def\XINT_FL_pfac_end_ #1.#2.%
2982 {%
2983 \expandafter\XINT_FL_fac_out
2984 \romannumeral0\expandafter\XINT_FL_fac_mul
2985 \the\numexpr \xint_c_x^viii+#1!%
2986 }%

```

## 8.76 \xintFloatBinomial, \XINTinFloatBinomial

1.2f. We compute  $\text{binomial}(x,y)$  as  $\text{pfac}(x-y,x)/y!$ , where the numerator and denominator are computed with a relative error at most  $4 \cdot 10^{-P-2}$ , then rounded (once I have a float truncation, I will use truncation rather) to  $P+3$  digits, and finally the quotient is correctly rounded to  $P$  digits. This will guarantee that the exact value  $X$  differs from the computed one  $Y$  by at most  $0.6 \text{ ulp}(Y)$ . (2015/12/01).

2016/11/19 for 1.2h. As for `\xintiiBinomial`, hard to understand why last year I coded this to raise an error if  $y < 0$  or  $y > x$  ! The question of the Gamma function is for another occasion, here  $x$  and  $y$  must be (small) integers.

```

2987 \def\xintFloatBinomial {\romannumeral0\xintfloatbinomial}%
2988 \def\xintfloatbinomial #1{\XINT_flbinom_chkopt \xintfloat #1\xint:}%
2989 \def\XINTinFloatBinomial {\romannumeral0\XINTinfloatbinomial}%
2990 \def\XINTinfloatbinomial #1{\XINT_flbinom_chkopt \XINTinfloat #1\xint:}%
2991 \def\XINT_flbinom_chkopt #1#2%
2992 {%
2993 \ifx [#2\expandafter\XINT_flbinom_opt
2994 \else\expandafter\XINT_flbinom_noopt
2995 \fi #1#2%
2996 }%
2997 \def\XINT_flbinom_noopt #1#2\xint:#3%
2998 {%
2999 \expandafter\XINT_FL_binom_a
3000 \the\numexpr\xintNum{#2}\expandafter.\the\numexpr\xintNum{#3}.\XINTdigits.#1%
3001 }%
3002 \def\XINT_flbinom_opt #1[\xint:#2]#3#4%
3003 {%
3004 \expandafter\XINT_FL_binom_a
3005 \the\numexpr\xintNum{#3}\expandafter.\the\numexpr\xintNum{#4}\expandafter.%
3006 \the\numexpr #2.#1%
3007 }%
3008 \def\XINT_FL_binom_a #1.#2.%
3009 {%
3010 \expandafter\XINT_FL_binom_fork \the\numexpr #1-#2.#2.#1.%
3011 }%
3012 \def\XINT_FL_binom_fork #1#2.#3#4.#5#6.%
3013 {%

```

```

3014 \if-#5\xint_dothis \XINT_FL_binom_neg\fi
3015 \if-#1\xint_dothis \XINT_FL_binom_zero\fi
3016 \if-#3\xint_dothis \XINT_FL_binom_zero\fi
3017 \if0#1\xint_dothis \XINT_FL_binom_one\fi
3018 \if0#3\xint_dothis \XINT_FL_binom_one\fi
3019 \ifnum #5#6>\xint_c_x^viii_mone \xint_dothis\XINT_FL_binom_toobig\fi
3020 \ifnum #1#2>#3#4 \xint_dothis\XINT_FL_binom_ab \fi
3021 \xint_orthat\XINT_FL_binom_aa
3022 #1#2.#3#4.#5#6.%
3023 }%
3024 \def\XINT_FL_binom_neg #1.#2.#3.#4.#5%
3025 {%
3026 #5[#4]{\XINT_signalcondition{InvalidOperation}
3027 {binomial with first arg negative: #3}{\{0[0]\}}}%
3028 }%
3029 \def\XINT_FL_binom_toobig #1.#2.#3.#4.#5%
3030 {%
3031 #5[#4]{\XINT_signalcondition{InvalidOperation}
3032 {binomial with first arg too big: 99999999 < #3}{\{0[0]\}}}%
3033 }%
3034 \def\XINT_FL_binom_one #1.#2.#3.#4.#5{#5[#4]{1[0]}}%
3035 \def\XINT_FL_binom_zero #1.#2.#3.#4.#5{#5[#4]{0[0]}}%
3036 \def\XINT_FL_binom_aa #1.#2.#3.#4.#5%
3037 {%
3038 #5[#4]{\xintDiv{\XINT_FL_pfac_increaseP
3039 #2.#3.\xint_c_iv{#4+\xint_c_i}{\XINTinfloat[#4+\xint_c_iii]}}%
3040 {\XINT_FL_fac_fork_b
3041 #1.\xint_c_iv{#4+\xint_c_i}\XINT_FL_fac_out{\XINTinfloat[#4+\xint_c_iii]}}}%
3042 }%
3043 \def\XINT_FL_binom_ab #1.#2.#3.#4.#5%
3044 {%
3045 #5[#4]{\xintDiv{\XINT_FL_pfac_increaseP
3046 #1.#3.\xint_c_iv{#4+\xint_c_i}{\XINTinfloat[#4+\xint_c_iii]}}%
3047 {\XINT_FL_fac_fork_b
3048 #2.\xint_c_iv{#4+\xint_c_i}\XINT_FL_fac_out{\XINTinfloat[#4+\xint_c_iii]}}}%
3049 }%

```

## 8.77 \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.

```

3050 \def\xintFloatSqrt      {\romannumeral0\xintfloatsqr }%
3051 \def\xintfloatsqr      #1{\XINT_flsqr_chkopt \xintfloat #1\xint:}%
3052 \def\XINTinFloatSqr    {\romannumeral0\XINTinfloatsqr }%
3053 \def\XINTinfloatsqr    #1{\XINT_flsqr_chkopt \XINTinfloat #1\xint:}%
3054 \def\XINT_flsqr_chkopt #1#2%
3055 {%
3056   \ifx [#2\expandafter\XINT_flsqr_opt
3057     \else\expandafter\XINT_flsqr_noopt
3058   \fi #1#2%
3059 }%
3060 \def\XINT_flsqr_noopt #1#2\xint:%
3061 {%
3062   \expandafter\XINT_FL_sqrt_a
3063     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.#1%
3064 }%
3065 \def\XINT_flsqr_opt #1[\xint:#2]%#3%
3066 {%
3067   \expandafter\XINT_flsqr_opt_a\the\numexpr #2.#1%
3068 }%
3069 \def\XINT_flsqr_opt_a #1.#2#3%
3070 {%
3071   \expandafter\XINT_FL_sqrt_a\romannumeral0\XINTinfloat[#1]{#3}#1.#2%
3072 }%
3073 \def\XINT_FL_sqrt_a #1%
3074 {%
3075   \xint_UDzerominusfork
3076   #1-\XINT_FL_sqrt_iszero
3077   0#1\XINT_FL_sqrt_isneg
3078   0-\XINT_FL_sqrt_pos #1}%
3079 \krof
3080 }%[
3081 \def\XINT_FL_sqrt_iszero #1]#2.#3{#3[#2]{0[0]}}%
3082 \def\XINT_FL_sqrt_isneg #1]#2.#3%
3083 {%
3084   #3[#2]{\XINT_signalcondition{InvalidOperation}
3085     {Square root of negative: -#1]}}{0[0]}}%
3086 }%

3087 \def\XINT_FL_sqrt_pos #1[#2]#3.%
3088 {%
3089   \expandafter\XINT_flsqr
3090   \the\numexpr #3\ifodd #2 \xint_dothis {+\xint_c_iii.(#2+\xint_c_i).0}\fi
3091   \xint_orthat {+\xint_c_ii.#2.{}}#100.#3.%
3092 }%

3093 \def\XINT_flsqr #1.#2.%
3094 {%
3095   \expandafter\XINT_flsqr_a
3096   \the\numexpr #2/\xint_c_ii-(#1-\xint_c_i)/\xint_c_ii.#1.%
3097 }%

```

```

3098 \def\XINT_flsqrt_a #1.#2.#3#4.#5.%
3099 {%
3100     \expandafter\XINT_flsqrt_b
3101     \the\numexpr (#2-\xint_c_i)/\xint_c_ii\expandafter.%
3102     \romannumeral0\XINT_sqrt_start #2.#4#3.#5.#2.#4#3.#5.#1.%
3103 }%

3104 \def\XINT_flsqrt_b #1.#2#3%
3105 {%
3106     \expandafter\XINT_flsqrt_c
3107     \romannumeral0\xintiisub
3108     {\XINT_dsx_addzeros {#1}#2;}%
3109     {\xintiiDivRound{\XINT_dsx_addzeros {#1}#3;}%
3110         {\XINT_dbl#2\xint_bye2345678\xint_bye*\xint_c_ii\relax}}.%
3111 }%

3112 \def\XINT_flsqrt_c #1.#2.%
3113 {%
3114     \expandafter\XINT_flsqrt_d
3115     \romannumeral0\XINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..%
3116 }%

3117 \def\XINT_flsqrt_d #1.#2#3.%
3118 {%
3119     \ifnum #2=\xint_c_v
3120     \expandafter\XINT_flsqrt_f\else\expandafter\XINT_flsqrt_finish\fi
3121     #2#3.#1.%
3122 }%

3123 \def\XINT_flsqrt_finish #1#2.#3.#4.#5.#6.#7.#8{#8[#6]{#3#1[#7]}}%

3124 \def\XINT_flsqrt_f 5#1.%
3125     {\expandafter\XINT_flsqrt_g\romannumeral0\xintinum{#1}\relax.}%
3126 \def\XINT_flsqrt_g #1#2#3.{\if\relax#2\xint_dothis{\XINT_flsqrt_h #1}\fi
3127     \xint_orthat{\XINT_flsqrt_finish 5.}}%
3128 \def\XINT_flsqrt_h #1{\ifnum #1<\xint_c_iii\xint_dothis{\XINT_flsqrt_again}\fi
3129     \xint_orthat{\XINT_flsqrt_finish 5.}}%

3130 \def\XINT_flsqrt_again #1.#2.%
3131 {%
3132     \expandafter\XINT_flsqrt_again_a\the\numexpr #2+\xint_c_viii.%
3133 }%

3134 \def\XINT_flsqrt_again_a #1.#2.#3.%
3135 {%
3136     \expandafter\XINT_flsqrt_b
3137     \the\numexpr (#1-\xint_c_i)/\xint_c_ii\expandafter.%
3138     \romannumeral0\XINT_sqrt_start #1.#2000000000.#3.%
3139     #1.#2000000000.#3.%
3140 }%

```



## 8.78 \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].

```

3141 \def\xintFloatE    {\romannumeral0\xintfloate }%
3142 \def\xintfloate #1{\XINT_float_chkopt #1\xint:}%
3143 \def\XINT_float_chkopt #1%
3144 {%
3145     \ifx [#1\expandafter\XINT_float_opt
3146         \else\expandafter\XINT_float_noopt
3147         \fi #1%
3148 }%
3149 \def\XINT_float_noopt #1\xint:%
3150 {%
3151     \expandafter\XINT_float_post
3152     \romannumeral0\XINTinfloat[\XINTdigits]{#1}\XINTdigits.%
3153 }%
3154 \def\XINT_float_opt [\xint:#1]%
3155 {%
3156     \expandafter\XINT_float_opt_a\the\numexpr #1.%
3157 }%
3158 \def\XINT_float_opt_a #1.#2%
3159 {%
3160     \expandafter\XINT_float_post
3161     \romannumeral0\XINTinfloat[#1]{#2}#1.%
3162 }%
3163 \def\XINT_float_post #1%
3164 {%
3165     \xint_UDzerominusfork
3166     #1-\XINT_float_zero
3167     0#1\XINT_float_neg
3168     0-\XINT_float_pos
3169     \krof #1%
3170 }%[
3171 \def\XINT_float_zero #1]#2.#3{ 0.e0}%
3172 \def\XINT_float_neg-{\expandafter-\romannumeral0\XINT_float_pos}%

3173 \def\XINT_float_pos #1#2[#3]#4.#5%
3174 {%
3175     \expandafter\XINT_float_pos_done\the\numexpr#3+#4+#5-\xint_c_i.#1.#2;%
3176 }%
3177 \def\XINTinFloatE {\romannumeral0\XINTinfloate }%
3178 \def\XINTinfloate
3179     {\expandafter\XINT_infloate\romannumeral0\XINTinfloat[\XINTdigits]}%
3180 \def\XINT_infloate #1[#2]#3%
3181     {\expandafter\XINT_infloate_end\the\numexpr #3+#2.{#1}}%
3182 \def\XINT_infloate_end #1.#2{ #2[#1]}%
```

## 8.79 \XINTinFloatMod

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrc](#), [xintexpr](#), [indices](#)

1.1. Pour emploi dans xintexpr. Code shortened at 1.2p.

```
3183 \def\XINTinFloatMod {\romannumeral0\XINTinfloatmod [\XINTdigits]]}%
3184 \def\XINTinfloatmod [#1]#2#3%
3185 {%
3186     \XINTinfloat[#1]{\xintMod
3187         {\romannumeral0\XINTinfloat[#1]{#2}}}%
3188         {\romannumeral0\XINTinfloat[#1]{#3}}}%
3189 }%
```

## 8.80 \XINTinFloatDivFloor

1.2p. Formerly // and /: in \xintfloatexpr used \xintDivFloor and \xintMod, hence did not round their operands to float precision beforehand.

```
3190 \def\XINTinFloatDivFloor {\romannumeral0\XINTinfloatdivfloor [\XINTdigits]]}%
3191 \def\XINTinfloatdivfloor [#1]#2#3%
3192 {%
3193     \xintdivfloor
3194         {\romannumeral0\XINTinfloat[#1]{#2}}}%
3195         {\romannumeral0\XINTinfloat[#1]{#3}}}%
3196 }%
```

## 8.81 \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.

```
3197 \def\XINTinFloatDivMod {\romannumeral0\XINTinfloatdivmod [\XINTdigits]]}%
3198 \def\XINTinfloatdivmod [#1]#2#3%
3199 {%
3200     \expandafter\XINT_infloatdivmod
3201     \romannumeral0\xintdivmod
3202         {\romannumeral0\XINTinfloat[#1]{#2}}}%
3203         {\romannumeral0\XINTinfloat[#1]{#3}}}%
3204     {#1}%
3205 }%
3206 \def\XINT_infloatdivmod #1#2#3{ #1,\XINTinFloat[#3]{#2}}%
```

## 8.82 \xintiffloatInt

1.3a for ifint() function in \xintfloatexpr.

```
3207 \def\xintiffloatInt {\romannumeral0\xintiffloatint}%
3208 \def\xintiffloatint #1{\expandafter\XINT_iffloatint
3209     \romannumeral0\xintrez{\XINTinFloat[\XINTdigits]{#1}}}%
3210 \def\XINT_iffloatint #1#2/1[#3]%
3211 {%
3212     \if 0#1\xint_dothis\xint_stop_atfirstoftwo\fi
3213     \ifnum#3<\xint_c\xint_dothis\xint_stop_atsecondoftwo\fi
3214     \xint_orthat\xint_stop_atfirstoftwo
3215 }%
```

## 8.83 \xintFloatIsInt

1.3d for `isint()` function in `\xintfloatexpr`.

```
3216 \def\xintFloatIsInt {\romannumeral0\xintfloatisint}%
3217 \def\xintfloatisint #1{\expandafter\xint_iffloatint
3218     \romannumeral0\xintrez{\XINTinFloat[\XINTdigits]{#1}}10}%
```

## 8.84 (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.

```
3219 \def\XINTinRandomFloatS{\romannumeral0\XINTinrandomfloatS}%
3220 \def\XINTinRandomFloatSdigits{\XINTinRandomFloatS[\XINTdigits]}%
3221 \def\XINTinrandomfloatS[#1]%
3222 {%
3223     \expandafter\xint_inrandomfloatS\the\numexpr\xint_c-(#1)\xint:
3224 }%
3225 \def\XINT_inrandomfloatS-#1\xint:
3226 {%
3227     \expandafter\XINT_inrandomfloatS_a
3228     \romannumeral0\xinrandomdigits{#1}[-#1]%
3229 }%
```

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.

```
3230 \def\XINT_inrandomfloatS_a#1%
3231 {%
3232     \if#10\xint_dothis{\XINT_inrandomfloatS_b}\fi
3233     \xint_orthat{ #1}%
3234 }%[
3235 \def\XINT_inrandomfloatS_b#1%
3236 {%
3237     \if#1[\xint_dothis{\XINT_inrandomfloatS_zero}\fi% ]
3238     \if#10\xint_dothis{\XINT_inrandomfloatS_b}\fi
3239     \xint_orthat{ #1}%
3240 }%[
3241 \def\XINT_inrandomfloatS_zero#1]{ 0[0]}%
```

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

## 8.85 (WIP) \XINTinRandomFloatSixteen

1.3b. Support for `grand()` function.

```
3242 \def\XINTinRandomFloatSixteen%
3243 {%
3244     \romannumeral0\expandafter\XINT_inrandomfloatS_a
3245     \romannumeral`&&\expandafter\XINT_eightrandomdigits
3246         \romannumeral`&&\XINT_eightrandomdigits[-16]%
3247 }%
3248 \XINT_restorecatcodes_endinput%
```

## 9 Package [xintseries](#) implementation

.1	Catcodes, $\varepsilon$ -TeX and reload detection . . .	261	.7	<code>\xintRationalSeries</code> . . . . .	264
.2	Package identification . . . . .	262	.8	<code>\xintRationalSeriesX</code> . . . . .	265
.3	<code>\xintSeries</code> . . . . .	262	.9	<code>\xintFxFtPowerSeries</code> . . . . .	266
.4	<code>\xintiSeries</code> . . . . .	262	.10	<code>\xintFxFtPowerSeriesX</code> . . . . .	267
.5	<code>\xintPowerSeries</code> . . . . .	263	.11	<code>\xintFloatPowerSeries</code> . . . . .	267
.6	<code>\xintPowerSeriesX</code> . . . . .	264	.12	<code>\xintFloatPowerSeriesX</code> . . . . .	269

The commenting is currently (2019/01/06) 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 [2019/01/06 1.3d 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_iseries\expandafter
77   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
78 }%
79 \def\XINT_iseries #1#2#3%
80 {%
81   \ifnum #2<#1
82     \xint_afterfi { 0}%
83   \else

```

```

84     \xint_afterfi {\XINT_iseries_loop {#1}{0}{#2}{#3}}%
85   \fi
86 }%
87 \def\XINT_iseries_loop #1#2#3#4%
88 {%
89   \ifnum #3>#1 \else \XINT_iseries_exit \fi
90   \expandafter\XINT_iseries_loop\expandafter
91   {\the\numexpr #1+1\expandafter }\expandafter
92   {\romannumeral0\xintiadd {#2}{#4{#1}}}%
93   {#3}{#4}%
94 }%
95 \def\XINT_iseries_exit \fi #1#2#3#4#5#6#7#8%
96 {%
97   \fi\xint_gobble_ii #6%
98 }%

```

## 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 {%

```

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices*

```

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,rationfunction,x*

This computes  $F(a,x)+\dots+F(b,x)$  on the basis of the value of  $F(a,x)$  and the ratios  $F(n,x)/F(n-1,x)$ . The argument  $x$  is first expanded and it is the value resulting from this which is used then throughout. The initial term  $F(a,x)$  must be defined as one-parameter macro which will be given  $x$ . Modified in 1.06 to give the indices first to a `\numexpr` rather than expanding twice. I just use `\the\numexpr` and maintain the previous code after that. 1.08a adds the forgotten optimization following that previous change.

```

189 \def\xintRationalSeriesX {\romannumeral0\xintratseriesx}%
190 \def\xintratseriesx #1#2%
191 {%
192   \expandafter\XINT_ratseriesx\expandafter
193   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
194}%
195 \def\XINT_ratseriesx #1#2#3#4#5%
196 {%
197   \ifnum #2<#1
198     \xint_afterfi { 0/1[0]}%
199   \else
200     \xint_afterfi
201     {\expandafter\XINT_ratseriesx_pre\expandafter
202      {\romannumeral`&&@#5}{#2}{#1}{#4}{#3}}%
203  }%

```

```

204 \fi
205 }%
206 \def\XINT_ratseriesx_pre #1#2#3#4#5%
207 {%
208 \XINT_ratseries_loop {#2}{1}{#3}{#4{#1}}{#5{#1}}%
209 }%

```

## 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\xintiiadd {#4}{\xintiTrunc {#7}{\xintMul {#6{#2}}{#1}}}}%
251     {#1}{#3}{#5}{#6}{#7}%
252 }%
253 \def\xINT_fppowseries_exit_i\fi\expandafter\xINT_fppowseries_loop_ii
254     {\fi \expandafter\xINT_fppowseries_exit_ii }%
255 \def\xINT_fppowseries_exit_ii #1#2#3#4#5#6#7%
256 {%
257     \xinttrunc {#7}
258     {\xintiiadd {#4}{\xintiTrunc {#7}{\xintMul {#6{#2}}{#1}}}{-#7}}%
259 }%

```

## 9.10 \xintFxFtPowerSeriesX

*a,b,coeff,x,D*

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

```

260 \def\xintFxFtPowerSeriesX {\romannumeral0\xintfxptpowerseriesx }%
261 \def\xintfxptpowerseriesx #1#2%
262 {%
263     \expandafter\xINT_fppowseriesx\expandafter
264     {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
265 }%
266 \def\xINT_fppowseriesx #1#2#3#4#5%
267 {%
268     \ifnum #2<#1
269         \xint_afterfi { 0}%
270     \else
271         \xint_afterfi
272         {\expandafter \XINT_fppowseriesx_pre \expandafter
273         {\romannumeral`&&@#4}{#1}{#2}{#3}{#5}%
274         }%
275     \fi
276 }%
277 \def\xINT_fppowseriesx_pre #1#2#3#4#5%
278 {%
279     \expandafter\xINT_fppowseries_loop_pre\expandafter
280     {\romannumeral0\xinttrunc {#5}{\xintPow {#1}{#2}}}%
281     {#2}{#1}{#3}{#4}{#5}%
282 }%

```

## 9.11 \xintFloatPowerSeries

1.08a. I still have to re-visit \xintFxFtPowerSeries; temporarily I just adapted the code to the case of floats.

```

283 \def\xintFloatPowerSeries {\romannumeral0\xintfloatpowerseries }%
284 \def\xintfloatpowerseries #1{\XINT_flpowseries_chkopt #1\xint:}%
285 \def\xINT_flpowseries_chkopt #1%
286 {%
287     \ifx [#1\expandafter\xINT_flpowseries_opt

```

```

288     \else\expandafter\XINT_flpowseries_noopt
289     \fi
290     #1%
291 }%
292 \def\XINT_flpowseries_noopt #1\xint:#2%
293 {%
294     \expandafter\XINT_flpowseries\expandafter
295     {\the\numexpr #1\expandafter}\expandafter
296     {\the\numexpr #2}\XINTdigits
297 }%
298 \def\XINT_flpowseries_opt [\xint:#1]#2#3%
299 {%
300     \expandafter\XINT_flpowseries\expandafter
301     {\the\numexpr #2\expandafter}\expandafter
302     {\the\numexpr #3\expandafter}{\the\numexpr #1}%
303 }%
304 \def\XINT_flpowseries #1#2#3#4#5%
305 {%
306     \ifnum #2<#1
307         \xint_afterfi { 0.e0}%
308     \else
309         \xint_afterfi
310         {\expandafter\XINT_flpowseries_loop_pre\expandafter
311         {\romannumeral0\XINTinfloatpow [#3]{#5}{#1}}%
312         {#1}{#5}{#2}{#4}{#3}%
313         }%
314     \fi
315 }%
316 \def\XINT_flpowseries_loop_pre #1#2#3#4#5#6%
317 {%
318     \ifnum #4>#2 \else\XINT_flpowseries_dont_i \fi
319     \expandafter\XINT_flpowseries_loop_i\expandafter
320     {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
321     {\romannumeral0\XINTinfloatmul [#6]{#5}{#2}}{#1}%
322     {#1}{#3}{#4}{#5}{#6}%
323 }%
324 \def\XINT_flpowseries_dont_i \fi\expandafter\XINT_flpowseries_loop_i
325     {\fi \expandafter\XINT_flpowseries_dont_ii }%
326 \def\XINT_flpowseries_dont_ii #1#2#3#4#5#6#7{\xintfloat [#7]{#2}}%
327 \def\XINT_flpowseries_loop_i #1#2#3#4#5#6#7%
328 {%
329     \ifnum #5>#1 \else \XINT_flpowseries_exit_i \fi
330     \expandafter\XINT_flpowseries_loop_ii\expandafter
331     {\romannumeral0\XINTinfloatmul [#7]{#3}{#4}}%
332     {#1}{#4}{#2}{#5}{#6}{#7}%
333 }%
334 \def\XINT_flpowseries_loop_ii #1#2#3#4#5#6#7%
335 {%
336     \expandafter\XINT_flpowseries_loop_i\expandafter
337     {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
338     {\romannumeral0\XINTinfloatadd [#7]{#4}%
339     {\XINTinfloatmul [#7]{#6}{#2}}{#1}}}%

```

```

340      {#1}{#3}{#5}{#6}{#7}%
341 }%
342 \def\XINT_flpowseries_exit_i\fi\expandafter\XINT_flpowseries_loop_ii
343   {\fi \expandafter\XINT_flpowseries_exit_ii }%
344 \def\XINT_flpowseries_exit_ii #1#2#3#4#5#6#7%
345 {%
346   \xintfloatadd [#7]{#4}{\XINTinfloatmul [#7]{#6{#2}}{#1}}%
347 }%

```

## 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

.1	Catcodes, $\varepsilon$ -TeX and reload detection . . .	270	.16	<code>\xintiGctoF</code> . . . . .	282
.2	Package identification . . . . .	271	.17	<code>\xintCtoCv</code> , <code>\xintCstoCv</code> . . . . .	283
.3	<code>\xintCFrac</code> . . . . .	271	.18	<code>\xintiCstoCv</code> . . . . .	284
.4	<code>\xintGCFrac</code> . . . . .	272	.19	<code>\xintGctoCv</code> . . . . .	284
.5	<code>\xintGGCFrac</code> . . . . .	274	.20	<code>\xintiGctoCv</code> . . . . .	286
.6	<code>\xintGctoGCx</code> . . . . .	275	.21	<code>\xintFtoCv</code> . . . . .	287
.7	<code>\xintFtoCs</code> . . . . .	275	.22	<code>\xintFtoCCv</code> . . . . .	287
.8	<code>\xintFtoCx</code> . . . . .	276	.23	<code>\xintCntoF</code> . . . . .	287
.9	<code>\xintFtoC</code> . . . . .	276	.24	<code>\xintGCntoF</code> . . . . .	288
.10	<code>\xintFtoGC</code> . . . . .	277	.25	<code>\xintCntoCs</code> . . . . .	289
.11	<code>\xintFGtoC</code> . . . . .	277	.26	<code>\xintCntoGC</code> . . . . .	289
.12	<code>\xintFtoCC</code> . . . . .	278	.27	<code>\xintGCntoGC</code> . . . . .	290
.13	<code>\xintCtoF</code> , <code>\xintCstoF</code> . . . . .	279	.28	<code>\xintCstoGC</code> . . . . .	291
.14	<code>\xintiCstoF</code> . . . . .	280	.29	<code>\xintGctoGC</code> . . . . .	291
.15	<code>\xintGctoF</code> . . . . .	280			

The commenting is currently (2019/01/06) very sparse. Release 1.09m (2014/02/26) has modified a few things: `\xintFtoCs` and `\xintCntoCs` insert spaces after the commas, `\xintCstoF` and `\xintCstoCv` authorize spaces in the input also before the commas, `\xintCntoCs` does not brace the produced coefficients, new macros `\xintFtoC`, `\xintCtoF`, `\xintCtoCv`, `\xintFGtoC`, and `\xintGGCFrac`.

There is partial dependency on [xinttools](#) due to `\xintCstoF` and `\xintCsToCv`.

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

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

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

```

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

```

```

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

```

## 10.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintcfrac}%
46 [2019/01/06 1.3d Expandable continued fractions with xint package (JFB)]%

```

## 10.3 \xintCFrac

```

47 \def\xintCFrac {\romannumeral0\xintcfrac}%
48 \def\xintcfrac #1%
49 {%
50   \XINT_cfrac_opt_a #1\xint:
51 }%
52 \def\XINT_cfrac_opt_a #1%
53 {%
54   \ifx[#1\XINT_cfrac_opt_b\fi \XINT_cfrac_noopt #1%
55 }%
56 \def\XINT_cfrac_noopt #1\xint:
57 {%
58   \expandafter\XINT_cfrac_A\romannumeral0\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{#11#2}{#3}}%
115 }%
116 \def\XINT_cfrac_end\Z\XINT_cfrac_T #1#2#3%
117 {%
118    \XINT_cfrac_end_b #3%
119 }%
120 \def\XINT_cfrac_end_b \Z+\cfrac#1#2{ #2}%

```

#### 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\xintrawwithzeros {#1}\Z
246 }%
247 \def\XINT_ftc_A #1/#2\Z
248 {%
249     \expandafter\XINT_ftc_B\romannumeral0\xintiidivision {#1}{#2}{#2}%
250 }%
251 \def\XINT_ftc_B #1#2%
252 {%
253     \XINT_ftc_C #2.{#1}%
254 }%
255 \def\XINT_ftc_C #1%
256 {%
257     \xint_gob_til_zero #1\XINT_ftc_integer 0\XINT_ftc_D #1%
258 }%
259 \def\XINT_ftc_integer 0\XINT_ftc_D 0#1.#2#3{ #2}%
260 \def\XINT_ftc_D #1.#2#3{\XINT_ftc_loop_a {#1}{#3}{#1}{#2, }}% 1.09m adds a space
261 \def\XINT_ftc_loop_a
262 {%
263     \expandafter\XINT_ftc_loop_d\romannumeral0\XINT_div_prepare
264 }%
265 \def\XINT_ftc_loop_d #1#2%
266 {%
267     \XINT_ftc_loop_e #2.{#1}%
268 }%
269 \def\XINT_ftc_loop_e #1%
```

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices*

```

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 {}

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfraction](#), [xintexpr](#), [indices](#)

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

```

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

```

## 10.12 \xintFtoCC

```

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

```

```

409 }%
410 \def\XINT_ftcc_loop_c #1#2%
411 {%
412     \expandafter\XINT_ftcc_loop_d
413     \romannumeral0\xintsub {#2}{#1[0]}\Z {#1}%
414 }%
415 \def\XINT_ftcc_loop_d #1%
416 {%
417     \xint_UDzerominusfork
418     #1-\XINT_ftcc_end
419     0#1\XINT_ftcc_loop_N
420     0-\XINT_ftcc_loop_P #1}%
421 \krof
422 }%
423 \def\XINT_ftcc_end #1\Z #2#3{ #3#2}%
424 \def\XINT_ftcc_loop_P #1\Z #2#3%
425 {%
426     \expandafter\XINT_ftcc_loop_a\expandafter
427     {\romannumeral0\xintdiv {1[0]}\{#1}\{#3#2+1/}%
428 }%
429 \def\XINT_ftcc_loop_N #1\Z #2#3%
430 {%
431     \expandafter\XINT_ftcc_loop_a\expandafter
432     {\romannumeral0\xintdiv {1[0]}\{#1}\{#3#2+-1/}%
433 }%

```

### 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{\xintrowwithzeros {#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\xintraawithzeros {#1/#2}}{#5}{#1}{#2}{#3}{#4}%
659 }%
660 \def\XINT_ctcv_loop_g #1#2{\XINT_ctcv_loop_a {#2{#1}}}% 1.09b removes [0]
661 \def\XINT_ctcv_end #1.#2#3#4#5#6{ #6}%

```

## 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\xintraawithzeros {#1/#2}}{#1}{#2}}%
692 }%
693 \def\XINT_icstcv_loop_e #1#2#3#4{\XINT_icstcv_loop_a {#4{#1}}#2#3}%
694 \def\XINT_icstcv_end #1.#2#3#4#5#6{ #6}% 1.09b removes [0]

```

## 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 \xintiGtoCv

```

768 \def\xintiGtoCv {\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}}%

```

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices*

```

805 }%
806 \def\XINT_igctcv_loop_h #1#2%
807 {%
808     \expandafter\XINT_igctcv_loop_i\expandafter {\expandafter{#2}{#1}}%
809 }%
810 \def\XINT_igctcv_loop_i #1#2{\XINT_igctcv_loop_k #2{#2#1}}%
811 \def\XINT_igctcv_loop_k #1#2%
812 {%
813     \expandafter\XINT_igctcv_loop_l\expandafter
814     {\romannumeral0\xintraawithzeros {#1/#2}}%
815 }%
816 \def\XINT_igctcv_loop_l #1#2#3{\XINT_igctcv_loop_a {#3{#1}}#2}%1.09i removes [0]
817 \def\XINT_igctcv_end_a #1.#2#3#4#5%
818 {%
819     \expandafter\XINT_igctcv_end_b\expandafter
820     {\romannumeral0\xintraawithzeros {#2/#3}}%
821 }%
822 \def\XINT_igctcv_end_b #1#2{ #2{#1}}% 1.09b removes [0]

```

## 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}%

```



*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrac, xintexpr, indices*

```

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, indices*

```
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 . . . . .	294
11.2	Catcodes, $\epsilon$ -TeX and reload detection . . . . .	295
11.3	Package identification . . . . .	296
11.4	Locking and unlocking . . . . .	296
11.5	<code>\XINT_expr_wrap</code> , <code>\XINT_iexpr_wrap</code> . . . . .	297
11.6	<code>\XINT_protectii</code> , <code>\XINT_expr_usethe</code> . . . . .	297
11.7	<code>\XINT_expr_print</code> , <code>\XINT_iexpr_print</code> , <code>\XINT_boolexpr_print</code> . . . . .	297
11.8	<code>\xintexpr</code> , <code>\xintiexpr</code> , <code>\xintfloatexpr</code> , <code>\xintiiepr</code> . . . . .	297
11.9	<code>\xinttheexpr</code> , <code>\xinttheiexpr</code> , <code>\xintthefloatexpr</code> , <code>\xinttheiiepr</code> . . . . .	297
11.10	<code>\xinteval</code> , <code>\xintieval</code> , <code>\xintfloateval</code> , <code>\xintiieval</code> . . . . .	297
11.11	<code>\xintthe</code> , <code>\xintthe_o</code> . . . . .	297
11.12	<code>\thexintexpr</code> , <code>\thexintiexpr</code> , <code>\thexintfloatexpr</code> , <code>\thexintiiepr</code> . . . . .	298
11.13	<code>\xintthecoords</code> . . . . .	298
11.14	<code>\xintbareeval</code> , <code>\xintbarefloateval</code> , <code>\xintbareiieval</code> . . . . .	298
11.15	<code>\xintthebareeval</code> , <code>\xintthebarefloateval</code> , <code>\xintthebareiieval</code> . . . . .	298
11.16	<code>\xintexprpro</code> , <code>\xintiexprpro</code> . . . . .	298
11.17	<code>\xintiexprpro</code> , <code>\XINT_iexpr_wrap</code> . . . . .	299
11.18	<code>\xintfloatexprpro</code> , <code>\XINT_flexpr_wrap</code> , <code>\XINT_flexpr_print</code> . . . . .	299
11.19	<code>\xintboolexpr</code> , <code>\xinttheboolexpr</code> , <code>\thexintboolexpr</code> . . . . .	300
11.20	<code>\xintifboolexpr</code> , <code>\xintifboolfloatexpr</code> , <code>\xintifbooliiepr</code> . . . . .	300
11.21	<code>\xintifsgnexpr</code> , <code>\xintifsgnfloatexpr</code> , <code>\xintifsgniiepr</code> . . . . .	300
11.22	Hooks for the functioning of <code>\xintNewExpr</code> and <code>\xintdeffunc</code> . . . . .	300
11.23	Macros handling csv lists on output (for <code>\XINT_expr_print</code> et al. routines) . . . . .	300
11.24	<code>\XINT_expr_getnext</code> : fetching some number then an operator . . . . .	302
11.25	<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 . . . . .	303
11.26	<code>\XINT_expr_getop</code> : finding the next operator or closing parenthesis or end of expression . . . . .	310
11.27	Expansion spanning; opening and closing parentheses . . . . .	312
11.28	<code> </code> , <code>  </code> , <code>&amp;</code> , <code>&amp;&amp;</code> , <code>&lt;</code> , <code>&gt;</code> , <code>=</code> , <code>==</code> , <code>&lt;=</code> , <code>&gt;=</code> , <code>!=</code> , <code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>^</code> , <code>**</code> , <code>//</code> , <code>/:</code> , <code>..</code> , <code>..[</code> , <code>]</code> , <code>]</code> , <code>]</code> , <code>]</code> , <code>]</code> , and <code>++</code> operators . . . . .	313
11.29	Macros for list selectors: <code>[list][N]</code> , <code>[list][:b]</code> , <code>[list][a:]</code> , <code>[list][a:b]</code> . . . . .	319
11.30	Macros for a..b list generation . . . . .	325
11.31	Macros for a..[d]..b list generation . . . . .	326
11.32	The comma as binary operator . . . . .	328
11.33	The minus as prefix operator of variable precedence level . . . . .	329
11.34	<code>?</code> as two-way and <code>??</code> as three-way conditionals with braced branches . . . . .	330
11.35	<code>!</code> as postfix factorial operator . . . . .	330
11.36	The A/B[N] mechanism . . . . .	331
11.37	<code>\XINT_expr_op_`</code> for recognizing functions . . . . .	331
11.38	The <code>\bool()</code> , <code>\togl()</code> , <code>\protect()</code> pseudo “functions” . . . . .	332
11.39	The <code>\break()</code> function . . . . .	332
11.40	The <code>\qraw()</code> , <code>\qint()</code> , <code>\qfrac()</code> , and <code>\qfloat()</code> “functions” . . . . .	332
11.41	The <code>\random()</code> and <code>\grand()</code> “functions” . . . . .	332
11.42	<code>\XINT_expr_op__</code> for recognizing variables . . . . .	333
11.43	User defined variables: <code>\xintdefvar</code> , <code>\xintdefiivar</code> , <code>\xintdeffloatvar</code> . . . . .	333
11.44	<code>\xintunassignvar</code> . . . . .	335
11.45	<code>seq</code> and the implementation of dummy variables . . . . .	335
11.46	<code>\add()</code> , <code>\mul()</code> . . . . .	341

11.47	<code>\subs()</code>	342
11.48	<code>\rseq()</code>	343
11.49	<code>\iter()</code>	344
11.50	<code>\rrseq()</code>	346
11.51	<code>\iterr()</code>	348
11.52	Macros handling csv lists for functions with multiple comma separated arguments in expressions	350
11.53	Auxiliary wrappers for function macros	354
11.54	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>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	355
11.55	f-expandable versions of the <code>\xintSeqB::csv</code> and alike routines, for <code>\xintNewExpr</code>	366
11.56	User defined functions: <code>\xintdeffunc</code> , <code>\xintdefiifunc</code> , <code>\xintdeffloatfunc</code>	368
11.57	<code>\xintunassignexprfunc</code> , <code>\xintunassigniifunc</code> , <code>\xintunassignfloatexprfunc</code>	369
11.58	<code>\xintNewFunction</code>	370
11.59	<code>\xintNewExpr</code> , <code>\xintNewIExpr</code> , <code>\xintNewFloatExpr</code> , <code>\xintNewIIExpr</code>	371

This is release 1.3d of [2019/01/06].

## 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 [l3fp-parse.dtx](#) (in its version as available in April-May 2013). One will recognize in particular the idea of the 'until' macros; I have not looked into the actual [l3fp](#) code beyond the very useful comments provided in its documentation.

A main worry was that my data has no a priori bound on its size; to keep the code reasonably efficient, I experimented with a technique of storing and retrieving data expandably as *names* of control sequences. Intermediate computation results are stored as control sequences `\.=a/b[n]`.

Roughly speaking, the parser mechanism is as follows: at any given time the last found 'operator' has its associated `until` macro awaiting some news from the token flow; first `getnext` expands forward in the hope to construct some number, which may come from a parenthesized sub-expression, from some braced material, or from a digit by digit scan. After this number has been formed the next operator is looked for by the `getop` macro. Once `getop` has finished its job, `until` is presented with three tokens: the first one is the precedence level of the new found operator (which may be an end of expression marker), the second is the operator character token (earlier versions had here already some macro name, but in order to keep as much common code to `expr` and `floatexpr` common as possible, this was modified) of the new found operator, and the third one is the newly found number (which was encountered just before the new operator).

The `until` macro of the earlier operator examines the precedence level of the new found one, and either executes the earlier operator (in the case of a binary operation, with the found number and a previously stored one) or it delays execution, giving the hand to the `until` macro of the operator having been found of higher precedence.

A minus sign acting as prefix gets converted into a (unary) operator inheriting the precedence level of the previous operator.

Once the end of the expression is found (it has to be marked by a `\relax`) the final result is output as four tokens (five tokens since 1.09j) the first one a catcode 11 exclamation mark, the second one an error generating macro, the third one is a protection mechanism, the fourth one a printing macro and the fifth is `\.=a/b[n]`. The prefix `\xintthe` makes the output printable by killing the first three tokens.

## 11.2 Catcodes, $\epsilon$ -TeX and reload detection

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

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

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2  \catcode13=5      % ^^M
3  \endlinechar=13 %
4  \catcode123=1     % {
5  \catcode125=2     % }
6  \catcode64=11     % @
7  \catcode35=6      % #
8  \catcode44=12     % ,
9  \catcode45=12     % -
10 \catcode46=12     % .
11 \catcode58=12     % :
12 \def\z {\endgroup}%
13 \expandafter\let\expandafter\x\csname ver@xintexpr.sty\endcsname
14 \expandafter\let\expandafter\w\csname ver@xintfrac.sty\endcsname
15 \expandafter\let\expandafter\t\csname ver@xinttools.sty\endcsname
16 \expandafter
17   \ifx\csname PackageInfo\endcsname\relax
18     \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
19   \else
20     \def\y#1#2{\PackageInfo{#1}{#2}}%
21   \fi
22 \expandafter
23 \ifx\csname numexpr\endcsname\relax
24   \y{xintexpr}{\numexpr not available, aborting input}%
25   \aftergroup\endinput
26 \else
27   \ifx\x\relax    % plain-TeX, first loading of xintexpr.sty
28     \ifx\w\relax % but xintfrac.sty not yet loaded.
29       \expandafter\def\expandafter\z\expandafter
30         {\z\input xintfrac.sty\relax}%
31     \fi
32     \ifx\t\relax % but xinttools.sty not yet loaded.
33       \expandafter\def\expandafter\z\expandafter
34         {\z\input xinttools.sty\relax}%
35     \fi
36   \else
37     \def\empty {}%
38     \ifx\x\empty % LaTeX, first loading,
39       % variable is initialized, but \ProvidesPackage not yet seen
40       \ifx\w\relax % xintfrac.sty not yet loaded.
41         \expandafter\def\expandafter\z\expandafter
42           {\z\RequirePackage{xintfrac}}%
43       \fi
44       \ifx\t\relax % xinttools.sty not yet loaded.
45         \expandafter\def\expandafter\z\expandafter
46           {\z\RequirePackage{xinttools}}%
47       \fi
48     \else
49       \aftergroup\endinput % xintexpr already loaded.

```

```
50      \fi
51      \fi
52      \fi
53 \z%
54 \XINTsetupcatcodes%
```

### 11.3 Package identification

```
55 \XINT_providespackage
56 \ProvidesPackage{xintexpr}%
57 [2019/01/06 1.3d Expandable expression parser (JFB)]%
58 \catcode`! 11
59 \let\XINT_Cmp \xintiiCmp
```

### 11.4 Locking and unlocking

Some renaming and modifications here with release 1.2 to switch from using chains of `\romannu-  
meral-`0` in order to gather numbers, possibly hexadecimal, to using a `\csname` governed expan-  
sion. 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 `\romannu-meral` 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 `\romannu-meral-`0` had ended  
gathering digits; this uses has been replaced by direct processing inside a `\csname...\endcsname`  
and the macro is kept only for matters of dummy variables.

Currently, the parsing of hexadecimal numbers needs two nested `\csname...\endcsname`, first to  
gather the letters (possibly with a hexadecimal fractional part), and in a second stage to apply  
`\xintHexToDec` to do the actual conversion. This should be faster than updating on the fly the  
number (which would be hard for the fraction part...).

```
60 \def\xint_gob_til_! #1!{% ! with catcode 11
61 \def\XINT_expr_lockscan#1{% not used for decimal numbers in xintexpr 1.2
62 \def\XINT_expr_lockscan##1!{\expandafter#1\csname .##1\endcsname}%
63 }\XINT_expr_lockscan{ }%
64 \def\XINT_expr_lockit#1{%
65 \def\XINT_expr_lockit##1{\expandafter#1\csname .##1\endcsname}%
66 }\XINT_expr_lockit{ }%
67 \def\XINT_expr_unlock_hex_in #1% expanded inside \csname..\endcsname
68 {\expandafter\XINT_expr_inhex\romannu-meral`&&\XINT_expr_unlock#1;%}
69 \def\XINT_expr_inhex #1.#2#3;% expanded inside \csname..\endcsname
70 {%
71      \if#2>%
72      \xintHexToDec{#1}%
73      \else
74      \xintiiMul{\xintiiPow{625}{\xintLength{#3}}}{\xintHexToDec{#1#3}}%
75      [\the\numexpr-4*\xintLength{#3}]%
76      \fi
77 }%
78 \def\XINT_expr_unlock {\expandafter\XINT_expr_unlock_a\string }%
79 \def\XINT_expr_unlock_a #1.={}%
80 \def\XINT_expr_unexpectedtoken {\xintError:ignored }%
```



```
81 \let\XINT_expr_done\space
```

### 11.5 \XINT\_expr\_wrap, \XINT\_iiexpr\_wrap

```
82 \def\XINT_expr_wrap { !\XINT_expr_usethe\XINT_protectii\XINT_expr_print }%
83 \def\XINT_iiexpr_wrap { !\XINT_expr_usethe\XINT_protectii\XINT_iiexpr_print }%
```

### 11.6 \XINT\_protectii, \XINT\_expr\_usethe

```
84 \def\XINT_protectii #1{\noexpand\XINT_protectii\noexpand #1\noexpand }%
85 \protected\def\XINT_expr_usethe\XINT_protectii {\xintError:missing_xintthe!}%
```

### 11.7 \XINT\_expr\_print, \XINT\_iiexpr\_print, \XINT\_boolexpr\_print

See also the \XINT\_flexpr\_print which is special, below.

```
86 \def\XINT_expr_print #1{\xintSPraw::csv {\XINT_expr_unlock #1}}%
87 \def\XINT_iiexpr_print #1{\xintCSV::csv {\XINT_expr_unlock #1}}%
88 \def\XINT_boolexpr_print #1{\xintIsTrue::csv {\XINT_expr_unlock #1}}%
```

### 11.8 \xintexpr, \xintiexpr, \xintfloatexpr, \xintiiepr

ATTENTION! 1.3d renamed \xinteval to \xintexpro etc...

```
89 \def\xintexpr {\romannumeral0\xintexpro }%
90 \def\xintiexpr {\romannumeral0\xintiexpro }%
91 \def\xintfloatexpr {\romannumeral0\xintfloatexpro }%
92 \def\xintiiepr {\romannumeral0\xintiiepro }%
```

### 11.9 \xinttheexpr, \xinttheiexpr, \xintthefloatexpr, \xinttheiiepr

```
93 \def\xinttheexpr
94 {\romannumeral`&&\expandafter\XINT_expr_print\romannumeral0\xintbareeval}%
95 \def\xinttheiexpr
96 {\romannumeral`&&\expandafter\xint_gobble_iii\romannumeral`&&\xintiexpr}%
97 \def\xintthefloatexpr
98 {\romannumeral`&&\expandafter\xint_gobble_iii\romannumeral`&&\xintfloatexpr}%
99 \def\xinttheiiepr
100 {\romannumeral`&&\expandafter\XINT_iiexpr_print\romannumeral0\xintbareiieval}%
```

### 11.10 \xinteval, \xintieval, \xintfloateval, \xintiieval

```
101 \def\xinteval #1%
102 {\romannumeral`&&\expandafter\XINT_expr_print\romannumeral0\xintbareeval#1\relax}%
103 \def\xintieval #1%
104 {\romannumeral`&&\expandafter\xint_gobble_iii\romannumeral`&&\xintiexpr#1\relax}%
105 \def\xintfloateval #1%
106 {\romannumeral`&&\expandafter\xint_gobble_iii\romannumeral`&&\xintfloatexpr#1\relax}%
107 \def\xintiieval #1%
108 {\romannumeral`&&\expandafter\XINT_iiexpr_print\romannumeral0\xintbareiieval#1\relax}%
```

### 11.11 \xintthe, \xintthe\_o

```
109 \def\xintthe #1{\romannumeral`&&\expandafter\xint_gobble_iii\romannumeral`&&#1}%
110 \def\xintthe_o #1{\expandafter\xint_gobble_iii\romannumeral`&&#1}%
```

## 11.12 `\thexintexpr`, `\thexintiexpr`, `\thexintfloatexpr`, `\thexintiiepr`

New with 1.2h. I have been three years long very strict in terms of prefixing macros, but well.

```
111 \let\thexintexpr      \xinttheexpr
112 \let\thexintiexpr    \xinttheiexpr
113 \let\thexintfloatexpr\xintthefloatexpr
114 \let\thexintiiepr    \xinttheiiepr
```

## 11.13 `\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!

```
115 \def\xintthecoords #1{\romannumeral`&&\expandafter\expandafter\expandafter
116      \XINT_thecoords_a
117      \expandafter\xint_gobble_iii\romannumeral0#1}%
118 \def\XINT_thecoords_a #1#2% #1=print macro, indispensable for scientific notation
119   {\expandafter\XINT_expr_unlock\csname.=\expandafter\XINT_thecoords_b
120      \romannumeral`&&@#1#2,!,!,^ \endcsname }%
121 \def\XINT_thecoords_b #1#2,#3#4,%
122   {\xint_gob_til_! #3\XINT_thecoords_c ! (#1#2, #3#4)\XINT_thecoords_b }%
123 \def\XINT_thecoords_c #1^{}%
```

## 11.14 `\xintbareeval`, `\xintbarefloateval`, `\xintbareiieval`

```
124 \def\xintbareeval
125   {\expandafter\XINT_expr_until_end_a\romannumeral`&&\XINT_expr_getnext }%
126 \def\xintbarefloateval
127   {\expandafter\XINT_flexpr_until_end_a\romannumeral`&&\XINT_expr_getnext }%
128 \def\xintbareiieval
129   {\expandafter\XINT_iiexpr_until_end_a\romannumeral`&&\XINT_expr_getnext }%
```

## 11.15 `\xintthebareeval`, `\xintthebarefloateval`, `\xintthebareiieval`

```
130 \def\xintthebareeval      {\expandafter\XINT_expr_unlock\romannumeral0\xintbareeval}%
131 \def\xintthebarefloateval {\expandafter\XINT_expr_unlock\romannumeral0\xintbarefloateval}%
132 \def\xintthebareiieval    {\expandafter\XINT_expr_unlock\romannumeral0\xintbareiieval}%
```

## 11.16 `\xintexpro`, `\xintiexpro`

ATTENTION! 1.3d renamed `\xinteval` to `\xintexpro` etc...

```
133 \def\xintexpro   {\expandafter\XINT_expr_wrap\romannumeral0\xintbareeval }%
134 \def\xintiexpro  {\expandafter\XINT_iiexpr_wrap\romannumeral0\xintbareiieval }%
```

## 11.17 \xintiexpr, \XINT\_iexpr\_wrap

Optional argument since 1.1.

ATTENTION! 1.3d renamed \xinteval to \xintexpr etc...

```

135 \def\xintiexpr #1%
136   {\ifx [#1\expandafter\XINT_iexpr_withopt\else\expandafter\XINT_iexpr_noopt \fi #1}%
137 \def\XINT_iexpr_noopt
138   {\expandafter\XINT_iexpr_wrap \expandafter 0\romannumeral0\xintbareeval}%
139 \def\XINT_iexpr_withopt [#1]%
140 {%
141   \expandafter\XINT_iexpr_wrap\expandafter
142   {\the\numexpr \xint_zapspaces #1 \xint_gobble_i\expandafter}%
143   \romannumeral0\xintbareeval
144}%
145 \def\XINT_iexpr_wrap #1#2%
146 {%
147   \expandafter\XINT_expr_wrap
148   \csname .=\xintRound::csv {#1}{\XINT_expr_unlock #2}\endcsname
149}%

```

## 11.18 \xintfloatexpr, \XINT\_flexpr\_wrap, \XINT\_flexpr\_print

Optional argument since 1.1

ATTENTION! 1.3d renamed \xinteval to \xintexpr etc...

```

150 \def\xintfloatexpr #1%
151 {%
152   \ifx [#1\expandafter\XINT_flexpr_withopt_a\else\expandafter\XINT_flexpr_noopt
153   \fi #1%
154}%
155 \def\XINT_flexpr_noopt
156 {%
157   \expandafter\XINT_flexpr_withopt_b\expandafter\xinttheDigits
158   \romannumeral0\xintbarefloateval
159}%
160 \def\XINT_flexpr_withopt_a [#1]%
161 {%
162   \expandafter\XINT_flexpr_withopt_b\expandafter
163   {\the\numexpr\xint_zapspaces #1 \xint_gobble_i\expandafter}%
164   \romannumeral0\xintbarefloateval
165}%
166 \def\XINT_flexpr_withopt_b #1#2%
167 {%
168   \expandafter\XINT_flexpr_wrap\csname .;#1.=% ; and not : as before b'cause NewExpr
169   \XINTinFloat::csv {#1}{\XINT_expr_unlock #2}\endcsname
170}%
171 \def\XINT_flexpr_wrap { !\XINT_expr_usethe\XINT_protectii\XINT_flexpr_print}%
172 \def\XINT_flexpr_print #1%
173 {%
174   \expandafter\xintPFfloat::csv
175   \romannumeral`&&\expandafter\XINT_expr_unlock_sp\string #1!%

```

```
176 }%
177 \def\XINT_expr_unlock_sp #1.;#2.=#3!{{#2}{#3}}%
```

## 11.19 \xintboolexpr, \xinttheboolexpr, \thexintboolexpr

ATTENTION! 1.3d renamed \xinteval to \xintexpr etc...

```
178 \def\xintboolexpr      {\romannumeral0\expandafter\expandafter\expandafter
179   \XINT_boolexpr_done \expandafter\xint_gobble_iv\romannumeral0\xintexpr }%
180 \def\xinttheboolexpr   {\romannumeral`&&\expandafter\expandafter\expandafter
181   \XINT_boolexpr_print\expandafter\xint_gobble_iv\romannumeral0\xintexpr }%
182 \let\thexintboolexpr\xinttheboolexpr
183 \def\XINT_boolexpr_done { !\XINT_expr_usethe\XINT_protectii\XINT_boolexpr_print }%
```

## 11.20 \xintifboolexpr, \xintifboolfloatexpr, \xintifbooliiexpr

Do not work with comma separated expressions.

```
184 \def\xintifboolexpr    #1{\romannumeral0\xintiiifnotzero {\xinttheexpr #1\relax}}%
185 \def\xintifboolfloatexpr #1{\romannumeral0\xintiiifnotzero {\xintthefloatexpr #1\relax}}%
186 \def\xintifbooliiexpr  #1{\romannumeral0\xintiiifnotzero {\xinttheiiexpr #1\relax}}%
```

## 11.21 \xintifsgnexpr, \xintifsgnfloatexpr, \xintifsgniiexpr

### 1.3d.

Do not work with comma separated expressions.

```
187 \def\xintifsgnexpr     #1{\romannumeral0\xintiiifsgn {\xinttheexpr #1\relax}}%
188 \def\xintifsgnfloatexpr #1{\romannumeral0\xintiiifsgn {\xintthefloatexpr #1\relax}}%
189 \def\xintifsgniiexpr   #1{\romannumeral0\xintiiifsgn {\xinttheiiexpr #1\relax}}%
```

## 11.22 Hooks for the functioning of \xintNewExpr and \xintdeffunc

This is new with 1.3. See \XINT\_expr\_redefinemacros.

```
190 \let\XINT:NEhook:two\empty
191 \let\XINT:NEhook:one\empty
192 \let\XINT:NEhook:csv\empty
```

## 11.23 Macros handling csv lists on output (for \XINT\_expr\_print et al. routines)

11.23.1	\XINT::_end . . . . .	301
11.23.2	\xintCSV::csv . . . . .	301
11.23.3	\xintSPRaw, \xintSPRaw::csv . . . . .	301
11.23.4	\xintIsTrue::csv . . . . .	301
11.23.5	\xintRound::csv . . . . .	301
11.23.6	\XINTinFloat::csv . . . . .	301
11.23.7	\xintPFloat::csv . . . . .	302

Changed completely for 1.1, which adds the optional arguments to \xintexpr and \xintfloatexpr.

### 11.23.1 \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

```
193 \def\XINT_::_end #1,#2{\xint_gobble_i #2}%
```

### 11.23.2 \xintCSV::csv

```
194 \def\xintCSV::csv #1{\expandafter\XINT_csv::_a\romannumeral`&&@#1,^}%
195 \def\XINT_csv::_a {\XINT_csv::_b {}}%
196 \def\XINT_csv::_b #1#2,{\expandafter\XINT_csv::_c \romannumeral`&&@#2,{#1}}%
197 \def\XINT_csv::_c #1{\if ^#1\expandafter\XINT_::_end\fi\XINT_csv::_d #1}%
198 \def\XINT_csv::_d #1,#2{\XINT_csv::_b {#2, #1}}% possibly, item #1 is empty.
```

### 11.23.3 \xintSPRaw, \xintSPRaw::csv

```
199 \def\xintSPRaw {\romannumeral0\xintspraw }%
200 \def\xintspraw #1{\expandafter\XINT_spraw\romannumeral`&&@#1[\W]}%
201 \def\XINT_spraw #1[#2#3]{\xint_gob_til_W #2\XINT_spraw_a\W\XINT_spraw_p #1[#2#3]}%
202 \def\XINT_spraw_a\W\XINT_spraw_p #1[\W]{ #1}%
203 \def\XINT_spraw_p #1[\W]{\xintpraw {#1}}%
204 \def\xintSPRaw::csv #1{\romannumeral0\expandafter\XINT_spraw::_a\romannumeral`&&@#1,^}%
205 \def\XINT_spraw::_a {\XINT_spraw::_b {}}%
206 \def\XINT_spraw::_b #1#2,{\expandafter\XINT_spraw::_c \romannumeral`&&@#2,{#1}}%
207 \def\XINT_spraw::_c #1{\if ,#1\xint_dothis\XINT_spraw::_e\fi
208     \if ^#1\xint_dothis\XINT_::_end\fi
209     \xint_orthat\XINT_spraw::_d #1}%
210 \def\XINT_spraw::_d #1,{\expandafter\XINT_spraw::_e\romannumeral0\XINT_spraw #1[\W],}%
211 \def\XINT_spraw::_e #1,#2{\XINT_spraw::_b {#2, #1}}%
```

### 11.23.4 \xintIsTrue::csv

```
212 \def\xintIsTrue::csv #1{\romannumeral0\expandafter\XINT_istrue::_a\romannumeral`&&@#1,^}%
213 \def\XINT_istrue::_a {\XINT_istrue::_b {}}%
214 \def\XINT_istrue::_b #1#2,{\expandafter\XINT_istrue::_c \romannumeral`&&@#2,{#1}}%
215 \def\XINT_istrue::_c #1{\if ,#1\xint_dothis\XINT_istrue::_e\fi
216     \if ^#1\xint_dothis\XINT_::_end\fi
217     \xint_orthat\XINT_istrue::_d #1}%
218 \def\XINT_istrue::_d #1,{\expandafter\XINT_istrue::_e\romannumeral0\xintisnotzero {#1},}%
219 \def\XINT_istrue::_e #1,#2{\XINT_istrue::_b {#2, #1}}%
```

### 11.23.5 \xintRound::csv

```
220 \def\XINT_:::_end #1,#2#3{\xint_gobble_i #3}%
221 \def\xintRound::csv #1#2{\romannumeral0\expandafter\XINT_round::_b\expandafter
222     {\the\numexpr#1\expandafter}\expandafter{\expandafter}\romannumeral`&&@#2,^}%
223 \def\XINT_round::_b #1#2#3,{\expandafter\XINT_round::_c \romannumeral`&&@#3,{#1}{#2}}%
224 \def\XINT_round::_c #1{\if ,#1\xint_dothis\XINT_round::_e\fi
225     \if ^#1\xint_dothis\XINT_:::_end\fi
226     \xint_orthat\XINT_round::_d #1}%
227 \def\XINT_round::_d #1,#2{%
228     \expandafter\XINT_round::_e\romannumeral0\ifnum#2>\xint_c_
229     \expandafter\xintround\else\expandafter\xintround\fi {#2}{#1},{#2}}%
230 \def\XINT_round::_e #1,#2#3{\XINT_round::_b {#2}{#3, #1}}%
```

### 11.23.6 \XINTinFloat::csv

### 11.23.7 \xintPFloat::csv

### 11.24 \XINT\_expr\_getnext: fetching some number then an operator

```

249 \def\XINT_expr_getnext #1%
250 {%
251     \expandafter\XINT_expr_getnext_a\romannumeral`&&@#1%
252 }%
253 \def\XINT_expr_getnext_a #1%
254 {% screens out sub-expressions and \count or \dimen registers/variables
255     \xint_gob_til_! #1\XINT_expr_subexpr !% recall this ! has catcode 11
256     \ifcat\relax#1% \count or \numexpr etc... token or count, dimen, skip cs
257         \expandafter\XINT_expr_countetc
258     \else
259         \expandafter\expandafter\expandafter\XINT_expr_getnextfork\expandafter\string
260     \fi
261     #1%
262 }%
263 \def\XINT_expr_subexpr !#1\fi !{\expandafter\XINT_expr_getop\xint_gobble_iii }%

```

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

```
272 \expandafter\XINT_expr_getnext\number #1%
```

```

273 }%
274 \def\XINT_expr_unpackvar\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
275   \expandafter\XINT_expr_getnext\number #1%
276   {\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
277   \expandafter\XINT_expr_getop\csname .=\number#1\endcsname }%
278 \begingroup
279 \lccode`*=`#
280 \lowercase{\endgroup
281 \def\XINT_expr_getnextfork #1{%
282   \if#1*\xint_dothis {\XINT_expr_scan_macropar *}\fi
283   \if#1[\xint_dothis {\xint_c_xviii ({} }\fi
284   \if#1+\xint_dothis \XINT_expr_getnext \fi
285   \if#1.\xint_dothis {\XINT_expr_startdec}\fi
286   \if#1-\xint_dothis -\fi
287   \if#1(\xint_dothis {\xint_c_xviii ({} }\fi
288   \xint_orthat {\XINT_expr_scan_nbr_or_func #1}%
289 }}%
290 \def\XINT_expr_scan_macropar #1#2{\expandafter\XINT_expr_getop\csname .=#1#2\endcsname }%

```

## 11.25 \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

11.25.1	Integral part (skipping zeroes)	304
11.25.2	Fractional part	306
11.25.3	Scientific notation	306
11.25.4	Hexadecimal numbers	307
11.25.5	\XINT_expr_scanfunc: parsing names of functions and variables	309

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 `\xintfloatexpr`, 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|flexpr}_func_+` (or \*). Documentation says to use ``+`(...)`, but ``+(...)` is also valid. The opening parenthesis must be there, it is not allowed to come from expansion.

```

291 \catcode96 11 % `
292 \def\XINT_expr_scan_nbr_or_func #1% this #1 has necessarily here catcode 12
293 {%
294   \if #1\xint_dothis \XINT_expr_gotnil \fi
295   \if "#1\xint_dothis \XINT_expr_scanhex_I\fi
296   \if `#1\xint_dothis {\XINT_expr_onliteral_}\fi
297   \ifnum \xint_c_ix<1#1 \xint_dothis \XINT_expr_startint\fi
298   \xint_orthat \XINT_expr_scanfunc #1%
299 }%
300 \def\XINT_expr_gotnil{\expandafter\XINT_expr_getop\csname.= \endcsname}%
301 \def\XINT_expr_onliteral_` #1#2#3({\xint_c_xviii `{#2}}%
302 \catcode96 12 % `
303 \def\XINT_expr_startint #1%
304 {%
305   \if #10\expandafter\XINT_expr_gobz_a\else\XINT_expr_scanint_a\fi #1%
306 }%
307 \def\XINT_expr_scanint_a #1#2%
308   {\expandafter\XINT_expr_getop\csname.=#1%
309     \expandafter\XINT_expr_scanint_b\romannumeral`&&@#2}%
310 \def\XINT_expr_gobz_a #1%
311   {\expandafter\XINT_expr_getop\csname.=%
312     \expandafter\XINT_expr_gobz_scanint_b\romannumeral`&&@#1}%
313 \def\XINT_expr_startdec #1%
314   {\expandafter\XINT_expr_getop\csname.=%
315     \expandafter\XINT_expr_scandec_a\romannumeral`&&@#1}%

```

### 11.25.1 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.

```

316 \def\XINT_expr_scanint_b #1%
317 {%
318   \ifcat \relax #1\expandafter\XINT_expr_scanint_endbycs\expandafter #1\fi
319   \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanint_c\fi
320   \string#1\XINT_expr_scanint_d
321 }%
322 \def\XINT_expr_scanint_d #1%
323 {%
324   \expandafter\XINT_expr_scanint_b\romannumeral`&&@#1%
325 }%
326 \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 1.21 to ignore underscore character \_ if encountered within digits; so it can serve as separator for better readability.

```

327 \def\XINT_expr_scanint_c\string #1\XINT_expr_scanint_d
328 {%
329   \if _#1\xint_dothis\XINT_expr_scanint_d\fi
330   \if e#1\xint_dothis{[\the\numexpr0\XINT_expr_scanexp_a +}\fi
331   \if E#1\xint_dothis{[\the\numexpr0\XINT_expr_scanexp_a +}\fi
332   \if .#1\xint_dothis{\XINT_expr_startdec_a .}\fi
333   \xint_orthat {\endcsname #1}%
334 }%
335 \def\XINT_expr_startdec_a .#1%
336 {%
337   \expandafter\XINT_expr_scandec_a\romannumeral`&&@#1%
338 }%
339 \def\XINT_expr_scandec_a #1%
340 {%
341   \if .#1\xint_dothis{\endcsname.}\fi
342   \xint_orthat {\XINT_expr_scandec_b 0.#1}%
343 }%
344 \def\XINT_expr_gobz_scanint_b #1%
345 {%
346   \ifcat \relax #1\expandafter\XINT_expr_gobz_scanint_endbycs\expandafter #1\fi
347   \ifnum\xint_c_x<1\string#1 \else\expandafter\XINT_expr_gobz_scanint_c\fi
348   \string#1\XINT_expr_scanint_d
349 }%
350 \def\XINT_expr_gobz_scanint_endbycs#1#2\XINT_expr_scanint_d{0\endcsname #1}%
351 \def\XINT_expr_gobz_scanint_c\string #1\XINT_expr_scanint_d
352 {%
353   \if _#1\xint_dothis\XINT_expr_gobz_scanint_d\fi
354   \if e#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
355   \if E#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
356   \if .#1\xint_dothis{\XINT_expr_gobz_startdec_a .}\fi
357   \if 0#1\xint_dothis\XINT_expr_gobz_scanint_d\fi
358   \xint_orthat {0\endcsname #1}%
359 }%
360 \def\XINT_expr_gobz_scanint_d #1%
361 {%
362   \expandafter\XINT_expr_gobz_scanint_b\romannumeral`&&@#1%
363 }%
364 \def\XINT_expr_gobz_startdec_a .#1%
365 {%
366   \expandafter\XINT_expr_gobz_scandec_a\romannumeral`&&@#1%
367 }%
368 \def\XINT_expr_gobz_scandec_a #1%
369 {%
370   \if .#1\xint_dothis{0\endcsname.}\fi
371   \xint_orthat {\XINT_expr_gobz_scandec_b 0.#1}%
372 }%

```

### 11.25.2 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 ?

```

373 \def\XINT_expr_scandec_b #1.#2%
374 {%
375   \ifcat \relax #2\expandafter\XINT_expr_scandec_endbycs\expandafter#2\fi
376   \ifnum\xint_c_ix<1\string#2 \else\expandafter\XINT_expr_scandec_c\fi
377   \string#2\expandafter\XINT_expr_scandec_d\the\numexpr #1-\xint_c_i.%
378 }%
379 \def\XINT_expr_scandec_endbycs #1#2\XINT_expr_scandec_d
380   \the\numexpr#3-\xint_c_i.{[#3]\endcsname #1}%
381 \def\XINT_expr_scandec_d #1.#2%
382 {%
383   \expandafter\XINT_expr_scandec_b
384   \the\numexpr #1\expandafter.\romannumeral`&&@#2%
385 }%
386 \def\XINT_expr_scandec_c\string #1#2\the\numexpr#3-\xint_c_i.%
387 {%
388   \if _#1\xint_dothis{\XINT_expr_scandec_d#3.}\fi
389   \if e#1\xint_dothis{[\the\numexpr#3\XINT_expr_scanexp_a +]\fi
390   \if E#1\xint_dothis{[\the\numexpr#3\XINT_expr_scanexp_a +]\fi
391   \xint_orthat {[#3]\endcsname #1}%
392 }%
393 \def\XINT_expr_gobz_scandec_b #1.#2%
394 {%
395   \ifcat \relax #2\expandafter\XINT_expr_gobz_scandec_endbycs\expandafter#2\fi
396   \ifnum\xint_c_ix<1\string#2 \else\expandafter\XINT_expr_gobz_scandec_c\fi
397   \if0#2\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
398   {\expandafter\XINT_expr_gobz_scandec_b}%
399   {\string#2\expandafter\XINT_expr_scandec_d}\the\numexpr#1-\xint_c_i.%
400 }%
401 \def\XINT_expr_gobz_scandec_endbycs #1#2\xint_c_i.{0[0]\endcsname #1}%
402 \def\XINT_expr_gobz_scandec_c\if0#1#2\fi #3\numexpr#4-\xint_c_i.%
403 {%
404   \if _#1\xint_dothis{\XINT_expr_gobz_scandec_b #4.}\fi
405   \if e#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +]\fi
406   \if E#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +]\fi
407   \xint_orthat {0[0]\endcsname #1}%
408 }%

```

### 11.25.3 Scientific notation

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

```

409 \def\XINT_expr_scanexp_a #1#2%
410 {%

```

```

411      #1\expandafter\XINT_expr_scanexp_b\romannumeral`&&@#2%
412 }%
413 \def\XINT_expr_scanexp_b #1%
414 {%
415     \ifcat \relax #1\expandafter\XINT_expr_scanexp_endbycs\expandafter #1\fi
416     \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanexp_c\fi
417     \string#1\XINT_expr_scanexp_d
418 }%
419 \def\XINT_expr_scanexp_endbycs#1#2\XINT_expr_scanexp_d []\endcsname #1}%
420 \def\XINT_expr_scanexp_d #1%
421 {%
422     \expandafter\XINT_expr_scanexp_bb\romannumeral`&&@#1%
423 }%
424 \def\XINT_expr_scanexp_c\string #1\XINT_expr_scanexp_d
425 {%
426     \if _#1\xint_dothis \XINT_expr_scanexp_d \fi
427     \if +#1\xint_dothis {\XINT_expr_scanexp_a +}\fi
428     \if -#1\xint_dothis {\XINT_expr_scanexp_a -}\fi
429     \xint_orthat []\endcsname #1}%
430 }%
431 \def\XINT_expr_scanexp_bb #1%
432 {%
433     \ifcat \relax #1\expandafter\XINT_expr_scanexp_endbycs_b\expandafter #1\fi
434     \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanexp_cb\fi
435     \string#1\XINT_expr_scanexp_db
436 }%
437 \def\XINT_expr_scanexp_endbycs_b#1#2\XINT_expr_scanexp_db []\endcsname #1}%
438 \def\XINT_expr_scanexp_db #1%
439 {%
440     \expandafter\XINT_expr_scanexp_bb\romannumeral`&&@#1%
441 }%
442 \def\XINT_expr_scanexp_cb\string #1\XINT_expr_scanexp_db
443 {%
444     \if _#1\xint_dothis\XINT_expr_scanexp_d\fi
445     \xint_orthat[]\endcsname #1}%
446 }%

```

#### 11.25.4 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.

```

447 \def\XINT_expr_scanhex_I #1% #1="
448 {%
449     \expandafter\XINT_expr_getop\csname.=\expandafter
450     \XINT_expr_unlock_hex_in\csname.=\XINT_expr_scanhexI_a
451 }%
452 \def\XINT_expr_scanhexI_a #1%
453 {%

```

```

454 \ifcat #1\relax\xint_dothis{.>\endcsname\endcsname #1}\fi
455 \ifx !#1\xint_dothis{.>\endcsname\endcsname !}\fi
456 \xint_orthat {\expandafter\XINT_expr_scanhexI_aa\string #1}%
457 }%
458 \def\XINT_expr_scanhexI_aa #1%
459 {%
460 \if\ifnum`#1>`/
461 \ifnum`#1>`9
462 \ifnum`#1>`@
463 \ifnum`#1>`F
464 0\else1\fi\else0\fi\else1\fi\else0\fi 1%
465 \expandafter\XINT_expr_scanhexI_b
466 \else
467 \if_#1\xint_dothis{\expandafter\XINT_expr_scanhexI_bgob}\fi
468 \if .#1\xint_dothis{\expandafter\XINT_expr_scanhex_transition}\fi
469 \xint_orthat % gather what we got so far, leave catcode 12 #1 in stream
470 {\xint_afterfi {.>\endcsname\endcsname}}%
471 \fi
472 #1%
473 }%
474 \def\XINT_expr_scanhexI_b #1#2%
475 {%
476 #1\expandafter\XINT_expr_scanhexI_a\romannumeral`&&@#2%
477 }%
478 \def\XINT_expr_scanhexI_bgob #1#2%
479 {%
480 \expandafter\XINT_expr_scanhexI_a\romannumeral`&&@#2%
481 }%
482 \def\XINT_expr_scanhex_transition .#1%
483 {%
484 \expandafter.\expandafter.\expandafter
485 \XINT_expr_scanhexII_a\romannumeral`&&@#1%
486 }%
487 \def\XINT_expr_scanhexII_a #1%
488 {%
489 \ifcat #1\relax\xint_dothis{\endcsname\endcsname#1}\fi
490 \ifx !#1\xint_dothis{\endcsname\endcsname !}\fi
491 \xint_orthat {\expandafter\XINT_expr_scanhexII_aa\string #1}%
492 }%
493 \def\XINT_expr_scanhexII_aa #1%
494 {%
495 \if\ifnum`#1>`/
496 \ifnum`#1>`9
497 \ifnum`#1>`@
498 \ifnum`#1>`F
499 0\else1\fi\else0\fi\else1\fi\else0\fi 1%
500 \expandafter\XINT_expr_scanhexII_b
501 \else
502 \if_#1\xint_dothis{\expandafter\XINT_expr_scanhexII_bgob}\fi
503 \xint_orthat{\xint_afterfi {\endcsname\endcsname}}%
504 \fi
505 #1%

```

```

506 }%
507 \def\XINT_expr_scanhexII_b #1#2%
508 {%
509     #1\expandafter\XINT_expr_scanhexII_a\romannumeral`&&@#2%
510 }%
511 \def\XINT_expr_scanhexII_bgob #1#2%
512 {%
513     \expandafter\XINT_expr_scanhexII_a\romannumeral`&&@#2%
514 }%

```

### 11.25.5 \XINT\_expr\_scanfunc: parsing names of functions and variables

```

515 \def\XINT_expr_scanfunc
516 {%
517     \expandafter\XINT_expr_func\romannumeral`&&\XINT_expr_scanfunc_a
518 }%
519 \def\XINT_expr_scanfunc_a #1#2%
520 {%
521     \expandafter #1\romannumeral`&&\expandafter\XINT_expr_scanfunc_b\romannumeral`&&@#2%
522 }%

```

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.

```

523 \def\XINT_expr_scanfunc_b #1%
524 {%
525     \ifx !#1\xint_dothis{(_}\fi
526     \ifcat \relax#1\xint_dothis{(_}\fi
527     \if (#1\xint_dothis{\xint_firstoftwo{(_}}\fi
528     \if @#1\xint_dothis \XINT_expr_scanfunc_a \fi
529     \if _#1\xint_dothis \XINT_expr_scanfunc_a \fi
530     \ifnum \xint_c_ix<1\string#1 \xint_dothis \XINT_expr_scanfunc_a \fi
531     \ifcat a#1\xint_dothis \XINT_expr_scanfunc_a \fi
532     \xint_orthat {(_}%
533     #1%
534 }%

```

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.

```
535 \def\XINT_expr_func #1(#2{\if _#2\xint_dothis\XINT_expr_op__\fi
536 \xint_orthat{\xint_c_xviii #2}{#1}}%
```

## 11.26 `\XINT_expr_getop`: finding the next operator or closing parenthesis or end of expression

Release 1.1 implements multi-character operators.

1.2d adds tacit mutiplication also in front of variable or functions names starting with a letter, not only a `@` or a `_` as was already the case. This is for `(x+y)z` situations. It also applies higher precedence in cases like `x/2y` or `x/2@`, or `x/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!`

```
537 \def\XINT_expr_getop #1#2% this #1 is the current locked computed value
538 {%
539 \expandafter\XINT_expr_getop_a\expandafter #1\romannumeral`&&@#2%
540 }%

541 \catcode`* 11
542 \def\XINT_expr_getop_a #1#2%
543 {%
544 \ifx \relax #2\xint_dothis\xint_firstofthree\fi
545 \ifcat \relax #2\xint_dothis\xint_secondofthree\fi
546 \ifnum\xint_c_ix<1\string#2 \xint_dothis\xint_secondofthree\fi
547 \if _#2\xint_dothis \xint_secondofthree\fi
```

```

548 \if @#2\xint_dothis \xint_secondofthree\fi
549 \if (#2\xint_dothis \xint_secondofthree\fi
550 \ifcat a#2\xint_dothis \xint_secondofthree\fi
551 \xint_orthat \xint_thirdofthree
552 {\XINT_expr_foundend #1}%
553 {\XINT_expr_precedence_*** *#1#2}% tacit multiplication with higher precedence
554 {\expandafter\XINT_expr_getop_b \string#2#1}%
555 }%
556 \catcode`* 12
557 \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.

```

558 \def\XINT_expr_getop_b #1%
559 {%
560 \if '#1\xint_dothis{\XINT_expr_binopwrdrd } \fi
561 \if ?#1\xint_dothis{\XINT_expr_precedence_? ?} \fi
562 \xint_orthat {\XINT_expr_scanop_a #1}%
563 }%
564 \def\XINT_expr_binopwrdrd #1#2' {\expandafter\XINT_expr_foundop_a
565 \csname XINT_expr_itself_\xint_zapspace #2 \xint_gobble_i\endcsname #1}%
566 \def\XINT_expr_scanop_a #1#2#3%
567 {\expandafter\XINT_expr_scanop_b\expandafter #1\expandafter #2\romannumeral`&&@#3}%
568 \def\XINT_expr_scanop_b #1#2#3%
569 {%
570 \ifcat#3\relax\xint_dothis{\XINT_expr_foundop_a #1#2#3}\fi
571 \ifcsname XINT_expr_itself_#1#3\endcsname
572 \xint_dothis
573 {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#3\endcsname #2}\fi
574 \xint_orthat {\XINT_expr_foundop_a #1#2#3}%
575 }%
576 \def\XINT_expr_scanop_c #1#2#3%
577 {%
578 \expandafter\XINT_expr_scanop_d\expandafter #1\expandafter #2\romannumeral`&&@#3%
579 }%
580 \def\XINT_expr_scanop_d #1#2#3%
581 {%
582 \ifcat#3\relax \xint_dothis{\XINT_expr_foundop #1#2#3}\fi
583 \ifcsname XINT_expr_itself_#1#3\endcsname
584 \xint_dothis
585 {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#3\endcsname #2}\fi
586 \xint_orthat {\csname XINT_expr_precedence_#1\endcsname #1#2#3}%
587 }%
588 \def\XINT_expr_foundop_a #1%
589 {%
590 \ifcsname XINT_expr_precedence_#1\endcsname
591 \csname XINT_expr_precedence_#1\endcsname\expandafter\endcsname
592 \expandafter #1%
593 \else
594 \xint_afterfi{\XINT_expr_unknown_operator {#1}\XINT_expr_getop}%
595 \fi

```

```
596 }%
597 \def\XINT_expr_unknown_operator #1{\xintError:removed \xint_gobble_i {#1}}%
598 \def\XINT_expr_foundedop #1{\csname XINT_expr_precedence_#1\endcsname #1}%
```

## 11.27 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.

```
599 \catcode`) 11
600 \def\XINT_tmpa #1#2#3#4%
601 {%
602     \def#1##1%
603     {%
604         \xint_UDsignfork
605             ##1{\expandafter#1\romannumeral`&&@#3}%
606             -{#2##1}%
607         \krof
608     }%
609     \def#2##1##2%
610     {%
611         \ifcase ##1\expandafter\XINT_expr_done
612         \or\xint_afterfi{\XINT_expr_extra_)
613             \expandafter #1\romannumeral`&&\XINT_expr_getop }%
614         \else
615         \xint_afterfi{\expandafter#1\romannumeral`&&\csname XINT_#4_op_##2\endcsname }%
616         \fi
617     }%
618 }%
619 \def\XINT_expr_extra_) {\xintError:removed }%
620 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
621     \expandafter\XINT_tmpa
622     \csname XINT_#1_until_end_a\expandafter\endcsname
623     \csname XINT_#1_until_end_b\expandafter\endcsname
624     \csname XINT_#1_op_-vi\endcsname
625     {#1}%
626 }%
627 \def\XINT_tmpa #1#2#3#4#5#6%
628 {%
629     \def #1##1{\expandafter #3\romannumeral`&&\XINT_expr_getnext }%
630     \def #2{\expandafter #3\romannumeral`&&\XINT_expr_getnext }%
631     \def #3##1{\xint_UDsignfork
632         ##1{\expandafter #3\romannumeral`&&@#5}%
633         -{#4##1}%
634         \krof }%
635     \def #4##1##2{\ifcase ##1\expandafter\XINT_expr_missing_)
636     \or \csname XINT_#6_op_##2\expandafter\endcsname
637     \else
638     \xint_afterfi{\expandafter #3\romannumeral`&&\csname XINT_#6_op_##2\endcsname }%
639     \fi
640 }%
641 }%
```



```
642 \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.

```
643 \catcode`) 12
644 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
645   \expandafter\XINT_tmpa
646   \csname XINT_#1_op_(\expandafter\endcsname
647   \csname XINT_#1_oparen\expandafter\endcsname
648   \csname XINT_#1_until_)_a\expandafter\endcsname
649   \csname XINT_#1_until_)_b\expandafter\endcsname
650   \csname XINT_#1_op_-vi\endcsname
651   {#1}%
652 }%
653 \expandafter\let\csname XINT_expr_precedence_)\endcsname\xint_c_i
```

## 11.28 |, ||, &, &&, <, >, =, ==, <=, >=, !=, +, -, \*, /, ^, \*\*, //, /:, .., ..[, ].., ][, ][:, :], and ++ operators

- 11.28.1 Square brackets for lists, the !? for omit and abort, and the ++ postfix construct . . . . 313
- 11.28.2 The |, &, xor, <, >, =, <=, >=, !=, //, /:, .., +, -, \*, /, ^, ..[, and ].. operators for  
expr, floatexpr and iiexpr operators . . . . . 314
- 11.28.3 The ]+, ]-, ]\*, ]/, ]^, +[, -[, \*[, /[, and ^[ list operators . . . . . 316
- 11.28.4 The 'and', 'or', 'xor', and 'mod' as infix operator words . . . . . 318
- 11.28.5 The ||, &&, \*\*, \*\*[, ]\*\* operators as synonyms . . . . . 319

### 11.28.1 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.

```
654 \expandafter\let\csname XINT_expr_precedence_]\endcsname\xint_c_i
655 \expandafter\let\csname XINT_expr_precedence_;\endcsname\xint_c_i
656 \let\XINT_expr_precedence_a \xint_c_xviii
657 \let\XINT_expr_precedence_!? \xint_c_ii
658 \expandafter\let\csname XINT_expr_precedence_++)\endcsname \xint_c_i
```

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

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

xintexpr

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.)

```

659 \catcode`. 11 \catcode`= 11 \catcode`+ 11
660 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
661   \expandafter\let\csname XINT_#1_op_\}\endcsname \XINT_expr_getop
662   \expandafter\let\csname XINT_#1_op_;\}\endcsname \space
663   \expandafter\def\csname XINT_#1_op_]\}\endcsname ##1{\XINT_expr_scanop_a a##1}%
664   \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.

```

665 \expandafter\def\csname XINT_#1_op_+\endcsname ##1##2\relax
666 {\expandafter\XINT_expr_founded\expandafter
667   {\expandafter\.=+ \csname .=\XINT:NEhook:one\xintiCeil{\XINT_expr_unlock ##1}\endcsname }}%
668 }%
669 \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.

```
670 \catcode\& 12
671 \xintFor* #1 in {{=}}{<=}}{>=}}{!=}}{&&}}{|}|}{**}}{/}/{:}}{..}}{..[]{}..}}{..}%
672 {+[]}{-[]}[*]}{/[]}{**}[{}^]{{a+}}{a-}}{a*}}{a/}}{a**}}{a^}%
673 {}[]{}[:]}{:}}{!}?{++}}{++}}}%{***}}
674 \do{\expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}}%
675 \catcode\& 7
676 \expandafter\let\csname XINT_expr_precedence_***\endcsname \xint_c_viii
```

**11.28.2** The `|`, `&`, `xor`, `<`, `>`, `=`, `<=`, `>=`, `!=`, `//`, `/:`, `..`, `+`, `-`, `*`, `/`, `^`, `..[`, and `]`.. operators for `expr`, `floatexpr` and `iiexpr` operators

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

```

677 \def\XINT_expr_defbin_c #1#2#3#4#5#6#7#8#9%
678 {%
679   \def #1##1% \XINT_expr_op_<op> ou flexpr ou iiexpr
680   {% keep value, get next number and operator, then do until
681     \expandafter #2\expandafter ##1%
682     \romannumeral`&&@\expandafter\XINT_expr_getnext }%
683   \def #2##1##2% \XINT_expr_until_<op>_a ou flexpr ou iiexpr
684   {\xint_UDsignfork ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
685     -{#3##1##2}%
686     \krof }%
687   \def #3##1##2##3##4% \XINT_expr_until_<op>_b ou flexpr ou iiexpr
688   {% either execute next operation now, or first do next (possibly unary)

```

```

689 \ifnum ##2>#7%
690 \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
691 \csname XINT_#8_op_##3\endcsname {##4}}%
692 \else \xint_afterfi {\expandafter ##2\expandafter ##3%
693 \csname .=#9#6{\XINT_expr_unlock ##1}{\XINT_expr_unlock ##4}\endcsname }%
694 \fi}%
695 \let #7#5%
696}%
697\def\XINT_expr_defbin_b #1#2#3#4#5%
698{%
699 \expandafter\XINT_expr_defbin_c
700 \csname XINT_#1_op_#2\expandafter\endcsname
701 \csname XINT_#1_until_#2_a\expandafter\endcsname
702 \csname XINT_#1_until_#2_b\expandafter\endcsname
703 \csname XINT_#1_op_#4\expandafter\endcsname
704 \csname xint_c_#3\expandafter\endcsname
705 \csname #5\expandafter\endcsname
706 \csname XINT_expr_precedence_#2\endcsname {#1}\XINT:NEhook:two
707}%
708\XINT_expr_defbin_b {expr} | {iii}{vi} {xintOR}%
709\XINT_expr_defbin_b {flexpr} | {iii}{vi} {xintOR}%
710\XINT_expr_defbin_b {iiexpr} | {iii}{vi} {xintOR}%
711\XINT_expr_defbin_b {expr} & {iv}{vi} {xintAND}%
712\XINT_expr_defbin_b {flexpr} & {iv}{vi} {xintAND}%
713\XINT_expr_defbin_b {iiexpr} & {iv}{vi} {xintAND}%
714\XINT_expr_defbin_b {expr} {xor}{iii}{vi} {xintXOR}%
715\XINT_expr_defbin_b {flexpr}{xor}{iii}{vi} {xintXOR}%
716\XINT_expr_defbin_b {iiexpr}{xor}{iii}{vi} {xintXOR}%
717\XINT_expr_defbin_b {expr} < {v}{vi} {xintLt}%
718\XINT_expr_defbin_b {flexpr} < {v}{vi} {xintLt}%
719\XINT_expr_defbin_b {iiexpr} < {v}{vi} {xintiiLt}%
720\XINT_expr_defbin_b {expr} > {v}{vi} {xintGt}%
721\XINT_expr_defbin_b {flexpr} > {v}{vi} {xintGt}%
722\XINT_expr_defbin_b {iiexpr} > {v}{vi} {xintiiGt}%
723\XINT_expr_defbin_b {expr} = {v}{vi} {xintEq}%
724\XINT_expr_defbin_b {flexpr} = {v}{vi} {xintEq}%
725\XINT_expr_defbin_b {iiexpr} = {v}{vi} {xintiiEq}%
726\XINT_expr_defbin_b {expr} {<=} {v}{vi} {xintLtorEq}%
727\XINT_expr_defbin_b {flexpr}{<=} {v}{vi} {xintLtorEq}%
728\XINT_expr_defbin_b {iiexpr}{<=} {v}{vi} {xintiiLtorEq}%
729\XINT_expr_defbin_b {expr} {>=} {v}{vi} {xintGtorEq}%
730\XINT_expr_defbin_b {flexpr}{>=} {v}{vi} {xintGtorEq}%
731\XINT_expr_defbin_b {iiexpr}{>=} {v}{vi} {xintiiGtorEq}%
732\XINT_expr_defbin_b {expr} {!=} {v}{vi} {xintNotEq}%
733\XINT_expr_defbin_b {flexpr}{!=} {v}{vi} {xintNotEq}%
734\XINT_expr_defbin_b {iiexpr}{!=} {v}{vi} {xintiiNotEq}%
735\XINT_expr_defbin_b {expr} {//} {vii}{vii}{xintDivFloor}% CHANGED IN 1.2p!
736\XINT_expr_defbin_b {flexpr}{//} {vii}{vii}{XINTinFloatDivFloor}% "
737\XINT_expr_defbin_b {iiexpr}{//} {vii}{vii}{xintiiDivFloor}% "
738\XINT_expr_defbin_b {expr} {/:} {vii}{vii}{xintMod}% "
739\XINT_expr_defbin_b {flexpr}{/:} {vii}{vii}{XINTinFloatMod}% "
740\XINT_expr_defbin_b {iiexpr}{/:} {vii}{vii}{xintiiMod}% "

```

```

741 \XINT_expr_defbin_b {expr} + {vi}{vi} {xintAdd}%
742 \XINT_expr_defbin_b {flexpr} + {vi}{vi} {XINTinFloatAdd}%
743 \XINT_expr_defbin_b {iiexpr} + {vi}{vi} {xintiiAdd}%
744 \XINT_expr_defbin_b {expr} - {vi}{vi} {xintSub}%
745 \XINT_expr_defbin_b {flexpr} - {vi}{vi} {XINTinFloatSub}%
746 \XINT_expr_defbin_b {iiexpr} - {vi}{vi} {xintiiSub}%
747 \XINT_expr_defbin_b {expr} * {vii}{vii} {xintMul}%
748 \XINT_expr_defbin_b {flexpr} * {vii}{vii} {XINTinFloatMul}%
749 \XINT_expr_defbin_b {iiexpr} * {vii}{vii} {xintiiMul}%
750 \XINT_expr_defbin_b {expr} / {vii}{vii} {xintDiv}%
751 \XINT_expr_defbin_b {flexpr} / {vii}{vii} {XINTinFloatDiv}%
752 \XINT_expr_defbin_b {iiexpr} / {vii}{vii} {xintiiDivRound}% CHANGED IN 1.1!
753 \XINT_expr_defbin_b {expr} ^ {ix}{ix} {xintPow}%
754 \XINT_expr_defbin_b {flexpr} ^ {ix}{ix} {XINTinFloatPowerH}%
755 \XINT_expr_defbin_b {iiexpr} ^ {ix}{ix} {xintiiPow}%
756 \XINT_expr_defbin_b {expr} {...}{iii}{vi} {xintSeqA::csv}%
757 \XINT_expr_defbin_b {flexpr}{...}{iii}{vi} {XINTinFloatSeqA::csv}%
758 \XINT_expr_defbin_b {iiexpr}{...}{iii}{vi} {xintiiSeqA::csv}%
759 \def\XINT_expr_defbin_b #1#2#3#4#5%
760 {%
761 \expandafter\XINT_expr_defbin_c
762 \csname XINT_#1_op_#2\expandafter\endcsname
763 \csname XINT_#1_until_#2_a\expandafter\endcsname
764 \csname XINT_#1_until_#2_b\expandafter\endcsname
765 \csname XINT_#1_op_#4\expandafter\endcsname
766 \csname xint_c_#3\expandafter\endcsname
767 \csname #5\expandafter\endcsname
768 \csname XINT_expr_precedence_#2\endcsname {#1}{}%
769 }%
770 \XINT_expr_defbin_b {expr} {...}{iii}{vi} {xintSeq::csv}%
771 \XINT_expr_defbin_b {flexpr}{...}{iii}{vi} {xintSeq::csv}%
772 \XINT_expr_defbin_b {iiexpr}{...}{iii}{vi} {xintiiSeq::csv}%
773 \XINT_expr_defbin_b {expr} {...}{iii}{vi} {xintSeqB::csv}%
774 \XINT_expr_defbin_b {flexpr}{...}{iii}{vi} {XINTinFloatSeqB::csv}%
775 \XINT_expr_defbin_b {iiexpr}{...}{iii}{vi} {xintiiSeqB::csv}%

```

### 11.28.3 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.

```

776 \def\XINT_expr_binop_inline#1%
777 {\XINT_expr_binop_inline_a\expandafter\XINT:NEhook:two\expandafter#1}%
778 \def\XINT_expr_binop_inline_a
779 {\expandafter\xint_gobble_i\romannumeral`&&\XINT_expr_binop_inline_b}%
780 \def\XINT_expr_binop_inline_b #1#2,{\XINT_expr_binop_inline_c #2,{#1}}%
781 \def\XINT_expr_binop_inline_c #1{%
782 \if ,#1\xint_dothis\XINT_expr_binop_inline_e\fi
783 \if ^#1\xint_dothis\XINT_expr_binop_inline_end\fi
784 \xint_orthat\XINT_expr_binop_inline_d #1}%
785 \def\XINT_expr_binop_inline_d #1,#2{,#2{#1}\XINT_expr_binop_inline_b {#2}}%
786 \def\XINT_expr_binop_inline_e #1,#2{,\XINT_expr_binop_inline_b {#2}}%
787 \def\XINT_expr_binop_inline_end #1,#2{}%

```

```

788 \def\XINT_expr_deflistopr_c #1#2#3#4#5#6#7#8%
789 {%
790   \def #1##1% \XINT_expr_op_<op> ou flexpr ou iiexpr
791   {% keep value, get next number and operator, then do until
792     \expandafter #2\expandafter ##1%
793     \romannumeral`&&\expandafter\XINT_expr_getnext }%
794   \def #2##1##2% \XINT_expr_until_<op>_a ou flexpr ou iiexpr
795   {\xint_UDsignfork ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
796     -{#3##1##2}%
797     \krof }%
798   \def #3##1##2##3##4% \XINT_expr_until_<op>_b ou flexpr ou iiexpr
799   {% either execute next operation now, or first do next (possibly unary)
800     \ifnum ##2>#7%
801       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
802         \csname XINT_#8_op_##3\endcsname {##4}}%
803     \else \xint_afterfi {\expandafter ##2\expandafter ##3%
804       \csname .=\expandafter\XINT_expr_binop_inline\expandafter
805         {\expandafter#6\expandafter\xint_exchangetwo_keepbraces\expandafter
806           {\expandafter\XINT_expr_unlock\expandafter ##4\expandafter}\expandafter}%
807         \romannumeral`&&\XINT_expr_unlock ##1,^,\endcsname }%
808     \fi }%
809   \let #7#5%
810 }%
811 \def\XINT_expr_deflistopr_b #1#2#3#4%
812 {%
813   \expandafter\XINT_expr_deflistopr_c
814   \csname XINT_#1_op_#2\expandafter\endcsname
815   \csname XINT_#1_until_#2_a\expandafter\endcsname
816   \csname XINT_#1_until_#2_b\expandafter\endcsname
817   \csname XINT_#1_op_-#3\expandafter\endcsname
818   \csname xint_c_#3\expandafter\endcsname
819   \csname #4\expandafter\endcsname
820   \csname XINT_expr_precedence_#2\endcsname {#1}%
821 }%

```

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

```

822 \XINT_expr_deflistopr_b {expr} {a+}{vi} {xintAdd}%
823 \XINT_expr_deflistopr_b {expr} {a-}{vi} {xintSub}%
824 \XINT_expr_deflistopr_b {expr} {a*}{vii} {xintMul}%
825 \XINT_expr_deflistopr_b {expr} {a/}{vii} {xintDiv}%
826 \XINT_expr_deflistopr_b {expr} {a^}{ix} {xintPow}%
827 \XINT_expr_deflistopr_b {iiexpr}{a+}{vi} {xintiiAdd}%
828 \XINT_expr_deflistopr_b {iiexpr}{a-}{vi} {xintiiSub}%
829 \XINT_expr_deflistopr_b {iiexpr}{a*}{vii} {xintiiMul}%
830 \XINT_expr_deflistopr_b {iiexpr}{a/}{vii} {xintiiDivRound}%
831 \XINT_expr_deflistopr_b {iiexpr}{a^}{ix} {xintiiPow}%
832 \XINT_expr_deflistopr_b {flexpr}{a+}{vi} {XINTinFloatAdd}%
833 \XINT_expr_deflistopr_b {flexpr}{a-}{vi} {XINTinFloatSub}%
834 \XINT_expr_deflistopr_b {flexpr}{a*}{vii} {XINTinFloatMul}%
835 \XINT_expr_deflistopr_b {flexpr}{a/}{vii} {XINTinFloatDiv}%
836 \XINT_expr_deflistopr_b {flexpr}{a^}{ix} {XINTinFloatPowerH}%

```

```

837 \def\XINT_expr_deflistopl_c #1#2#3#4#5#6#7%
838 {%
839   \def #1#1{\expandafter#2\expandafter##1\romannumeral`&&@%
840     \expandafter #3\romannumeral`&&@\XINT_expr_getnext }%
841   \def #2##1##2##3##4%
842   {% either execute next operation now, or first do next (possibly unary)
843     \ifnum ##2>#6%
844       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
845         \csname XINT_#7_op_#3\endcsname {##4}}%
846     \else \xint_afterfi {\expandafter ##2\expandafter ##3%
847       \csname .=\expandafter\XINT_expr_binop_inline\expandafter
848         {\expandafter#5\expandafter
849           {\expandafter\XINT_expr_unlock\expandafter ##1\expandafter}\expandafter}%
850         \romannumeral`&&@\XINT_expr_unlock ##4,^,\endcsname }%
851     \fi }%
852   \let #6#4%
853 }%
854 \def\XINT_expr_deflistopl_b #1#2#3#4%
855 {%
856   \expandafter\XINT_expr_deflistopl_c
857   \csname XINT_#1_op_#2\expandafter\endcsname
858   \csname XINT_#1_until_#2\expandafter\endcsname
859   \csname XINT_#1_until_)_a\expandafter\endcsname
860   \csname xint_c_#3\expandafter\endcsname
861   \csname #4\expandafter\endcsname
862   \csname XINT_expr_precedence_#2\endcsname {#1}%
863 }%

```

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

```

864 \XINT_expr_deflistopl_b {expr} {+[] {vi} {xintAdd}%
865 \XINT_expr_deflistopl_b {expr} {-[] {vi} {xintSub}%
866 \XINT_expr_deflistopl_b {expr} {*[] {vii} {xintMul}%
867 \XINT_expr_deflistopl_b {expr} {/[] {vii} {xintDiv}%
868 \XINT_expr_deflistopl_b {expr} {^[] {ix} {xintPow}%
869 \XINT_expr_deflistopl_b {iiexpr}{+[] {vi} {xintiiAdd}%
870 \XINT_expr_deflistopl_b {iiexpr}{-[] {vi} {xintiiSub}%
871 \XINT_expr_deflistopl_b {iiexpr}{*[] {vii} {xintiiMul}%
872 \XINT_expr_deflistopl_b {iiexpr}{/[] {vii} {xintiiDivRound}%
873 \XINT_expr_deflistopl_b {iiexpr}{^[] {ix} {xintiiPow}%
874 \XINT_expr_deflistopl_b {flexpr}{+[] {vi} {XINTinFloatAdd}%
875 \XINT_expr_deflistopl_b {flexpr}{-[] {vi} {XINTinFloatSub}%
876 \XINT_expr_deflistopl_b {flexpr}{*[] {vii} {XINTinFloatMul}%
877 \XINT_expr_deflistopl_b {flexpr}{/[] {vii} {XINTinFloatDiv}%
878 \XINT_expr_deflistopl_b {flexpr}{^[] {ix} {XINTinFloatPowerH}%

```

#### 11.28.4 The 'and', 'or', 'xor', and 'mod' as infix operator words

```

879 \xintFor #1 in {and,or,xor,mod} \do {%
880   \expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}%
881   \expandafter\let\csname XINT_expr_precedence_and\expandafter\endcsname
882     \csname XINT_expr_precedence_&\endcsname
883   \expandafter\let\csname XINT_expr_precedence_or\expandafter\endcsname
884     \csname XINT_expr_precedence_| \endcsname

```

```

885 \expandafter\let\csname XINT_expr_precedence_mod\expandafter\endcsname
886       \csname XINT_expr_precedence_/\endcsname
887 \xintFor #1 in {expr, flexpr, iexpr} \do {%
888   \expandafter\let\csname XINT_#1_op_and\expandafter\endcsname
889       \csname XINT_#1_op_&\endcsname
890   \expandafter\let\csname XINT_#1_op_or\expandafter\endcsname
891       \csname XINT_#1_op_|\endcsname
892   \expandafter\let\csname XINT_#1_op_mod\expandafter\endcsname
893       \csname XINT_#1_op_/\endcsname
894 }%
```

### 11.28.5 The ||, &&, \*\*, \*\*[, ]\*\* operators as synonyms

```

895 \expandafter\let\csname XINT_expr_precedence_==\expandafter\endcsname
896       \csname XINT_expr_precedence_=\endcsname
897 \expandafter\let\csname XINT_expr_precedence_&\string&\expandafter\endcsname
898       \csname XINT_expr_precedence_&\endcsname
899 \expandafter\let\csname XINT_expr_precedence_||\expandafter\endcsname
900       \csname XINT_expr_precedence_|\endcsname
901 \expandafter\let\csname XINT_expr_precedence_**\expandafter\endcsname
902       \csname XINT_expr_precedence_^\endcsname
903 \expandafter\let\csname XINT_expr_precedence_a**\expandafter\endcsname
904       \csname XINT_expr_precedence_a^\endcsname
905 \expandafter\let\csname XINT_expr_precedence_**[\expandafter\endcsname
906       \csname XINT_expr_precedence_^\endcsname
907 \xintFor #1 in {expr, flexpr, iexpr} \do {%
908   \expandafter\let\csname XINT_#1_op_==\expandafter\endcsname
909       \csname XINT_#1_op_=\endcsname
910   \expandafter\let\csname XINT_#1_op_&\string&\expandafter\endcsname
911       \csname XINT_#1_op_&\endcsname
912   \expandafter\let\csname XINT_#1_op_||\expandafter\endcsname
913       \csname XINT_#1_op_|\endcsname
914   \expandafter\let\csname XINT_#1_op_**\expandafter\endcsname
915       \csname XINT_#1_op_^\endcsname
916   \expandafter\let\csname XINT_#1_op_a**\expandafter\endcsname
917       \csname XINT_#1_op_a^\endcsname
918   \expandafter\let\csname XINT_#1_op_**[\expandafter\endcsname
919       \csname XINT_#1_op_^\endcsname
920 }%
```

## 11.29 Macros for list selectors: [list][N], [list][:b], [list][a:], [list][a:b]

11.29.1	\xintListSel:x:csv	321
11.29.2	\xintListSel:f:csv	322
11.29.3	\xintKeep:x:csv	323
11.29.4	\xintKeep:f:csv	324
11.29.5	\xintTrim:f:csv	324
11.29.6	\xintNthEltPy:f:csv	324
11.29.7	\xintLength:f:csv	325
11.29.8	\xintReverse:f:csv	325

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.

```

921 \def\XINT_tmpa #1#2#3#4#5#6%
922 {%
923   \def #1##1% \XINT_expr_op_[
924   {%
925     \expandafter #2\expandafter ##1\romannumeral`&&\XINT_expr_getnext
926   }%
927   \def #2##1##2% \XINT_expr_until_[_a
928   {\xint_UDsignfork
929     ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
930     -{##3##1##2}%
931   \krof }%
932   \def #3##1##2##3##4% \XINT_expr_until_[_b
933   {%
934     \ifnum ##2>#5%
935       \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
936         \csname XINT_#6_op_##3\endcsname {##4}}%
937     \else
938       \xint_afterfi
939       {\expandafter ##2\expandafter ##3\csname
940         .=\expandafter\xintListSel:x:csv % will be \xintListSel:f:csv in \xintNewExpr output
941         \romannumeral`&&\XINT_expr_unlock ##4;% selector
942         \XINT_expr_unlock ##1;\endcsname % unlock already pre-positioned for \xintNewExpr
943       }%
944     \fi
945   }%
946   \let #5\xint_c_ii
947 }%
948 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
949 \expandafter\XINT_tmpa
950   \csname XINT_#1_op_[\expandafter\endcsname
951   \csname XINT_#1_until_[_a\expandafter\endcsname
952   \csname XINT_#1_until_[_b\expandafter\endcsname
953   \csname XINT_#1_op_-vi\expandafter\endcsname
954   \csname XINT_expr_precedence_[\endcsname {#1}%
955 }%
956 \def\XINT_tmpa #1#2#3#4#5#6%
957 {%
958   \def #1##1% \XINT_expr_op_:
959   {%
960     \expandafter #2\expandafter ##1\romannumeral`&&\XINT_expr_getnext
961   }%
962   \def #2##1##2% \XINT_expr_until_:_a
963   {\xint_UDsignfork
964     ##2{\expandafter #2\expandafter ##1\romannumeral`&&@#4}%
965     -{##3##1##2}%

```



```

966 \krof }%
967 \def #3##1##2##3##4% \XINT_expr_until:_b
968 {%
969 \ifnum ##2>#5%
970 \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
971 \csname XINT_#6_op_##3\endcsname {##4}}%
972 \else
973 \xint_afterfi
974 {\expandafter ##2\expandafter ##3\csname
975 .:=\XINT:NEhook:one\xintNum{\XINT_expr_unlock ##1};%
976 \XINT:NEhook:one\xintNum{\XINT_expr_unlock ##4}%
977 \endcsname
978 }%
979 \fi
980 }%
981 \let #5\xint_c_iii
982 }%
983 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
984 \expandafter\XINT_tmpa
985 \csname XINT_#1_op_:\expandafter\endcsname
986 \csname XINT_#1_until:_a\expandafter\endcsname
987 \csname XINT_#1_until:_b\expandafter\endcsname
988 \csname XINT_#1_op_-vi\expandafter\endcsname
989 \csname XINT_expr_precedence_:\endcsname {#1}%
990 }%

991 \catcode`[ 11 \catcode`] 11
992 \let\XINT_expr_precedence_:\xint_c_iii
993 \def\XINT_expr_op_:] #1%
994 {%
995 \expandafter\xint_c_i\expandafter )%
996 \csname .:=\XINT:NEhook:one\xintNum{\XINT_expr_unlock #1}\endcsname
997 }%
998 \let\XINT_flexpr_op_:] \XINT_expr_op_:]
999 \let\XINT_iiexpr_op_:] \XINT_expr_op_:]
1000 \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).

```

1001 \edef\XINT_expr_op_][: #1{\xint_c_ii\noexpand\XINT_expr_itself_][#10\string :}%
1002 \let\XINT_flexpr_op_][: \XINT_expr_op_][:
1003 \let\XINT_iiexpr_op_][: \XINT_expr_op_][:
1004 \catcode`[ 12 \catcode`] 12

```

### 11.29.1 `\xintListSel:x:csv`

1.2j. Because there is `\xintKeep:x:csv` which is faster than `\xintKeep:f:csv`.

```

1005 \def\xintListSel:x:csv #1%
1006 {%
1007 \if ]\noexpand#1\xint_dothis\XINT_listsel:_s\fi
1008 \if : \noexpand#1\xint_dothis\XINT_listxs sel:_:\fi

```

```

1009 \xint_orthat {\XINT_listsel:_nth #1}%
1010 }%
1011 \def\XINT_listsel:_s #1#2;#3;%
1012 {%
1013 \if-#1\expandafter\xintKeep:f:csv\else\expandafter\xintTrim:f:csv\fi
1014 {#1#2}{#3}%
1015 }%
1016 \def\XINT_listsel:_nth #1;#2;{\xintNthEltPy:f:csv {\xintNum{#1}}{#2}}%

    \XINT_listsel:_nth and \XINT_listsel:_s located in \xintListSel:f:csv.

1017 \def\XINT_listxsels:_ #1#2;#3#4;%
1018 {%
1019 \xint_UDsignsfork
1020 #1#3\XINT_listxsels:_N:N
1021 #1-\XINT_listxsels:_N:P
1022 -#3\XINT_listxsels:_P:N
1023 --\XINT_listxsels:_P:P
1024 \krof #1#2;#3#4;%
1025 }%
1026 \def\XINT_listxsels:_P:P #1;#2;#3;%
1027 {%
1028 \unless\ifnum #1<#2 \expandafter\xint_gobble_iii\fi
1029 \xintKeep:x:csv{#2-#1}{\xintTrim:f:csv{#1}{#3}}%
1030 }%
1031 \def\XINT_listxsels:_N:N #1;#2;#3;%
1032 {%
1033 \expandafter\XINT_listxsels:_N:N_a
1034 \the\numexpr #2-#1\expandafter;\the\numexpr#1+\xintLength:f:csv{#3};#3;%
1035 }%
1036 \def\XINT_listxsels:_N:N_a #1;#2;#3;%
1037 {%
1038 \unless\ifnum #1>\xint_c_ \expandafter\xint_gobble_iii\fi
1039 \xintKeep:x:csv{#1}{\xintTrim:f:csv{\ifnum#2<\xint_c_\xint_c_\else#2\fi}{#3}}%
1040 }%
1041 \def\XINT_listxsels:_N:P #1;#2;#3;{\expandafter\XINT_listxsels:_N:P_a
1042 \the\numexpr #1+\xintLength:f:csv{#3};#2;#3;%}
1043 \def\XINT_listxsels:_N:P_a #1#2;%
1044 {\if -#1\expandafter\XINT_listxsels:_O:P\fi\XINT_listxsels:_P:P #1#2;%}
1045 \def\XINT_listxsels:_O:P\XINT_listxsels:_P:P #1;{\XINT_listxsels:_P:P 0;%}
1046 \def\XINT_listxsels:_P:N #1;#2;#3;{\expandafter\XINT_listxsels:_P:N_a
1047 \the\numexpr #2+\xintLength:f:csv{#3};#1;#3;%}
1048 \def\XINT_listxsels:_P:N_a #1#2;#3;%
1049 {\if -#1\expandafter\XINT_listxsels:_P:O\fi\XINT_listxsels:_P:P #3;#1#2;%}
1050 \def\XINT_listxsels:_P:O\XINT_listxsels:_P:P #1;#2;{\XINT_listxsels:_P:P #1;0;%}

```

### 11.29.2 \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.

```

1051 \def\xintListSel:f:csv #1%

```

```

1052 {%
1053   \if ]\noexpand#1\xint_dothis{\expandafter\XINT_listsel:_s\romannumeral`&&@\fi
1054   \if :\noexpand#1\xint_dothis{\XINT_listsel:_:}\fi
1055   \xint_orthat {\XINT_listsel:_nth #1}%
1056 }%
1057 \def\XINT_listsel:_: #1;#2;%
1058 {%
1059   \expandafter\XINT_listsel:_:a
1060   \the\numexpr #1\expandafter;\the\numexpr #2\expandafter;\romannumeral`&&@%
1061 }%
1062 \def\XINT_listsel:_:a #1#2;#3#4;%
1063 {%
1064   \xint_UDsignsfork
1065     #1#3\XINT_listsel:_N:N
1066     #1-\XINT_listsel:_N:P
1067     -#3\XINT_listsel:_P:N
1068     --\XINT_listsel:_P:P
1069   \krof #1#2;#3#4;%
1070 }%
1071 \def\XINT_listsel:_P:P #1;#2;#3;%
1072 {%
1073   \unless\ifnum #1<#2 \xint_afterfi{\expandafter\space\xint_gobble_iii}\fi
1074   \xintKeep:f:csv{#2-#1}{\xintTrim:f:csv{#1}{#3}}%
1075 }%
1076 \def\XINT_listsel:_N:N #1;#2;#3;%
1077 {%
1078   \unless\ifnum #1<#2 \expandafter\XINT_listsel:_N:N_abort\fi
1079   \expandafter\XINT_listsel:_N:N_a
1080   \the\numexpr#1+\xintLength:f:csv{#3}\expandafter;\the\numexpr#2-#1;#3;%
1081 }%
1082 \def\XINT_listsel:_N:N_abort #1;#2;#3;{ }%
1083 \def\XINT_listsel:_N:N_a #1;#2;#3;%
1084 {%
1085   \xintKeep:f:csv{#2}{\xintTrim:f:csv{\ifnum#1<\xint_c\xint_c\else#1\fi}{#3}}%
1086 }%
1087 \def\XINT_listsel:_N:P #1;#2;#3;{\expandafter\XINT_listsel:_N:P_a
1088   \the\numexpr #1+\xintLength:f:csv{#3};#2;#3;%
1089 \def\XINT_listsel:_N:P_a #1#2;%
1090   {\if -#1\expandafter\XINT_listsel:_O:P\fi\XINT_listsel:_P:P #1#2;%}
1091 \def\XINT_listsel:_O:P\XINT_listsel:_P:P #1;{\XINT_listsel:_P:P 0;}%
1092 \def\XINT_listsel:_P:N #1;#2;#3;{\expandafter\XINT_listsel:_P:N_a
1093   \the\numexpr #2+\xintLength:f:csv{#3};#1;#3;%}
1094 \def\XINT_listsel:_P:N_a #1#2;#3;%
1095   {\if -#1\expandafter\XINT_listsel:_P:O\fi\XINT_listsel:_P:P #3;#1#2;%}
1096 \def\XINT_listsel:_P:O\XINT_listsel:_P:P #1;#2;{\XINT_listsel:_P:P #1;0;}%

```

### 11.29.3 \xintKeep:x:csv

1.2j. This macro is used only with positive first argument.

```

1097 \def\xintKeep:x:csv #1#2%
1098 {%
1099   \expandafter\xint_gobble_i

```

```

1100 \romannumeral0\expandafter\XINT_keep:x:csv_pos
1101 \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
1102 }%
1103 \def\XINT_keep:x:csv_pos #1.#2%
1104 {%
1105 \expandafter\XINT_keep:x:csv_loop\the\numexpr#1-\xint_c_viii.%
1106 #2\xint_Bye,\xint_Bye,\xint_Bye,\xint_Bye,%
1107 \xint_Bye,\xint_Bye,\xint_Bye,\xint_Bye,\xint_bye
1108 }%
1109 \def\XINT_keep:x:csv_loop #1%
1110 {%
1111 \xint_gob_til_minus#1\XINT_keep:x:csv_finish-%
1112 \XINT_keep:x:csv_loop_pickeight #1%
1113 }%
1114 \def\XINT_keep:x:csv_loop_pickeight #1.#2,#3,#4,#5,#6,#7,#8,#9,%
1115 {%
1116 ,#2,#3,#4,#5,#6,#7,#8,#9%
1117 \expandafter\XINT_keep:x:csv_loop\the\numexpr#1-\xint_c_viii.%
1118 }%
1119 \def\XINT_keep:x:csv_finish-\XINT_keep:x:csv_loop_pickeight -#1.%
1120 {%
1121 \csname XINT_keep:x:csv_finish#1\endcsname
1122 }%
1123 \expandafter\def\csname XINT_keep:x:csv_finish1\endcsname
1124 #1,#2,#3,#4,#5,#6,#7,{,#1,#2,#3,#4,#5,#6,#7\xint_Bye}%
1125 \expandafter\def\csname XINT_keep:x:csv_finish2\endcsname
1126 #1,#2,#3,#4,#5,#6,{,#1,#2,#3,#4,#5,#6\xint_Bye}%
1127 \expandafter\def\csname XINT_keep:x:csv_finish3\endcsname
1128 #1,#2,#3,#4,#5,{,#1,#2,#3,#4,#5\xint_Bye}%
1129 \expandafter\def\csname XINT_keep:x:csv_finish4\endcsname
1130 #1,#2,#3,#4,{,#1,#2,#3,#4\xint_Bye}%
1131 \expandafter\def\csname XINT_keep:x:csv_finish5\endcsname
1132 #1,#2,#3,{,#1,#2,#3\xint_Bye}%
1133 \expandafter\def\csname XINT_keep:x:csv_finish6\endcsname
1134 #1,#2,{,#1,#2\xint_Bye}%
1135 \expandafter\def\csname XINT_keep:x:csv_finish7\endcsname
1136 #1,{,#1\xint_Bye}%
1137 \expandafter\let\csname XINT_keep:x:csv_finish8\endcsname\xint_Bye

```

#### 11.29.4 \xintKeep:f:csv

1.2g. moved to [xinttools](#).

#### 11.29.5 \xintTrim:f:csv

1.2g. moved to [xinttools](#).

#### 11.29.6 \xintNthEltPy:f:csv

1.2g. moved to [xinttools](#).

### 11.29.7 `\xintLength:f:csv`

1.2g. moved to [xinttools](#).

### 11.29.8 `\xintReverse:f:csv`

1.2g. moved to [xinttools](#).

## 11.30 Macros for a..b list generation

11.30.1	<code>\xintSeq::csv</code>	325
11.30.2	<code>\xintiiSeq::csv</code>	326

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`.

### 11.30.1 `\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.

```

1138 \def\xintSeq::csv {\romannumeral0\xintseq::csv}%
1139 \def\xintseq::csv #1#2%
1140 {%
1141   \expandafter\XINT_seq::csv\expandafter
1142     {\the\numexpr \xintiCeil{#1}\expandafter}\expandafter
1143     {\the\numexpr \xintiFloor{#2}}}%
1144}%
1145 \def\XINT_seq::csv #1#2%
1146 {%
1147   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
1148     \expandafter\XINT_seq::csv_z
1149   \or
1150     \expandafter\XINT_seq::csv_p
1151   \else
1152     \expandafter\XINT_seq::csv_n
1153   \fi
1154   {#2}{#1}}%
1155}%
1156 \def\XINT_seq::csv_z #1#2{ #1/1[0]}%
1157 \def\XINT_seq::csv_p #1#2%
1158 {%
1159   \ifnum #1>#2
1160     \expandafter\expandafter\expandafter\XINT_seq::csv_p
1161   \else
1162     \expandafter\XINT_seq::csv_e
1163   \fi
1164   \expandafter{\the\numexpr #1-\xint_c_i}{#2},#1/1[0]}%
1165}%
1166 \def\XINT_seq::csv_n #1#2%

```

```

1167 {%
1168   \ifnum #1<#2
1169     \expandafter\expandafter\expandafter\XINT_seq::csv_n
1170   \else
1171     \expandafter\XINT_seq::csv_e
1172   \fi
1173   \expandafter{\the\numexpr #1+\xint_c_i}{#2},#1/1[0]%
1174 }%
1175 \def\XINT_seq::csv_e #1,{ }%

```

### 11.30.2 \xintiiSeq::csv

```

1176 \def\xintiiSeq::csv {\romannumeral0\xintiiSeq::csv }%
1177 \def\xintiiSeq::csv #1#2%
1178 {%
1179   \expandafter\XINT_iiseq::csv\expandafter
1180   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
1181 }%
1182 \def\XINT_iiseq::csv #1#2%
1183 {%
1184   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
1185   \expandafter\XINT_iiseq::csv_z
1186   \or
1187   \expandafter\XINT_iiseq::csv_p
1188   \else
1189   \expandafter\XINT_iiseq::csv_n
1190   \fi
1191   {#2}{#1}%
1192 }%
1193 \def\XINT_iiseq::csv_z #1#2{ #1}%
1194 \def\XINT_iiseq::csv_p #1#2%
1195 {%
1196   \ifnum #1>#2
1197     \expandafter\expandafter\expandafter\XINT_iiseq::csv_p
1198   \else
1199     \expandafter\XINT_seq::csv_e
1200   \fi
1201   \expandafter{\the\numexpr #1-\xint_c_i}{#2},#1%
1202 }%
1203 \def\XINT_iiseq::csv_n #1#2%
1204 {%
1205   \ifnum #1<#2
1206     \expandafter\expandafter\expandafter\XINT_iiseq::csv_n
1207   \else
1208     \expandafter\XINT_seq::csv_e
1209   \fi
1210   \expandafter{\the\numexpr #1+\xint_c_i}{#2},#1%
1211 }%
1212 \def\XINT_seq::csv_e #1,{ }%

```

## 11.31 Macros for a..[d]..b list generation

11.31.1	\xintSeqA::csv, \xintiiSeqA::csv, \XINTinFloatSeqA::csv . . . . .	327
11.31.2	\xintSeqB::csv . . . . .	327

11.31.3	<code>\xintiiSeqB::csv</code>	327
11.31.4	<code>\XINTinFloatSeqB::csv</code>	328

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`.

### 11.31.1 `\xintSeqA::csv`, `\xintiiSeqA::csv`, `\XINTinFloatSeqA::csv`

```

1213 \def\xintSeqA::csv #1%
1214   {\expandafter\XINT_seqa::csv\expandafter{\romannumeral0\xintra {#1}}}%
1215 \def\XINT_seqa::csv #1#2{\expandafter\XINT_seqa::csv_a \romannumeral0\xintra {#2};#1;}%
1216 \def\xintiiSeqA::csv #1{\expandafter\XINT_iiseqa::csv\expandafter{\romannumeral`&&@#1}}%
1217 \def\XINT_iiseqa::csv #1#2{\expandafter\XINT_seqa::csv_a\romannumeral`&&@#2;#1;}%
1218 \def\XINTinFloatSeqA::csv #1{\expandafter\XINT_flseqa::csv\expandafter
1219   {\romannumeral0\XINTinfloat [\XINTdigits]{#1}}}%
1220 \def\XINT_flseqa::csv #1#2%
1221   {\expandafter\XINT_seqa::csv_a\romannumeral0\XINTinfloat [\XINTdigits]{#2};#1;}%
1222 \def\XINT_seqa::csv_a #1{\xint_UDzerominusfork
1223   #1-{z}%
1224   0#1{n}%
1225   0-{p}%
1226   \krof #1}%

```

### 11.31.2 `\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`.

```

1227 \def\xintSeqB::csv #1#2%
1228   {\expandafter\XINT_seqb::csv \expandafter{\romannumeral0\xintra{#2}}{#1}}%
1229 \def\XINT_seqb::csv #1#2{\expandafter\XINT_seqb::csv_a\romannumeral`&&@#2#1!}%
1230 \def\XINT_seqb::csv_a #1#2;#3;#4!\expandafter\XINT_expr_seq_empty?
1231   \romannumeral0\cename XINT_seqb::csv_#1\endcsname {#3}{#4}{#2}}%
1232 \def\XINT_seqb::csv_p #1#2#3%
1233 {%
1234   \xintifCmp {#1}{#2}{, #1\expandafter\XINT_seqb::csv_p\expandafter}%
1235   {, #1\xint_gobble_iii}{\xint_gobble_iii}%

```

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

```

1236   {\romannumeral0\xintadd {#3}{#1}{#2}{#3}}%
1237 }%
1238 \def\XINT_seqb::csv_n #1#2#3%
1239 {%
1240   \xintifCmp {#1}{#2}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1241   {, #1\expandafter\XINT_seqb::csv_n\expandafter}%
1242   {\romannumeral0\xintadd {#3}{#1}{#2}{#3}}%
1243 }%
1244 \def\XINT_seqb::csv_z #1#2#3{, #1}%

```

### 11.31.3 `\xintiiSeqB::csv`

```

1245 \def\xintiiSeqB::csv #1#2{\XINT_iiseqb::csv #1#2}%
1246 \def\XINT_iiseqb::csv #1#2#3#4%

```

```

1247 {\expandafter\XINT_iiseqb::csv_a
1248 \romannumeral`&&\expandafter \XINT_expr_unlock\expandafter#2%
1249 \romannumeral`&&\XINT_expr_unlock #4!}%
1250 \def\XINT_iiseqb::csv_a #1#2;#3;#4!\expandafter\XINT_expr_seq_empty?
1251 \romannumeral`&&\csname XINT_iiseqb::csv_#1\endcsname {#3}{#4}{#2}}%
1252 \def\XINT_iiseqb::csv_p #1#2#3%
1253 {%
1254 \xintSgnFork{\XINT_Cmp {#1}{#2}}{, #1\expandafter\XINT_iiseqb::csv_p\expandafter}%
1255 {, #1\xint_gobble_iii}{\xint_gobble_iii}%
1256 {\romannumeral0\xintiadd {#3}{#1}}{#2}{#3}%
1257 }%
1258 \def\XINT_iiseqb::csv_n #1#2#3%
1259 {%
1260 \xintSgnFork{\XINT_Cmp {#1}{#2}}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1261 {, #1\expandafter\XINT_iiseqb::csv_n\expandafter}%
1262 {\romannumeral0\xintiadd {#3}{#1}}{#2}{#3}%
1263 }%
1264 \def\XINT_iiseqb::csv_z #1#2#3{, #1}%

```

#### 11.31.4 \XINTinFloatSeqB::csv

```

1265 \def\XINTinFloatSeqB::csv #1#2{\expandafter\XINT_flseqb::csv \expandafter
1266 {\romannumeral0\XINTinfloat [\XINTdigits]{#2}}{#1}}%
1267 \def\XINT_flseqb::csv #1#2{\expandafter\XINT_flseqb::csv_a\romannumeral`&&#2#1!}%
1268 \def\XINT_flseqb::csv_a #1#2;#3;#4!\expandafter\XINT_expr_seq_empty?
1269 \romannumeral`&&\csname XINT_flseqb::csv_#1\endcsname {#3}{#4}{#2}}%
1270 \def\XINT_flseqb::csv_p #1#2#3%
1271 {%
1272 \xintifCmp {#1}{#2}{, #1\expandafter\XINT_flseqb::csv_p\expandafter}%
1273 {, #1\xint_gobble_iii}{\xint_gobble_iii}%
1274 {\romannumeral0\XINTinfloatadd {#3}{#1}}{#2}{#3}%
1275 }%
1276 \def\XINT_flseqb::csv_n #1#2#3%
1277 {%
1278 \xintifCmp {#1}{#2}{\xint_gobble_iii}{, #1\xint_gobble_iii}%
1279 {, #1\expandafter\XINT_flseqb::csv_n\expandafter}%
1280 {\romannumeral0\XINTinfloatadd {#3}{#1}}{#2}{#3}%
1281 }%
1282 \def\XINT_flseqb::csv_z #1#2#3{, #1}%

```

### 11.32 The comma as binary operator

New with 1.09a. Suffices to set its precedence level to two.

```

1283 \def\XINT_tmpa #1#2#3#4#5#6%
1284 {%
1285 \def #1##1% \XINT_expr_op_,
1286 {%
1287 \expandafter #2\expandafter ##1\romannumeral`&&\XINT_expr_getnext
1288 }%
1289 \def #2##1##2% \XINT_expr_until_,_a
1290 {\xint_UDsignfork
1291 ##2{\expandafter #2\expandafter ##1\romannumeral`&&#4}%
1292 -{#3##1##2}%
1293 \krof }%

```



```

1294 \def #3##1##2##3##4% \XINT_expr_until_,_b
1295 {%
1296 \ifnum ##2>\xint_c_ii
1297 \xint_afterfi {\expandafter #2\expandafter ##1\romannumeral`&&@%
1298 \csname XINT_#6_op_##3\endcsname {##4}}%
1299 \else
1300 \xint_afterfi
1301 {\expandafter ##2\expandafter ##3%
1302 \csname .=\XINT_expr_unlock ##1,\XINT_expr_unlock ##4\endcsname }%
1303 \fi
1304 }%
1305 \let #5\xint_c_ii
1306 }%
1307 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1308 \expandafter\XINT_tmpa
1309 \csname XINT_#1_op_,\expandafter\endcsname
1310 \csname XINT_#1_until_,_a\expandafter\endcsname
1311 \csname XINT_#1_until_,_b\expandafter\endcsname
1312 \csname XINT_#1_op_-vi\expandafter\endcsname
1313 \csname XINT_expr_precedence_,\endcsname {#1}%
1314 }%

```

### 11.33 The minus as prefix operator of variable precedence level

Inherits the precedence level of the previous infix operator.

```

1315 \def\XINT_tmpa #1#2#3%
1316 {%
1317 \expandafter\XINT_tmppb
1318 \csname XINT_#1_op_-#3\expandafter\endcsname
1319 \csname XINT_#1_until_-#3_a\expandafter\endcsname
1320 \csname XINT_#1_until_-#3_b\expandafter\endcsname
1321 \csname xint_c_#3\endcsname {#1}#2%
1322 }%
1323 \def\XINT_tmppb #1#2#3#4#5#6%
1324 {%
1325 \def #1% \XINT_expr_op_-<level>
1326 {% get next number+operator then switch to _until macro
1327 \expandafter #2\romannumeral`&&@\XINT_expr_getnext
1328 }%
1329 \def #2##1% \XINT_expr_until_-<l>_a
1330 {\xint_UDsignfork
1331 ##1{\expandafter #2\romannumeral`&&@#1}%
1332 -{#3##1}%
1333 \krof }%
1334 \def #3##1##2##3% \XINT_expr_until_-<l>_b
1335 {% _until tests precedence level with next op, executes now or postpones
1336 \ifnum ##1>#4%
1337 \xint_afterfi {\expandafter #2\romannumeral`&&@%
1338 \csname XINT_#5_op_##2\endcsname {##3}}%
1339 \else
1340 \xint_afterfi {\expandafter ##1\expandafter ##2%
1341 \csname .=%

```

```

1342          \XINT:NEhook:one#6{\XINT_expr_unlock ##3}\endcsname }%
1343      \fi
1344  }%
1345 }%

```

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.

```

1346 \xintApplyInline{\XINT_tmpa {expr}\xintOpp}{\vi{\vii{\viii}{ix}}}%
1347 \xintApplyInline{\XINT_tmpa {flexpr}\xintOpp}{\vi{\vii{\viii}{ix}}}%
1348 \xintApplyInline{\XINT_tmpa {iiexpr}\xintiiOpp}{\vi{\vii{\viii}{ix}}}%

```

### 11.34 ? 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.

```

1349 \let\XINT_expr_precedence_? \xint_c_x
1350 \def\XINT_expr_op_? #1#2%
1351   {\XINT_expr_op_?checka #2!\xint_bye\XINT_expr_op_?a #1{#2}}%
1352 \def\XINT_expr_op_?checka #1{\expandafter\XINT_expr_op_?checkb\detokenize{#1}}%
1353 \def\XINT_expr_op_?checkb #1{\if ?#1\expandafter\XINT_expr_op_?checkc
1354   \else\expandafter\xint_bye\fi }%
1355 \def\XINT_expr_op_?checkc #1{\xint_gob_til! #1\XINT_expr_op_?? !\xint_bye}%
1356 \def\XINT_expr_op_?a #1#2#3%
1357 {%
1358   \xintiiifNotZero{\XINT_expr_unlock #1}{\XINT_expr_getnext #2}{\XINT_expr_getnext #3}%
1359 }%
1360 \let\XINT_flexpr_op_?\XINT_expr_op_?
1361 \let\XINT_iiexpr_op_?\XINT_expr_op_?
1362 \def\XINT_expr_op_?? !\xint_bye\xint_bye\XINT_expr_op_?a #1#2#3#4#5%
1363 {%
1364   \xintiiifSgn {\XINT_expr_unlock #1}%
1365   {\XINT_expr_getnext #3}{\XINT_expr_getnext #4}{\XINT_expr_getnext #5}%
1366 }%

```

### 11.35 ! as postfix factorial operator

```

1367 \let\XINT_expr_precedence_! \xint_c_x
1368 \def\XINT_expr_op_! #1{\expandafter\XINT_expr_getop
1369   \csname .=\XINT:NEhook:one\xintFac{\XINT_expr_unlock #1}\endcsname }%
1370 \def\XINT_flexpr_op_! #1{\expandafter\XINT_expr_getop
1371   \csname .=\XINT:NEhook:one\XINTinFloatFac{\XINT_expr_unlock #1}\endcsname }%
1372 \def\XINT_iiexpr_op_! #1{\expandafter\XINT_expr_getop
1373   \csname .=\XINT:NEhook:one\xintiiFac{\XINT_expr_unlock #1}\endcsname }%

```

## 11.36 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, \xintiE, 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.

```

1374 \catcode`[ 11
1375 \let\XINT_expr_precedence_ \xint_c_vii
1376 \def\XINT_expr_op_[ #1#2]{\expandafter\XINT_expr_getop
1377     \csname .=\xintE{\XINT_expr_unlock #1}%
1378     {\xint_zapspaces #2 \xint_gobble_i}\endcsname}%
1379 \def\XINT_iiexpr_op_[ #1#2]{\expandafter\XINT_expr_getop
1380     \csname .=\xintiE{\XINT_expr_unlock #1}%
1381     {\xint_zapspaces #2 \xint_gobble_i}\endcsname}%
1382 \def\XINT_flexpr_op_[ #1#2]{\expandafter\XINT_expr_getop
1383     \csname .=\XINTinFloatE{\XINT_expr_unlock #1}%
1384     {\xint_zapspaces #2 \xint_gobble_i}\endcsname}%
1385 \catcode`[ 12

```

## 11.37 \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.

```

1386 \def\XINT_tmpa #1#2#3{%
1387     \def #1##1%
1388     {%
1389         \ifcsname XINT_#3_func_##1\endcsname
1390         \xint_dothis{\expandafter\expandafter
1391             \csname XINT_#3_func_##1\endcsname\romannumeral`&&@#2}\fi
1392         \ifcsname XINT_expr_onliteral_##1\endcsname
1393         \xint_dothis{\csname XINT_expr_onliteral_##1\endcsname}\fi
1394         \xint_orthat{\XINT_expr_unknown_function {##1}%
1395             \expandafter\XINT_expr_func_unknown\romannumeral`&&@#2}%
1396     }%
1397 }%
1398 \def\XINT_expr_unknown_function #1{\xintError:removed \xint_gobble_i {#1}}%
1399 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1400     \expandafter\XINT_tmpa
1401     \csname XINT_#1_op_`\expandafter\endcsname
1402     \csname XINT_#1_oparen\endcsname

```

```

1403             {#1}%
1404 }%
1405 \def\XINT_expr_func_unknown #1#2#3%
1406     {\expandafter #1\expandafter #2\csname .:=0\endcsname }%

```

### 11.38 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”.

```

1407 \def\XINT_expr_onliteral_bool #1)%
1408     {\expandafter\XINT_expr_getop\csname .:=\xintBool{#1}\endcsname }%
1409 \def\XINT_expr_onliteral_togl #1)%
1410     {\expandafter\XINT_expr_getop\csname .:=\xintToggle{#1}\endcsname }%
1411 \def\XINT_expr_onliteral_protect #1)%
1412     {\expandafter\XINT_expr_getop\csname .:=\detokenize{#1}\endcsname }%

```

### 11.39 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.

```

1413 \def\XINT_expr_func_break #1#2#3%
1414     {\expandafter #1\expandafter #2\csname .:=?\romannumeral`&&@\XINT_expr_unlock #3\endcsname }%
1415 \let\XINT_flexpr_func_break \XINT_expr_func_break
1416 \let\XINT_iexpr_func_break \XINT_expr_func_break

```

### 11.40 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.

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.

```

1417 \def\XINT_expr_onliteral_qint #1)%
1418     {\expandafter\XINT_expr_getop\csname .:=\xintiNum{#1}\endcsname }%
1419 \def\XINT_expr_onliteral_qfrac #1)%
1420     {\expandafter\XINT_expr_getop\csname .:=\xintRaw{#1}\endcsname }%
1421 \def\XINT_expr_onliteral_qfloat #1)%
1422     {\expandafter\XINT_expr_getop\csname .:=\XINTinFloatdigits{#1}\endcsname }%
1423 \def\XINT_expr_onliteral_qraw #1)%
1424     {\expandafter\XINT_expr_getop\csname .:=#1\endcsname }%

```

### 11.41 The `\random()` and `\qrand()` “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.

```

1425 \def\XINT_expr_onliteral_random #1)%

```

```

1426 {\expandafter\XINT_expr_getop\csname .=\XINTinRandomFloatSdigits\endcsname }%
1427 \def\XINT_expr_onliteral_grand #1)%
1428 {\expandafter\XINT_expr_getop\csname .=\XINTinRandomFloatSixteen\endcsname }%

```

## 11.42 \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`.

```

1429 \def\XINT_expr_op__ #1% op__ with two _'s
1430 {%
1431     \ifcsname XINT_expr_var_#1\endcsname
1432     \expandafter\xint_firstoftwo
1433     \else
1434     \expandafter\xint_secondoftwo
1435     \fi
1436     {\expandafter\expandafter\expandafter
1437     \XINT_expr_getop\csname XINT_expr_var_#1\endcsname}%
1438     {\XINT_expr_unknown_variable {#1}%
1439     \expandafter\XINT_expr_getop\csname .:=0\endcsname}%
1440 }%
1441 \def\XINT_expr_unknown_variable #1{\xintError:removed \xint_gobble_i {#1}}%
1442 \let\XINT_flexpr_op__ \XINT_expr_op__
1443 \let\XINT_iexpr_op__ \XINT_expr_op__

```

## 11.43 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.

```

1444 \catcode`* 11
1445 \def\XINT_expr_defvar_one #1#2%
1446 {%
1447     \XINT_global
1448     \expandafter\edef\csname XINT_expr_var_#1\endcsname

```

```

1449         {\expandafter\noexpand#2}%
1450 \XINT_global
1451 \expandafter\edef\csname XINT_expr_onliteral_#1\endcsname
1452     {\XINT_expr_precedence_*** *\expandafter\noexpand#2}%
1453 \ifxintverbose\xintMessage{xintexpr}{Info}
1454     {Variable "#1" \ifxintglobaldefs globally \fi
1455     defined with value \expandafter\XINT_expr_unlock#2.}%
1456 \fi
1457 }%
1458 \catcode`* 12
1459 \catcode`~ 13
1460 \catcode`: 12
1461 \def\XINT_expr_defvar_getname #1:#2~{\endgroup
1462     \def\XINT_expr_tmpa{#1}\edef\XINT_expr_tmpc{\xintCSVLength{#1}}}%
1463 \def\XINT_expr_defvar #1#2#3;%
1464 {%
1465     \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..).

```

1466 \begingroup\lccode`~: \lowercase{\let~}\empty
1467 \edef\XINT_expr_tmpa{#2}%
1468 \edef\XINT_expr_tmpa{\xint_zapspaces_o\XINT_expr_tmpa}%
1469 \expandafter\XINT_expr_defvar_getname
1470     \detokenize\expandafter{\XINT_expr_tmpa}:~%
1471 \ifcase\XINT_expr_tmpc
1472     \xintMessage {xintexpr}{Warning}
1473     {Aborting: not allowed to declare variable with empty name.}%
1474 \or
1475     \edef\XINT_expr_tmpb{\romannumeral0#1#3\relax}%
1476     \XINT_expr_defvar_one\XINT_expr_tmpa\XINT_expr_tmpb
1477 \else
1478     \edef\XINT_expr_tmpb
1479         {\expandafter\XINT_expr_unlock\romannumeral0#1#3\relax}%
1480     \edef\XINT_expr_tmpd{\xintCSVLength{\XINT_expr_tmpb}}%
1481     \ifnum\XINT_expr_tmpc=\XINT_expr_tmpd\space
1482         \xintAssignArray\xintCSVtoList\XINT_expr_tmpa\to\XINT_expr_tmpvar
1483         \xintAssignArray
1484         \xintApply\XINT_expr_lockit{\xintCSVtoList\XINT_expr_tmpb}%
1485         \to\XINT_expr_tmpval
1486     \def\XINT_expr_tmpd{1}%
1487     \xintloop
1488         \expandafter\XINT_expr_defvar_one
1489         \csname XINT_expr_tmpvar\XINT_expr_tmpd\expandafter\endcsname
1490         \csname XINT_expr_tmpval\XINT_expr_tmpd\endcsname
1491     \ifnum\XINT_expr_tmpd<\XINT_expr_tmpc\space
1492         \edef\XINT_expr_tmpd{\the\numexpr\XINT_expr_tmpd+1}%
1493     \repeat
1494     \xintRelaxArray\XINT_expr_tmpvar
1495     \xintRelaxArray\XINT_expr_tmpval
1496 \else

```

```

1497      \xintMessage {xintexpr}{Warning}
1498      {Aborting: mismatch between number of variables (\XINT_expr_tmpc)
1499      and number of values (\XINT_expr_tmpd).}%
1500    \fi
1501  \fi
1502 }%
1503 \catcode`~ 3
1504 \catcode`: 11

```

This SafeCatcodes is mainly in the hope that semi-colon ending the expression can still be sanitized.

```

1505 \def\xintdefvar      {\xintexprSafeCatcodes\xintdefvar_a}%
1506 \def\xintdefiivar    {\xintexprSafeCatcodes\xintdefiivar_a}%
1507 \def\xintdeffloatvar {\xintexprSafeCatcodes\xintdeffloatvar_a}%
1508 \def\xintdefvar_a    #1={\XINT_expr_defvar\xintbareeval    {#1}}%
1509 \def\xintdefiivar_a  #1={\XINT_expr_defvar\xintbareiieval  {#1}}%
1510 \def\xintdeffloatvar_a #1={\XINT_expr_defvar\xintbarefloateval {#1}}%

```

## 11.44 \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``.

```

1511 \def\xintunassignvar #1{%
1512   \edef\XINT_expr_tmpa{#1}%
1513   \edef\XINT_expr_tmpa {\xint_zapspace_o\XINT_expr_tmpa}%
1514   \ifcsname XINT_expr_var_\XINT_expr_tmpa\endcsname
1515     \ifnum\expandafter\xintLength\expandafter{\XINT_expr_tmpa}=\@ne
1516       \expandafter\xintnewdummy\XINT_expr_tmpa
1517     \else
1518       \XINT_global\expandafter
1519       \let\csname XINT_expr_var_\XINT_expr_tmpa\endcsname\xint_undefined
1520       \XINT_global\expandafter
1521       \let\csname XINT_expr_onliteral_\XINT_expr_tmpa\endcsname\xint_undefined
1522       \ifxintverbose\xintMessage {xintexpr}{Info}
1523       {Variable \XINT_expr_tmpa\space has been
1524       \ifxintglobaldefs globally \fi ``unassigned''.}%
1525     \fi
1526   \fi
1527 \else
1528   \xintMessage {xintexpr}{Warning}
1529   {Error: there was no such variable \XINT_expr_tmpa\space to unassign.}%
1530 \fi
1531 }%

```

## 11.45 seq and the implementation of dummy variables

11.45.1	All letters usable as dummy variables . . . . .	336
11.45.2	\omit() and \abort() . . . . .	337

11.45.3	The special variables @, @1, @2, @3, @4, @@, @@(1), ..., @@@, @@@(1), ... for recursion	338
11.45.4	\XINT_expr_onliteral_seq . . . . .	339
11.45.5	\XINT_expr_onliteral_seq_a . . . . .	339
11.45.6	\XINT_isbalanced_a for \XINT_expr_onliteral_seq_a . . . . .	339
11.45.7	\XINT_allexpr_func_seqx . . . . .	340
11.45.8	Evaluation over list, \XINT_expr_seq:_a with break, abort, omit . . . . .	340
11.45.9	Evaluation over ++ generated lists with \XINT_expr_seq:_A . . . . .	341

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.

### 11.45.1 All letters usable as dummy variables

The nil variable was introduced in 1.1 but isn't used under that name. However macros handling a..[d]..b, or for seq with dummy variable where omit has omitted everything may in practice inject a nil value as current number.

1.2c has changed the way variables are disambiguated from functions and for this it has added here the definitions of \XINT\_expr\_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.

```

1532 \catcode`* 11
1533 \def\XINT_expr_makedummy #1%
1534 {%
1535   \XINT_global
1536   \expandafter\def\csname XINT_expr_var_#1\endcsname ##1\relax !#1##2%

```



```

1537      {##2##1\relax !#1##2}%
1538  \XINT_global
1539  \expandafter\def\csname XINT_expr_onliteral_#1\endcsname ##1\relax !#1##2%
1540      {\XINT_expr_precedence_*** *##2(##1\relax !#1##2}%
1541 }%
1542 \xintApplyUnbraced \XINT_expr_makedummy {abcdefghijklmnopqrstuvwxyz}%
1543 \xintApplyUnbraced \XINT_expr_makedummy {ABCDEFGHIJKLMNOPQRSTUVWXYZ}%
1544 \def\xintnewdummy #1{%
1545     \XINT_expr_makedummy{#1}%
1546     \ifxintverbose\xintMessage {xintexpr}{Info}%
1547         {#1 (with letter catcode) now
1548         \ifxintglobaldefs globally \fi usable as dummy variable.}%
1549     \fi
1550 }%
1551 \edef\XINT_expr_var_nil {\expandafter\noexpand\csname . = \endcsname}%
1552 \edef\XINT_expr_onliteral_nil
1553     {\XINT_expr_precedence_*** *\expandafter\noexpand\csname . = \endcsname (}%
1554 \catcode`* 12

```

### 11.45.2 \omit() and \abort()

June 24 and 25, 2014.

Added comments 2015/11/13:

Et la documentation ? on n'y comprend plus rien. Trop rusé.

```
\def\XINT_expr_var_omit #1\relax !{1^C!}{ }{ }{ } \.=!\relax !}
```

```
\def\XINT_expr_var_abort #1\relax !{1^C!}{ }{ }{ } \.=^\relax !}
```

C'était accompagné de \XINT\_expr\_precedence\_^C=0 et d'un hack au sein même des macros until de plus bas niveau.

Le mécanisme sioux était le suivant: ^C est déclaré comme un opérateur de précedence nulle. Lorsque le parseur trouve un "omit" dans un seq ou autre, il va insérer dans le stream \XINT\_expr\_getop suivi du texte de remplacement. Donc ici on avait un 1 comme place holder, puis l'opérateur ^C. Celui-ci étant de précedence zéro provoque la finalisation de tous les calculs antérieurs dans le sous-bareeval. Mais j'ai dû hacker le until\_end\_b (et le until\_)\_b) qui confronté à ^C, va se relancer à zéro, le getnext va trouver le !{ }{ }{ } \.=! et ensuite il y aura \relax, et le résultat sera \.=! pour omit ou \.=^ pour abort. Les routines des boucles seq, iter, etc... peuvent alors repérer le ! ou ^ et agir en conséquence (un long paragraphe pour ne décrire que partiellement une ou deux lignes de codes...).

Mais ^C a été fait alors que je n'avais pas encore les variables muettes. Je dois trouver autre chose, car seq(2^C, C=1..5) est alors impossible. De toute façon ce ^C était à usage interne uniquement.

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

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

```

1555 \edef\XINT_expr_var_omit #1\relax !{1\string !?!\relax !}%
1556 \edef\XINT_expr_var_abort #1\relax !{1\string !?^\relax !}%
1557 \def\XINT_expr_op_!? #1#2\relax {\expandafter\XINT_expr_foundend\csname . = #2\endcsname}%
1558 \let\XINT_iiexpr_op_!? \XINT_expr_op_!?
1559 \let\XINT_flexpr_op_!? \XINT_expr_op_!?

```

### 11.45.3 The special variables @, @1, @2, @3, @4, @@, @@(1), ..., @@@, @@@(1), ... for recursion

October 2014: I had completely forgotten what the @@@ etc... stuff were supposed to do: this is for nesting recursions! (I was mad back in June). @@(N) gives the Nth back, @@@(N) gives the Nth back of the higher recursion!

1.2c adds the needed "onlital" 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.

```

1560 \catcode`? 3 \catcode`* 11
1561 \def\XINT_expr_var_@ #1~#2{#2#1~#2}%
1562 \expandafter\let\csname XINT_expr_var_@1\endcsname \XINT_expr_var_@
1563 \expandafter\def\csname XINT_expr_var_@2\endcsname #1~#2#3{#3#1~#2#3}%
1564 \expandafter\def\csname XINT_expr_var_@3\endcsname #1~#2#3#4{#4#1~#2#3#4}%
1565 \expandafter\def\csname XINT_expr_var_@4\endcsname #1~#2#3#4#5{#5#1~#2#3#4#5}%
1566 \def\XINT_expr_onlital_@ #1~#2{\XINT_expr_precedence_*** *#2(#1~#2)%
1567 \expandafter\let\csname XINT_expr_onlital_@1\endcsname \XINT_expr_onlital_@
1568 \expandafter\def\csname XINT_expr_onlital_@2\endcsname #1~#2#3%
1569         {\XINT_expr_precedence_*** *#3(#1~#2#3)%
1570 \expandafter\def\csname XINT_expr_onlital_@3\endcsname #1~#2#3#4%
1571         {\XINT_expr_precedence_*** *#4(#1~#2#3#4)%
1572 \expandafter\def\csname XINT_expr_onlital_@4\endcsname #1~#2#3#4#5%
1573         {\XINT_expr_precedence_*** *#5(#1~#2#3#4#5)%
1574 \catcode`* 12

1575 \def\XINT_expr_func_@@ #1#2#3#4~#5?%
1576 {%
1577     \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1578         {\xintNum{\XINT_expr_unlock#3}}{#5}#4~#5?%
1579 }%
1580 \def\XINT_expr_func_@@@ #1#2#3#4~#5~#6?%
1581 {%
1582     \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1583     {\xintNum{\XINT_expr_unlock#3}}{#6}#4~#5~#6?%
1584 }%
1585 \def\XINT_expr_func_@@@@ #1#2#3#4~#5~#6~#7?%
1586 {%
1587     \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1588     {\xintNum{\XINT_expr_unlock#3}}{#7}#4~#5~#6~#7?%
1589 }%
1590 \let\XINT_flexpr_func_@@\XINT_expr_func_@@
1591 \let\XINT_flexpr_func_@@@\XINT_expr_func_@@@
1592 \let\XINT_flexpr_func_@@@@\XINT_expr_func_@@@@
1593 \def\XINT_iexpr_func_@@ #1#2#3#4~#5?%
1594 {%
1595     \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1596     {\XINT_expr_unlock#3}{#5}#4~#5?%
1597 }%
1598 \def\XINT_iexpr_func_@@@ #1#2#3#4~#5~#6?%
1599 {%
1600     \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1601     {\XINT_expr_unlock#3}{#6}#4~#5~#6?%
1602 }%
1603 \def\XINT_iexpr_func_@@@@ #1#2#3#4~#5~#6~#7?%

```

```
1604 {%
1605   \expandafter#1\expandafter#2\romannumeral0\xintntheltnoexpand
1606   {\XINT_expr_unlock#3}{#7}#4~#5~#6~#7?%
1607 }%
1608 \catcode`? 11
```

#### 11.45.4 \XINT\_expr\_onliteral\_seq

```
1609 \def\XINT_expr_onliteral_seq
1610 {\expandafter\XINT_expr_onliteral_seq_f\romannumeral`&&\XINT_expr_onliteral_seq_a {}}%
1611 \def\XINT_expr_onliteral_seq_f #1#2{\xint_c_xviii `{seqx}#2)\relax #1}%
```

#### 11.45.5 \XINT\_expr\_onliteral\_seq\_a

```
1612 \def\XINT_expr_onliteral_seq_a #1#2,%
1613 {%
1614   \ifcase\XINT_isbalanced_a \relax #1#2(\xint_bye)\xint_bye
1615   \expandafter\XINT_expr_onliteral_seq_c
1616   \or\expandafter\XINT_expr_onliteral_seq_b
1617   \else\expandafter\xintError:we_are_doomed
1618   \fi {#1#2},%
1619 }%
1620 \def\XINT_expr_onliteral_seq_b #1,{\XINT_expr_onliteral_seq_a {#1},}%
1621 \def\XINT_expr_onliteral_seq_c #1,#2#3% #3 pour absorber le =
1622 {%
1623   \XINT_expr_onliteral_seq_d {#2{#1}}}%
1624 }%
1625 \def\XINT_expr_onliteral_seq_d #1#2#3)%
1626 {%
1627   \ifcase\XINT_isbalanced_a \relax #2#3(\xint_bye)\xint_bye
1628   \or\expandafter\XINT_expr_onliteral_seq_e
1629   \else\expandafter\xintError:we_are_doomed
1630   \fi
1631   {#1}{#2#3}%
1632 }%
1633 \def\XINT_expr_onliteral_seq_e #1#2{\XINT_expr_onliteral_seq_d {#1}{#2}}%
```

#### 11.45.6 \XINT\_isbalanced\_a for \XINT\_expr\_onliteral\_seq\_a

Expands to \xint\_c\_mone in case a closing ) had no opening ( matching it, to \@ne if opening ) had no closing ) matching it, to \z@ if expression was balanced.

```
1634 % use as \XINT_isbalanced_a \relax #1(\xint_bye)\xint_bye
1635 \def\XINT_isbalanced_a #1({\XINT_isbalanced_b #1)\xint_bye }%
1636 \def\XINT_isbalanced_b #1)#2%
1637   {\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

1638 \def\XINT_isbalanced_error #1)\xint_bye {\xint_c_mone}%

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

1639 \def\XINT_isbalanced_c\xint_bye\XINT_isbalanced_error #1%
1640   {\xint_bye #1\XINT_isbalanced_yes\xint_bye\XINT_isbalanced_d #1}%

```

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfac, xintexpr, indices*

#1 is \xint\_bye, there was never ( nor ) in original #1, hence OK.

```
1641 \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.

```
1642 \def\xINT_isbalanced_d #1)#2%
```

```
1643 {\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.

```
1644 \def\xINT_isbalanced_no\xint_bye #1\xint_bye\xint_bye {\xint_c_i }%
```

### 11.45.7 \XINT\_allexpr\_func\_seqx

1.2c uses \xintthebareval, ... which strangely were not available at 1.1 time. This spares some tokens from \XINT\_expr\_seq:\_d and cousins. Also now variables have changed their mode of operation they pick only one token which must be an already encapsulated value.

In \XINT\_allexp\_seqx, #2 is the list, evaluated and encapsulated, #3 is the dummy variable, #4 is the expression to evaluate repeatedly.

A special case is a list generated by <variable>+: then #2 is {\.=+\.=<start>}.

```
1645 \def\xINT_expr_func_seqx #1#2{\XINT_allexpr_seqx \xintthebareval }%
1646 \def\xINT_flexpr_func_seqx #1#2{\XINT_allexpr_seqx \xintthebarefloateval}%
1647 \def\xINT_iexpr_func_seqx #1#2{\XINT_allexpr_seqx \xintthebareiieval }%
1648 \def\xINT_allexpr_seqx #1#2#3#4%
1649 {%
1650   \expandafter \XINT_expr_getop
1651   \csname .=\expandafter\xINT_expr_seq:_aa
1652     \romannumeral`&&\XINT_expr_unlock #2!{#1#4\relax !#3}\endcsname
1653 }%
1654 \def\xINT_expr_seq:_aa #1{\if +#1\expandafter\xINT_expr_seq:_A\else
1655   \expandafter\xINT_expr_seq:_a\fi #1}%

```

### 11.45.8 Evaluation over list, \XINT\_expr\_seq:\_a with break, abort, omit

The #2 here is \...bareval <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.)

```
1656 \def\xINT_expr_seq:_a #1!#2{\expandafter\xINT_expr_seq_empty?
1657   \romannumeral0\xINT_expr_seq:_b {#2}#1,^,%}
1658 \def\xINT_expr_seq:_b #1#2#3,{%
1659   \if ,#2\xint_dothis\xINT_expr_seq:_noop\fi
1660   \if ^#2\xint_dothis\xINT_expr_seq:_end\fi
1661   \xint_orthat{\expandafter\xINT_expr_seq:_c}\csname.=#2#3\endcsname {#1}%
1662 }%
1663 \def\xINT_expr_seq:_noop\csname.=#1\endcsname #2{\XINT_expr_seq:_b {#2}#1,%}

```

```

1664 \def\XINT_expr_seq:_end \csname.=^ \endcsname #1{%
1665 \def\XINT_expr_seq:_c #1#2{\expandafter\XINT_expr_seq:_d\romannumeral`&&@#2#1{#2}}%
1666 \def\XINT_expr_seq:_d #1{\if #1^ \xint_dothis\XINT_expr_seq:_abort\fi
1667         \if #1? \xint_dothis\XINT_expr_seq:_break\fi
1668         \if #1! \xint_dothis\XINT_expr_seq:_omit\fi
1669         \xint_orthat{\XINT_expr_seq:_goon #1}}%
1670 \def\XINT_expr_seq:_abort #1!#2#3#4#5^,{}%
1671 \def\XINT_expr_seq:_break #1!#2#3#4#5^,{,#1}%
1672 \def\XINT_expr_seq:_omit #1!#2#3#4{\XINT_expr_seq:_b {#4}}%
1673 \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`.

```

1674 \def\XINT_expr_seq_empty? #1{%
1675 \def\XINT_expr_seq_empty? ##1{\if ,##1\expandafter\xint_gobble_i\fi #1\endcsname }}%
1676 \XINT_expr_seq_empty? { }%

```

#### 11.45.9 Evaluation over ++ generated lists with `\XINT_expr_seq:_A`

This is for index lists generated by `n++`. The starting point will have been replaced by its ceil (added: in fact with version 1.1. the ceil was not yet evaluated, but `_var_<letter>` did an expansion of what they fetch). We use `\numexpr` rather than `\xintInc`, hence the indexing is limited to small integers.

The 1.2c version of `n++` produces a `#1` here which is already a single `\.=<value>` token.

```

1677 \def\XINT_expr_seq:_A +#1!%
1678   {\expandafter\XINT_expr_seq_empty?\romannumeral0\XINT_expr_seq:_D #1}%
1679 \def\XINT_expr_seq:_D #1#2{\expandafter\XINT_expr_seq:_E\romannumeral`&&@#2#1{#2}}%
1680 \def\XINT_expr_seq:_E #1{\if #1^ \xint_dothis\XINT_expr_seq:_Abort\fi
1681         \if #1? \xint_dothis\XINT_expr_seq:_Break\fi
1682         \if #1! \xint_dothis\XINT_expr_seq:_Omit\fi
1683         \xint_orthat{\XINT_expr_seq:_Goon #1}}%
1684 \def\XINT_expr_seq:_Abort #1!#2#3#4{%
1685 \def\XINT_expr_seq:_Break #1!#2#3#4{,#1}%
1686 \def\XINT_expr_seq:_Omit #1!#2#3%
1687   {\expandafter\XINT_expr_seq:_D
1688     \csname.= \the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
1689 \def\XINT_expr_seq:_Goon #1!#2#3%
1690   {,#1\expandafter\XINT_expr_seq:_D
1691     \csname.= \the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%

```

#### 11.46 `\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.

```

1692 \def\XINT_expr_onliteral_add
1693   {\expandafter\XINT_expr_onliteral_add_f\romannumeral`&&\XINT_expr_onliteral_seq_a {}}%
1694 \def\XINT_expr_onliteral_add_f #1#2{\xint_c_xviii `{opxadd}#2)\relax #1}%
1695 \def\XINT_expr_onliteral_mul

```

```
1696 {\expandafter\XINT_expr_onliteral_mul_f\romannumeral`&&\XINT_expr_onliteral_seq_a {}}%
1697 \def\XINT_expr_onliteral_mul_f #1#2{\xint_c_xviii `{opxmul}#2)\relax #1}%
```

### 11.46.1 \XINT\_expr\_func\_opxadd, \XINT\_flexpr\_func\_opxadd, \XINT\_iiexpr\_func\_opxadd and same for mul

modified 1.2c.

```
1698 \def\XINT_expr_func_opxadd #1#2{\XINT_allexpr_opx \xintbareeval {\xintAdd 0}}%
1699 \def\XINT_flexpr_func_opxadd #1#2{\XINT_allexpr_opx \xintbarefloateval {\XINTinFloatAdd 0}}%
1700 \def\XINT_iiexpr_func_opxadd #1#2{\XINT_allexpr_opx \xintbareiieval {\xintiiAdd 0}}%
1701 \def\XINT_expr_func_opxmul #1#2{\XINT_allexpr_opx \xintbareeval {\xintMul 1}}%
1702 \def\XINT_flexpr_func_opxmul #1#2{\XINT_allexpr_opx \xintbarefloateval {\XINTinFloatMul 1}}%
1703 \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

```
1704 \def\XINT_allexpr_opx #1#2#3#4#5%
1705 {%
1706   \expandafter\XINT_expr_getop
1707   \csname.=\romannumeral`&&\expandafter\XINT_expr_op:_a
1708     \romannumeral`&&\XINT_expr_unlock #3!{#1#5\relax !#4}{#2}\endcsname
1709 }%
1710 \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.

```
1711 \def\XINT_expr_op:_b #1#2#3#4#5,{%
1712   \if ,#4\xint_dothis\XINT_expr_op:_noop\fi
1713   \if ^#4\xint_dothis\XINT_expr_op:_end\fi
1714   \xint_orthat{\expandafter\XINT_expr_op:_c}\csname.=#4#5\endcsname {#3}#1{#2}%
1715 }%
1716 \def\XINT_expr_op:_c #1#2#3#4%
1717   {\expandafter\XINT_expr_op:_d\romannumeral0#2#1#3{#4}{#2}}%
1718 \def\XINT_expr_op:_d #1!#2#3#4#5%
1719   {\expandafter\XINT_expr_op:_b\expandafter #4\expandafter
1720     \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.

```
1721 \def\XINT_expr_op:_noop\csname.=,#1\endcsname #2#3#4{\XINT_expr_op:_b #3{#4}{#2}#1,}%
1722 \def\XINT_expr_op:_end \csname.=^ \endcsname #1#2#3{#3}%
```

### 11.47 \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.

```
1723 \def\XINT_expr_onliteral_subs
1724   {\expandafter\XINT_expr_onliteral_subs_f\romannumeral`&&\XINT_expr_onliteral_seq_a {}}%
1725 \def\XINT_expr_onliteral_subs_f #1#2{\xint_c_xviii `{subx}#2)\relax #1}%

```

```

1726 \def\XINT_expr_func_subx #1#2{\XINT_allexpr_subx \xintbareeval }%
1727 \def\XINT_flexpr_func_subx #1#2{\XINT_allexpr_subx \xintbarefloateval}%
1728 \def\XINT_iiexpr_func_subx #1#2{\XINT_allexpr_subx \xintbareiieval }%
1729 \def\XINT_allexpr_subx #1#2#3#4% #2 is the value to assign to the dummy variable
1730 {% #3 is the dummy variable, #4 is the expression to evaluate
1731     \expandafter\expandafter\expandafter\XINT_expr_getop
1732     \expandafter\XINT_expr_subx:_end\romannumeral0#1#4\relax !#3#2%
1733 }%
1734 \def\XINT_expr_subx:_end #1!#2#3{#1}%

```

## 11.48 \rseq()

11.48.1	\XINT_expr_rseqx	343
11.48.2	\XINT_expr_rseqy	343
11.48.3	\XINT_expr_rseq:_a etc...	344
11.48.4	\XINT_expr_rseq:_A etc...	344

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...

```

1735 \def\XINT_expr_func_rseq {\XINT_allexpr_rseq \xintbareeval \xintthebareeval }%
1736 \def\XINT_flexpr_func_rseq {\XINT_allexpr_rseq \xintbarefloateval \xintthebarefloateval }%
1737 \def\XINT_iiexpr_func_rseq {\XINT_allexpr_rseq \xintbareiieval \xintthebareiieval }%
1738 \def\XINT_allexpr_rseq #1#2#3%
1739 {%
1740     \expandafter\XINT_expr_rseqx\expandafter #1\expandafter#2\expandafter
1741     #3\romannumeral`&&\XINT_expr_onliteral_seq_a {}%
1742 }%

```

### 11.48.1 \XINT\_expr\_rseqx

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

```

1743 \def\XINT_expr_rseqx #1#2#3#4#5%
1744 {%
1745     \expandafter\XINT_expr_rseqy\romannumeral0#1(#5)\relax #3#4#2%
1746 }%

```

### 11.48.2 \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

```

1747 \def\XINT_expr_rseqy #1#2#3#4#5%
1748 {%
1749     \expandafter \XINT_expr_getop
1750     \csname .=\XINT_expr_unlock #2%
1751     \expandafter\XINT_expr_rseq:_aa
1752     \romannumeral`&&\XINT_expr_unlock #1!{#5#4\relax !#3}#2\endcsname
1753 }%

```



```
1754 \def\XINT_expr_rseq:_aa #1{\if +#1\expandafter\XINT_expr_rseq:_A\else
1755 \expandafter\XINT_expr_rseq:_a\fi #1}%
```

### 11.48.3 \XINT\_expr\_rseq:\_a etc...

```
1756 \def\XINT_expr_rseq:_a #1!#2#3{\XINT_expr_rseq:_b {#3}{#2}#1,^,%}
1757 \def\XINT_expr_rseq:_b #1!#2#3#4,{%
1758 \if ,#3\xint_dothis\XINT_expr_rseq:_noop\fi
1759 \if ^#3\xint_dothis\XINT_expr_rseq:_end\fi
1760 \xint_orthat{\expandafter\XINT_expr_rseq:_c}\csname.=#3#4\endcsname
1761 {#1}{#2}%
1762 }%
1763 \def\XINT_expr_rseq:_noop\csname.=,#1\endcsname #2#3{\XINT_expr_rseq:_b {#2}{#3}#1,%}
1764 \def\XINT_expr_rseq:_end \csname.=^\endcsname #1#2{%}
1765 \def\XINT_expr_rseq:_c #1!#2#3%
1766 {\expandafter\XINT_expr_rseq:_d\romannumeral`&&@#3#1~#2{#3}}%
1767 \def\XINT_expr_rseq:_d #1{%
1768 \if ^#1\xint_dothis\XINT_expr_rseq:_abort\fi
1769 \if ?#1\xint_dothis\XINT_expr_rseq:_break\fi
1770 \if !#1\xint_dothis\XINT_expr_rseq:_omit\fi
1771 \xint_orthat{\XINT_expr_rseq:_goon #1}%}
1772 \def\XINT_expr_rseq:_goon #1!#2#3~#4#5{,#1\expandafter\XINT_expr_rseq:_b
1773 \romannumeral0\XINT_expr_lockit {#1}{#5}}%
1774 \def\XINT_expr_rseq:_omit #1!#2#3~{\XINT_expr_rseq:_b }%
1775 \def\XINT_expr_rseq:_abort #1!#2#3~#4#5#6^,{,%}
1776 \def\XINT_expr_rseq:_break #1!#2#3~#4#5#6^,{,#1}%
```

### 11.48.4 \XINT\_expr\_rseq:\_A etc...

*n++ for rseq. With 1.2c dummy variables pick a single token.*

```
1777 \def\XINT_expr_rseq:_A +#1!#2#3{\XINT_expr_rseq:_D #1#3{#2}}%
1778 \def\XINT_expr_rseq:_D #1!#2#3%
1779 {\expandafter\XINT_expr_rseq:_E\romannumeral`&&@#3#1~#2{#3}}%
1780 \def\XINT_expr_rseq:_E #1{\if #1^\xint_dothis\XINT_expr_rseq:_Abort\fi
1781 \if #1?\xint_dothis\XINT_expr_rseq:_Break\fi
1782 \if #1!\xint_dothis\XINT_expr_rseq:_Omit\fi
1783 \xint_orthat{\XINT_expr_rseq:_Goon #1}}%
1784 \def\XINT_expr_rseq:_Goon #1!#2#3~#4#5%
1785 {,#1\expandafter\XINT_expr_rseq:_D
1786 \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1787 \romannumeral0\XINT_expr_lockit{#1}{#5}}%
1788 \def\XINT_expr_rseq:_Omit #1!#2#3~%#4#5%
1789 {\expandafter\XINT_expr_rseq:_D
1790 \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname }%
1791 \def\XINT_expr_rseq:_Abort #1!#2#3~#4#5{%}
1792 \def\XINT_expr_rseq:_Break #1!#2#3~#4#5{,#1}%
```

## 11.49 \iter()

11.49.1	\XINT_expr_iterx	345
11.49.2	\XINT_expr_itory	345
11.49.3	\XINT_expr_iter:_a etc...	345
11.49.4	\XINT_expr_iter:_A etc...	346



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.

```

1793 \def\XINT_expr_func_iter    {\XINT_allexpr_iter \xintbareeval      \xintthebareeval      }%
1794 \def\XINT_flexpr_func_iter  {\XINT_allexpr_iter \xintbarefloateval \xintthebarefloateval }%
1795 \def\XINT_iiexpr_func_iter  {\XINT_allexpr_iter \xintbareiieval    \xintthebareiieval    }%
1796 \def\XINT_allexpr_iter      #1#2#3%
1797 {%
1798     \expandafter\XINT_expr_iterx\expandafter #1\expandafter#2\expandafter
1799     #3\romannumeral`&&\XINT_expr_onliteral_seq_a }%
1800 }%

```

### 11.49.1 \XINT\_expr\_iterx

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

```

1801 \def\XINT_expr_iterx #1#2#3#4#5%
1802 {%
1803     \expandafter\XINT_expr_itery\romannumeral0#1(#5)\relax #3#4#2%
1804 }%

```

### 11.49.2 \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

```

1805 \def\XINT_expr_itery #1#2#3#4#5%
1806 {%
1807     \expandafter \XINT_expr_getop
1808     \csname .=%
1809     \expandafter\XINT_expr_iter:_aa
1810     \romannumeral`&&\XINT_expr_unlock #1!{#5#4\relax !#3}#2\endcsname
1811 }%
1812 \def\XINT_expr_iter:_aa #1{\if +#1\expandafter\XINT_expr_iter:_A\else
1813     \expandafter\XINT_expr_iter:_a\fi #1}%

```

### 11.49.3 \XINT\_expr\_iter:\_a etc...

```

1814 \def\XINT_expr_iter:_a #1!#2#3{\XINT_expr_iter:_b {#3}{#2}#1,^,%}
1815 \def\XINT_expr_iter:_b #1#2#3#4,{%
1816     \if ,#3\xint_dothis\XINT_expr_iter:_noop\fi
1817     \if ^#3\xint_dothis\XINT_expr_iter:_end\fi
1818     \xint_orthat{\expandafter\XINT_expr_iter:_c}%
1819     \csname.=#3#4\endcsname {#1}{#2}%
1820 }%
1821 \def\XINT_expr_iter:_noop\csname.=#1\endcsname #2#3{\XINT_expr_iter:_b {#2}{#3}#1,%}
1822 \def\XINT_expr_iter:_end \csname.=^'\endcsname #1#2{\XINT_expr:_unlock #1}%
1823 \def\XINT_expr_iter:_c #1#2#3%
1824     {\expandafter\XINT_expr_iter:_d\romannumeral`&&#3#1~#2{#3}}%
1825 \def\XINT_expr_iter:_d #1{%

```

```

1826 \if ^#1\xint_dothis\XINT_expr_iter:_abort\fi
1827 \if ?#1\xint_dothis\XINT_expr_iter:_break\fi
1828 \if !#1\xint_dothis\XINT_expr_iter:_omit\fi
1829 \xint_orthat{\XINT_expr_iter:_goon #1}%
1830 \def\XINT_expr_iter:_goon #1!#2#3~#4#5%
1831 {\expandafter\XINT_expr_iter:_b\romannumeral0\XINT_expr_lockit {#1}{#5}}%
1832 \def\XINT_expr_iter:_omit #1!#2#3~{\XINT_expr_iter:_b }%
1833 \def\XINT_expr_iter:_abort #1!#2#3~#4#5#6^,{\XINT_expr_unlock #4}%
1834 \def\XINT_expr_iter:_break #1!#2#3~#4#5#6^,{#1}%

```

#### 11.49.4 \XINT\_expr\_iter:\_A etc...

n++ for iter. With 1.2c dummy variables pick a single token.

```

1835 \def\XINT_expr_iter:_A +#1!#2#3{\XINT_expr_iter:_D #1#3{#2}}%
1836 \def\XINT_expr_iter:_D #1#2#3%
1837 {\expandafter\XINT_expr_iter:_E\romannumeral`&&@#3#1~#2{#3}}%
1838 \def\XINT_expr_iter:_E #1{\if #1^\xint_dothis\XINT_expr_iter:_Abort\fi
1839 \if #1?\xint_dothis\XINT_expr_iter:_Break\fi
1840 \if #1!\xint_dothis\XINT_expr_iter:_Omit\fi
1841 \xint_orthat{\XINT_expr_iter:_Goon #1}}%
1842 \def\XINT_expr_iter:_Goon #1!#2#3~#4#5%
1843 {\expandafter\XINT_expr_iter:_D
1844 \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1845 \romannumeral0\XINT_expr_lockit{#1}{#5}}%
1846 \def\XINT_expr_iter:_Omit #1!#2#3~%#4#5%
1847 {\expandafter\XINT_expr_iter:_D
1848 \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname }%
1849 \def\XINT_expr_iter:_Abort #1!#2#3~#4#5{\XINT_expr:_unlock #4}%
1850 \def\XINT_expr_iter:_Break #1!#2#3~#4#5{#1}%

```

### 11.50 \rrseq()

11.50.1	\XINT_expr_rrseqx . . . . .	346
11.50.2	\XINT_expr_rrseqy . . . . .	347
11.50.3	\XINT_expr_rrseq:_a etc. . . . .	347
11.50.4	\XINT_expr_rrseq:_A etc. . . . .	348

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.

```

1851 \def\XINT_expr_func_rrseq {\XINT_allexpr_rrseq \xintbareeval \xintthebareeval }%
1852 \def\XINT_flexpr_func_rrseq {\XINT_allexpr_rrseq \xintbarefloateval \xintthebarefloateval }%
1853 \def\XINT_iiexpr_func_rrseq {\XINT_allexpr_rrseq \xintbareiieval \xintthebareiieval }%
1854 \def\XINT_allexpr_rrseq #1#2#3%
1855 {%
1856 \expandafter\XINT_expr_rrseqx\expandafter #1\expandafter#2\expandafter
1857 #3\romannumeral`&&\XINT_expr_onliteral_seq_a {}}%
1858 }%

```

#### 11.50.1 \XINT\_expr\_rrseqx

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

```

1859 \def\XINT_expr_rrseqx #1#2#3#4#5%
1860 {%
1861     \expandafter\XINT_expr_rrseq\romannumeral0#1(#5)\expandafter\relax
1862     \expandafter{\romannumeral0\xintapply \XINT_expr_lockit
1863         {\xintRevWithBraces{\xintCSVtoListNonStripped{\XINT_expr_unlock #3}}}}%
1864     #3#4#2%
1865 }%

```

### 11.50.2 \XINT\_expr\_rrseq

#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 \xintthebare-floateval ou \xintthebareiieval

```

1866 \def\XINT_expr_rrseq #1#2#3#4#5#6%
1867 {%
1868     \expandafter \XINT_expr_getop
1869     \csname .=\XINT_expr_unlock #3%
1870     \expandafter\XINT_expr_rrseq:_aa
1871         \romannumeral`&&@\XINT_expr_unlock #1!{#6#5\relax !#4}{#2}\endcsname
1872 }%
1873 \def\XINT_expr_rrseq:_aa #1{\if +#1\expandafter\XINT_expr_rrseq:_A\else
1874     \expandafter\XINT_expr_rrseq:_a\fi #1}%

```

### 11.50.3 \XINT\_expr\_rrseq:\_a etc...

Attention que ? a catcode 3 ici et dans iter.

```

1875 \catcode`? 3
1876 \def\XINT_expr_rrseq:_a #1!#2#3{\XINT_expr_rrseq:_b {#3}{#2}#1,^,%}
1877 \def\XINT_expr_rrseq:_b #1#2#3#4,{%
1878     \if ,#3\xint_dothis\XINT_expr_rrseq:_noop\fi
1879     \if ^#3\xint_dothis\XINT_expr_rrseq:_end\fi
1880     \xint_orthat{\expandafter\XINT_expr_rrseq:_c}\csname.=#3#4\endcsname
1881     {#1}{#2}%
1882 }%
1883 \def\XINT_expr_rrseq:_noop\csname.=,#1\endcsname #2#3{\XINT_expr_rrseq:_b {#2}{#3}#1,%}
1884 \def\XINT_expr_rrseq:_end \csname.=^\endcsname #1#2{%}
1885 \def\XINT_expr_rrseq:_c #1#2#3%
1886     {\expandafter\XINT_expr_rrseq:_d\romannumeral`&&@#3#1~#2?{#3}}%
1887 \def\XINT_expr_rrseq:_d #1{%
1888     \if ^#1\xint_dothis\XINT_expr_rrseq:_abort\fi
1889     \if ?#1\xint_dothis\XINT_expr_rrseq:_break\fi
1890     \if !#1\xint_dothis\XINT_expr_rrseq:_omit\fi
1891     \xint_orthat{\XINT_expr_rrseq:_goon #1}%
1892 }%
1893 \def\XINT_expr_rrseq:_goon #1!#2#3~#4?#5{,#1\expandafter\XINT_expr_rrseq:_b\expandafter
1894     {\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}{#4}}{#5}}%
1895 \def\XINT_expr_rrseq:_omit #1!#2#3~{\XINT_expr_rrseq:_b }%
1896 \def\XINT_expr_rrseq:_abort #1!#2#3~#4?#5#6^,{,%}
1897 \def\XINT_expr_rrseq:_break #1!#2#3~#4?#5#6^,{,#1}%

```

#### 11.50.4 \XINT\_expr\_rrseq:\_A etc...

*n++ for rrseq. With 1.2C, the #1 in \XINT\_expr\_rrseq:\_A is a single token.*

```
1898 \def\XINT_expr_rrseq:_A +#1!#2#3{\XINT_expr_rrseq:_D #1{#3}{#2}}%
1899 \def\XINT_expr_rrseq:_D #1#2#3%
1900   {\expandafter\XINT_expr_rrseq:_E\romannumeral`&&@#3#1~#2?{#3}}%
1901 \def\XINT_expr_rrseq:_Goon #1!#2#3~#4?#5%
1902   {,#1\expandafter\XINT_expr_rrseq:_D
1903     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1904     \expandafter{\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}#4}}{#5}}%
1905 \def\XINT_expr_rrseq:_Omit #1!#2#3~%#4?#5%
1906   {\expandafter\XINT_expr_rrseq:_D
1907     \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
1908 \def\XINT_expr_rrseq:_Abort #1!#2#3~#4?#5{}%
1909 \def\XINT_expr_rrseq:_Break #1!#2#3~#4?#5{,#1}%
1910 \def\XINT_expr_rrseq:_E #1{\if #1^\xint_dothis\XINT_expr_rrseq:_Abort\fi
1911     \if #1?\xint_dothis\XINT_expr_rrseq:_Break\fi
1912     \if #1!\xint_dothis\XINT_expr_rrseq:_Omit\fi
1913     \xint_orthat{\XINT_expr_rrseq:_Goon #1}}%
```

### 11.51 \iterr()

11.51.1	\XINT_expr_iterrx . . . . .	348
11.51.2	\XINT_expr_itterry . . . . .	348
11.51.3	\XINT_expr_iterr:_a etc. . . . .	349
11.51.4	\XINT_expr_iterr:_A etc. . . . .	349

```
1914 \def\XINT_expr_func_iterr {\XINT_allexpr_iterr \xintbareeval \xintthebareeval }%
1915 \def\XINT_flexpr_func_iterr {\XINT_allexpr_iterr \xintbarefloateval \xintthebarefloateval }%
1916 \def\XINT_iiexpr_func_iterr {\XINT_allexpr_iterr \xintbareiieval \xintthebareiieval }%
1917 \def\XINT_allexpr_iterr #1#2#3%
1918 {%
1919   \expandafter\XINT_expr_iterrx\expandafter #1\expandafter #2\expandafter
1920   #3\romannumeral`&&@\XINT_expr_onliteral_seq_a {}%
1921 }%
```

#### 11.51.1 \XINT\_expr\_iterrx

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

```
1922 \def\XINT_expr_iterrx #1#2#3#4#5%
1923 {%
1924   \expandafter\XINT_expr_itterry\romannumeral0#1(#5)\expandafter\relax
1925   \expandafter{\romannumeral0\xintapply \XINT_expr_lockit
1926     {\xintRevWithBraces{\xintCSVtoListNonStripped{\XINT_expr_unlock #3}}}%
1927   #3#4#2%
1928 }%
```

#### 11.51.2 \XINT\_expr\_itterry

*#1=valeurs pour variable (locked), #2=initial values (reversed, one (braced) token each) #3=toutes les valeurs initiales (csv,locked), #4=variable, #5=expr, #6=\xintbareeval ou \xintbarefloateval ou \xintbareiieval*

```

1929 \def\XINT_expr_iterrry #1#2#3#4#5#6%
1930 {%
1931     \expandafter \XINT_expr_getop
1932     \csname .=%
1933     \expandafter\XINT_expr_iterr:_aa
1934     \romannumeral`&&\XINT_expr_unlock #1!{#6#5\relax !#4}{#2}\endcsname
1935 }%
1936 \def\XINT_expr_iterr:_aa #1{\if +#1\expandafter\XINT_expr_iterr:_A\else
1937     \expandafter\XINT_expr_iterr:_a\fi #1}%

```

### 11.51.3 \XINT\_expr\_iterr:\_a etc...

```

1938 \def\XINT_expr_iterr:_a #1!#2#3{\XINT_expr_iterr:_b {#3}{#2}#1,^,%}
1939 \def\XINT_expr_iterr:_b #1#2#3#4,{%
1940     \if ,#3\xint_dothis\XINT_expr_iterr:_noop\fi
1941     \if ^#3\xint_dothis\XINT_expr_iterr:_end\fi
1942     \xint_orthat{\expandafter\XINT_expr_iterr:_c}%
1943     \csname.=#3#4\endcsname {#1}{#2}%
1944 }%
1945 \def\XINT_expr_iterr:_noop\csname.=,#1\endcsname #2#3{\XINT_expr_iterr:_b {#2}{#3}#1,%}
1946 \def\XINT_expr_iterr:_end \csname.=^{\endcsname #1#2%
1947     {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1948         {,\XINT_expr:_unlock}{\xintReverseOrder{#1\space}}}%
1949 \def\XINT_expr_iterr:_c #1#2#3%
1950     {\expandafter\XINT_expr_iterr:_d\romannumeral`&&@#3#1~#2?{#3}}%
1951 \def\XINT_expr_iterr:_d #1{%
1952     \if ^#1\xint_dothis\XINT_expr_iterr:_abort\fi
1953     \if ?#1\xint_dothis\XINT_expr_iterr:_break\fi
1954     \if !#1\xint_dothis\XINT_expr_iterr:_omit\fi
1955     \xint_orthat{\XINT_expr_iterr:_goon #1}%
1956 }%
1957 \def\XINT_expr_iterr:_goon #1!#2#3~#4?#5{\expandafter\XINT_expr_iterr:_b\expandafter
1958     {\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}{#4}}{#5}}%
1959 \def\XINT_expr_iterr:_omit #1!#2#3~{\XINT_expr_iterr:_b }%
1960 \def\XINT_expr_iterr:_abort #1!#2#3~#4?#5#6^,%
1961     {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1962         {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}}}%
1963 \def\XINT_expr_iterr:_break #1!#2#3~#4?#5#6^,%
1964     {\expandafter\xint_gobble_iv\romannumeral0\xintapplyunbraced
1965         {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}},#1}%
1966 \def\XINT_expr:_unlock #1{\XINT_expr_unlock #1}%

```

### 11.51.4 \XINT\_expr\_iterr:\_A etc...

*n++ for iterr. ? is of catcode 3 here.*

```

1967 \def\XINT_expr_iterr:_A +#1!#2#3{\XINT_expr_iterr:_D #1{#3}{#2}}%
1968 \def\XINT_expr_iterr:_D #1#2#3%
1969     {\expandafter\XINT_expr_iterr:_E\romannumeral`&&@#3#1~#2?{#3}}%
1970 \def\XINT_expr_iterr:_Goon #1!#2#3~#4?#5%
1971     {\expandafter\XINT_expr_iterr:_D
1972         \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\expandafter\endcsname
1973         \expandafter{\romannumeral0\xinttrim{-1}{\XINT_expr_lockit{#1}{#4}}{#5}}%
1974 \def\XINT_expr_iterr:_Omit #1!#2#3~#4?#5%

```

```

1975     {\expandafter\XINT_expr_iterr:_D
1976       \csname.=\the\numexpr \XINT_expr_unlock#3+\xint_c_i\endcsname}%
1977 \def\XINT_expr_iterr:_Abort #1!#2#3~#4?#5%
1978   {\expandafter\xint_gobble_i\romannumeral0\xintapplyunbraced
1979     {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}}}%
1980 \def\XINT_expr_iterr:_Break #1!#2#3~#4?#5%
1981   {\expandafter\xint_gobble_iv\romannumeral0\xintapplyunbraced
1982     {,\XINT_expr:_unlock}{\xintReverseOrder{#4\space}},#1}%
1983 \def\XINT_expr_iterr:_E #1{\if #1^\xint_dothis\XINT_expr_iterr:_Abort\fi
1984       \if #1?\xint_dothis\XINT_expr_iterr:_Break\fi
1985       \if #1!\xint_dothis\XINT_expr_iterr:_Omit\fi
1986       \xint_orthat{\XINT_expr_iterr:_Goon #1}}%
1987 \catcode`? 11

```

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

11.52.1	\xintANDof:csv	350
11.52.2	\xintORof:csv	351
11.52.3	\xintXORof:csv	351
11.52.4	Generic csv routine (\XINT_oncsv:_a)	351
11.52.5	\xintMaxof:csv, \xintiiMaxof:csv	351
11.52.6	\xintMinof:csv, \xintiiMinof:csv	352
11.52.7	\xintSum:csv, \xintiiSum:csv	352
11.52.8	\xintPrd:csv, \xintiiPrd:csv	352
11.52.9	\xintGCDof:csv, \xintLCMof:csv	352
11.52.10	\xintiiGCDof:csv, \xintiiLCMof:csv	354
11.52.11	\XINTinFloatdigits, \XINTinFloatSqrtdigits, \XINTinFloatFacdigits	354
11.52.12	\XINTinFloatMaxof:csv, \XINTinFloatMinof:csv	354
11.52.13	\XINTinFloatSum:csv, \XINTinFloatPrd:csv	354

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)

### 11.52.1 \xintANDof:csv

1.09a. For use by \xintexpr inside \csname. 1.1, je remplace ifTrueAelseB par iiNotZero pour des raisons d'optimisations.

```

1988 \def\xintANDof:csv #1{\expandafter\XINT_andof:_a\romannumeral`&&@#1,^}%
1989 \def\XINT_andof:_a #1{\if ,#1\expandafter\XINT_andof:_e
1990   \else\expandafter\XINT_andof:_c\fi #1}%
1991 \def\XINT_andof:_c #1,{\xintiiifNotZero {#1}{\XINT_andof:_a}{\XINT_andof:_no}}%
1992 \def\XINT_andof:_no #1^{0}%
1993 \def\XINT_andof:_e #1^{1}% works with empty list

```

[TOC](#), [xintkernel](#), [xinttools](#), [xintcore](#), [xint](#), [xintbinhex](#), [xintgcd](#), [xintfrac](#), [xintseries](#), [xintcfrc](#), [xintexpr](#), [indices](#)

### 11.52.2 \xintORof:csv

1.09a. For use by \xintexpr.

```
1994 \def\xintORof:csv #1{\expandafter\XINT_orof:_a\romannumeral`&&@#1,^}%
1995 \def\XINT_orof:_a #1{\if ,#1\expandafter\XINT_orof:_e
1996         \else\expandafter\XINT_orof:_c\fi #1}%
1997 \def\XINT_orof:_c #1,{\xintiifNotZero{#1}{\XINT_orof:_yes}{\XINT_orof:_a}}%
1998 \def\XINT_orof:_yes #1^{1}%
1999 \def\XINT_orof:_e #1^{0}% works with empty list
```

### 11.52.3 \xintXORof:csv

1.09a. For use by \xintexpr (inside a \csname..\endcsname).

```
2000 \def\xintXORof:csv #1{\expandafter\XINT_xorof:_a\expandafter 0\romannumeral`&&@#1,^}%
2001 \def\XINT_xorof:_a #1#2,{\XINT_xorof:_b #2,#1}%
2002 \def\XINT_xorof:_b #1{\if ,#1\expandafter\XINT_xorof:_e
2003         \else\expandafter\XINT_xorof:_c\fi #1}%
2004 \def\XINT_xorof:_c #1,#2%
2005         {\xintiifNotZero {#1}{\if #20\xint_afterfi{\XINT_xorof:_a 1}%
2006                 \else\xint_afterfi{\XINT_xorof:_a 0}\fi}%
2007         {\XINT_xorof:_a #2}%
2008         }%
2009 \def\XINT_xorof:_e ,#1#2^{#1}% allows empty list (then returns 0)
```

### 11.52.4 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.

```
2010 \def\XINT_oncsv:_empty #1,^,#2{#2}%
2011 \def\XINT_oncsv:_end ^,#1#2#3#4{#1}%
2012 \def\XINT_oncsv:_a #1#2#3%
2013     {\if ,#3\expandafter\XINT_oncsv:_empty\else\expandafter\XINT_oncsv:_b\fi #1#2#3}%
2014 \def\XINT_oncsv:_b #1#2#3,%
2015     {\expandafter\XINT_oncsv:_c \expandafter{\romannumeral`&&@#2{#3}}#1#2}%
2016 \def\XINT_oncsv:_c #1#2#3#4,{\expandafter\XINT_oncsv:_d \romannumeral`&&@#4,{#1}#2#3}%
2017 \def\XINT_oncsv:_d #1%
2018     {\if ^#1\expandafter\XINT_oncsv:_end\else\expandafter\XINT_oncsv:_e\fi #1}%
2019 \def\XINT_oncsv:_e #1,#2#3#4%
2020     {\expandafter\XINT_oncsv:_c\expandafter {\romannumeral`&&@#3{#4{#1}}{#2}}#3#4}%

```

### 11.52.5 \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 xintiexpr, dans il faut bien sûr ne pas faire de xintNum, donc il faut un iimax.

```
2021 \def\xintMaxof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmax
2022         \expandafter\xint_firstofone\romannumeral`&&@#1,^{0/1[0]}}%
2023 \def\xintiiMaxof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiimax
2024         \expandafter\xint_firstofone\romannumeral`&&@#1,^{,0}}%
```



*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrc, xintexpr, indices*

### 11.52.6 \xintMinof:csv, \xintiiMinof:csv

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

```
2025 \def\xintMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmin
2026 \expandafter\xint_firstofone\romannumeral`&&@#1,^{0/1[0]}}%
2027 \def\xintiiMinof:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiimin
2028 \expandafter\xint_firstofone\romannumeral`&&@#1,^{0}%
```

### 11.52.7 \xintSum:csv, \xintiiSum:csv

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

```
2029 \def\xintSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintadd
2030 \expandafter\xint_firstofone\romannumeral`&&@#1,^{0/1[0]}}%
2031 \def\xintiiSum:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiiaadd
2032 \expandafter\xint_firstofone\romannumeral`&&@#1,^{0}%
```

### 11.52.8 \xintPrd:csv, \xintiiPrd:csv

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

```
2033 \def\xintPrd:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintmul
2034 \expandafter\xint_firstofone\romannumeral`&&@#1,^{1/1[0]}}%
2035 \def\xintiiPrd:csv #1{\expandafter\XINT_oncsv:_a\expandafter\xintiimul
2036 \expandafter\xint_firstofone\romannumeral`&&@#1,^{1}%
```

### 11.52.9 \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.

```
2037 \def\xintGCDof:csv #1{\expandafter\XINT_gcdof:_a\romannumeral`&&@#1,^{1/1[0]}}%
2038 \def\XINT_gcdof:_a #1%
2039 {\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.

```
2040 \def\XINT_gcdof:_b #1,%
2041 {\expandafter\XINT_gcdof:_c\romannumeral0\xintiiabs#1\xint:}%
2042 \def\XINT_gcdof:_c #1\xint:#2,%
2043 {\expandafter\XINT_gcdof:_d\romannumeral0\xintiiabs#2\xint:#1\xint:}%
2044 \def\XINT_gcdof:_d #1%
2045 {\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.

```
2046 \def\XINT_gcdof:_e#1#2\xint:#3\xint:
2047 {%
2048 \if0#1\expandafter\XINT_gcdof:_f\fi
```



```

2049 \expandafter\XINT_gcdof:_e\romannumeral0\xintmod{#3}{#1#2}\xint:#1#2\xint:
2050 }%
2051 \def\XINT_gcdof:_f
2052 \expandafter\XINT_gcdof:_e\romannumeral0\xintmod#1#2\xint:#3\xint:#4,%
2053 {%
2054 \expandafter\XINT_gcdof:_d\romannumeral0\xintiiabs#4\xint:#1\xint:
2055 }%

```

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 `\csize...\endcsize`, and there is no triggering `\romannumeral`, attention to not leave a space upfront.

```

2056 \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.

```

2057 \def\xintLCMof:csv #1{\expandafter\XINT_lcmof:_a\romannumeral`&&@#1,^{0/1[0]}}%
2058 \def\XINT_lcmof:_a #1%
2059 {\if ,#1\expandafter\XINT_oncsv:_empty\else\expandafter\XINT_lcmof:_b\fi #1}%
2060 \def\XINT_lcmof:_b #1,%
2061 {\expandafter\XINT_lcmof:_c\romannumeral0\xintiiabs\xintRaw{#1}\xint:}%
2062 \def\XINT_lcmof:_c #1{\if0#1\expandafter\XINT_lcmof:_zero\fi
2063 \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.

```

2064 \def\XINT_lcmof:_d #1\xint:#2,%
2065 {\expandafter\XINT_lcmof:_e\romannumeral0\xintiiabs#2\xint:#1\xint:}%
2066 \def\XINT_lcmof:_e #1%
2067 {\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`.

```

2068 \def\XINT_lcmof:_f#1#2\xint:
2069 {%
2070 \if0#1\expandafter\XINT_lcmof:_zero\fi
2071 \expandafter\XINT_lcmof:_g\romannumeral0\expandafter\XINT_inv
2072 \romannumeral0\xintraw{#1#2}\xint:
2073 }%
2074 \def\XINT_lcmof:_g #1#2\xint:#3\xint:
2075 {%
2076 \if0#1\expandafter\XINT_lcmof:_h\fi
2077 \expandafter\XINT_lcmof:_g\romannumeral0\xintmod{#3}{#1#2}\xint:#1#2\xint:
2078 }%
2079 \def\XINT_lcmof:_h
2080 \expandafter\XINT_lcmof:_g\romannumeral0\xintmod#1#2\xint:#3\xint:#4,%
2081 {%
2082 \expandafter\XINT_lcmof:_e\romannumeral0\xintiiabs#4\xint:#1\xint:
2083 }%
2084 \def\XINT_lcmof:_zero #1^,#2{0/1[0]}}%

```

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfac, xintexpr, indices*

We need this `\romannumeral0` to remove the up-front space token which will be left by `\XINT_inv`, in case of `\csname..\endcsname` expansion.

```
2085 \def\XINT_lcmof:_end ^\xint:#1\xint:#2{\romannumeral0\XINT_inv #1}%
```

### 11.52.10 `\xintiigcdof:csv, \xintiilcmof:csv`

**1.1a.** For `\xintiexpr`. Requires the `xintgcd` provided macros.

```
2086 \def\xintiigcdof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintiigcd
2087           \expandafter\xint_firstofone\romannumeral`&&@#1,^,1}%
2088 \def\xintiilcmof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintiilcm
2089           \expandafter\xint_firstofone\romannumeral`&&@#1,^,0}%
```

### 11.52.11 `\XINTinFloatdigits, \XINTinFloatSqrtdigits, \XINTinFloatFacdigits`

for `\xintNewExpr` matters, mainly.

```
2090 \def\XINTinFloatdigits {\XINTinFloat [\XINTdigits]]}%
2091 \def\XINTinFloatSqrtdigits {\XINTinFloatSqrt[\XINTdigits]]}%
2092 \def\XINTinFloatFacdigits {\XINTinFloatFac [\XINTdigits]]}%
```

### 11.52.12 `\XINTinFloatMaxof:csv, \XINTinFloatMinof:csv`

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

```
2093 \def\XINTinFloatMaxof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintmax
2094           \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^{,0[0]}}}%
2095 \def\XINTinFloatMinof:csv #1{\expandafter\XINT_onscv:_a\expandafter\xintmin
2096           \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^{,0[0]}}}%
```

### 11.52.13 `\XINTinFloatSum:csv, \XINTinFloatPrd:csv`

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

```
2097 \def\XINTinFloatSum:csv #1{\expandafter\XINT_onscv:_a\expandafter\XINTinfloatadd
2098           \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^{,{0[0]}}}%
2099 \def\XINTinFloatPrd:csv #1{\expandafter\XINT_onscv:_a\expandafter\XINTinfloatmul
2100           \expandafter\XINTinFloatdigits\romannumeral`&&@#1,^{,{1[0]}}}%
```

## 11.53 Auxiliary wrappers for function macros

```
2101 \def\XINT:expr:one:and:opt #1,#2,#3!#4#5%
2102 {%
2103   \ifrelax#3\relax\expandafter\xint_firstoftwo\else
2104     \expandafter\xint_secondoftwo\fi
2105   {#4}{#5[\xintNum{#2}]]}{#1}%
2106 }%
2107 \def\XINT:expr:tacitzeroifonearg #1,#2,#3!#4#5%
2108 {%
2109   \ifrelax#3\relax\expandafter\xint_firstoftwo\else
2110     \expandafter\xint_secondoftwo\fi
2111   {#4{0}}{#5{\xintNum{#2}]]}{#1}%
2112 }%
```

```

2113 \def\XINT:iiexpr:tacitzeroifonearg #1,#2,#3!#4%
2114 {%
2115     \if\relax#3\relax\expandafter\xint_firstoftwo\else
2116         \expandafter\xint_secondoftwo\fi
2117     {#4{0}}{#4{#2}}{#1}%
2118 }%
2119 \def\XINT:expr:totwo #1#2{#1,#2}%
2120 \def\XINT:expr:two:to:two #1,#2,!#3%
2121 {%
2122     \expandafter\XINT:expr:totwo\romannumeral`&&@%
2123     #3{#1}{#2}%
2124 }%
2125 \def\XINT:expr:two:to:one #1,#2,!#3%
2126 {%
2127     #3{#1}{#2}%
2128 }%
2129 \def\XINT:flexpr:two:to:one #1,#2,!#3%
2130 {%
2131     #3{#1}{#2}%
2132 }%
2133 \let\XINT:flexpr:two:to:two\XINT:flexpr:two:to:one

```

#### 11.54 The num(), reduce(), preduce(), abs(), sgn(), frac(), floor(), ceil(), sqr(), sqrt(), sqrtr(), float(), round(), trunc(), mod(), quo(), rem(), divmod(), gcd(), lcm(), max(), min(), `+`(), `\*`(), ?(), !(), not(), all(), any(), xor(), if(), ifsgn(), ifint(), ifone(), even(), odd(), isint(), isone(), first(), last(), len(), reversed(), factorial(), binomial() and randrange() functions

```

2134 \def\XINT_expr_func_num #1#2#3%
2135 {%
2136     \expandafter #1\expandafter #2\csname.=%
2137     \XINT:NEhook:one\xintNum{\XINT_expr_unlock #3}\endcsname
2138 }%
2139 \let\XINT_flexpr_func_num\XINT_expr_func_num
2140 \let\XINT_iiexpr_func_num\XINT_expr_func_num
2141 \def\XINT_expr_func_reduce #1#2#3%
2142 {%
2143     \expandafter #1\expandafter #2\csname.=%
2144     \XINT:NEhook:one\xintIrr{\XINT_expr_unlock #3}[0]\endcsname
2145 }%
2146 \let\XINT_flexpr_func_reduce\XINT_expr_func_reduce
2147 \def\XINT_expr_func_preduce #1#2#3%
2148 {%
2149     \expandafter #1\expandafter #2\csname.=%
2150     \XINT:NEhook:one\xintPIrr{\XINT_expr_unlock #3}\endcsname
2151 }%
2152 \let\XINT_flexpr_func_preduce\XINT_expr_func_preduce
2153 \def\XINT_expr_func_abs #1#2#3%
2154 {%
2155     \expandafter #1\expandafter #2\csname.=%
2156     \XINT:NEhook:one\xintAbs{\XINT_expr_unlock #3}\endcsname
2157 }%

```

```

2158 \let\XINT_flexpr_func_abs\XINT_expr_func_abs
2159 \def\XINT_iiexpr_func_abs #1#2#3%
2160 {%
2161   \expandafter #1\expandafter #2\csname.=%
2162   \XINT:NEhook:one\xintiiAbs{\XINT_expr_unlock #3}\endcsname
2163 }%
2164 \def\XINT_expr_func_sgn #1#2#3%
2165 {%
2166   \expandafter #1\expandafter #2\csname.=%
2167   \XINT:NEhook:one\xintSgn{\XINT_expr_unlock #3}\endcsname
2168 }%
2169 \let\XINT_flexpr_func_sgn\XINT_expr_func_sgn
2170 \def\XINT_iiexpr_func_sgn #1#2#3%
2171 {%
2172   \expandafter #1\expandafter #2\csname.=%
2173   \XINT:NEhook:one\xintiiSgn{\XINT_expr_unlock #3}\endcsname
2174 }%
2175 \def\XINT_expr_func_frac #1#2#3%
2176 {%
2177   \expandafter #1\expandafter #2\csname.=%
2178   \XINT:NEhook:one\xintTFrac{\XINT_expr_unlock #3}\endcsname
2179 }%
2180 \def\XINT_flexpr_func_frac #1#2#3%
2181 {%
2182   \expandafter #1\expandafter #2\csname.=%
2183   \XINT:NEhook:one\XINTinFloatFracdigits{\XINT_expr_unlock #3}\endcsname
2184 }%

```

*no \XINT\_iiexpr\_func\_frac*

```

2185 \def\XINT_expr_func_floor #1#2#3%
2186 {%
2187   \expandafter #1\expandafter #2\csname.=%
2188   \XINT:NEhook:one\xintFloor{\XINT_expr_unlock #3}\endcsname
2189 }%
2190 \let\XINT_flexpr_func_floor\XINT_expr_func_floor

```

*The floor and ceil functions in \xintiiexpr require protect(a/b) or, better, \qfrac(a/b); else the / will be executed first and do an integer rounded division.*

```

2191 \def\XINT_iiexpr_func_floor #1#2#3%
2192 {%
2193   \expandafter #1\expandafter #2\csname.=%
2194   \XINT:NEhook:one\xintiFloor{\XINT_expr_unlock #3}\endcsname
2195 }%
2196 \def\XINT_expr_func_ceil #1#2#3%
2197 {%
2198   \expandafter #1\expandafter #2\csname.=%
2199   \XINT:NEhook:one\xintCeil{\XINT_expr_unlock #3}\endcsname
2200 }%
2201 \let\XINT_flexpr_func_ceil\XINT_expr_func_ceil
2202 \def\XINT_iiexpr_func_ceil #1#2#3%
2203 {%
2204   \expandafter #1\expandafter #2\csname.=%
2205   \XINT:NEhook:one\xintiCeil{\XINT_expr_unlock #3}\endcsname

```

```

2206 }%
2207 \def\XINT_expr_func_sqr #1#2#3%
2208 {%
2209     \expandafter #1\expandafter #2\csname.=%
2210     \XINT:NEhook:one\xintSqr{\XINT_expr_unlock #3}\endcsname
2211 }%
2212 \def\XINTinFloatSqr#1{\XINTinFloatMul{#1}{#1}}% revoir après
2213 \def\XINT_flexpr_func_sqr #1#2#3%
2214 {%
2215     \expandafter #1\expandafter #2\csname.=%
2216     \XINT:NEhook:one\XINTinFloatSqr{\XINT_expr_unlock #3}\endcsname
2217 }%
2218 \def\XINT_iiexpr_func_sqr #1#2#3%
2219 {%
2220     \expandafter #1\expandafter #2\csname.=%
2221     \XINT:NEhook:one\xintiiSqr{\XINT_expr_unlock #3}\endcsname
2222 }%
2223 \def\XINT_expr_func_? #1#2#3%
2224 {%
2225     \expandafter #1\expandafter #2\csname.=%
2226     \XINT:NEhook:one\xintiiIsNotZero{\XINT_expr_unlock #3}\endcsname
2227 }%
2228 \let\XINT_flexpr_func_? \XINT_expr_func_?
2229 \let\XINT_iiexpr_func_? \XINT_expr_func_?
2230 \def\XINT_expr_func_! #1#2#3%
2231 {%
2232     \expandafter #1\expandafter #2\csname.=%
2233     \XINT:NEhook:one\xintiiIsZero{\XINT_expr_unlock #3}\endcsname
2234 }%
2235 \let\XINT_flexpr_func_! \XINT_expr_func_!
2236 \let\XINT_iiexpr_func_! \XINT_expr_func_!
2237 \def\XINT_expr_func_not #1#2#3%
2238 {%
2239     \expandafter #1\expandafter #2\csname.=%
2240     \XINT:NEhook:one\xintiiIsZero{\XINT_expr_unlock #3}\endcsname
2241 }%
2242 \let\XINT_flexpr_func_not \XINT_expr_func_not
2243 \let\XINT_iiexpr_func_not \XINT_expr_func_not
2244 \def\XINT_expr_func_odd #1#2#3%
2245 {%
2246     \expandafter #1\expandafter #2\csname.=%
2247     \XINT:NEhook:one\xintOdd{\XINT_expr_unlock #3}\endcsname
2248 }%
2249 \let\XINT_flexpr_func_odd\XINT_expr_func_odd
2250 \def\XINT_iiexpr_func_odd #1#2#3%
2251 {%
2252     \expandafter #1\expandafter #2\csname.=%
2253     \XINT:NEhook:one\xintiiOdd{\XINT_expr_unlock #3}\endcsname
2254 }%
2255 \def\XINT_expr_func_even #1#2#3%
2256 {%
2257     \expandafter #1\expandafter #2\csname.=%

```

```

2258 \XINT:NEhook:one\xintEven{\XINT_expr_unlock #3}\endcsname
2259 }%
2260 \let\XINT_flexpr_func_even\XINT_expr_func_even
2261 \def\XINT_iiexpr_func_even #1#2#3%
2262 {%
2263 \expandafter #1\expandafter #2\csname.=%
2264 \XINT:NEhook:one\xintiiEven{\XINT_expr_unlock #3}\endcsname
2265 }%
2266 \def\XINT_expr_func_isint #1#2#3%
2267 {%
2268 \expandafter #1\expandafter #2\csname.=%
2269 \XINT:NEhook:one\xintIsInt{\XINT_expr_unlock #3}\endcsname
2270 }%
2271 \def\XINT_flexpr_func_isint #1#2#3%
2272 {%
2273 \expandafter #1\expandafter #2\csname.=%
2274 \XINT:NEhook:one\xintFloatIsInt{\XINT_expr_unlock #3}\endcsname
2275 }%
2276 \let\XINT_iiexpr_func_isint\XINT_expr_func_isint % ? perhaps rather always 1
2277 \def\XINT_expr_func_isone #1#2#3%
2278 {%
2279 \expandafter #1\expandafter #2\csname.=%
2280 \XINT:NEhook:one\xintIsOne{\XINT_expr_unlock #3}\endcsname
2281 }%
2282 \let\XINT_flexpr_func_isone\XINT_expr_func_isone
2283 \def\XINT_iiexpr_func_isone #1#2#3%
2284 {%
2285 \expandafter #1\expandafter #2\csname.=%
2286 \XINT:NEhook:one\xintiiIsOne{\XINT_expr_unlock #3}\endcsname
2287 }%
2288 % REVOIR nuple
2289 \def\XINT_expr_func_nuple #1#2#3%
2290 {\expandafter #1\expandafter #2\csname.=\XINT_expr_unlock #3\endcsname }%
2291 \let\XINT_flexpr_func_nuple\XINT_expr_func_nuple
2292 \let\XINT_iiexpr_func_nuple\XINT_expr_func_nuple
2293 \def\XINT_expr_func_factorial #1#2#3%
2294 {%
2295 \expandafter #1\expandafter #2\csname.=%
2296 \expandafter\XINT:expr:one:and:opt
2297 \romannumeral`&&\XINT_expr_unlock#3,,!\xintFac\XINTinFloatFac
2298 \endcsname
2299 }%
2300 \def\XINT_flexpr_func_factorial #1#2#3%
2301 {%
2302 \expandafter #1\expandafter #2\csname.=%
2303 \expandafter\XINT:expr:one:and:opt
2304 \romannumeral`&&\XINT_expr_unlock#3,,!\XINTinFloatFacdigits\XINTinFloatFac
2305 \endcsname
2306 }%
2307 \def\XINT_iiexpr_func_factorial #1#2#3%
2308 {%
2309 \expandafter #1\expandafter #2\csname.=%

```

```

2310 \XINT:NEhook:one\xintiiFac{\XINT_expr_unlock #3}\endcsname
2311 }%
2312 \def\XINT_expr_func_sqrt #1#2#3%
2313 {%
2314 \expandafter #1\expandafter #2\csname.=%
2315 \expandafter\XINT:expr:one:and:opt
2316 \romannumeral`&&\XINT_expr_unlock#3,,!\XINTinFloatSqrtdigits\XINTinFloatSqrt
2317 \endcsname
2318 }%
2319 \let\XINT_flexpr_func_sqrt\XINT_expr_func_sqrt
2320 \def\XINT_iiexpr_func_sqrt #1#2#3%
2321 {%
2322 \expandafter #1\expandafter #2\csname.=%
2323 \XINT:NEhook:one\xintiiSqrt{\XINT_expr_unlock #3}\endcsname
2324 }%
2325 \def\XINT_iiexpr_func_sqrtr #1#2#3%
2326 {%
2327 \expandafter #1\expandafter #2\csname.=%
2328 \XINT:NEhook:one\xintiiSqrtr{\XINT_expr_unlock #3}\endcsname
2329 }%
2330 \def\XINT_expr_func_round #1#2#3%
2331 {%
2332 \expandafter #1\expandafter #2\csname.=%
2333 \expandafter\XINT:expr:tacitzeroifonearg
2334 \romannumeral`&&\XINT_expr_unlock #3,,!\xintiRound\xintRound
2335 \endcsname
2336 }%
2337 \let\XINT_flexpr_func_round\XINT_expr_func_round
2338 \def\XINT_iiexpr_func_round #1#2#3%
2339 {%
2340 \expandafter #1\expandafter #2\csname.=%
2341 \expandafter\XINT:iiexpr:tacitzeroifonearg
2342 \romannumeral`&&\XINT_expr_unlock #3,,!\xintiRound
2343 \endcsname
2344 }%
2345 \def\XINT_expr_func_trunc #1#2#3%
2346 {%
2347 \expandafter #1\expandafter #2\csname.=%
2348 \expandafter\XINT:expr:tacitzeroifonearg
2349 \romannumeral`&&\XINT_expr_unlock #3,,!\xintiTrunc\xintTrunc
2350 \endcsname
2351 }%
2352 \let\XINT_flexpr_func_trunc\XINT_expr_func_trunc
2353 \def\XINT_iiexpr_func_trunc #1#2#3%
2354 {%
2355 \expandafter #1\expandafter #2\csname.=%
2356 \expandafter\XINT:iiexpr:tacitzeroifonearg
2357 \romannumeral`&&\XINT_expr_unlock #3,,!\xintiTrunc
2358 \endcsname
2359 }%
2360 \def\XINT_expr_func_float #1#2#3%
2361 {%

```

```

2362 \expandafter #1\expandafter #2\csname.=%
2363 \expandafter\XINT:expr:one:and:opt
2364 \romannumeral`&&\XINT_expr_unlock #3,,!\XINTinFloatdigits\XINTinFloat
2365 \endcsname
2366 }%
2367 \let\XINT_flexpr_func_float\XINT_expr_func_float
2368 % \XINT_iiexpr_func_float not defined
2369 \def\XINT_expr_func_divmod #1#2#3%
2370 {%
2371 \expandafter #1\expandafter #2\csname.=%
2372 \expandafter\XINT:expr:two:to:two
2373 \romannumeral`&&\XINT_expr_unlock #3,!\xintDivMod
2374 \endcsname
2375 }%
2376 \def\XINT_flexpr_func_divmod #1#2#3%
2377 {%
2378 \expandafter #1\expandafter #2\csname.=%
2379 \expandafter\XINT:flexpr:two:to:two
2380 \romannumeral`&&\XINT_expr_unlock #3,!\XINTinFloatDivMod
2381 \endcsname
2382 }%
2383 \def\XINT_iiexpr_func_divmod #1#2#3%
2384 {%
2385 \expandafter #1\expandafter #2\csname.=%
2386 \expandafter\XINT:expr:two:to:two
2387 \romannumeral`&&\XINT_expr_unlock #3,!\xintiiDivMod
2388 \endcsname
2389 }%
2390 \def\XINT_expr_func_mod #1#2#3%
2391 {%
2392 \expandafter #1\expandafter #2\csname.=%
2393 \expandafter\XINT:expr:two:to:one
2394 \romannumeral`&&\XINT_expr_unlock #3,!\xintMod
2395 \endcsname
2396 }%
2397 \def\XINT_flexpr_func_mod #1#2#3%
2398 {%
2399 \expandafter #1\expandafter #2\csname.=%
2400 \expandafter\XINT:flexpr:two:to:one
2401 \romannumeral`&&\XINT_expr_unlock #3,!\XINTinFloatMod
2402 \endcsname
2403 }%
2404 \def\XINT_iiexpr_func_mod #1#2#3%
2405 {%
2406 \expandafter #1\expandafter #2\csname.=%
2407 \expandafter\XINT:expr:two:to:one
2408 \romannumeral`&&\XINT_expr_unlock #3,!\xintiiMod
2409 \endcsname
2410 }%
2411 \def\XINT_expr_func_binomial #1#2#3%
2412 {%
2413 \expandafter #1\expandafter #2\csname.=%

```



```

2414 \expandafter\XINT:expr:two:to:one
2415 \romannumeral`&&\XINT_expr_unlock #3,! \xintBinomial
2416 \endcsname
2417 }%
2418 \def\XINT_flexpr_func_binomial #1#2#3%
2419 {%
2420 \expandafter #1\expandafter #2\csname.=%
2421 \expandafter\XINT:flexpr:two:to:one
2422 \romannumeral`&&\XINT_expr_unlock #3,! \XINTinFloatBinomial
2423 \endcsname
2424 }%
2425 \def\XINT_iiexpr_func_binomial #1#2#3%
2426 {%
2427 \expandafter #1\expandafter #2\csname.=%
2428 \expandafter\XINT:expr:two:to:one
2429 \romannumeral`&&\XINT_expr_unlock #3,! \xintiiBinomial
2430 \endcsname
2431 }%
2432 \def\XINT_expr_func_pfactorial #1#2#3%
2433 {%
2434 \expandafter #1\expandafter #2\csname.=%
2435 \expandafter\XINT:expr:two:to:one
2436 \romannumeral`&&\XINT_expr_unlock #3,! \xintPFactorial
2437 \endcsname
2438 }%
2439 \def\XINT_flexpr_func_pfactorial #1#2#3%
2440 {%
2441 \expandafter #1\expandafter #2\csname.=%
2442 \expandafter\XINT:flexpr:two:to:one
2443 \romannumeral`&&\XINT_expr_unlock #3,! \XINTinFloatPFactorial
2444 \endcsname
2445 }%
2446 \def\XINT_iiexpr_func_pfactorial #1#2#3%
2447 {%
2448 \expandafter #1\expandafter #2\csname.=%
2449 \expandafter\XINT:expr:two:to:one
2450 \romannumeral`&&\XINT_expr_unlock #3,! \xintiiPFactorial
2451 \endcsname
2452 }%
2453 \def\XINT_expr_func_randrange #1#2#3%
2454 {%
2455 \expandafter #1\expandafter #2\csname.=%
2456 \expandafter\XINT:expr:randrange
2457 \romannumeral`&&\XINT_expr_unlock #3,,!%
2458 \endcsname
2459 }%
2460 \let\XINT_flexpr_func_randrange\XINT_expr_func_randrange
2461 \def\XINT_iiexpr_func_randrange #1#2#3%
2462 {%
2463 \expandafter #1\expandafter #2\csname.=%
2464 \expandafter\XINT:iiexpr:randrange
2465 \romannumeral`&&\XINT_expr_unlock #3,,!%

```

```

2466 \endcsname
2467 }%
2468 \def\XINT:expr:randrange #1,#2,#3!%
2469 {%
2470 \if\relax#3\relax\expandafter\xint_firstoftwo\else
2471 \expandafter\xint_secondoftwo\fi
2472 {\xintiiRandRange{\XINT:NEhook:one\xintNum{#1}}}%
2473 {\xintiiRandRangeAtoB{\XINT:NEhook:one\xintNum{#1}}}%
2474 {\XINT:NEhook:one\xintNum{#2}}}%
2475 }%
2476 \def\XINT:iiexpr:randrange #1,#2,#3!%
2477 {%
2478 \if\relax#3\relax\expandafter\xint_firstoftwo\else
2479 \expandafter\xint_secondoftwo\fi
2480 {\xintiiRandRange{#1}}{\xintiiRandRangeAtoB{#1}{#2}}}%
2481 }%
2482 \def\XINT_expr_func_quo #1#2#3%
2483 {%
2484 \expandafter #1\expandafter #2\csname.=%
2485 \expandafter\XINT:expr:two:to:one
2486 \romannumeral`&&\XINT_expr_unlock #3,! \xintiQuo
2487 \endcsname
2488 }%
2489 \let\XINT_flexpr_func_quo\XINT_expr_func_quo
2490 \def\XINT_iiexpr_func_quo #1#2#3%
2491 {%
2492 \expandafter #1\expandafter #2\csname.=%
2493 \expandafter\XINT:expr:two:to:one
2494 \romannumeral`&&\XINT_expr_unlock #3,! \xintiiQuo
2495 \endcsname
2496 }%
2497 \def\XINT_expr_func_rem #1#2#3%
2498 {%
2499 \expandafter #1\expandafter #2\csname.=%
2500 \expandafter\XINT:expr:two:to:one
2501 \romannumeral`&&\XINT_expr_unlock #3,! \xintiRem
2502 \endcsname
2503 }%
2504 \let\XINT_flexpr_func_rem\XINT_expr_func_rem
2505 \def\XINT_iiexpr_func_rem #1#2#3%
2506 {%
2507 \expandafter #1\expandafter #2\csname.=%
2508 \expandafter\XINT:expr:two:to:one
2509 \romannumeral`&&\XINT_expr_unlock #3,! \xintiiRem
2510 \endcsname
2511 }%
2512 \def\XINT_expr_func_gcd #1#2#3%
2513 {%
2514 \expandafter #1\expandafter #2\csname.=%
2515 \XINT:NEhook:csv\xintGCDof:csv{\XINT_expr_unlock #3}\endcsname
2516 }%
2517 \let\XINT_flexpr_func_gcd\XINT_expr_func_gcd

```

```

2518 \def\XINT_iiexpr_func_gcd #1#2#3%
2519 {%
2520     \expandafter #1\expandafter #2\csname.=%
2521     \XINT:NEhook:csv\xintiiGCDof:csv{\XINT_expr_unlock #3}\endcsname
2522 }%
2523 \def\XINT_expr_func_lcm #1#2#3%
2524 {%
2525     \expandafter #1\expandafter #2\csname.=%
2526     \XINT:NEhook:csv\xintLCMof:csv{\XINT_expr_unlock #3}\endcsname
2527 }%
2528 \let\XINT_flexpr_func_lcm\XINT_expr_func_lcm
2529 \def\XINT_iiexpr_func_lcm #1#2#3%
2530 {%
2531     \expandafter #1\expandafter #2\csname.=%
2532     \XINT:NEhook:csv\xintiiLCMof:csv{\XINT_expr_unlock #3}\endcsname
2533 }%
2534 \def\XINT_expr_func_max #1#2#3%
2535 {%
2536     \expandafter #1\expandafter #2\csname.=%
2537     \XINT:NEhook:csv\xintMaxof:csv{\XINT_expr_unlock #3}\endcsname
2538 }%
2539 \def\XINT_iiexpr_func_max #1#2#3%
2540 {%
2541     \expandafter #1\expandafter #2\csname.=%
2542     \XINT:NEhook:csv\xintiiMaxof:csv{\XINT_expr_unlock #3}\endcsname
2543 }%
2544 \def\XINT_flexpr_func_max #1#2#3%
2545 {%
2546     \expandafter #1\expandafter #2\csname.=%
2547     \XINT:NEhook:csv\XINTinFloatMaxof:csv{\XINT_expr_unlock #3}\endcsname
2548 }%
2549 \def\XINT_expr_func_min #1#2#3%
2550 {%
2551     \expandafter #1\expandafter #2\csname.=%
2552     \XINT:NEhook:csv\xintMinof:csv{\XINT_expr_unlock #3}\endcsname
2553 }%
2554 \def\XINT_iiexpr_func_min #1#2#3%
2555 {%
2556     \expandafter #1\expandafter #2\csname.=%
2557     \XINT:NEhook:csv\xintiiMinof:csv{\XINT_expr_unlock #3}\endcsname
2558 }%
2559 \def\XINT_flexpr_func_min #1#2#3%
2560 {%
2561     \expandafter #1\expandafter #2\csname.=%
2562     \XINT:NEhook:csv\XINTinFloatMinof:csv{\XINT_expr_unlock #3}\endcsname
2563 }%
2564 \expandafter
2565 \def\csname XINT_expr_func_+\endcsname #1#2#3%
2566 {%
2567     \expandafter #1\expandafter #2\csname.=%
2568     \XINT:NEhook:csv\xintSum:csv{\XINT_expr_unlock #3}\endcsname
2569 }%

```

```

2570 \expandafter
2571 \def\csname XINT_flexpr_func_+\endcsname #1#2#3%
2572 {%
2573     \expandafter #1\expandafter #2\csname.=%
2574     \XINT:NEhook:csv\XINTinFloatSum:csv{\XINT_expr_unlock #3}\endcsname
2575 }%
2576 \expandafter
2577 \def\csname XINT_iiexpr_func_+\endcsname #1#2#3%
2578 {%
2579     \expandafter #1\expandafter #2\csname.=%
2580     \XINT:NEhook:csv\xintiSum:csv{\XINT_expr_unlock #3}\endcsname
2581 }%
2582 \expandafter
2583 \def\csname XINT_expr_func_*\endcsname #1#2#3%
2584 {%
2585     \expandafter #1\expandafter #2\csname.=%
2586     \XINT:NEhook:csv\xintPrd:csv{\XINT_expr_unlock #3}\endcsname
2587 }%
2588 \expandafter
2589 \def\csname XINT_flexpr_func_*\endcsname #1#2#3%
2590 {%
2591     \expandafter #1\expandafter #2\csname.=%
2592     \XINT:NEhook:csv\XINTinFloatPrd:csv{\XINT_expr_unlock #3}\endcsname
2593 }%
2594 \expandafter
2595 \def\csname XINT_iiexpr_func_*\endcsname #1#2#3%
2596 {%
2597     \expandafter #1\expandafter #2\csname.=%
2598     \XINT:NEhook:csv\xintiiPrd:csv{\XINT_expr_unlock #3}\endcsname
2599 }%
2600 \def\XINT_expr_func_all #1#2#3%
2601 {%
2602     \expandafter #1\expandafter #2\csname.=%
2603     \XINT:NEhook:csv\xintANDof:csv{\XINT_expr_unlock #3}\endcsname
2604 }%
2605 \let\XINT_flexpr_func_all\XINT_expr_func_all
2606 \let\XINT_iiexpr_func_all\XINT_expr_func_all
2607 \def\XINT_expr_func_any #1#2#3%
2608 {%
2609     \expandafter #1\expandafter #2\csname.=%
2610     \XINT:NEhook:csv\xintORof:csv{\XINT_expr_unlock #3}\endcsname
2611 }%
2612 \let\XINT_flexpr_func_any\XINT_expr_func_any
2613 \let\XINT_iiexpr_func_any\XINT_expr_func_any
2614 \def\XINT_expr_func_xor #1#2#3%
2615 {%
2616     \expandafter #1\expandafter #2\csname.=%
2617     \XINT:NEhook:csv\xintXORof:csv{\XINT_expr_unlock #3}\endcsname
2618 }%
2619 \let\XINT_flexpr_func_xor\XINT_expr_func_xor
2620 \let\XINT_iiexpr_func_xor\XINT_expr_func_xor
2621 \def\XINT_expr_func_len #1#2#3%

```

```

2622 {%
2623     \expandafter#1\expandafter#2\csname.=%
2624     \XINT:NEhook:csv\xintLength:f:csv{\XINT_expr_unlock#3}\endcsname
2625 }%
2626 \let\XINT_flexpr_func_len \XINT_expr_func_len
2627 \let\XINT_iiexpr_func_len \XINT_expr_func_len

```

1.2k has \xintFirstItem:f:csv for improved \xintNewExpr compatibility.

```

2628 \def\XINT_expr_func_first #1#2#3%
2629 {%
2630     \expandafter #1\expandafter #2\csname.=%
2631     \XINT:NEhook:csv\xintFirstItem:f:csv{\XINT_expr_unlock #3}\endcsname
2632 }%
2633 \let\XINT_flexpr_func_first\XINT_expr_func_first
2634 \let\XINT_iiexpr_func_first\XINT_expr_func_first

```

1.2k has \xintLastItem:f:csv for efficiency and improved \xintNewExpr compatibility.

```

2635 \def\XINT_expr_func_last #1#2#3%
2636 {%
2637     \expandafter #1\expandafter #2\csname.=%
2638     \XINT:NEhook:csv\xintLastItem:f:csv{\XINT_expr_unlock #3}\endcsname
2639 }%
2640 \let\XINT_flexpr_func_last\XINT_expr_func_last
2641 \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.

```

2642 \def\XINT_expr_func_reversed #1#2#3%
2643 {%
2644     \expandafter #1\expandafter #2\csname.=%
2645     \XINT:NEhook:csv\xintReverse:f:csv{\XINT_expr_unlock #3}\endcsname
2646 }%
2647 \let\XINT_flexpr_func_reversed\XINT_expr_func_reversed
2648 \let\XINT_iiexpr_func_reversed\XINT_expr_func_reversed
2649 \def\xintiiifNotZero: #1,#2,#3,{\xintiiifNotZero{#1}{#2}{#3}}%
2650 \def\XINT_expr_func_if #1#2#3%
2651 {%
2652     \expandafter #1\expandafter #2\csname.=%
2653     \expandafter\xintiiifNotZero:%
2654     \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2655 }%
2656 \let\XINT_flexpr_func_if\XINT_expr_func_if
2657 \let\XINT_iiexpr_func_if\XINT_expr_func_if
2658 \def\xintifInt: #1,#2,#3,{\xintifInt{#1}{#2}{#3}}%
2659 \def\XINT_expr_func_ifint #1#2#3%
2660 {%
2661     \expandafter #1\expandafter #2\csname.=%
2662     \expandafter\xintifInt:%
2663     \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2664 }%
2665 \let\XINT_iiexpr_func_ifint\XINT_expr_func_ifint

```

```

2666 \def\xintifFloatInt: #1,#2,#3,{\xintifFloatInt{#1}{#2}{#3}}%
2667 \def\XINT_flexpr_func_ifint #1#2#3%
2668 {%
2669   \expandafter #1\expandafter #2\csname.=%
2670   \expandafter\xintifFloatInt:%
2671   \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2672 }%
2673 \def\xintifOne: #1,#2,#3,{\xintifOne{#1}{#2}{#3}}%
2674 \def\XINT_expr_func_ifone #1#2#3%
2675 {%
2676   \expandafter #1\expandafter #2\csname.=%
2677   \expandafter\xintifOne:%
2678   \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2679 }%
2680 \let\XINT_flexpr_func_ifone\XINT_expr_func_ifone
2681 \def\xintiifOne: #1,#2,#3,{\xintiifOne{#1}{#2}{#3}}%
2682 \def\XINT_iiexpr_func_ifone #1#2#3%
2683 {%
2684   \expandafter #1\expandafter #2\csname.=%
2685   \expandafter\xintiifOne:%
2686   \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2687 }%
2688 \def\xintiifSgn: #1,#2,#3,#4,{\xintiifSgn{#1}{#2}{#3}{#4}}%
2689 \def\XINT_expr_func_ifsgn #1#2#3%
2690 {%
2691   \expandafter #1\expandafter #2\csname.=%
2692   \expandafter\xintiifSgn:%
2693   \romannumeral`&&\XINT_expr_unlock #3,\endcsname
2694 }%
2695 \let\XINT_flexpr_func_ifsgn\XINT_expr_func_ifsgn
2696 \let\XINT_iiexpr_func_ifsgn\XINT_expr_func_ifsgn

```

## 11.55 f-expandable versions of the `\xintSeqB::csv` and alike routines, for `\xintNewExpr`

11.55.1	<code>\xintSeqB:f:csv</code> . . . . .	366
11.55.2	<code>\xintiiSeqB:f:csv</code> . . . . .	367
11.55.3	<code>\XINTinFloatSeqB:f:csv</code> . . . . .	368

### 11.55.1 `\xintSeqB:f:csv`

Produces in f-expandable way. If the step is zero, gives empty result except if start and end coincide.

```

2697 \def\xintSeqB:f:csv #1#2%
2698   {\expandafter\XINT_seqb:f:csv \expandafter{\romannumeral0\xintra{#2}}{#1}}%
2699 \def\XINT_seqb:f:csv #1#2{\expandafter\XINT_seqb:f:csv_a\romannumeral`&&@#2#1!}%
2700 \def\XINT_seqb:f:csv_a #1#2;#3;#4!{%
2701   \expandafter\xint_gobble_i\romannumeral`&&@%
2702   \xintifCmp {#3}{#4}\XINT_seqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_seqb:f:csv_bg
2703   #1{#3}{#4}{#2}}%
2704 \def\XINT_seqb:f:csv_be #1#2#3#4#5{,#2}%
2705 \def\XINT_seqb:f:csv_bl #1{\if #1p\expandafter\XINT_seqb:f:csv_pa\else
2706   \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%

```

```

2707 \def\XINT_seqb:f:csv_pa #1#2#3#4{\expandafter\XINT_seqb:f:csv_p\expandafter
2708         {\romannumeral0\xintadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2709 \def\XINT_seqb:f:csv_p #1#2%
2710 {%
2711     \xintifCmp {#1}{#2}\XINT_seqb:f:csv_pa\XINT_seqb:f:csv_pb\XINT_seqb:f:csv_pc
2712     {#1}{#2}%
2713 }%
2714 \def\XINT_seqb:f:csv_pb #1#2#3#4{#3,#1}%
2715 \def\XINT_seqb:f:csv_pc #1#2#3#4{#3}%
2716 \def\XINT_seqb:f:csv_bg #1{\if #1n\expandafter\XINT_seqb:f:csv_na\else
2717         \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2718 \def\XINT_seqb:f:csv_na #1#2#3#4{\expandafter\XINT_seqb:f:csv_n\expandafter
2719         {\romannumeral0\xintadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2720 \def\XINT_seqb:f:csv_n #1#2%
2721 {%
2722     \xintifCmp {#1}{#2}\XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_seqb:f:csv_na
2723     {#1}{#2}%
2724 }%
2725 \def\XINT_seqb:f:csv_nb #1#2#3#4{#3,#1}%
2726 \def\XINT_seqb:f:csv_nc #1#2#3#4{#3}%

```

### 11.55.2 \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).

```

2727 \def\xintiiSeqB:f:csv #1#2%
2728     {\expandafter\XINT_iiseqb:f:csv \expandafter{\romannumeral`&&@#2}{#1}}%
2729 \def\XINT_iiseqb:f:csv #1#2{\expandafter\XINT_iiseqb:f:csv_a\romannumeral`&&@#2#1!}%
2730 \def\XINT_iiseqb:f:csv_a #1#2;#3;#4!{%
2731     \expandafter\xint_gobble_i\romannumeral`&&@%
2732     \xintSgnFork{\XINT_Cmp {#3}{#4}}%
2733         \XINT_iiseqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_iiseqb:f:csv_bg
2734     #1{#3}{#4}}{#2}}%
2735 \def\XINT_iiseqb:f:csv_bl #1{\if #1p\expandafter\XINT_iiseqb:f:csv_pa\else
2736         \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2737 \def\XINT_iiseqb:f:csv_pa #1#2#3#4{\expandafter\XINT_iiseqb:f:csv_p\expandafter
2738         {\romannumeral0\xintiiadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2739 \def\XINT_iiseqb:f:csv_p #1#2%
2740 {%
2741     \xintSgnFork{\XINT_Cmp {#1}{#2}}%
2742     \XINT_iiseqb:f:csv_pa\XINT_iiseqb:f:csv_pb\XINT_iiseqb:f:csv_pc {#1}{#2}%
2743 }%
2744 \def\XINT_iiseqb:f:csv_pb #1#2#3#4{#3,#1}%
2745 \def\XINT_iiseqb:f:csv_pc #1#2#3#4{#3}%
2746 \def\XINT_iiseqb:f:csv_bg #1{\if #1n\expandafter\XINT_iiseqb:f:csv_na\else
2747         \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2748 \def\XINT_iiseqb:f:csv_na #1#2#3#4{\expandafter\XINT_iiseqb:f:csv_n\expandafter
2749         {\romannumeral0\xintiiadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2750 \def\XINT_iiseqb:f:csv_n #1#2%

```

```
2751 {%
2752     \xintSgnFork{\XINT_Cmp {#1}{#2}}%
2753     \XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_iiseqb:f:csv_na {#1}{#2}%
2754 }%
```

### 11.55.3 \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.

```
2755 \def\XINTinFloatSeqB:f:csv #1#2{\expandafter\XINT_flseqb:f:csv \expandafter
2756     {\romannumeral0\XINTinfloat [\XINTdigits]{#2}}{#1}}%
2757 \def\XINT_flseqb:f:csv #1#2{\expandafter\XINT_flseqb:f:csv_a\romannumeral`&&@#2#1!}%
2758 \def\XINT_flseqb:f:csv_a #1#2;#3;#4!{%
2759     \expandafter\xint_gobble_i\romannumeral`&&@%
2760     \xintifCmp {#3}{#4}\XINT_flseqb:f:csv_bl\XINT_seqb:f:csv_be\XINT_flseqb:f:csv_bg
2761     #1{#3}{#4}}{#2}}%
2762 \def\XINT_flseqb:f:csv_bl #1{\if #1p\expandafter\XINT_flseqb:f:csv_pa\else
2763     \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2764 \def\XINT_flseqb:f:csv_pa #1#2#3#4{\expandafter\XINT_flseqb:f:csv_p\expandafter
2765     {\romannumeral0\XINTinfloatadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2766 \def\XINT_flseqb:f:csv_p #1#2%
2767 {%
2768     \xintifCmp {#1}{#2}%
2769     \XINT_flseqb:f:csv_pa\XINT_flseqb:f:csv_pb\XINT_flseqb:f:csv_pc {#1}{#2}%
2770 }%
2771 \def\XINT_flseqb:f:csv_pb #1#2#3#4{#3,#1}%
2772 \def\XINT_flseqb:f:csv_pc #1#2#3#4{#3}%
2773 \def\XINT_flseqb:f:csv_bg #1{\if #1n\expandafter\XINT_flseqb:f:csv_na\else
2774     \xint_afterfi{\expandafter,\xint_gobble_iv}\fi }%
2775 \def\XINT_flseqb:f:csv_na #1#2#3#4{\expandafter\XINT_flseqb:f:csv_n\expandafter
2776     {\romannumeral0\XINTinfloatadd{#4}{#1}}{#2}{#3,#1}{#4}}%
2777 \def\XINT_flseqb:f:csv_n #1#2%
2778 {%
2779     \xintifCmp {#1}{#2}%
2780     \XINT_seqb:f:csv_nc\XINT_seqb:f:csv_nb\XINT_flseqb:f:csv_na {#1}{#2}%
2781 }%
```

## 11.56 User defined functions: \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.

```
2782 \def\XINT_tmpa #1#2#3#4%
```



```

2783 {%
2784 \def #1##1(##2)##3==##4;{%
2785 \edef\XINT_expr_tmpa {##1}%
2786 \edef\XINT_expr_tmpa {\xint_zapspaces_o \XINT_expr_tmpa}%
2787 \def\XINT_expr_tmpb {0}%
2788 \def\XINT_expr_tmpc {(##4)}%
2789 \xintFor ####1 in {##2} \do
2790 {\edef\XINT_expr_tmpb {\the\numexpr\XINT_expr_tmpb+\xint_c_i}%
2791 \edef\XINT_expr_tmpc {subs(\unexpanded\expandafter{\XINT_expr_tmpc},%
2792 #####1=#####\XINT_expr_tmpb)}}%
2793 }%
2794 \expandafter\XINT_expr_defuserfunc
2795 \csname XINT_#2_func_\XINT_expr_tmpa\expandafter\endcsname
2796 \expandafter{\XINT_expr_tmpa}{#2}%
2797 \expandafter#3\csname XINT_#2_userfunc_\XINT_expr_tmpa\endcsname
2798 [\XINT_expr_tmpb]{\XINT_expr_tmpc}%
2799 \ifxintverbose\xintMessage {xintexpr}{Info}
2800 {Function \XINT_expr_tmpa\space for \string\xint #4 parser
2801 associated to \string\XINT_#2_userfunc_\XINT_expr_tmpa\space
2802 with \ifxintglobaldefs global \fi meaning \expandafter\meaning
2803 \csname XINT_#2_userfunc_\XINT_expr_tmpa\endcsname}%
2804 \fi
2805 \xintexprRestoreCatcodes
2806 }%
2807 }%
2808 \def\xintdeffunc {\xintexprSafeCatcodes\xintdeffunc_a}%
2809 \def\xintdefiifunc {\xintexprSafeCatcodes\xintdefiifunc_a}%
2810 \def\xintdeffloatfunc {\xintexprSafeCatcodes\xintdeffloatfunc_a}%
2811 \XINT_tmpa\xintdeffunc_a {expr} \XINT_NewFunc {expr}%
2812 \XINT_tmpa\xintdefiifunc_a {iiexpr}\XINT_NewIIFunc {iiexpr}%
2813 \XINT_tmpa\xintdeffloatfunc_a{floatexpr}\XINT_NewFloatFunc{floatexpr}%
2814 \def\XINT_expr_defuserfunc #1#2#3%
2815 {%
2816 \XINT_global
2817 \def #1##1##2##3{\expandafter ##1\expandafter ##2%
2818 \csname .=\XINT:expr:userfunc{#3}{#2}{\XINT_expr_unlock ##3}\endcsname
2819 }%
2820 }%
2821 \def\XINT:expr:userfunc #1#2#3%
2822 {\csname XINT_#1_userfunc_#2\expandafter\endcsname
2823 \romannumeral0\xintcsvtolistnonstripped{#3}}%
2824 \catcode`~ 12
2825 \def\XINT:newexpr:userfunc #1#2#3%
2826 {\~xintExpandArgs{XINT_#1_userfunc_#2}{\xintCSVtoListNonStripped{#3}}}%
2827 \catcode`~ 3

```

### 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.

```

2828 \def\XINT_tmpa #1{\expandafter\def\csname xintunassign#1func\endcsname ##1{%
2829   \edef\XINT_expr_tmpa{##1}%
2830   \edef\XINT_expr_tmpa {\xint_zapspaces_o\XINT_expr_tmpa}%
2831   \XINT_global\expandafter
2832     \let\csname XINT_#1_func_\XINT_expr_tmpa\endcsname\xint_undefined
2833   \XINT_global\expandafter
2834     \let\csname XINT_#1_userfunc_\XINT_expr_tmpa\endcsname\xint_undefined
2835   \ifxintverbose\xintMessage {xintexpr}{Info}
2836     {Function \XINT_expr_tmpa space for \string\xint #1 parser now
2837     \ifxintglobaldefs globally \fi undefined.}%
2838   \fi}}%
2839 \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 `\xintdefunc`.

```

2840 \def\XINT_expr_wrapit #1{\expandafter\XINT_expr_wrap\csname.=#1\endcsname}%
2841 \def\xintNewFunction #1#2[#3]#4%
2842 {%
2843   \edef\XINT_expr_tmpa {#1}%
2844   \edef\XINT_expr_tmpa {\xint_zapspaces_o \XINT_expr_tmpa}%
2845   \def\XINT_expr_tmpb ##1##2##3##4##5##6##7##8##9{#4}%
2846   \begingroup
2847     \ifcase #3\relax
2848       \toks0{}%
2849     \or \toks0{##1}%
2850     \or \toks0{##1##2}%
2851     \or \toks0{##1##2##3}%
2852     \or \toks0{##1##2##3##4}%
2853     \or \toks0{##1##2##3##4##5}%
2854     \or \toks0{##1##2##3##4##5##6}%
2855     \or \toks0{##1##2##3##4##5##6##7}%
2856     \or \toks0{##1##2##3##4##5##6##7##8}%
2857     \else \toks0{##1##2##3##4##5##6##7##8##9}%
2858     \fi
2859     \expandafter
2860   \endgroup\expandafter
2861   \XINT_global\expandafter
2862   \def\csname XINT_expr_macrofunc_\XINT_expr_tmpa\expandafter\endcsname
2863   \the\toks0\expandafter{\XINT_expr_tmpb
2864     {\XINT_expr_wrapit{##1}}{\XINT_expr_wrapit{##2}}{\XINT_expr_wrapit{##3}}%
2865     {\XINT_expr_wrapit{##4}}{\XINT_expr_wrapit{##5}}{\XINT_expr_wrapit{##6}}%
2866     {\XINT_expr_wrapit{##7}}{\XINT_expr_wrapit{##8}}{\XINT_expr_wrapit{##9}}}%
2867   \expandafter\XINT_expr_newfunction
2868     \csname XINT_expr_func_\XINT_expr_tmpa\expandafter\endcsname
2869     \expandafter{\XINT_expr_tmpa}{eval}\xintbareeval
2870   \expandafter\XINT_expr_newfunction
2871     \csname XINT_iiexpr_func_\XINT_expr_tmpa\expandafter\endcsname
2872     \expandafter{\XINT_expr_tmpa}{iieval}\xintbareiieval

```

```

2873 \expandafter\XINT_expr_newfunction
2874   \csname XINT_flexpr_func_\XINT_expr_tmpa\expandafter\endcsname
2875   \expandafter{\XINT_expr_tmpa}{floateval}\xintbarefloateval
2876 \ifxintverbose
2877   \xintMessage {xintexpr}{Info}
2878     {Function \XINT_expr_tmpa\space for the expression parsers is
2879     associated to \string\XINT_expr_macrofunc_\XINT_expr_tmpa\space
2880     with \ifxintglobaldefs global \fi meaning \expandafter\meaning
2881     \csname XINT_expr_macrofunc_\XINT_expr_tmpa\endcsname}%
2882 \fi
2883 }%
2884 \def\XINT_expr_newfunction #1#2#3#4%
2885 {%
2886   \XINT_global
2887   \def##1##2##3{\expandafter ##1\expandafter ##2\romannumeral0%
2888     \XINT:expr_macrofunc{#4}{#3}{#2}{\XINT_expr_unlock##3}}%
2889 }%
2890 \def\XINT:expr_macrofunc #1#2#3#4%
2891 {%
2892   #1\csname XINT_expr_macrofunc_#3\expandafter\endcsname
2893   \romannumeral0\xintcsvtolistnonstripped{#4}\relax
2894 }%
2895 \catcode~ 12
2896 \def\XINT:newexpr_macrofunc #1{%
2897 \def\XINT:newexpr_macrofunc ##1##2##3##4%
2898 {%
2899   \expandafter#1\csname.=~XINT:newexpr_macrofunc:a{##2}{##3}%
2900   {\xintCSVtoListNonStripped{##4}}\endcsname
2901 }%
2902 }\XINT:newexpr_macrofunc { }%
2903 \catcode~ 3
2904 \def\XINT:newexpr_macrofunc:a #1#2#3%
2905 {%
2906   \expandafter\XINT_expr_unlock\romannumeral0\csname xintbare#1\endcsname
2907   \csname XINT_expr_macrofunc_#2\endcsname#3\relax
2908 }%

```

## 11.59 \xintNewExpr, \xintNewIExpr, \xintNewFloatExpr, \xintNewIIExpr

11.59.1	\xintApply::csv and \xintApply:::csv . . . . .	372
11.59.2	Mysterious stuff . . . . .	373
11.59.3	\XINT_expr_redefinemacros . . . . .	376
11.59.4	\XINT_expr_redefineprints . . . . .	377
11.59.5	\xintNewExpr, ..., at last. . . . .	377
11.59.6	\ifxintexprsafecatcodes, \xintexprSafeCatcodes, \xintexprRestoreCatcodes . .	379

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 `\csname` 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`

```
2909 \def\xintApply::csv #1#2%
2910   {\expandafter\XINT_applyon::_a\expandafter {\romannumeral`&&#2}{#1}}%
```

```

2911 \def\XINT_applyon::_a #1#2{\XINT_applyon::_b {#2}{#1},}%
2912 \def\XINT_applyon::_b #1#2#3,{\expandafter\XINT_applyon::_c \romannumeral`&&@#3,{#1}{#2}}%
2913 \def\XINT_applyon::_c #1{\if #1,\expandafter\XINT_applyon::_end
2914 \else\expandafter\XINT_applyon::_d\fi #1}%
2915 \def\XINT_applyon::_d #1,#2{\expandafter\XINT_applyon::_e\romannumeral`&&@#2{#1},{#2}}%
2916 \def\XINT_applyon::_e #1,#2#3{\XINT_applyon::_b {#2}{#3}, #1}%
2917 \def\XINT_applyon::_end #1,#2#3{\xint_secondoftwo #3}%
2918 \def\xintApply:::csv #1#2#3%
2919 {\expandafter\XINT_applyon::_a\expandafter{\romannumeral`&&@#2{#1}{#3}}%
2920 \def\XINT_applyon::_a #1#2#3{\XINT_applyon::_b {#2}{#3}{#1},}%
2921 \def\XINT_applyon::_b #1#2#3#4,%
2922 {\expandafter\XINT_applyon::_c \romannumeral`&&@#4,{#1}{#2}{#3}}%
2923 \def\XINT_applyon::_c #1{\if #1,\expandafter\XINT_applyon::_end
2924 \else\expandafter\XINT_applyon::_d\fi #1}%
2925 \def\XINT_applyon::_d #1,#2#3%
2926 {\expandafter\XINT_applyon::_e\expandafter
2927 {\romannumeral`&&\xintApply:::csv {#2{#1}}{#3}}{#2}{#3}}%
2928 \def\XINT_applyon::_e #1,#2#3#4{\XINT_applyon::_b {#2}{#3}{#4}, #1}%
2929 \def\XINT_applyon::_end #1,#2#3#4{\xint_secondoftwo #4}%

```

### 11.59.2 Mysterious stuff

~ and \$ of catcode 12 in what follows.

```

2930 \catcode`~ 12
2931 \catcode`$ 12 % $
2932 \def\xint_ddfork #1$#2#3\krof {#2}% $$
2933 \def\XINT:NE:RApply:::csv #1#2#3#4%
2934 {\xintApply:::csv{\expandafter #2~\xint_exchangetwo_keepbraces{#4}}{#3}}%
2935 \def\XINT:NE:LApply:::csv #1#2#3{\xintApply:::csv{#2{#3}}}%
2936 \def\XINT:NE:RApply:::csv #1{\xintApply:::csv}%
2937 \def\XINT:NE:two#1{\XINT:NE:two_{#1}{\detokenize{#1}}}%
2938 \def\XINT:NE:two_#1#2#3#4%
2939 {\expandafter\XINT:NE:two_a\romannumeral`&&@#4!{#3}{#1}{#2}}%
2940 \def\XINT:NE:two_a#1#2!#3#4#5%
2941 {\expandafter\XINT:NE:two_b\romannumeral`&&@#3!#1{#4}{#5}{#1#2}}%
2942 \def\XINT:NE:two_b#1#2!#3#4#5{\XINT:NE:two_fork_dd#1#3{#4}{#5}{#1#2}}%
2943 \def\XINT:NE:two_fork_dd #1#2{%
2944 \xint_ddfork
2945 #1#2\XINT:NE:RApply:::csv
2946 #1$\XINT:NE:RApply:::csv% $
2947 $#2\XINT:NE:LApply:::csv% $
2948 $$\XINT:NE:two_fork_nn #1#2}% $$
2949 \krof
2950}%
2951 \def\XINT:NE:two_fork_nn #1#2#3#4{%
2952 \if #1##\xint_dothis{#4}\fi
2953 \if #1~\xint_dothis{#4}\fi
2954 \if #2##\xint_dothis{#4}\fi
2955 \if #2~\xint_dothis{#4}\fi
2956 \xint_orthat{#3}%
2957}%
2958 \def\XINT:NE:one#1#2{\expandafter\XINT:NE:one_a\romannumeral`&&@#2!#1}%
2959 \def\XINT:NE:one_a#1#2!#3{%

```

```

2960 \if ##1\xint_dothis {\detokenize{#3}}\fi
2961 \if ~#1\xint_dothis {\detokenize{#3}}\fi
2962 \if $#1\xint_dothis {\xintApply::csv{\detokenize{#3}}}\fi %$
2963 \xint_orthat #3{#1#2}%
2964}%
2965\def\xINT:NE:oneopt#1[#2]#3%
2966 {\expandafter\xINT:NE:oneopt_a\romannumeral`&&@#3!{#2}#1}%
2967\def\xINT:NE:oneopt_a#1#2!#3#4%
2968 {\expandafter\xINT:NE:oneopt_b\romannumeral`&&@#3!#1#4{#1#2}}%
2969\def\xINT:NE:oneopt_b#1#2!#3#4%
2970 {\expandafter\xINT:NE:oneopt_fork#1#3#4{#1#2}}%
2971\def\xINT:NE:oneopt_fork#1#2#3#4%
2972 \if1\if###11\else\if~#11\else\if###21\else\if~#21\else0\fi\fi\fi\fi
2973 \xint_dothis {\detokenize{#3}[#4]}\fi
2974 \if $#2\xint_dothis {\xintApply::csv{\detokenize{#3}[#4]}\fi %$
2975 \xint_orthat{#3[#4]}%
2976}% pas complètement général, mais bon
2977\def\xINT:NE:csv #1{\detokenize{#1}}% radicalement fainéant
2978\def\xINT:newexpr:one:and:opt #1,#2,#3!#4#5%
2979{%
2980 \if\relax#3\relax\expandafter\xint_firstoftwo\else
2981 \expandafter\xint_secondoftwo\fi
2982 {\XINT:NE:one#4}{\XINT:NE:oneopt#5[\XINT:NE:one\xintNum{#2}]}{#1}%
2983}%
2984\def\xINT:newexpr:tacitzeroifonearg #1,#2,#3!#4#5%
2985{%
2986 \if\relax#3\relax\expandafter\xint_firstoftwo\else
2987 \expandafter\xint_secondoftwo\fi
2988 {\XINT:NE:two#4{0}}{\XINT:NE:two#5[\XINT:NE:one\xintNum{#2}]}{#1}%
2989}%
2990\def\xINT:newiexpr:tacitzeroifonearg #1,#2,#3!#4%
2991{%
2992 \if\relax#3\relax\expandafter\xint_firstoftwo\else
2993 \expandafter\xint_secondoftwo\fi
2994 {\XINT:NE:two#4{0}}{\XINT:NE:two#4{#2}}{#1}%
2995}%
2996\def\xINT:newexpr:insertdollar~{$noexpand$}%
2997\def\xINT:newexpr:two:to:two #1,#2,!#3%
2998{%
2999 \XINT:NE:two_
3000 {\expandafter\xINT:expr:totwo\romannumeral`&&@#3}%
3001 {$noexpand$\expandafter~\XINT:expr:totwo~\romannumeral-`0\detokenize{#3}}%
3002 {#1}{#2}%
3003}%
3004\def\xINT:newflexpr:two:to:two #1,#2,!#3%
3005{%
3006 \XINT:NE:two_
3007 {#3}%
3008 {\expandafter\xINT:newexpr:insertdollar\detokenize{#3}}%
3009 {#1}{#2}%
3010}%
3011\def\xINT:newexpr:two:to:one #1,#2,!#3%

```

```

3012 {%
3013     \XINT:NE:two#3{#1}{#2}%
3014 }%
3015 \let\XINT:newflexpr:two:to:one\XINT:newexpr:two:to:one
3016 \def\xintiiifNotZeroNE:#1#2,#3,#4,%
3017 {%
3018     \if1\if###11\else\if~#11\else\if$#11\else0%$
3019         \fi\fi\fi
3020     \xint_dothis{~xintiiifNotZero}\fi
3021     \xint_orthat\xintiiifNotZero
3022     {#1#2}{#3}{#4}%
3023 }%
3024 \def\xintifIntNE:#1#2,#3,#4,%
3025 {%
3026     \if1\if###11\else\if~#11\else\if$#11\else0%$
3027         \fi\fi\fi
3028     \xint_dothis{~xintifInt}\fi
3029     \xint_orthat\xintifInt
3030     {#1#2}{#3}{#4}%
3031 }%
3032 \def\xintifFloatIntNE:#1#2,#3,#4,%
3033 {%
3034     \if1\if###11\else\if~#11\else\if$#11\else0%$
3035         \fi\fi\fi
3036     \xint_dothis{~xintifFloatInt}\fi
3037     \xint_orthat\xintifFloatInt
3038     {#1#2}{#3}{#4}%
3039 }%
3040 \def\xintiiifOneNE:#1#2,#3,#4,%
3041 {%
3042     \if1\if###11\else\if~#11\else\if$#11\else0%$
3043         \fi\fi\fi
3044     \xint_dothis{~xintiiifOne}\fi
3045     \xint_orthat\xintiiifOne
3046     {#1#2}{#3}{#4}%
3047 }%
3048 \def\xintifOneNE:#1#2,#3,#4,%
3049 {%
3050     \if1\if###11\else\if~#11\else\if$#11\else0%$
3051         \fi\fi\fi
3052     \xint_dothis{~xintifOne}\fi
3053     \xint_orthat\xintifOne
3054     {#1#2}{#3}{#4}%
3055 }%
3056 \def\xintiiifSgnNE:#1#2,#3,#4,#5,%
3057 {%
3058     \if1\if###11\else\if~#11\else\if$#11\else0%$
3059         \fi\fi\fi
3060     \xint_dothis{~xintiiifSgn}\fi
3061     \xint_orthat\xintiiifSgn
3062     {#1#2}{#3}{#4}{#5}%
3063 }%

```



### 11.59.3 \XINT\_expr\_redefinemacros

Completely refactored at 1.3.

```

3064 \def\XINT_expr_redefinemacros {%
3065   \let\XINT:NEhook:one\XINT:NE:one
3066   \let\XINT:NEhook:two\XINT:NE:two
3067   \let\XINT:NEhook:csv\XINT:NE:csv
3068   \let\XINT:expr:one:and:opt      \XINT:newexpr:one:and:opt
3069   \let\XINT:expr:one:or:two:nums  \XINT:newexpr:one:or:two:nums
3070   \let\XINT:iiexpr:one:or:two:    \XINT:newiiexpr:one:or:two:
3071   \let\XINT:expr:tacitzeroifonearg \XINT:newexpr:tacitzeroifonearg
3072   \let\XINT:iiexpr:tacitzeroifonearg \XINT:newiiexpr:tacitzeroifonearg
3073   \let\XINT:expr:two:to:two      \XINT:newexpr:two:to:two
3074   \let\XINT:expr:two:to:one      \XINT:newexpr:two:to:one
3075   \let\XINT:flexpr:two:to:one     \XINT:newflexpr:two:to:one
3076   \let\XINT:expr:two:to:one      \XINT:newexpr:two:to:one
3077   \let\XINT:flexpr:two:to:two     \XINT:newflexpr:two:to:two
3078   \let\XINT:flexpr:two:to:one     \XINT:newflexpr:two:to:one
3079   \let\xintiiifNotZero:          \xintiiifNotZeroNE:
3080   \let\xintifInt:                \xintifIntNE:
3081   \let\xintifFloatInt:           \xintifFloatIntNE:
3082   \let\xintiiifOne:              \xintiiifOneNE:
3083   \let\xintifOne:                \xintifOneNE:
3084   \let\xintiiifSgn:              \xintiiifSgnNE:
3085   \let\xintSeqNumeric::csv       \xintSeq::csv
3086   \let\xintiiSeqNumeric::csv     \xintiiSeq::csv
3087   \let\XINTinFloatSeqNumeric::csv \XINTinFloatSeq::csv
3088   \let\xintSeqBNumeric::csv      \xintSeqB::csv
3089   \let\xintiiSeqBNumeric::csv    \xintiiSeqB::csv
3090   \let\XINTinFloatSeqBNumeric::csv \XINTinFloatSeqB::csv
3091   \def\xintSeq::csv
3092     {\XINT:NE:two_\xintSeqNumeric::csv{$noexpand\xintSeq::csv}}%
3093   \def\xintiiSeq::csv
3094     {\XINT:NE:two_\xintiiSeqNumeric::csv{$noexpand\xintiiSeq::csv}}%
3095   \def\XINTinFloatSeq::csv
3096     {\XINT:NE:two_\XINTinFloatSeqNumeric::csv{$noexpand\XINTinFloatSeq::csv}}%
3097   \def\xintSeqB::csv
3098     {\XINT:NE:two_\xintSeqBNumeric::csv{$noexpand\xintSeqB:f:csv}}%
3099   \def\xintiiSeqB::csv
3100     {\XINT:NE:two_\xintiiSeqBNumeric::csv{$noexpand\xintiiSeqB:f:csv}}%
3101   \def\XINTinFloatSeqB::csv
3102     {\XINT:NE:two_\XINTinFloatSeqBNumeric::csv{$noexpand\XINTinFloatSeqB:f:csv}}%
3103   \def\xintListSel:x:csv {\~xintListSel:f:csv}%
3104   \let\XINT:expr:userfunc \XINT:newexpr:userfunc
3105   \let\XINT:expr:macrofunc \XINT:newexpr:macrofunc
3106   \def\XINTinRandomFloatSdigits{\~XINTinRandomFloatSdigits}%
3107   \def\XINTinRandomFloatSixteen{\~XINTinRandomFloatSixteen}%
3108   \def\xintiiRandRange{\~xintiiRandRange}%
3109   \def\xintiiRandRangeAtoB{\~xintiiRandRangeAtoB}%
3110 }%
```



#### 11.59.4 \XINT\_expr\_redefineprints

This is used by \xintNewExpr but not by \xintdeffunc.

```

3111 \def\XINT_expr_redefineprints
3112 {%
3113   \def\XINT_flexpr_noopt
3114     {\expandafter\XINT_flexpr_withopt_b\expandafter-\romannumeral0\xintbarefloateval }%
3115   \def\XINT_flexpr_withopt_b ##1##2%
3116     {\expandafter\XINT_flexpr_wrap\csname .;##1.=\XINT_expr_unlock ##2\endcsname }%
3117   \def\XINT_expr_unlock_sp ##1.;##2##3.##4!%
3118     {\if -##2\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
3119     \XINTdigits{##2##3}{##4}}%
3120   \def\XINT_expr_print ##1%
3121     {\expandafter\xintSPRaw::csv\expandafter
3122     {\romannumeral`&&\XINT_expr_unlock ##1}}%
3123   \def\XINT_iexpr_print ##1%
3124     {\expandafter\xintCSV::csv\expandafter
3125     {\romannumeral`&&\XINT_expr_unlock ##1}}%
3126   \def\XINT_boolexpr_print ##1%
3127     {\expandafter\xintIsTrue::csv\expandafter
3128     {\romannumeral`&&\XINT_expr_unlock ##1}}%
3129   \def\xintCSV::csv {\xintCSV::csv }%
3130   \def\xintSPRaw::csv {\xintSPRaw::csv }%
3131   \def\xintPFloat::csv {\xintPFloat::csv }%
3132   \def\xintIsTrue::csv {\xintIsTrue::csv }%
3133   \def\xintRound::csv {\xintRound::csv }%
3134 }%

```

#### 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.

```

3135 \def\xintNewExpr {\XINT_NewExpr\XINT_expr_redefineprints\xint_firstofone
3136                  \xinttheexpr\XINT_newexpr_clean}%
3137 \def\xintNewFloatExpr{\XINT_NewExpr\XINT_expr_redefineprints\xint_firstofone
3138                  \xintthefloatexpr\XINT_newexpr_clean}%
3139 \def\xintNewIExpr {\XINT_NewExpr\XINT_expr_redefineprints\xint_firstofone
3140                  \xinttheiexpr\XINT_newexpr_clean}%
3141 \def\xintNewIIExpr {\XINT_NewExpr\XINT_expr_redefineprints\xint_firstofone
3142                  \xinttheiexpr\XINT_newexpr_clean}%
3143 \def\xintNewBoolExpr {\XINT_NewExpr\XINT_expr_redefineprints\xint_firstofone
3144                  \xinttheboolexpr\XINT_newexpr_clean}%
3145 \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.

```

3146 \def\XINT_NewFunc
3147   {\XINT_NewExpr}\xint_gobble_i\xintthebareeval\XINT_newfunc_clean}%
3148 \def\XINT_NewFloatFunc
3149   {\XINT_NewExpr}\xint_gobble_i\xintthebarefloateval\XINT_newfunc_clean}%
3150 \def\XINT_NewIIFunc

```

```
3151 {\XINT_NewExpr}\xint_gobble_i\xintthebareiieval\XINT_newfunc_clean}%
3152 \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.

```
3153 \def\XINT_NewExpr #1#2#3#4#5#6[#7]%
3154 {%
3155 \begingroup
3156 \ifcase #7\relax
3157 \toks0 {\endgroup\XINT_global\def#5}%
3158 \or \toks0 {\endgroup\XINT_global\def#5##1}%
3159 \or \toks0 {\endgroup\XINT_global\def#5##1##2}%
3160 \or \toks0 {\endgroup\XINT_global\def#5##1##2##3}%
3161 \or \toks0 {\endgroup\XINT_global\def#5##1##2##3##4}%
3162 \or \toks0 {\endgroup\XINT_global\def#5##1##2##3##4##5}%
3163 \or \toks0 {\endgroup\XINT_global\def#5##1##2##3##4##5##6}%
3164 \or \toks0 {\endgroup\XINT_global\def#5##1##2##3##4##5##6##7}%
3165 \or \toks0 {\endgroup\XINT_global\def#5##1##2##3##4##5##6##7##8}%
3166 \or \toks0 {\endgroup\XINT_global\def#5##1##2##3##4##5##6##7##8##9}%
3167 \fi
3168 #2\xintexprSafeCatcodes
3169 \XINT_expr_redefinmacros
3170 #1%
3171 \XINT_NewExpr_a #2#3#4#5%
3172 }%
```

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`.

```
3173 \catcode`~ 13 \catcode`@ 14 \catcode`\% 6 \catcode`# 12 \catcode`$ 11 @ $
3174 \def\XINT_NewExpr_a %1%2%3%4%5@
3175 {@
3176 \def\XINT_tmpa %1%2%3%4%5%6%7%8%9{%5}@
3177 \def~{$noexpand$}@
3178 \catcode`: 11 \catcode`_ 11
3179 \catcode`# 12 \catcode`~ 13 \escapechar 126
3180 \endlinechar -1 \everyeof {\noexpand }@
3181 \edef\XINT_tmppb
3182 {\scantokens\expandafter{\romannumeral`&&\expandafter
3183 %2\XINT_tmpa{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}\relax}@
3184 }@
3185 \escapechar 92 \catcode`# 6 \catcode`$ 0 @ $
3186 \edef\XINT_tmppa %1%2%3%4%5%6%7%8%9@
3187 {\scantokens\expandafter{\expandafter%3\meaning\XINT_tmppb}}@
3188 \the\toks0\expandafter
3189 {\XINT_tmppa{%1}{%2}{%3}{%4}{%5}{%6}{%7}{%8}{%9}}@
```

```

3190 %1{\ifxintverbose
3191   \xintMessage{xintexpr}{Info}@
3192   {\string%4\space now with @
3193   \ifxintglobaldefs global \fi meaning \meaning%4}@
3194   \fi}@
3195 }@
3196 \catcode`% 14

```

## 11.59.6 \ifxintexprsafecatcodes, \xintexprSafeCatcodes, \xintexprRestoreCatcodes

### 1.3c (2018/06/17).

Added \ifxintexprsafecatcodes to allow nesting

```

3197 \newif\ifxintexprsafecatcodes
3198 \let\xintexprRestoreCatcodes\empty
3199 \def\xintexprSafeCatcodes
3200 {%
3201   \unless\ifxintexprsafecatcodes
3202     \edef\xintexprRestoreCatcodes {%
3203       \catcode59=\the\catcode59 % ;
3204       \catcode34=\the\catcode34 % "
3205       \catcode63=\the\catcode63 % ?
3206       \catcode124=\the\catcode124 % |
3207       \catcode38=\the\catcode38 % &
3208       \catcode33=\the\catcode33 % !
3209       \catcode93=\the\catcode93 % ]
3210       \catcode91=\the\catcode91 % [
3211       \catcode94=\the\catcode94 % ^
3212       \catcode95=\the\catcode95 % _
3213       \catcode47=\the\catcode47 % /
3214       \catcode41=\the\catcode41 % )
3215       \catcode40=\the\catcode40 % (
3216       \catcode42=\the\catcode42 % *
3217       \catcode43=\the\catcode43 % +
3218       \catcode62=\the\catcode62 % >
3219       \catcode60=\the\catcode60 % <
3220       \catcode58=\the\catcode58 % :
3221       \catcode46=\the\catcode46 % .
3222       \catcode45=\the\catcode45 % -
3223       \catcode44=\the\catcode44 % ,
3224       \catcode61=\the\catcode61 % =
3225       \catcode96=\the\catcode96 % `
3226       \catcode32=\the\catcode32\relax % space
3227       \noexpand\xintexprsafecatcodesfalse
3228     }%
3229   \fi
3230   \xintexprsafecatcodestrue
3231     \catcode59=12 % ;
3232     \catcode34=12 % "
3233     \catcode63=12 % ?
3234     \catcode124=12 % |
3235     \catcode38=4 % &
3236     \catcode33=12 % !

```

*TOC, xintkernel, xinttools, xintcore, xint, xintbinhex, xintgcd, xintfrac, xintseries, xintcfrc, xintexpr, indices*

```

3237      \catcode93=12 % ]
3238      \catcode91=12 % [
3239      \catcode94=7  % ^
3240      \catcode95=8  % _
3241      \catcode47=12 % /
3242      \catcode41=12 % )
3243      \catcode40=12 % (
3244      \catcode42=12 % *
3245      \catcode43=12 % +
3246      \catcode62=12 % >
3247      \catcode60=12 % <
3248      \catcode58=12 % :
3249      \catcode46=12 % .
3250      \catcode45=12 % -
3251      \catcode44=12 % ,
3252      \catcode61=12 % =
3253      \catcode96=12 % `
3254      \catcode32=10 % space
3255 }%
3256 \let\XINT_tmpa\relax \let\XINT_tmpp\relax \let\XINT_tmppc\relax
3257 \XINT_restorecatcodes_endinput%

```

## 12 Per package indices of control sequences

### 12.1 Index of [xintkernel](#)

<code>\_!_/</code> .....	18		
<b>C</b>			
<code>\catcode</code> .....	5--7, 18		
<code>\char</code> .....	10		
<code>\chardef</code> .....	8		
<code>\csname</code> .....	5, 6, 8, 14--17		
<b>E</b>			
<code>\edef</code> .....	6, 7, 15--17		
<code>\endcsname</code> .....	5, 6, 8, 14--17		
<code>\endinput</code> .....	5--7		
<code>\endlinechar</code> .....	5--7		
<b>F</b>			
<code>\fdef</code> .....	11		
<b>I</b>			
<code>\if</code> .....	17, 18		
<code>\ifdefined</code> .....	8, 9, 11, 17		
<code>\ifxintglobaldefs</code> .....	18		
<code>\ifxintverbose</code> .....	18		
<code>\immediate</code> .....	5, 8, 9, 18		
<code>\inputlineno</code> .....	18		
<b>K</b>			
<code>\krof</code> .....	10, 13, 14		
<b>L</b>			
<code>\lccode</code> .....	18		
<b>M</b>			
<code>\mathchardef</code> .....	8		
<b>N</b>			
<code>\newif</code> .....	18		
<code>\noexpand</code> .....	6, 7, 15--17		
<code>\numexpr</code> .....	5, 12--15, 17, 18		
<b>O</b>			
<code>\odef</code> .....	11		
<code>\oodef</code> .....	11		
<b>P</b>			
<code>\pdfuniformdeviate</code> .....	8		
<code>\PrepareCatcodes</code> .....	6, 7		
<code>\ProvidesPackage</code> .....	8		
<b>R</b>			
<code>\R</code> .....	9		
<code>\romannumeral</code> .....	7, 11--13		
<b>U</b>			
<code>\uniformdeviate</code> .....	8		
<b>W</b>			
<code>\W</code> .....	9, 10		
<code>\write</code> .....	5, 8, 9, 18		
<b>X</b>			
<code>\xint:</code> .....	10--13, 17, 18		
<code>\xint_afterfi</code> .....	10		
<code>\xint_bracedstopper</code> .....	10		
<code>\xint_Bye</code> .....	10		
<code>\xint_bye</code> .....	10--13		
<code>\xint_c_</code> .....	8, 12--15		
<code>\xint_c_i</code> .....	8, 12, 13, 15		
<code>\xint_c_ii</code> .....	8, 12, 13		
<code>\xint_c_ii^v</code> .....	8		
<code>\xint_c_ii^vi</code> .....	8		
<code>\xint_c_ii^vii</code> .....	8, 17		
<code>\xint_c_ii^viii</code> .....	8		
<code>\xint_c_ii^xii</code> .....	8		
<code>\xint_c_ii^xiv</code> .....	8, 17		
<code>\xint_c_ii^xxi</code> .....	8, 17		
<code>\xint_c_iii</code> .....	8, 12, 13		
<code>\xint_c_iv</code> .....	8, 12, 13		
<code>\xint_c_ix</code> .....	8, 12, 15		
<code>\xint_c_v</code> .....	8, 12, 13, 15		
<code>\xint_c_vi</code> .....	8, 12, 13		
<code>\xint_c_vii</code> .....	8, 12, 13		
<code>\xint_c_viii</code> .....	8, 12, 13		
<code>\xint_c_x</code> .....	8		
<code>\xint_c_x^iv</code> .....	8		
<code>\xint_c_xii</code> .....	8		
<code>\xint_c_xiv</code> .....	8		
<code>\xint_c_xvi</code> .....	8		
<code>\xint_c_xviii</code> .....	8		
<code>\xint_c_xxii</code> .....	8		
<code>\xint_dothis</code> .....	10		
<code>\xint_exchangetwo_keepbraces</code> .....	9		
<code>\XINT_expandableerror</code> .....	18		
<code>\XINT_expandableerror_continue</code> .....	18		
<code>\xint_firstofone</code> .....	9, 10, 18		
<code>\xint_firstoftwo</code> .....	9, 17		
<code>\XINT_global</code> .....	18		
<code>\xint_gob_andstop_</code> .....	9		
<code>\xint_gob_andstop_i</code> .....	9		
<code>\xint_gob_andstop_ii</code> .....	9		
<code>\xint_gob_andstop_iii</code> .....	9		
<code>\xint_gob_andstop_iv</code> .....	9		
<code>\xint_gob_andstop_v</code> .....	9		
<code>\xint_gob_andstop_vi</code> .....	9		
<code>\xint_gob_andstop_vii</code> .....	9		



## 12.2 Index of [xinttools](#)

<b>A</b>	
<code>\aftergroup</code> .....	20
<b>B</b>	
<code>\bgroup</code> .....	43
<b>C</b>	
<code>\catcode</code> .....	19, 37--40, 42, 43
<code>\csname</code> .....	19, 20, 28, 29, 31, 38--46, 49--52
<b>D</b>	
<code>\dimexpr</code> .....	40, 41
<b>E</b>	
<code>\edef</code> .....	38, 39, 41--45
<code>\endcsname</code> ....	19, 20, 28, 29, 31, 38--46, 49--52
<code>\endinput</code> .....	20
<code>\endlinechar</code> .....	19
<code>\escapechar</code> .....	44, 45
<b>F</b>	
<code>\futurelet</code> .....	36, 38, 39
<b>I</b>	
<code>\if</code> .....	52
<code>\ifcsname</code> .....	43, 44
<code>\immediate</code> .....	19
<b>K</b>	
<code>\krof</code> .....	26--32, 48--52
<b>N</b>	
<code>\newtoks</code> .....	20
<code>\noexpand</code> .....	39, 41--43
<code>\numexpr</code> .....	20, 26--36, 38--43, 45, 47--52
<b>P</b>	
<code>\ProvidesPackage</code> .....	20
<b>R</b>	
<code>\R</code> .....	21, 24
<code>\repeat</code> .....	35, 44
<code>\romannumeral</code> ....	20--33, 36, 38, 40--43, 45--54
<b>T</b>	
<code>\to</code> .....	43--45
<b>W</b>	
<code>\w</code> .....	19, 20
<code>\write</code> .....	19
<b>X</b>	
<code>\x</code> .....	19, 20
<code>\xint:</code> ...	21--23, 26--31, 43--45, 47--49, 51--54
<code>\XINT?expr_D</code> .....	41
<code>\XINT?expr_Da</code> .....	40
<code>\XINT?expr_Di</code> .....	41
<code>\XINT?expr_U</code> .....	41
<code>\XINT?expr_Ua</code> .....	40
<code>\XINT?expr_Ui</code> .....	41
<code>\XINT?expr_V</code> .....	41
<code>\XINT?expr_Va</code> .....	40
<code>\XINT?expr_Vb</code> .....	40
<code>\XINT?expr_Vc</code> .....	40
<code>\XINT?expr_Vd</code> .....	40, 41
<code>\XINT?expr_Ve</code> .....	41
<code>\XINT?expr_Vf</code> .....	40, 41
<code>\XINT?expr_Vi</code> .....	41
<code>\XINT?expr_Vx</code> .....	41
<code>\XINT?expr_Vy</code> .....	41
<code>\xint_afterfi</code> .....	34, 38, 45
<code>\XINT_apply</code> .....	32
<code>\XINT_apply_end</code> .....	32
<code>\XINT_apply_loop_a</code> .....	32
<code>\XINT_apply_loop_b</code> .....	32
<code>\XINT_applyunbr</code> .....	32
<code>\XINT_applyunbr_end</code> .....	32, 33
<code>\XINT_applyunbr_loop_a</code> .....	32, 33
<code>\XINT_applyunbr_loop_b</code> .....	32, 33
<code>\xint_arrayname</code> .....	44, 45
<code>\XINT_assign_a</code> .....	43
<code>\XINT_assign_b</code> .....	43
<code>\XINT_assign_c</code> .....	43, 44
<code>\XINT_assign_d</code> .....	44
<code>\XINT_assign_def</code> .....	43, 44
<code>\XINT_assign_e</code> .....	44
<code>\XINT_assign_f</code> .....	43, 44
<code>\XINT_assign_fork</code> .....	43
<code>\XINT_assign_opt</code> .....	43
<code>\XINT_assignarray</code> .....	44, 45
<code>\XINT_assignarray_def</code> .....	44, 45
<code>\XINT_assignarray_end</code> .....	45
<code>\XINT_assignarray_fork</code> .....	44
<code>\XINT_assignarray_loop</code> .....	45
<code>\XINT_assignarray_opt</code> .....	44
<code>\xint_bracedstopper</code> .....	44, 45
<code>\xint_bye</code> .....	21--34, 47--54
<code>\xint_c_</code> .....	26--31, 34, 44, 45, 47, 51
<code>\xint_c_i</code> ....	26--31, 33, 42, 45, 47, 49, 51, 52
<code>\xint_c_ii</code> .....	26--31, 42, 47, 49, 51, 52
<code>\xint_c_iii</code> .....	26--31, 43, 47, 49, 51, 52
<code>\xint_c_iv</code> .....	26--31, 47, 49, 51, 52
<code>\xint_c_ix</code> .....	30, 31, 47, 49, 51, 52
<code>\xint_c_v</code> .....	26--31, 47, 49, 51, 52
<code>\xint_c_vi</code> .....	26--31, 47, 49, 51, 52
<code>\xint_c_vii</code> .....	26--31, 47, 49, 51, 52
<code>\xint_c_viii</code> .....	26--32, 47--52
<code>\xint_c_x</code> .....	27
<code>\XINT_csvtol_finish_a</code> .....	24
<code>\XINT_csvtol_finish_b</code> .....	24

\XINT_csvtol_finish_di .....	24, 25	\XINT_inline_w .....	37
\XINT_csvtol_finish_dii .....	24, 25	\XINT_item .....	37
\XINT_csvtol_finish_diii .....	24	\xint_itemcount .....	45
\XINT_csvtol_finish_div .....	24	\XINT_keep:f:csv_a .....	48
\XINT_csvtol_finish_dv .....	24	\XINT_keep:f:csv_keepall .....	49
\XINT_csvtol_finish_dvi .....	24	\XINT_keep:f:csv_keepnone .....	48, 49
\XINT_csvtol_finish_dvii .....	24	\XINT_keep:f:csv_loop .....	49--51
\XINT_csvtol_finish_dviii .....	24	\XINT_keep:f:csv_loop_end .....	50
\XINT_csvtol_loop_a .....	24	\XINT_keep:f:csv_loop_pickeight .....	50
\XINT_csvtol_loop_b .....	24	\XINT_keep:f:csv_neg .....	49
\XINT_expandableerror .....	45	\XINT_keep:f:csv_neg_a .....	49
\XINT_first:f:csv_a .....	53	\XINT_keep:f:csv_neg_done .....	49
\xint_firstofone .....	20, 22, 23, 46, 53	\XINT_keep:f:csv_pos .....	49
\xint_firstoftwo .....	38, 39, 42	\XINT_keep:f:csv_pos_fork .....	49
\XINT_flet_macro .....	38, 43, 44	\XINT_keep:f:csv_pos_keepall .....	49, 50
\XINT_flet_sp? .....	38	\XINT_keep:f:csv_trimloop .....	49, 52
\XINT_flet_zapsp .....	38, 43, 44	\XINT_keep:f:csv_trimloop_finish .....	49
\XINT_for .....	38	\XINT_keep:f:csv_trimloop_trimnine .....	49
\XINT_for_d .....	38, 39	\XINT_keep_a .....	27
\XINT_for_forever? .....	38	\XINT_keep_keepall .....	28
\XINT_for_ifstar .....	38	\XINT_keep_keepnone .....	27, 29
\XINT_for_last? .....	39	\XINT_keep_loop .....	28, 30
\XINT_for_last?yes .....	39, 42, 43	\XINT_keep_loop_end .....	28
\XINT_forever .....	39, 40	\XINT_keep_loop_pickeight .....	28
\XINT_forever_a .....	40, 41	\XINT_keep_neg .....	27, 29
\XINT_forever_b .....	41	\XINT_keep_neg_a .....	27
\XINT_forever_c .....	41	\XINT_keep_pos .....	27, 28
\XINT_forever_d .....	41	\XINT_keepunbr_a .....	29
\XINT_forever_opt_a .....	41	\XINT_keepunbr_loop .....	29, 32
\XINT_forever_opt_c .....	41	\XINT_keepunbr_loop_end .....	29
\XINT_forfour_d .....	42, 43	\XINT_keepunbr_loop_pickeight .....	29
\XINT_forpair_d .....	42	\XINT_keepunbr_pos .....	29
\XINT_forthree_d .....	42	\XINT_last:f:csv_loop .....	54
\XINT_forx .....	38, 39	\XINT_last_loop_enda .....	54
\XINT_forx_d .....	39	\XINT_last_loop_endb .....	54
\XINT_forx_empty? .....	39	\XINT_last_loop_endc .....	54
\XINT_forx_forever? .....	39	\XINT_last_loop_endd .....	54
\xint_gob_til_minus .....	28, 29, 31, 49--51	\XINT_last_loop_ende .....	54
\xint_gob_til_R .....	21, 24	\XINT_last_loop_endf .....	54
\xint_gob_til_xint: .....		\XINT_last_loop_endg .....	54
.....	21, 23, 27, 31, 47, 48, 51, 52, 54	\XINT_last_loop_endh .....	54
\xint_gob_til_Z .....	24	\XINT_length:f:csv_a .....	47, 49, 51, 52
\XINT_gobble .....	26, 28	\XINT_length:f:csv_finish .....	47
\xint_gobble_i .....	21, 35, 37, 40	\XINT_length:f:csv_loop .....	47
\xint_gobble_ii .....	21, 27	\XINT_length_loop .....	26, 27, 30, 31
\xint_gobble_iii .....	21, 35, 36	\XINT_lengthupto:f:csv_a .....	47--49
\xint_gobble_iv .....	21, 34	\XINT_lengthupto:f:csv_empty .....	48
\xint_gobble_v .....	21	\XINT_lengthupto:f:csv_finish_a .....	48
\xint_gobble_vi .....	21	\XINT_lengthupto:f:csv_finish_b .....	48
\xint_gobble_vii .....	21	\XINT_lengthupto:f:csv_gt .....	48
\xint_gobble_viii .....	21	\XINT_lengthupto:f:csv_loop_a .....	48
\XINT_inline_b .....	37	\XINT_lengthupto:f:csv_loop_b .....	48
\XINT_inline_d .....	37	\XINT_lengthupto_loop .....	28, 29
\XINT_inline_e .....	37	\XINT_lws .....	25
\XINT_inline_f .....	37	\XINT_lws_e .....	25, 26
\XINT_inline_g .....	37	\XINT_lws_e_i .....	25, 26
\XINT_inline_macro .....	37	\XINT_lws_e_ii .....	25, 26



<code>\XINT_lws_e_iii</code>	25, 26	<code>\XINT_to_forever</code>	38, 39
<code>\XINT_lws_e_iv</code>	25	<code>\XINT_to_forxever</code>	39
<code>\XINT_lws_e_v</code>	25	<code>\XINT_token</code>	36--39, 43, 44
<code>\XINT_lws_e_vi</code>	25	<code>\XINT_tokenB</code>	36
<code>\XINT_lws_loop_a</code>	25	<code>\XINT_toks</code>	20, 38, 39, 41--43
<code>\XINT_lws_loop_b</code>	25, 26	<code>\XINT_trim:f:csv_a</code>	50
<code>\XINT_nthelt:f:csv_a</code>	52	<code>\XINT_trim:f:csv_finish</code>	51
<code>\XINT_nthelt:f:csv_neg</code>	52	<code>\XINT_trim:f:csv_loop</code>	51, 52
<code>\XINT_nthelt:f:csv_neg_done</code>	52	<code>\XINT_trim:f:csv_loop_trimnine</code>	51
<code>\XINT_nthelt:f:csv_neg_fork</code>	52	<code>\XINT_trim:f:csv_neg</code>	50
<code>\XINT_nthelt:f:csv_pos</code>	52	<code>\XINT_trim:f:csv_neg_a</code>	51
<code>\XINT_nthelt:f:csv_pos_cleanup</code>	52, 53	<code>\XINT_trim:f:csv_pos</code>	50, 51
<code>\XINT_nthelt:f:csv_pos_done</code>	52, 53	<code>\XINT_trim:f:csv_pos_done</code>	51, 52
<code>\XINT_nthelt_a</code>	26	<code>\XINT_trim:f:csv_toofew</code>	51
<code>\XINT_nthelt_neg</code>	26	<code>\XINT_trim:f:csv_trimall</code>	51
<code>\XINT_nthelt_neg_a</code>	26	<code>\XINT_trim:f:csv_trimnone</code>	50
<code>\XINT_nthelt_neg_b</code>	26	<code>\XINT_trim_a</code>	30
<code>\XINT_nthelt_pos</code>	26, 27	<code>\XINT_trim_finish</code>	31
<code>\XINT_nthelt_pos_done</code>	27	<code>\XINT_trim_loop</code>	27, 30, 31
<code>\XINT_nthelt_zero</code>	26	<code>\XINT_trim_loop_trimnine</code>	31
<code>\XINT_providespackage</code>	20	<code>\XINT_trim_neg</code>	30
<code>\XINT_restorecatcodes_endinput</code>	54	<code>\XINT_trim_neg_a</code>	30
<code>\XINT_restoreescapechar</code>	44, 45	<code>\XINT_trim_pos</code>	30, 31
<code>\XINT_reverse:f:csv_cleanup</code>	53	<code>\XINT_trim_pos_done</code>	30, 31
<code>\XINT_reverse:f:csv_finish</code>	53	<code>\XINT_trim_toofew</code>	31
<code>\XINT_reverse:f:csv_loop</code>	53	<code>\XINT_trim_trimall</code>	30, 32
<code>\XINT_revwbr_finish_a</code>	21	<code>\XINT_trim_trimnone</code>	30, 31
<code>\XINT_revwbr_finish_b</code>	21	<code>\XINT_trimunbr_a</code>	31
<code>\XINT_revwbr_finish_c</code>	21	<code>\XINT_trimunbr_neg</code>	31
<code>\XINT_revwbr_loop</code>	21	<code>\XINT_trimunbr_neg_a</code>	31
<code>\xint_secondoftwo</code>	38, 39, 41--43	<code>\xint_UDsignfork</code>	27, 30, 31, 48, 49, 51, 52
<code>\XINT_seq</code>	33	<code>\xint_UDzerominusfork</code>	26, 27, 29--31, 48, 50
<code>\XINT_seq_chkopt</code>	33	<code>\XINT_x</code>	39--43
<code>\XINT_seq_e</code>	33	<code>\XINT_xflet</code>	36, 37, 39
<code>\XINT_seq_n</code>	33	<code>\XINT_xflet_eq?</code>	36
<code>\XINT_seq_noopt</code>	33	<code>\XINT_xflet_macro</code>	36
<code>\XINT_seq_opt</code>	33, 34	<code>\XINT_xflet_sp?</code>	36
<code>\XINT_seq_p</code>	33	<code>\XINT_xflet_spB?</code>	36
<code>\XINT_seqo</code>	34	<code>\XINT_xflet_zapsp</code>	36
<code>\XINT_seqo_a</code>	34	<code>\XINT_xflet_zapspB</code>	36
<code>\XINT_seqo_na</code>	34	<code>\XINT_y</code>	39, 41--43
<code>\XINT_seqo_nb</code>	35	<code>\XINT_zapbsp_a</code>	21, 22
<code>\XINT_seqo_nc</code>	35	<code>\XINT_zapbsp_again</code>	22
<code>\XINT_seqo_nd</code>	35	<code>\XINT_zapbsp_again?</code>	22
<code>\XINT_seqo_o</code>	34, 35	<code>\XINT_zapbsp_b</code>	22
<code>\XINT_seqo_pa</code>	34	<code>\XINT_zapesp_a</code>	22, 23
<code>\XINT_seqo_pb</code>	34	<code>\XINT_zapesp_b</code>	22
<code>\XINT_seqo_pc</code>	34	<code>\XINT_zapesp_e</code>	22
<code>\XINT_seqo_pd</code>	34	<code>\XINT_zapesp_end</code>	22
<code>\XINT_sptoken</code>	20, 36, 38	<code>\XINT_zapesp_end?</code>	22
<code>\xint_stop_afterbye</code>	26, 52	<code>\XINT_zapsp_a</code>	23
<code>\xint_stop_aftergobble</code>	48, 50	<code>\XINT_zapsp_again</code>	23
<code>\xint_stop_atfirstoftwo</code>	33	<code>\XINT_zapsp_again?</code>	23
<code>\xint_temp</code>	44, 45	<code>\XINT_zapsp_b</code>	23
<code>\XINT_tmpa</code>	38, 54	<code>\XINT_zapsp_c</code>	23
<code>\XINT_tmpb</code>	38, 54	<code>\XINT_zapspb_bracedorone</code>	23
<code>\XINT_tmppc</code>	38, 54	<code>\XINT_zapspb_one?</code>	23

<code>\XINT_zapspb_onlyspaces</code> .....	23	<code>\xinttiloop_again_b</code> .....	35, 36
<code>\xintApply</code> .....	23, 24, 32	<code>\xinttiloopindex</code> .....	35, 44
<code>\xintapply</code> .....	32, 46	<code>\xinttiloopskipandredo</code> .....	36
<code>\xintApplyInline</code> .....	37, 38	<code>\xinttiloopskiptonext</code> .....	36
<code>\xintApplyNoExpand</code> .....	32	<code>\xintintegers</code> .....	40
<code>\xintapplynoexpand</code> .....	32	<code>\xintKeep</code> .....	27
<code>\xintApplyUnbraced</code> .....	32, 38	<code>\xintkeep</code> .....	27
<code>\xintapplyunbraced</code> .....	32	<code>\xintKeep:f:csv</code> .....	48, 54
<code>\xintApplyUnbracedNoExpand</code> .....	32	<code>\xintkeep:f:csv</code> .....	48
<code>\xintapplyunbracednoexpand</code> .....	32	<code>\xintKeepNoExpand</code> .....	27
<code>\xintAssign</code> .....	43	<code>\xintkeepnoexpand</code> .....	27
<code>\xintAssignArray</code> .....	44, 45	<code>\xintKeepUnbraced</code> .....	28
<code>\xintbracediloopindex</code> .....	35	<code>\xintkeepunbraced</code> .....	28
<code>\xintbracedouteriloopindex</code> .....	36	<code>\xintKeepUnbracedNoExpand</code> .....	28
<code>\xintBreakFor</code> .....	38, 39	<code>\xintkeepunbracednoexpand</code> .....	28, 29
<code>\xintBreakForAndDo</code> .....	38, 40	<code>\xintLastItem:f:csv</code> .....	54
<code>\xintbreakiloop</code> .....	35	<code>\xintlastitem:f:csv</code> .....	54
<code>\xintbreakiloopanddo</code> .....	35	<code>\xintlength</code> .....	26
<code>\xintbreakloop</code> .....	35	<code>\xintLength:f:csv</code> .....	47, 54
<code>\xintbreakloopanddo</code> .....	35	<code>\xintlength:f:csv</code> .....	47
<code>\xintCSVFirstItem</code> .....	54	<code>\xintLengthUpTo:f:csv</code> .....	47
<code>\xintCSVKeep</code> .....	54	<code>\xintlengthupto:f:csv</code> .....	47
<code>\xintCSVLastItem</code> .....	54	<code>\xintListWithSep</code> .....	25
<code>\xintCSVLLength</code> .....	54	<code>\xintlistwithsep</code> .....	25
<code>\xintCSVNthEltPy</code> .....	54	<code>\xintListWithSepNoExpand</code> .....	25
<code>\xintCSVReverse</code> .....	54	<code>\xintlistwithsepnoexpand</code> .....	25
<code>\xintCSVtoList</code> .....	23	<code>\xintloop</code> .....	35
<code>\xintcsvtolist</code> .....	23, 38, 39, 42, 43	<code>\xintloop_again</code> .....	35
<code>\xintCSVtoListNoExpand</code> .....	24	<code>\xintloopskiptonext</code> .....	35
<code>\xintcsvtolistnoexpand</code> .....	24	<code>\xintNthElt</code> .....	26
<code>\xintCSVtoListNonStripped</code> .....	24	<code>\xintnthelt</code> .....	26
<code>\xintcsvtolistnonstripped</code> .....	23, 24	<code>\xintNthEltNoExpand</code> .....	26
<code>\xintCSVtoListNonStrippedNoExpand</code> .....	24	<code>\xintntheltnoexpand</code> .....	26
<code>\xintcsvtolistnonstrippednoexpand</code> .....	24	<code>\xintNthEltPy:f:csv</code> .....	52, 54
<code>\xintCSVTrim</code> .....	54	<code>\xintntheltpy:f:csv</code> .....	52
<code>\xintDigitsOf</code> .....	45	<code>\xintodef</code> .....	20
<code>\xintdimensions</code> .....	40	<code>\xintoodef</code> .....	20
<code>\xintegers</code> .....	40	<code>\xintouteriloopindex</code> .....	36
<code>\xintExpandArgs</code> .....	46	<code>\xintrationals</code> .....	40
<code>\xintfdef</code> .....	20	<code>\xintrawwithzeros</code> .....	40
<code>\xintFirstItem:f:csv</code> .....	53, 54	<code>\xintRelaxArray</code> .....	44
<code>\xintfirstitem:f:csv</code> .....	53	<code>\xintReverse:f:csv</code> .....	53, 54
<code>\xintFor</code> .....	38	<code>\xintreverse:f:csv</code> .....	53
<code>\xintForfour</code> .....	42	<code>\xintRevWithBraces</code> .....	20
<code>\xintForpair</code> .....	42	<code>\xintrevwithbraces</code> .....	20, 21
<code>\xintForthree</code> .....	42	<code>\xintRevWithBracesNoExpand</code> .....	21
<code>\xintgfdef</code> .....	20	<code>\xintrevwithbracesnoexpand</code> .....	21
<code>\xintgodef</code> .....	20	<code>\xintSeq</code> .....	33
<code>\xintgoodef</code> .....	20	<code>\xintseq</code> .....	33
<code>\xintifEq</code> .....	40	<code>\XINTsetupcatcodes</code> .....	20
<code>\xintifForFirst</code> .....	38, 39, 41--43	<code>\xintTrim</code> .....	30
<code>\xintifForLast</code> .....	38, 39, 41--43	<code>\xinttrim</code> .....	30
<code>\xintiiadd</code> .....	41	<code>\xintTrim:f:csv</code> .....	50, 54
<code>\xintiimul</code> .....	40, 41	<code>\xinttrim:f:csv</code> .....	50
<code>\xintiloop</code> .....	35, 44	<code>\xintTrimNoExpand</code> .....	30
<code>\xintiloop_a</code> .....	35	<code>\xinttrimnoexpand</code> .....	30
<code>\xintiloop_again</code> .....	35, 36	<code>\xintTrimUnbraced</code> .....	31

**Y**

**Z**

## 12.3 Index of **xintcore**

<b>Symbols</b>	
\@backslashchar .....	112
<b>A</b>	
\aftergroup .....	56
<b>C</b>	
\catcode .....	55, 57, 58
\csname .....	56--59, 92, 111, 112
<b>E</b>	
\endcsname .....	56--59, 92, 112
\endinput .....	56
\endlinechar .....	55
<b>I</b>	
\if . 63, 64, 75, 87, 88, 91, 93, 98, 100, 102--104	
\ifcsname .....	57, 58
\ifdefined .....	58, 59
\immediate .....	56
<b>K</b>	
\krof .....	60--65, 72, 75, 76, 78, 81--83, 87, 99, 102--105, 107, 110
<b>N</b>	
\numexpr .....	56, 59, 60, 62--71, 73--76, 78--87, 89--101, 106, 108--112
<b>P</b>	
\ProvidesPackage .....	56
<b>R</b>	
\R .....	66, 68, 70--73, 76, 78, 79, 82--84, 88--90, 93--96, 98, 105, 106, 108, 110
\romannumeral .....	57, 59--65, 71--73, 75--79, 82--85, 87--96, 99, 102--108, 110
<b>U</b>	
\unless .....	57, 64, 86
<b>W</b>	
\W .....	68--85, 88--90, 93--96, 98, 99, 105, 106, 108--111
\w .....	56
\write .....	56
<b>X</b>	
\X .....	73, 78, 90
\x .....	56
\xint: .....	59--61, 69--109
\XINT_abs .....	61
\XINT_add_A .....	73
\XINT_add_a .....	73--75, 85
\XINT_add_aa .....	73
\XINT_add_aa_small .....	73
\XINT_add_b .....	74
\XINT_add_bi .....	74
\XINT_add_c .....	74
\XINT_add_checklengths .....	73
\XINT_add_d .....	74
\XINT_add_di .....	74
\XINT_add_e .....	74
\XINT_add_exchange .....	73
\XINT_add_f .....	74
\XINT_add_fi .....	74
\XINT_add_firstiszero .....	72
\XINT_add_fork .....	72
\XINT_add_g .....	74
\XINT_add_h .....	74
\XINT_add_hi .....	74, 75
\XINT_add_i .....	75
\XINT_add_k .....	74, 75
\XINT_add_ke .....	75
\XINT_add_kf .....	75
\XINT_add_l .....	75
\XINT_add_lf .....	75
\XINT_add_m .....	75
\XINT_add_minusminus .....	72
\XINT_add_minusplus .....	72
\XINT_add_n .....	75
\XINT_add_nfork .....	72
\XINT_add_o .....	75
\XINT_add_out .....	73
\XINT_add_plusminus .....	72
\XINT_add_plusplus .....	72, 73
\XINT_add_pp_a .....	72, 73, 78
\XINT_add_pp_b .....	72, 73
\XINT_add_secondiszero .....	72
\xint_afterfi .....	100, 107
\xint_Bye .....	63--65, 89, 103
\xint_bye ... ..	62--65, 67, 82, 89, 91, 94, 99, 103
\xint_c_ .....	60, 62, 66, 69, 73, 76, 78, 84, 90, 93, 99, 100, 105
\xint_c_i .....	60, 63, 64, 67, 70--73, 75, 76, 78--81, 83, 84, 89--91, 93, 94, 96, 98--101, 105, 106, 108, 109, 111, 112
\xint_c_ii .....	62--64, 72--76, 78, 83--85, 89, 90, 105, 108, 109, 111, 112
\xint_c_iii .....	76, 90, 111, 112
\xint_c_iv .....	73, 76, 78, 84, 90, 105, 112
\xint_c_ix .....	93
\xint_c_mone .....	58, 60
\xint_c_v .....	63, 64, 76, 89, 90
\xint_c_vi .....	72, 73, 76, 78, 84, 90, 92, 105
\xint_c_vii .....	76, 90, 92
\xint_c_viii .....	69, 72, 73, 78, 84, 90, 105
\xint_c_x .....	63--65, 72, 73, 78, 84, 89, 90, 103, 105
\xint_c_x^iv .....	110
\xint_c_x^ix .....	59, 85, 86, 97

\xint_c_x^viii .....	58, 86, 87, 89, 97, 100, 101, 111, 112
\xint_c_x^viii_mone .....	59, 101, 102
\xint_c_xi_e_viii_mone .....	59, 83, 100, 101
\xint_c_xii .....	72, 73, 78, 84, 90, 105
\xint_c_xii_e_viii .....	59
\xint_c_xiv .....	71
\xint_c_xxii .....	84
\XINT_Clamped.handler .....	58
\XINT_cmp_a .....	76, 77
\XINT_cmp_checklengths .....	76
\XINT_cmp_distinctlengths .....	76
\XINT_cmp_equal .....	77
\XINT_cmp_firstiszero .....	75, 76
\XINT_cmp_gt .....	77
\XINT_cmp_lt .....	77
\XINT_cmp_minusminus .....	75, 76
\XINT_cmp_minusplus .....	75, 76
\XINT_cmp_nfork .....	75
\XINT_cmp_plusminus .....	75, 76
\XINT_cmp_plusplus .....	75, 76
\XINT_cmp_pp .....	76
\XINT_cmp_secondiszero .....	75, 76
\XINT_cntSgn .....	60, 107
\XINT_ConversionSyntax-signal .....	57
\XINT_ConversionSyntax.handler .....	58
\XINT_cuz .....	66, 106
\XINT_cuz_byviii .....	66
\XINT_cuz_byviii_done .....	66
\XINT_cuz_byviii_e .....	66
\XINT_cuz_byviii_z .....	66
\XINT_cuz_cleantotend .....	66
\XINT_cuz_hitend .....	66
\XINT_cuz_loop .....	66, 68
\XINT_cuz_small .....	59, 71, 73, 82, 85
\XINT_dbl .....	62
\XINT_dbl_a .....	62
\XINT_dbl_e .....	62, 63
\XINT_dbl_fork .....	62
\XINT_dbl_neg .....	62
\XINT_dec .....	63, 64
\XINT_dec_a .....	64
\XINT_dec_bye .....	63, 64
\XINT_dec_e .....	64
\XINT_dec_fork .....	64
\XINT_dec_neg .....	64
\XINT_div_BisOne .....	88
\XINT_div_BisTwo .....	88
\XINT_div_BisTwo_a .....	89
\XINT_div_cleanR .....	91, 92
\XINT_div_dosmalldiv .....	89, 90
\XINT_div_dosmallsmall .....	89
\XINT_div_exittofinish .....	94
\XINT_div_finish .....	91
\XINT_div_finish_a .....	91
\XINT_div_finish_b .....	91
\XINT_div_finish_bRpos .....	91
\XINT_div_finish_bRzero .....	91
\XINT_div_I_a .....	92, 97
\XINT_div_I_b .....	92, 93
\XINT_div_I_c .....	92, 93
\XINT_div_I_czero .....	92, 93
\XINT_div_I_da .....	93
\XINT_div_I_db .....	93
\XINT_div_I_dc .....	93
\XINT_div_I_dd .....	93
\XINT_div_I_de .....	93, 94
\XINT_div_I_dN .....	93
\XINT_div_I_dP .....	93, 94
\XINT_div_I_dz .....	93
\XINT_div_I_g .....	93, 94
\XINT_div_I_h .....	94
\XINT_div_II_b .....	94
\XINT_div_II_c .....	95
\XINT_div_II_d .....	95
\XINT_div_II_e .....	95
\XINT_div_II_f .....	95, 96
\XINT_div_II_fa .....	96
\XINT_div_II_g .....	96
\XINT_div_II_h .....	96
\XINT_div_II_k .....	95, 96
\XINT_div_II_l .....	96
\XINT_div_II_m .....	96, 97
\XINT_div_II_skipc .....	95
\XINT_div_II_skipf .....	95, 96
\XINT_div_mini .....	95, 101
\XINT_div_mini_a .....	101
\XINT_div_mini_b .....	101
\XINT_div_mini_c .....	101
\XINT_div_mini_d .....	101, 102
\XINT_div_mini_w .....	101
\XINT_div_minimulwc_a .....	97
\XINT_div_minimulwc_b .....	97
\XINT_div_minimulwc_c .....	97
\XINT_div_prepare .....	88, 103, 104
\XINT_div_prepare_a .....	88
\XINT_div_prepare_b .....	88, 90
\XINT_div_prepare_c .....	90
\XINT_div_prepare_d .....	90
\XINT_div_prepare_e .....	90
\XINT_div_prepare_f .....	90
\XINT_div_prepare_g .....	90
\XINT_div_prepare_h .....	90, 91
\XINT_div_prepare_small .....	88
\XINT_div_small_a .....	88, 89
\XINT_div_small_b .....	89
\XINT_div_small_ba .....	89
\XINT_div_smallmul_a .....	95--97
\XINT_div_smallmul_e .....	97
\XINT_div_smallsmall .....	89
\XINT_div_smallsmallend .....	89
\XINT_div_start_a .....	91
\XINT_div_start_b .....	91
\XINT_div_start_c .....	91, 92

\XINT_div_start_c	92	\XINT_dsr_fork	64
\XINT_div_start_c_i	92	\XINT_dsr_neg	65
\XINT_div_start_c_ii	92	\XINT_dsrr	65, 103
\XINT_div_start_c_iii	92	\XINT_dsrr_a	65
\XINT_div_start_c_iv	92	\XINT_dsrr_b	65
\XINT_div_start_c_v	92	\XINT_dsrr_e	65
\XINT_div_start_c_vi	92	\XINT_dsrr_fork	65
\XINT_div_start_ca	92	\XINT_dsrr_neg	65
\XINT_div_start_cb	92	\xint_exchangetwo_keepbraces	88, 92, 105
\XINT_div_sub	93--96, 98	\XINT_expandableerror	58
\XINT_div_sub_a	98	\XINT_fac_bigloop_a	110, 111
\XINT_div_sub_b	98	\XINT_fac_bigloop_b	111
\XINT_div_sub_bi	98, 99	\XINT_fac_bigloop_exit	111
\XINT_div_sub_c	98, 99	\XINT_fac_bigloop_loop	111
\XINT_div_sub_clean	98	\XINT_fac_bigloop_mul	111
\XINT_div_sub_d	98	\XINT_fac_checksize	110
\XINT_div_sub_di	98, 99	\XINT_fac_fork	110
\XINT_div_sub_e	98, 99	\XINT_fac_loop_exit	111, 112
\XINT_div_sub_f	98	\XINT_fac_medloop_a	110, 111
\XINT_div_sub_fi	98, 99	\XINT_fac_medloop_b	111
\XINT_div_sub_g	98, 99	\XINT_fac_medloop_loop	111
\XINT_div_sub_h	98	\XINT_fac_medloop_mul	111
\XINT_div_sub_hi	98, 99	\XINT_fac_neg	110
\XINT_div_sub_i	98, 99	\XINT_fac_smallloop_a	110, 111
\XINT_div_sub_l	99	\XINT_fac_smallloop_loop	112
\XINT_div_sub_neg	98	\XINT_fac_smallloop_mul	112
\XINT_div_sub_r	99	\XINT_fac_toobig	110, 111
\XINT_div_unsepQ	67, 91	\XINT_fac_zero	110
\XINT_div_unsepQ_delim	67, 94	\XINT_FDg	61
\XINT_div_unsepQ_one	67, 68	\XINT_fdg	61
\XINT_div_unsepQ_x	67	\xint_firstofone	58
\XINT_div_unsepQ_y	67, 68	\xint_firstoftwo	58, 76, 77, 103
\XINT_div_unsepR	68, 94	\xint_gob_til_dot	90
\XINT_div_unsepR_x	68	\xint_gob_til_eightzeroes	66, 67, 86, 95
\XINT_div_verysmallisone	97	\xint_gob_til_exclam	96
\XINT_div_verysmallmul	93, 94, 97	\xint_gob_til_one	68, 95, 97
\XINT_div_verysmallmul_a	97	\xint_gob_til_R	66, 68, 70, 71, 88, 89
\XINT_div_verysmallmul_b	97	\xint_gob_til_sc	71,
\XINT_div_verysmallmul_e	97		74, 75, 77, 80--82, 85, 86, 97, 98, 100, 108
\XINT_div_xmini	95, 96	\xint_gob_til_xint:	60, 107
\XINT_div_xmini_a	95	\xint_gob_til_Z	67
\XINT_div_xmini_b	95	\xint_gob_til_zero	64, 86, 92, 95, 100, 101
\XINT_div_xmini_c	95	\xint_gobble_i	57, 58
\XINT_div_zeroQ	91	\xint_gobble_iv	112
\XINT_div_zeroQ_end	91	\XINT_half	63, 89
\XINT_DivisionByZero.handler	58	\XINT_half_a	63
\XINT_DivisionImpossible-signal	57	\XINT_half_b	63
\XINT_DivisionImpossible.handler	58	\XINT_half_e	63
\XINT_DivisionUndefined-signal	57	\XINT_half_fork	63
\XINT_DivisionUndefined.handler	58	\XINT_half_neg	63
\xint_dothis	57, 58, 64, 87, 102--104, 110	\XINT_iadd	72
\XINT_dsl	64	\XINT_icmp	75
\xint_dsl_zero	64	\XINT_idivround	102
\XINT_dsr	65	\XINT_ifFlagRaised	58, 112
\XINT_dsr_a	65	\XINT_iiadd	72
\XINT_dsr_b	65	\XINT_iicmp	75
\XINT_dsr_e	65	\XINT_iidivision	87

\XINT_iidivision_a .....	87	\XINT_ldg_fork .....	62
\XINT_iidivision_aiszero .....	87	\XINT_minimul_a .....	86, 100, 101, 106
\XINT_iidivision_aneg .....	87, 88	\XINT_minimul_b .....	86
\XINT_iidivision_aneg_b .....	88	\XINT_minimul_c .....	86, 87
\XINT_iidivision_aneg_end .....	88	\XINT_minimulwc_a .....	85, 86
\XINT_iidivision_aneg_rpos .....	88	\XINT_minimulwc_b .....	85
\XINT_iidivision_aneg_rzero .....	88	\XINT_minimulwc_c .....	85, 86
\XINT_iidivision_apos .....	87, 88	\XINT_mul_a .....	85
\XINT_iidivision_bneg .....	87	\XINT_mul_b .....	85
\XINT_iidivision_bpos .....	87	\XINT_mul_checklengths .....	84
\XINT_iidivision_divbyzero .....	87	\XINT_mul_e .....	85
\XINT_iidivmod .....	104	\XINT_mul_exchange .....	84
\XINT_iidivmod_a .....	104	\XINT_mul_fork .....	83
\XINT_iidivmod_aiszero .....	104	\XINT_mul_loop .....	85, 106, 108
\XINT_iidivmod_bneg .....	104, 105	\XINT_mul_minusminus .....	83
\XINT_iidivmod_bneg_finish .....	105	\XINT_mul_minusplus .....	83
\XINT_iidivmod_bpos .....	104, 105	\XINT_mul_nfork .....	83
\XINT_iidivmod_divbyzero .....	104	\XINT_mul_oneisone .....	84
\XINT_iidivround .....	102	\XINT_mul_out .....	84, 85, 106, 108--111
\XINT_iidivround_a .....	102	\XINT_mul_plusminus .....	83
\XINT_iidivround_aiszero .....	102--104	\XINT_mul_plusplus .....	83
\XINT_iidivround_bneg .....	102	\XINT_mul_pre_b .....	83, 84
\XINT_iidivround_bpos .....	102	\XINT_mul_smallbyfirst .....	84
\XINT_iidivround_divbyzero .....	102--104	\XINT_mul_smallbysecond .....	84
\XINT_iidivround_pos .....	102, 103	\XINT_mul_start .....	84, 85
\XINT_iidivtrunc .....	103	\XINT_mul_verysmall .....	84
\XINT_iidivtrunc_a .....	103	\XINT_mul_zero .....	83
\XINT_iidivtrunc_aiszero .....	103	\XINT_num .....	59
\XINT_iidivtrunc_bneg .....	103	\XINT_num_cleanup .....	59, 60, 82
\XINT_iidivtrunc_bpos .....	103	\XINT_num_end .....	60
\XINT_iidivtrunc_divbyzero .....	103	\XINT_num_loop .....	59, 60, 82
\XINT_iidivtrunc_pos .....	103	\XINT_Opp .....	61
\XINT_iimodtrunc .....	103	\XINT_opp .....	61, 72, 76, 78, 79, 87, 107
\XINT_iimodtrunc_a .....	103, 104	\xint_orthat .....	57, 58, 64, 87, 102--104, 110
\XINT_iimodtrunc_aiszero .....	104	\XINT_Overflow.handler .....	58
\XINT_iimodtrunc_bneg .....	104	\xint_pow .....	106
\XINT_iimodtrunc_bpos .....	104	\XINT_pow_AatleastTwo .....	107
\XINT_iimodtrunc_divbyzero .....	104	\XINT_pow_AisOne .....	107
\XINT_iimodtrunc_pos .....	104	\XINT_pow_AisZero .....	107
\XINT_iimul .....	83	\XINT_pow_Aneg .....	107
\XINT_iisub .....	77	\XINT_pow_Apos .....	107
\XINT_inc .....	63, 64	\XINT_pow_Apos_a .....	107
\XINT_inc_a .....	63	\XINT_pow_Apos_short .....	107
\XINT_inc_e .....	63, 65	\XINT_pow_BisNegative .....	107
\XINT_inc_fork .....	63	\XINT_pow_BisZero .....	107, 108
\XINT_inc_neg .....	63	\XINT_pow_I_exit .....	108
\XINT_Inexact.handler .....	58	\XINT_pow_I_in .....	107, 108
\XINT_InvalidContext-signal .....	57	\XINT_pow_I_loop .....	108
\XINT_InvalidContext.handler .....	58	\XINT_pow_I_squareit .....	108
\XINT_InvalidOperation.handler .....	58	\XINT_pow_II_even .....	109
\XINT_isub .....	78	\XINT_pow_II_exit .....	109
\XINT_ldg .....	62	\XINT_pow_II_in .....	108
\XINT_ldg_a .....	62	\XINT_pow_II_loop .....	108, 109
\XINT_ldg_b .....	62	\XINT_pow_II_odda .....	109
\XINT_ldg_c .....	62	\XINT_pow_II_oddb .....	109
\XINT_ldg_cbye .....	62	\XINT_pow_mul_small .....	108
\XINT_ldg_d .....	62	\XINT_pow_mulbutcheckifsmall .....	108, 109

\XINT_providespackage .....	56	\XINT_smallmul_f .....	86
\XINT_RaiseFlag .....	58	\XINT_smallunrevbyviii .....	71, 73
\XINT_resetFlag .....	58	\XINT_sqr .....	105
\XINT_resetFlags .....	58	\XINT_sqr_a .....	105, 106
\XINT_restorecatcodes_endinput .....	112	\XINT_sqr_small .....	106
\XINT_rev_nounsep .....	71, 93--96, 98	\XINT_sqr_small_out .....	106
\XINT_rev_nounsep_done .....	71	\XINT_sqr_start .....	106
\XINT_rev_nounsep_end .....	71	\XINT_sqr_verysmall .....	106
\XINT_Rounded.handler .....	58	\xint_stop_atfirstofone .....	58
\XINT_rsepbyviii .....	69, 70	\xint_stop_atfirstofthree .....	58
\XINT_rsepbyviii_b .....	69	\xint_stop_atfirstoftwo .....	102, 103, 105
\XINT_rsepbyviii_end_A .....		\xint_stop_atsecondoftwo .....	102, 104, 105
..... 70, 72, 73, 78, 83, 84, 90, 105, 108		\XINT_sub_a .....	79, 80
\XINT_rsepbyviii_end_B .....		\XINT_sub_aa .....	79
..... 70, 72, 73, 78, 83, 84, 90, 105, 108		\XINT_sub_b .....	80
\XINT_sdiv_out .....	90, 99	\XINT_sub_bi .....	80
\xint_secondoftwo .....	58, 76, 77	\XINT_sub_c .....	80
\XINT_sepandrev .....	70, 108	\XINT_sub_checklengths .....	78, 79
\XINT_sepandrev_a .....	70	\XINT_sub_comp_clean .....	83
\XINT_sepandrev_andcount .....		\XINT_sub_comp_finish .....	82
..... 70, 72, 73, 78, 83, 84, 90, 105		\XINT_sub_comp_loop .....	82, 83
\XINT_sepandrev_andcount_a .....	70	\XINT_sub_d .....	80
\XINT_sepandrev_andcount_b .....	70, 71	\XINT_sub_di .....	80, 81
\XINT_sepandrev_andcount_done .....	71	\XINT_sub_e .....	80, 81
\XINT_sepandrev_andcount_end .....	70, 71	\XINT_sub_exchange .....	79
\XINT_sepandrev_b .....	70	\XINT_sub_f .....	80
\XINT_sepandrev_done .....	70	\XINT_sub_fi .....	80, 81
\XINT_sepandrev_end .....	70	\XINT_sub_firstiszero .....	78
\XINT_sepbyviii .....	69	\XINT_sub_fix_cuz .....	82
\XINT_sepbyviii_andcount .....	69, 76, 90	\XINT_sub_fix_neg .....	82
\XINT_sepbyviii_andcount_a .....	69	\XINT_sub_fix_none .....	81, 82
\XINT_sepbyviii_andcount_b .....	69	\XINT_sub_g .....	80, 81
\XINT_sepbyviii_andcount_end .....	69	\XINT_sub_h .....	80
\XINT_sepbyviii_end .....	69, 76, 90	\XINT_sub_hi .....	80, 81
\XINT_sepbyviii_Z .....	68, 89	\XINT_sub_i .....	80, 81
\XINT_sepbyviii_Z_end .....	68, 89	\XINT_sub_k .....	81
\XINT_Sgn .....	60, 88	\XINT_sub_l .....	81, 82
\XINT_sgn .....	60	\XINT_sub_l_carry .....	81
\XINT_signalcondition .....		\XINT_sub_l_Ia .....	81
..... 57, 87, 102, 104, 107, 110, 111		\XINT_sub_l_Ib .....	81, 82
\XINT_signalcondition_a .....	57	\XINT_sub_l_Ic .....	81
\XINT_smallldiv_c .....	100	\XINT_sub_l_Ica .....	81
\XINT_smallldiv_d .....	100, 101	\XINT_sub_l_Id .....	81, 82
\XINT_smallldiv_e .....	100	\XINT_sub_l_Id_a .....	82
\XINT_smallldiv_end .....	100	\XINT_sub_l_Id_b .....	82
\XINT_smallldiv_f .....	100	\XINT_sub_l_Ida .....	82
\XINT_smallldiv_fz .....	100	\XINT_sub_minusminus .....	78
\XINT_smallldiv_g .....	100	\XINT_sub_minusplus .....	78
\XINT_smallldiv_h .....	100	\XINT_sub_mm_a .....	72, 78, 88
\XINT_smallldiv_i .....	100	\XINT_sub_mm_b .....	78
\XINT_smallldiv_j .....	100, 101	\XINT_sub_nfork .....	77, 78
\XINT_smallldiv_k .....	101	\XINT_sub_out .....	79
\XINT_smallldivx_a .....	90, 99	\XINT_sub_p .....	81, 82
\XINT_smallldivx_b .....	99	\XINT_sub_plusminus .....	78
\XINT_smallmul .....	84--86, 108, 111, 112	\XINT_sub_plusplus .....	78
\XINT_smallmul_a .....	86	\XINT_sub_secondiszero .....	78
\XINT_smallmul_e .....	86	\XINT_Subnormal.handler .....	58





## 12.4 Index of **xint**

<b>A</b>	
\aftergroup .....	114
<b>C</b>	
\catcode .....	113
\csname ...	113, 114, 117, 118, 135, 148, 150, 151
<b>E</b>	
\endcsname .....	113, 114, 116--118, 120, 135, 148, 150, 151
\endinput .....	114
\endlinechar .....	113
<b>I</b>	
\if .....	122, 123, 125--131, 134, 138--140, 146
\ifdefined .....	114, 153
\iftoggle .....	148
\immediate .....	114
<b>K</b>	
\krof .....	115--122, 129, 130, 133, 149, 152
<b>N</b>	
\numexpr ..	114--120, 133--138, 140--148, 150--152
<b>P</b>	
\ProvidesPackage .....	114
<b>R</b>	
\R .....	115, 116, 143, 146
\romannumeral .....	115--133, 135--139, 141--146, 148--153
<b>U</b>	
\unless .....	114, 123, 130, 131, 137, 146
<b>W</b>	
\W .....	115, 116, 143, 146
\w .....	114
\write .....	114
<b>X</b>	
\x .....	113, 114
\xint: .....	115, 118, 122, 128--133, 139, 143, 148--153
\XINT_add_fork .....	132
\xint_afterfi .....	123, 124, 128, 129, 146
\XINT_andof_a .....	128
\XINT_andof_b .....	128
\XINT_andof_c .....	128
\XINT_andof_e .....	128
\XINT_andof_no .....	128
\XINT_binom_a .....	140
\XINT_binom_b .....	140
\XINT_binom_bigloop .....	140, 142
\XINT_binom_bigloop_a .....	142
\XINT_binom_bigloop_b .....	142
\XINT_binom_div .....	141--144
\XINT_binom_end_ .....	141, 142, 144, 145
\XINT_binom_end_i .....	141, 142, 144, 145
\XINT_binom_end_ii .....	141, 144, 145
\XINT_binom_end_iii .....	141, 144
\XINT_binom_finish .....	144
\XINT_binom_fork .....	140
\XINT_binom_medloop .....	140--142
\XINT_binom_medloop_a .....	141
\XINT_binom_medloop_b .....	141, 142
\XINT_binom_mul .....	141--144
\XINT_binom_pre .....	140
\XINT_binom_smallloop .....	140, 141
\XINT_binom_smallloop_a .....	141, 143
\XINT_binom_smallloop_b .....	141
\XINT_binom_vbigloop .....	140, 142
\XINT_binom_vbigloop_a .....	142
\XINT_binom_vsmallend_ .....	143, 145
\XINT_binom_vsmallend_b .....	145
\XINT_binom_vsmallend_i .....	143, 145
\XINT_binom_vsmallend_ib .....	145
\XINT_binom_vsmallend_ii .....	143, 144
\XINT_binom_vsmallend_iib .....	145
\XINT_binom_vsmallend_iii .....	143, 144
\XINT_binom_vsmallend_iiib .....	144
\XINT_binom_vsmallfinish .....	144, 145
\XINT_binom_vsmallloop .....	140, 143
\XINT_binom_vsmallloop_a .....	143
\XINT_binom_vsmallloop_b .....	143
\XINT_binom_vsmallmuldiv .....	143--145
\xint_bye .....	115, 117, 118, 121, 137--139, 144
\xint_c_ .....	115, 118, 126, 152
\xint_c_i .....	115, 118, 134--136, 140--148, 152
\xint_c_ii .....	115, 118, 133--139, 141--145, 147, 148
\xint_c_ii^vii .....	150
\xint_c_ii^xiv .....	150
\xint_c_ii^xxi .....	150
\xint_c_iii .....	115, 118, 141, 143, 144, 147, 148, 150, 151
\xint_c_iv .....	115, 118, 136, 141, 143, 146
\xint_c_ix .....	135
\xint_c_nine_x^viii .....	114, 150
\xint_c_v .....	115, 118
\xint_c_vi .....	115, 118, 133
\xint_c_vii .....	115, 118
\xint_c_viii .....	115, 117, 118, 136, 137, 150--153
\xint_c_x .....	133, 152
\xint_c_x^viii .....	136, 143, 146--148, 150
\xint_c_x^viii_mone .....	140, 146
\xint_c_xvi .....	137
\XINT_cmp_plusplus .....	122
\XINT_dbl .....	137--139
\XINT_dec .....	139
\XINT_dec_bye .....	139

\XINT_div_prepare .....	149	\XINT_iigeq .....	122
\xint_dothis 123, 133--135, 137, 138, 140, 146, 152		\XINT_iilcm .....	149
\XINT_dsh_fork .....	119	\xint_iimax .....	129
\XINT_dsh_xisPos .....	119	\XINT_iimaxof_a .....	131
\XINT_dsh_xiszero .....	119	\XINT_iimaxof_b .....	131
\XINT_dshr_fork .....	119	\XINT_iimaxof_c .....	131
\XINT_dshr_xpositive .....	119	\XINT_iimaxof_d .....	131
\XINT_dshr_xzeroorneg .....	119	\XINT_iimaxof_e .....	131
\XINT_dsx_addzeros .....	120, 138	\xint_iimin .....	130
\XINT_dsx_addzerosnofuss .....	120, 138	\XINT_iiminof_a .....	131
\XINT_dsx_AisNeg .....	121	\XINT_iiminof_b .....	131
\XINT_dsx_AisNeg_checkiffirstempty .....	121	\XINT_iiminof_c .....	131
\XINT_dsx_AisNeg_finish_notzero .....	121	\XINT_iiminof_d .....	131
\XINT_dsx_AisNeg_finish_zero .....	121	\XINT_iiminof_e .....	131
\XINT_dsx_AisPos .....	121	\XINT_is_One .....	123
\XINT_dsx_AisPos_finish .....	121	\XINT_isOne .....	123
\XINT_dsx_AisZero .....	120, 121	\XINT_ison .....	123
\XINT_dsx_append .....	116, 120	\XINT_lcm_end .....	149
\XINT_dsx_end .....	121	\XINT_lcm_fork .....	149
\XINT_dsx_fork .....	120	\XINT_lcm_iszero .....	149
\XINT_dsx_xisNeg_Azero .....	120	\XINT_lcm_notzero .....	149
\XINT_dsx_xisNeg_checkA .....	119, 120	\XINT_len_fork .....	115
\XINT_dsx_xisPos .....	119, 120	\XINT_length_loop .....	115, 118
\XINT_dsx_xisZero .....	120	\XINT_max_fork .....	129
\XINT_eightrandomdigits .....	150, 151	\XINT_max_minusminus .....	129, 130
\XINT_expandableerror .....	152, 153	\XINT_max_minusplus .....	129, 130
\xint_firstofthree .....	114	\XINT_max_plusminus .....	129, 130
\xint_firstoftwo .....	128	\XINT_max_plusplus .....	129, 130
\XINT_gcd_AisZero .....	149	\XINT_max_pluszero .....	129, 130
\XINT_gcd_Aiszero .....	149	\XINT_max_zeroplus .....	129, 130
\XINT_gcd_BisZero .....	149	\XINT_max_zerozero .....	129, 130
\XINT_gcd_Biszero .....	149	\XINT_microrevsep .....	116
\XINT_gcd_CheckRem .....	149	\XINT_microrevsep_end .....	115, 116
\XINT_gcd_end .....	149	\XINT_min_fork .....	130
\XINT_gcd_fork .....	148, 149	\XINT_min_minusminus .....	130, 131
\XINT_gcd_loop .....	149	\XINT_min_minusplus .....	130, 131
\XINT_geq .....	122	\XINT_min_plusminus .....	130, 131
\XINT_geq_finish .....	122	\XINT_min_plusplus .....	130, 131
\XINT_geq_firstiszero .....	122	\XINT_min_pluszero .....	130, 131
\XINT_geq_fork .....	122	\XINT_min_zeroplus .....	130, 131
\XINT_geq_minusminus .....	122	\XINT_min_zerozero .....	130, 131
\XINT_geq_minusplus .....	122	\XINT_mul_fork .....	132
\XINT_geq_no .....	122	\XINT_mul_out .....	148
\XINT_geq_plusminus .....	122	\XINT_num .....	121
\XINT_geq_plusplus .....	122, 130, 131	\XINT_orof_a .....	128
\XINT_geq_secondiszero .....	122	\XINT_orof_b .....	128
\XINT_geq_yes .....	122	\XINT_orof_c .....	128
\xint_gob_til_dot .....	121	\XINT_orof_e .....	128
\xint_gob_til_R .....	116	\XINT_orof_yes .....	128
\xint_gob_til_xint: .....	128, 129, 131, 132	\xint_orthat 123, 133--135, 137, 138, 140, 146, 152	
\xint_gob_til_Z .....	116	\XINT_pfac_a .....	146
\xint_gob_til_zero .....	120, 149	\XINT_pfac_b .....	146
\xint_gobble_i .....	150	\XINT_pfac_bigloop .....	146, 147
\XINT_ii_e_a .....	116	\XINT_pfac_bigloop_a .....	147
\XINT_ii_e_fork .....	116	\XINT_pfac_bigloop_b .....	147
\XINT_ii_e_neg .....	116	\XINT_pfac_end_ .....	146--148
\XINT_iigcd .....	148	\XINT_pfac_end_i .....	146--148

\XINT_pfac_end_ii .....	146--148	\XINT_split_fromleft_end_a .....	117
\XINT_pfac_end_iii .....	146, 148	\XINT_split_fromleft_end_b .....	117, 118
\XINT_pfac_fork .....	145, 146	\XINT_split_fromright .....	117, 118
\XINT_pfac_medloop .....	146, 147	\XINT_split_fromright_a .....	118
\XINT_pfac_medloop_a .....	147	\XINT_split_fromright_Lempty .....	118
\XINT_pfac_medloop_b .....	147	\XINT_split_xfork .....	117, 118, 121
\XINT_pfac_neg .....	146	\XINT_split_zerosplit .....	117
\XINT_pfac_one .....	146	\XINT_splitl_finish .....	118
\XINT_pfac_outofrange .....	146	\XINT_splitr_finish .....	118
\XINT_pfac_smallloop .....	146, 147	\XINT_sqrt .....	133
\XINT_pfac_smallloop_a .....	146	\XINT_sqrt_a .....	133
\XINT_pfac_smallloop_b .....	146, 147	\XINT_sqrt_bE .....	133
\XINT_pfac_vbigloop .....	146--148	\XINT_sqrt_big_a .....	133
\XINT_pfac_vbigloop_a .....	148	\XINT_sqrt_big_d .....	133, 135
\XINT_pfac_zero .....	146	\XINT_sqrt_big_eE .....	135
\XINT_prdexpr .....	132	\XINT_sqrt_big_eE_a .....	135
\XINT_prod_finished .....	132	\XINT_sqrt_big_end .....	139
\XINT_prod_loop_a .....	132	\XINT_sqrt_big_e0 .....	135
\XINT_prod_loop_b .....	132	\XINT_sqrt_big_e0_a .....	135
\XINT_prod_loop_c .....	132	\XINT_sqrt_big_f .....	135, 136
\XINT_providespackage .....	114	\XINT_sqrt_big_fa .....	136
\XINT_randomdigits .....	150	\XINT_sqrt_big_ga .....	136
\XINT_randomdigits_a .....	150	\XINT_sqrt_big_gb .....	136
\XINT_randrange .....	151, 153	\XINT_sqrt_big_gc .....	136
\XINT_randrange_A .....	152, 153	\XINT_sqrt_big_gd .....	136, 137
\XINT_randrange_a .....	152	\XINT_sqrt_big_ge .....	137
\XINT_randrange_again .....	153	\XINT_sqrt_big_gf .....	137
\XINT_randrange_B .....	153	\XINT_sqrt_big_gg .....	137
\XINT_randrange_b .....	152	\XINT_sqrt_big_gi .....	137
\XINT_randrange_c .....	152, 153	\XINT_sqrt_big_gj .....	137
\XINT_randrange_d .....	152	\XINT_sqrt_big_gk .....	137
\XINT_randrange_E .....	153	\XINT_sqrt_big_gl .....	137, 138
\XINT_randrange_e .....	152	\XINT_sqrt_big_gloop .....	137, 138
\XINT_randrange_err:empty .....	151, 152	\XINT_sqrt_big_gm .....	138
\XINT_randrange_Z .....	152	\XINT_sqrt_big_gn .....	138
\XINT_randrangeAtoB_a .....	151	\XINT_sqrt_big_ka .....	136--138
\XINT_rdg .....	150	\XINT_sqrt_big_kb .....	138
\XINT_rdg_aux .....	150	\XINT_sqrt_big_kc .....	138
\XINT_rep .....	116, 120	\XINT_sqrt_big_ke .....	139
\XINT_replicate .....	150, 151	\XINT_sqrt_big_kend .....	138
\XINT_restorecatcodes_endinput .....	153	\XINT_sqrt_big_kf .....	139
\XINT_rev_nounsep .....	143	\XINT_sqrt_big_kg .....	139
\XINT_revdigits .....	115	\XINT_sqrt_big_kloop .....	138, 139
\XINT_revdigits_a .....	115	\XINT_sqrt_big_kz .....	138
\XINT_revdigits_b .....	116	\XINT_sqrt_bigormed_f .....	135
\XINT_revdigits_end .....	116	\XINT_sqrt_b0 .....	133
\xint_secondofthree .....	114	\XINT_sqrt_c .....	133
\xint_secondoftwo .....	128, 149	\XINT_sqrt_checkin .....	133
\XINT_Sgn .....	139	\XINT_sqrt_isneg .....	133
\XINT_signalcondition .....	133, 140, 146	\XINT_sqrt_iszero .....	133
\XINT_smalldivx_a .....	143	\XINT_sqrt_med_fa .....	136
\XINT_smallmul .....	143, 147, 148	\XINT_sqrt_med_fb .....	136
\XINT_split_finish .....	117	\XINT_sqrt_med_fv .....	136
\XINT_split_fromleft .....	117, 118, 138	\XINT_sqrt_med_fvi .....	136
\XINT_split_fromleft_a .....	117	\XINT_sqrt_med_fvii .....	136
\XINT_split_fromleft_b .....	117	\XINT_sqrt_med_fviii .....	136
\XINT_split_fromleft_clean .....	117	\XINT_sqrt_post .....	139

\XINT_sqrt_small_a .....	133	\xintiffFalseAelseB .....	127
\XINT_sqrt_small_d .....	133, 134	\xintiffTrueAelseB .....	127--129
\XINT_sqrt_small_e .....	134, 135	\xintiiabs .....	148, 149
\XINT_sqrt_small_ea .....	134, 136	\xintiiadd .....	136, 137, 139, 151
\XINT_sqrt_small_eb .....	134	\xintiiBinomial .....	139
\XINT_sqrt_small_ec .....	134	\xintiiibinomial .....	139, 140
\XINT_sqrt_small_end .....	134, 135	\xintiiCmp .....	125, 126
\XINT_sqrt_small_ez .....	134	\xintiidivision .....	137, 139
\XINT_sqrt_small_f .....	134, 135	\xintiiE .....	116
\XINT_sqrt_small_g .....	134	\xintiiE .....	116
\XINT_sqrt_small_h .....	135	\xintiiEq .....	121
\XINT_sqrt_start .....	133	\xintiiEven .....	124
\XINT_sqrtr_post .....	139	\xintiiEven .....	124
\xint_stop_atfirstofthree .....	114, 124, 125	\xintiiGCD .....	148
\xint_stop_atfirstoftwo .....	125--127, 129--131	\xintiigcd .....	148
\xint_stop_atsecondofthree .....	114, 124, 125	\xintiiGeq .....	122
\xint_stop_atsecondoftwo .....	119, 125--127, 129--131	\xintiigeq .....	122
\xint_stop_atthirdofthree .....	114, 124, 125	\xintiiGt .....	122
\XINT_sum_finished .....	132	\xintiiGtorEq .....	123
\XINT_sum_loop_a .....	132	\xintiiifCmp .....	125
\XINT_sum_loop_b .....	132	\xintiiifcmp .....	125
\XINT_sum_loop_c .....	132	\xintiiifEq .....	125
\XINT_sumexpr .....	132	\xintiiifeq .....	121, 125
\xint_texuniformdeviate .....	114, 150, 153	\xintiiifGt .....	125
\xint_thirdofthree .....	114	\xintiiifgt .....	122, 123, 125
\xint_UDsignfork .....	115--118	\xintiiiflt .....	126, 139, 153
\xint_UDsignsfork .....	122, 129, 130	\xintiiiflt .....	122, 123, 126
\xint_UDzerofork .....	122, 149	\xintiiifNotZero .....	126
\xint_UDzerominusfork .....	117, 119, 120, 133, 151	\xintiiifnotzero .....	126, 127
\xint_UDzerosfork .....	129, 130	\xintiiifOdd .....	127
\XINT_uniformdeviate .....	152	\xintiiifodd .....	127
\XINT_unsep_cuzsmall .....	144	\xintiiifOne .....	126
\XINT_XINT_rdg .....	150	\xintiiifone .....	126
\XINT_xorof_a .....	128, 129	\xintiiifSgn .....	125
\XINT_xorof_b .....	129	\xintiiifsgn .....	125
\XINT_xorof_c .....	129	\xintiiifZero .....	126
\XINT_xorof_e .....	129	\xintiiifzero .....	126, 127
\XINT_xrandomdigits .....	151	\xintiiIsNotZero .....	123
\XINT_xrandomdigits_a .....	151	\xintiiisnotzero .....	123, 128
\xintAND .....	128	\xintiiIsOne .....	123, 127
\xintand .....	128	\xintiiisone .....	123
\xintANDof .....	128	\xintiiIsZero .....	123, 128
\xintandof .....	128	\xintiiiszero .....	123, 127
\xintBool .....	148	\xintiiLCM .....	149
\xintDecSplit .....	117	\xintiiLCM .....	149
\xintdecsplit .....	117	\xintiiLt .....	122
\xintDecSplitL .....	118	\xintiiLtorEq .....	123
\xintdecsplitl .....	118	\xintiiMax .....	129
\xintDecSplitR .....	118	\xintiiimax .....	129, 131
\xintdecsplitr .....	118	\xintiiMaxof .....	131
\xintDouble .....	137	\xintiiimaxof .....	131
\xintDSH .....	119	\xintiiMin .....	130
\xintdsh .....	119	\xintiiimin .....	130, 131
\xintDSHr .....	119	\xintiiMinof .....	131
\xintdshr .....	119	\xintiiiminof .....	131
\xintDSx .....	120	\xintiiIMON .....	124
\xintdsx .....	120	\xintiiimmon .....	124

[illegible]Z

## 12.5 Index of xintbinhex

<b>A</b>	
<code>\aftergroup</code> .....	154, 155
<b>C</b>	
<code>\catcode</code> .....	154
<code>\csname</code> .....	154--156, 158--160, 163--165
<b>E</b>	
<code>\edef</code> .....	155
<code>\endcsname</code> .....	154--156, 158--160, 163--165
<code>\endinput</code> .....	154, 155
<code>\endlinechar</code> .....	154
<b>I</b>	
<code>\if</code> .....	159, 161, 162
<code>\immediate</code> .....	154
<b>K</b>	
<code>\krof</code> .....	157, 159, 160, 163--165
<b>N</b>	
<code>\numexpr</code> .....	154, 157--164
<b>P</b>	
<code>\ProvidesPackage</code> .....	155
<b>R</b>	
<code>\R</code> .....	156, 157, 159, 161, 163
<code>\romannumeral</code> .....	156, 157, 159--165
<b>U</b>	
<code>\unless</code> .....	159
<b>W</b>	
<code>\W</code> .....	156, 157, 159, 161, 163
<code>\w</code> .....	154, 155
<code>\write</code> .....	154
<b>X</b>	
<code>\x</code> .....	154, 155
<code>\xint:</code> .....	156--160, 162, 163
<code>\xint_afterfi</code> .....	158
<code>\XINT_btd_checkin</code> .....	162
<code>\XINT_btd_htd</code> .....	163
<code>\XINT_btd_main</code> .....	163
<code>\XINT_btd_N</code> .....	162, 163
<code>\XINT_bth_checkin</code> .....	163
<code>\XINT_bth_loop</code> .....	163, 164
<code>\XINT_bth_main</code> .....	163
<code>\XINT_bth_N</code> .....	163
<code>\xint_bye</code> .....	157, 159, 161--165
<code>\xint_c</code> .....	158
<code>\xint_c_i</code> .....	157--160
<code>\xint_c_ii^vii</code> .....	158, 159
<code>\xint_c_ii^viii</code> .....	158, 159
<code>\xint_c_ii^xv</code> .....	155, 157
<code>\xint_c_ii^xvi</code> .....	155, 157, 161
<code>\xint_c_viii</code> .....	158--160
<code>\xint_c_xvi</code> .....	158--160
<code>\XINT_chtb_checkin</code> .....	165
<code>\XINT_chtb_main</code> .....	165
<code>\XINT_chtb_N</code> .....	165
<code>\XINT_csbth_none</code> .....	156
<code>\XINT_cshtb_A</code> .....	156
<code>\XINT_cshtb_B</code> .....	156
<code>\XINT_cshtb_C</code> .....	156
<code>\XINT_cshtb_D</code> .....	156
<code>\XINT_cshtb_E</code> .....	156
<code>\XINT_cshtb_F</code> .....	156
<code>\XINT_cshtb_none</code> .....	156
<code>\xint_dothis</code> .....	158, 159
<code>\XINT_dtb_checkin</code> .....	159
<code>\XINT_dtb_finish</code> .....	159, 160
<code>\XINT_dtb_finish_a</code> .....	160
<code>\XINT_dtb_main</code> .....	159
<code>\XINT_dtb_neg</code> .....	159
<code>\XINT_dtb_tobin</code> .....	159, 160
<code>\XINT_dtb_tobin_a</code> .....	159
<code>\XINT_dth_checkin</code> .....	156
<code>\XINT_dth_finish</code> .....	157, 159
<code>\XINT_dth_main</code> .....	157
<code>\XINT_dth_neg</code> .....	156, 157
<code>\XINT_dth_tohex</code> .....	157--159
<code>\XINT_dth_tohex_a</code> .....	158
<code>\XINT_dthb_again</code> .....	157, 158
<code>\XINT_dthb_exclam</code> .....	157
<code>\XINT_dthb_lastpass</code> .....	157
<code>\XINT_dthb_nextfour</code> .....	157
<code>\XINT_dthb_small</code> .....	157
<code>\XINT_dthb_start</code> .....	157, 159
<code>\XINT_dthb_start_a</code> .....	157
<code>\XINT_dthb_update</code> .....	157, 158
<code>\XINT_dthb_update_a</code> .....	157
<code>\xint_firstoftwo</code> .....	158
<code>\xint_gob_til_R</code> .....	156
<code>\XINT_htb_checkin</code> .....	164
<code>\XINT_htb_cuz</code> .....	164
<code>\XINT_htb_loop</code> .....	164, 165
<code>\XINT_htb_main</code> .....	164
<code>\XINT_htb_N</code> .....	164
<code>\XINT_htd_A</code> .....	161, 162
<code>\XINT_htd_a</code> .....	161, 162
<code>\XINT_htd_Aa</code> .....	162
<code>\XINT_htd_Ab</code> .....	162
<code>\XINT_htd_again</code> .....	161, 162
<code>\XINT_htd_checkin</code> .....	160
<code>\XINT_htd_finish</code> .....	161, 162
<code>\XINT_htd_finish_cuz</code> .....	162
<code>\XINT_htd_main</code> .....	160
<code>\XINT_htd_neg</code> .....	160

**Y**

Z



## 12.6 Index of xintgcd

<b>A</b>		<b>V</b>	
\A	176--178	\V	178
\aftergroup	166, 167	\vbox	176, 177
<b>B</b>		<b>W</b>	
\B	176--178	\w	166, 167
\BEZ	177, 178	\write	166
\break	177	<b>X</b>	
<b>C</b>		\x	166, 167
\catcode	166	\xint:	167, 168, 171, 172, 174--176
\count	176--178	\XINT_bezalg	174
\cr	176, 177	\XINT_bezalg_a	174, 175
\csname	166	\XINT_bezalg_AisZero	174
<b>D</b>		\XINT_bezalg_b	174
\D	178	\XINT_bezalg_BisZero	174
<b>E</b>		\XINT_bezalg_c	174, 175
\edef	176--178	\XINT_bezalg_d	175
\endcsname	166	\XINT_bezalg_e	175
\endinput	166, 167	\XINT_bezalg_end	175
\endlinechar	166	\XINT_bezalg_end_a	175
\errmessage	176, 177	\XINT_bezalg_fork	174
<b>H</b>		\XINT_bezout	169
\halign	176, 177	\XINT_bezout_botharezero	169
\hfill	177	\XINT_bezout_exitA	171, 172
<b>I</b>		\XINT_bezout_exita	172
\if	171, 172	\XINT_bezout_firstiszero	169
\ifdefined	176, 177	\XINT_bezout_fork	169
\immediate	166	\XINT_bezout_loop_B	171, 172
\indent	177	\XINT_bezout_loop_b	172
<b>K</b>		\XINT_bezout_loop_C	171
\krof	167--170, 173, 174	\XINT_bezout_loop_c	172
<b>N</b>		\XINT_bezout_loop_D	171, 172
\N	176--178	\XINT_bezout_loop_d	172
\numexpr	166, 173, 174, 177	\XINT_bezout_loop_E	172
<b>P</b>		\XINT_bezout_loop_e	172
\ProvidesPackage	167	\XINT_bezout_loop_minusminus	169, 170
<b>R</b>		\XINT_bezout_loop_minusplus	169, 170
\repeat	177, 178	\XINT_bezout_mm_post	170
\romannumeral	167--176	\XINT_bezout_mm_postb	170
<b>T</b>		\XINT_bezout_mp_post	170
\times	177, 178	\XINT_bezout_plusminus	169, 170
\to	176, 177	\XINT_bezout_plusplus	169, 170
<b>U</b>		\XINT_bezout_pm_post	170
\U	176--178	\XINT_bezout_preloop_A	171
\unless	176, 177	\XINT_bezout_preloop_a	170, 171
		\XINT_bezout_preloop_exchange	171
		\XINT_bezout_preloop_exit	171
		\XINT_bezout_secondiszero	169
		\xint_bye	174, 175
		\xint_c_i	173, 174
		\XINT_div_prepare	168, 170--174
		\xint_dothis	171
		\XINT_euc	172, 173
		\XINT_euc_a	173

\XINT_euc_AisZero .....	173
\XINT_euc_b .....	173
\XINT_euc_BisZero .....	173
\XINT_euc_c .....	173
\XINT_euc_end .....	173
\XINT_euc_end_a .....	173, 174
\XINT_euc_fork .....	173
\xint_exchangetwo_keepbraces .....	170, 171
\XINT_gcd_AisZero .....	167
\XINT_gcd_Aiszero .....	167
\XINT_gcd_BisZero .....	167
\XINT_gcd_Biszero .....	167
\XINT_gcd_CheckRem .....	168
\XINT_gcd_end .....	168
\XINT_gcd_fork .....	167
\XINT_gcd_loop .....	167, 168
\XINT_gcddof_a .....	176
\XINT_gcddof_b .....	176
\XINT_gcddof_c .....	176
\XINT_gcddof_d .....	176
\XINT_gcddof_e .....	176
xint_gob_til_xint: .....	176
\xint_gob_til_zero .....	168, 173, 175
\xint_gobble_iii .....	176, 177
\XINT_iigcd .....	167
\XINT_iilmcm .....	168
\XINT_lcm_end .....	168
\XINT_lcm_fork .....	168
\XINT_lcm_iszero .....	168
\XINT_lcm_notzero .....	168
\XINT_lcmof_a .....	176
\XINT_lcmof_b .....	176
\XINT_lcmof_c .....	176
\XINT_lcmof_d .....	176
\XINT_lcmof_e .....	176
\XINT_mul_plusplus .....	171, 172
\xint_orthat .....	171
\XINT_providespackage .....	167
\XINT_restorecatcodes_endinput .....	178
\XINT_rord_main .....	174, 175
\xint_secondoftwo .....	168
\XINT_TypesetBezoutAlgorithm .....	177
\XINT_TypesetEuclideanAlgorithm .....	176
\xint_UDsignfork .....	169
\xint_UDsignsfork .....	169
\xint_UDzerofork .....	167, 168, 173, 174
\xint_UDzerosfork .....	169
\xintAssignArray .....	176, 177
\xintBezout .....	169
\xintbezout .....	169
\xintBezoutAlgorithm .....	174, 177
\xintbezoutalgorithm .....	174
\xintEuclideanAlgorithm .....	172, 176
\xinteuklidealgorithm .....	172
\xintGCD .....	167
\xintgcd .....	167, 176
\xintGCDoF .....	176
\xintgcdoF .....	176
\xintiiaBs .....	167, 168, 172--174
\xintiiaDd .....	171, 172, 175
\xintiiGCD .....	167
\xintiigcd .....	167
\xintiiLCM .....	168
\xintiilmcm .....	168
\xintiiMul .....	175
\xintiimul .....	168
\xintiioPp .....	170
\xintiiQuo .....	168
\xintLCM .....	168
\xintlcm .....	168, 176
\xintLCMoF .....	176
\xintlcmoF .....	176
\xintloop .....	177
\xintNum .....	167, 168, 172--174
\xintnum .....	169
\XINTSetupCatCodes .....	167
\xintTypesetBezoutAlgorithm .....	177
\xintTypesetEuclideanAlgorithm .....	176
<b>y</b>	
\y .....	166
<b>Z</b>	
\Z .....	169, 173--175
\z .....	166, 167

Z

## 12.7 Index of xintfrac

A		X	
<code>\afterassignment</code>	208	<code>\X</code>	204
<code>\aftergroup</code>	180	<code>\x</code>	180
C		<code>\xint:</code>	181--185, 191, 198--200, 202, 204--208, 210, 211, 213, 218--220, 222--224, 231, 233--237, 240, 242, 244, 249, 253, 255, 257, 259
<code>\catcode</code>	180, 192, 193	<code>\XINT_abs</code>	181, 222
<code>\csname</code>	180, 229, 232, 233, 247	<code>\xint_afterfi</code>	187, 192, 193, 195, 197, 201, 217, 219, 220, 240, 248, 250
E		<code>\XINT_binom_pre</code>	212
<code>\endcsname</code>	180, 205, 207, 208, 229, 232, 233, 238, 242, 247	<code>\xint_Bye</code>	202, 227, 243
<code>\endinput</code>	180	<code>\xint_bye</code>	181, 184, 185, 200, 202, 205, 206, 208, 227--229, 238, 243, 256
<code>\endlinechar</code>	180	<code>\xint_c_</code>	181, 184, 185, 188--191, 200, 201, 204, 206, 207, 229, 230, 232, 234, 238, 239, 243, 244, 249, 258, 259
F		<code>\xint_c_i</code>	181, 184--186, 200--206, 225, 227, 229--232, 238, 239, 243, 244, 246--257
<code>\frac</code>	192	<code>\xint_c_ii</code>	181, 184, 185, 200, 203, 204, 229, 239, 243, 244, 246--248, 250--253, 255, 256
I		<code>\xint_c_ii^v</code>	205, 206
<code>\if</code>	182, 186--190, 192, 193, 195--197, 199--202, 210, 214--217, 219, 220, 225, 226, 229--231, 238, 240, 241, 243, 248, 250, 254, 256, 258, 259	<code>\xint_c_ii^vi</code>	205, 206
<code>\iffalse</code>	238, 243	<code>\xint_c_iii</code>	181, 184, 185, 200, 203, 229, 237, 241, 242, 246--248, 251, 252, 254--256
<code>\iftrue</code>	238, 242	<code>\xint_c_iv</code>	181, 184, 185, 200, 204, 228, 246--248, 250, 251, 254
<code>\immediate</code>	180	<code>\xint_c_ix</code>	186
K		<code>\xint_c_v</code>	181, 184, 185, 200, 229, 232, 243, 256
<code>\krof</code>	183--185, 192--200, 202, 205, 207, 209, 210, 212, 215--223, 225--227, 232, 237, 238, 240, 242, 243, 245, 255, 257	<code>\xint_c_vi</code>	181, 184, 185, 200, 204
M		<code>\xint_c_vii</code>	181, 184, 185, 200
<code>\mathchardef</code>	208	<code>\xint_c_viii</code>	181, 184, 185, 200, 204, 246, 248--251, 256
N		<code>\xint_c_x</code>	202, 204, 227, 243
<code>\numexpr</code>	180, 181, 183--186, 194, 195, 199--209, 211--213, 215--218, 221, 222, 224--239, 242--244, 246--253, 255--257, 259	<code>\xint_c_x^iv</code>	245, 246, 250
O		<code>\xint_c_x^ix</code>	248
<code>\over</code>	193	<code>\xint_c_x^viii</code>	203, 246--253
P		<code>\xint_c_x^viii_mone</code>	245, 250, 254
<code>\ProvidesPackage</code>	180	<code>\xint_c_xii</code>	204
R		<code>\XINT_cmp_plusplus</code>	222
<code>\R</code>	202--204, 246	<code>\XINT_cntSgn</code>	181, 191, 218, 222
<code>\romannumeral</code>	181--244, 246, 248--253, 255--260	<code>\XINT_cntSgnFork</code>	181, 218, 222
U		<code>\XINT_dbl</code>	256
<code>\unless</code>	250	<code>\XINT_dectostr</code>	190
W		<code>\XINT_dectostr_a</code>	190
<code>\W</code>	202, 204, 246	<code>\XINT_dectostr_b</code>	190
<code>\w</code>	180	<code>\XINT_dectostr_z</code>	190
<code>\write</code>	180	<code>\XINT_denom_A</code>	191, 192
		<code>\XINT_denom_B</code>	191, 192
		<code>\XINT_denom_fork</code>	191
		<code>\XINT_div_prepare</code>	196, 198, 201, 209, 215--217
		<code>\XINT_div_prepare_g</code>	204
		<code>\XINT_div_small_b</code>	204
		<code>\XINT_div_small_ba</code>	204
		<code>\XINT_divmod_a</code>	215

\XINT_divmod_aiszero .....	215	\XINT_fcmp_E .....	222
\XINT_divmod_b .....	215	\XINT_fcmp_Fd .....	222
\XINT_divmod_bneg .....	215	\XINT_fcmp_Fe .....	222
\XINT_divmod_bneg_finish .....	216	\XINT_fcmp_firstneg .....	221
\XINT_divmod_bpos .....	215, 216	\XINT_fcmp_firstzero .....	221
\XINT_divmod_bpos_a .....	216	\XINT_fcmp_Fn .....	222
\XINT_divmod_bpos_finish .....	216	\XINT_fcmp_Fo .....	222
\XINT_divmod_divbyzero .....	215	\XINT_fcmp_minusminus .....	221
\xint_dothis .....	186, 190, 199--202, 214--216, 225, 226, 230, 232, 234, 238, 243, 245, 250, 251, 254--256, 258, 259	\XINT_fcmp_nonneg_a .....	221
\XINT_dsrr .....	202, 227	\XINT_fcmp_pos .....	221
\XINT_dsx_addzeros .....	190-- 192, 199, 200, 207, 209, 218, 222, 225, 256	\XINT_fcmp_secondneg .....	221
\XINT_dsx_addzerosnofuss .....	229	\XINT_fcmp_secondzero .....	221
\XINT_e .....	195	\XINT_fcmp_zerozero .....	221
\XINT_e_end .....	195	\XINT_fdiv .....	213
\XINT_eightrandomdigits .....	260	\XINT_fdiv_A .....	213
\xint_exchangetwo_keepbraces .....	216	\XINT_fdiv_B .....	213, 214
\XINT_fac_fork .....	212	\XINT_fdiv_C .....	214
\XINT_factortens .....	186, 194	\XINT_fgeq .....	218
\XINT_factortens_a .....	186	\XINT_fgeq_A .....	218--220
\XINT_factortens_b .....	186	\XINT_fgeq_B .....	218
\XINT_factortens_c .....	186	\XINT_fgeq_C .....	218
\XINT_factortens_cc .....	186	\XINT_fgeq_D .....	218
\XINT_factortens_d .....	186	\XINT_fgeq_E .....	218
\XINT_factortens_e .....	186	\XINT_fgeq_Fd .....	218
\XINT_factortens_f .....	186	\XINT_fgeq_Fe .....	218
\XINT_factortens_g .....	186	\XINT_fgeq_Fn .....	218
\XINT_factortens_x .....	186	\XINT_fgeq_Fo .....	218, 219
\XINT_factortens_y .....	186	\XINT_fgeq_Zi .....	218
\XINT_factortens_yy .....	186	\XINT_fgeq_Zii .....	218
\XINT_factortens_z .....	186	\xint_firstofone .....	231
\XINT_fadd .....	208	\xint_firstofthree .....	181
\XINT_fadd_a .....	208	\xint_firstoftwo .....	189, 201, 234, 249
\XINT_fadd_Aa .....	209	\XINT_FL_add_a .....	233--235
\XINT_fadd_Azero .....	208	\XINT_FL_add_b .....	234
\XINT_fadd_B .....	209	\XINT_FL_add_c .....	234
\XINT_fadd_b .....	208	\XINT_FL_add_d .....	234
\XINT_fadd_Ba .....	209	\XINT_FL_add_zero .....	234
\XINT_fadd_Bb .....	209	\XINT_FL_binom_a .....	253
\XINT_fadd_Bzero .....	208, 210	\XINT_FL_binom_aa .....	254
\XINT_fadd_C .....	209	\XINT_FL_binom_ab .....	254
\XINT_fadd_c .....	208, 210	\XINT_FL_binom_fork .....	253
\XINT_fadd_D_a .....	209	\XINT_FL_binom_neg .....	254
\XINT_fadd_D_b .....	209	\XINT_FL_binom_one .....	254
\XINT_fadd_D_c .....	209	\XINT_FL_binom_toobig .....	254
\XINT_fadd_D_exit .....	209	\XINT_FL_binom_zero .....	254
\XINT_fadd_E .....	209	\XINT_FL_div_a .....	236
\XINT_fadd_F .....	209, 210	\XINT_FL_div_b .....	236
\XINT_fadd_G .....	210	\XINT_FL_fac_addzeros .....	247, 248
\XINT_fadd_iszero .....	209, 210	\XINT_FL_fac_addzeros_exit .....	248
\XINT_fcmp .....	221	\XINT_FL_fac_big .....	245, 246
\XINT_fcmp_A .....	221	\XINT_FL_fac_bigloop_a .....	246
\XINT_fcmp_B .....	221	\XINT_FL_fac_bigloop_b .....	246
\XINT_fcmp_C .....	221	\XINT_FL_fac_bigloop_loop .....	246, 247
\XINT_fcmp_D .....	221, 222	\XINT_FL_fac_bigloop_mul .....	247
		\XINT_FL_fac_countdigits .....	246, 250
		\XINT_FL_fac_countdone .....	246
		\XINT_FL_fac_fork_a .....	244

\XINT_FL_fac_fork_b .....	245, 254	\XINT_FL_sqrt_iszero .....	255
\XINT_FL_fac_increasP .....	246	\XINT_FL_sqrt_pos .....	255
\XINT_FL_fac_isneg .....	244, 245	\XINT_fladd_chkopt .....	233
\XINT_FL_fac_iszero .....	244, 245	\XINT_fladd_noopt .....	233
\XINT_FL_fac_loop_exit .....	246--248	\XINT_fladd_opt .....	233
\XINT_FL_fac_med .....	245, 246	\XINT_fladd_opt_a .....	233, 234
\XINT_FL_fac_medloop_a .....	246, 247	\XINT_flbinom_chkopt .....	253
\XINT_FL_fac_medloop_b .....	247	\XINT_flbinom_noopt .....	253
\XINT_FL_fac_medloop_loop .....	247	\XINT_flbinom_opt .....	253
\XINT_FL_fac_medloop_mul .....	247	\XINT_fldiv_chkopt .....	236
\XINT_FL_fac_minimulwc_a .....	248, 249	\XINT_fldiv_noopt .....	236
\XINT_FL_fac_minimulwc_b .....	248	\XINT_fldiv_opt .....	236
\XINT_FL_fac_minimulwc_c .....	248	\XINT_fldiv_opt_a .....	236
\XINT_FL_fac_minimulwc_d .....	248	\XINT_flen .....	181
\XINT_FL_fac_minimulwc_e .....	248	\XINT_flfac_chkopt .....	244
\XINT_FL_fac_mul .....	246--248, 251--253	\XINT_flfac_noopt .....	244
\XINT_FL_fac_mul_a .....	248	\XINT_flfac_opt .....	244
\XINT_FL_fac_out .....	244, 246, 252--254	\XINT_flfac_opt_a .....	244
\XINT_FL_fac_small .....	245, 246	\XINT_flmul_chkopt .....	235
\XINT_FL_fac_smallloop_a .....	246, 247	\XINT_flmul_noopt .....	235
\XINT_FL_fac_smallloop_loop .....	248	\XINT_flmul_opt .....	235
\XINT_FL_fac_smallloop_mul .....	248	\XINT_flmul_opt_a .....	235
\XINT_FL_fac_smallmul .....	248, 249	\XINT_float_chkopt .....	224
\XINT_FL_fac_smallmul_end .....	249	\XINT_float_neg .....	225
\XINT_FL_fac_toobig .....	245	\XINT_float_noopt .....	224
\XINT_FL_fac_vbig .....	245, 246	\XINT_float_opt .....	224
\XINT_FL_fac_vbigloop_a .....	246	\XINT_float_opt_a .....	224
\XINT_FL_fac_vbigloop_loop .....	246	\XINT_float_pos .....	225
\XINT_FL_mul_a .....	235	\XINT_float_pos_done .....	225, 257
\XINT_FL_mul_b .....	235	\XINT_float_post .....	224, 225
\XINT_FL_pfac_a .....	250	\XINT_float_zero .....	225
\XINT_FL_pfac_addzeroes .....	250, 251	\XINT_floate_chkopt .....	257
\XINT_FL_pfac_addzeroes_exit .....	251	\XINT_floate_neg .....	257
\XINT_FL_pfac_b .....	250, 251	\XINT_floate_noopt .....	257
\XINT_FL_pfac_bigloop .....	251, 252	\XINT_floate_opt .....	257
\XINT_FL_pfac_bigloop_a .....	252	\XINT_floate_opt_a .....	257
\XINT_FL_pfac_bigloop_b .....	252	\XINT_floate_pos .....	257
\XINT_FL_pfac_end_ .....	251--253	\XINT_floate_post .....	257
\XINT_FL_pfac_end_i .....	251--253	\XINT_floate_zero .....	257
\XINT_FL_pfac_end_ii .....	251, 252	\XINT_flpfac_chkopt .....	249
\XINT_FL_pfac_end_iii .....	251, 252	\XINT_flpfac_noopt .....	249
\XINT_FL_pfac_fork .....	249, 250	\XINT_flpfac_opt .....	249
\XINT_FL_pfac_increasP .....	250, 254	\XINT_flpfac_opt_b .....	249, 250
\XINT_FL_pfac_medloop .....	251, 252	\XINT_flpow_a .....	238
\XINT_FL_pfac_medloop_a .....	251	\XINT_flpow_aa .....	237
\XINT_FL_pfac_medloop_b .....	251, 252	\XINT_flpow_ab .....	238
\XINT_FL_pfac_neg .....	250	\XINT_flpow_b .....	238
\XINT_FL_pfac_one .....	250	\XINT_flpow_BisZero .....	237
\XINT_FL_pfac_outofrange .....	250	\XINT_flpow_checkB_a .....	237
\XINT_FL_pfac_smallloop .....	251	\XINT_flpow_checkB_b .....	237
\XINT_FL_pfac_smallloop_a .....	251	\XINT_flpow_checkB_c .....	237
\XINT_FL_pfac_smallloop_b .....	251	\XINT_flpow_checkB_d .....	237
\XINT_FL_pfac_vbigloop .....	251, 252	\XINT_flpow_chkopt .....	236, 237
\XINT_FL_pfac_vbigloop_a .....	252	\XINT_flpow_III .....	238--240, 243, 244
\XINT_FL_pfac_zero .....	250	\XINT_flpow_IIIend .....	240
\XINT_FL_sqrt_a .....	255	\XINT_flpow_IItoIII .....	239
\XINT_FL_sqrt_isneg .....	255	\XINT_flpow_ItoIII .....	238

\XINT_flpow_loopI .....	238, 239	\XINT_flsqrt_noopt .....	255
\XINT_flpow_loopI_even .....	238, 239	\XINT_flsqrt_opt .....	255
\XINT_flpow_loopI_odd .....	238, 239	\XINT_flsqrt_opt_a .....	255
\XINT_flpow_loopII .....	239	\XINT_flsub_chkopt .....	234
\XINT_flpow_loopII_even .....	239	\XINT_flsub_noopt .....	234
\XINT_flpow_loopII_odd .....	239	\XINT_flsub_opt .....	234
\XINT_flpow_loopII_odda .....	239	\XINT_flsub_opt_a .....	235
\XINT_flpow_noopt .....	237	\XINT_fmax .....	219
\XINT_flpow_opt .....	237	\XINT_fmax_A .....	219
\XINT_flpow_opt_a .....	237	\XINT_fmax_firstneg .....	219
\XINT_flpow_truncate .....	238, 239, 243, 244	\XINT_fmax_minusminus .....	219
\XINT_flpow_truncate_a .....	238	\XINT_fmax_nonneg_a .....	219
\XINT_flpow_zero .....	238, 242	\XINT_fmax_nonneg_b .....	219, 220
\XINT_flpower_a .....	242	\XINT_fmax_secondneg .....	219
\XINT_flpower_aa .....	242	\XINT_fmin .....	220
\XINT_flpower_ab .....	242	\XINT_fmin_A .....	220
\XINT_flpower_b .....	242, 243	\XINT_fmin_firstneg .....	220
\XINT_flpower_BisZero .....	241, 242	\XINT_fmin_minusminus .....	220
\XINT_flpower_checkB_a .....	241, 242	\XINT_fmin_nonneg_a .....	220
\XINT_flpower_checkB_b .....	242	\XINT_fmin_nonneg_b .....	219, 220
\XINT_flpower_checkB_c .....	242	\XINT_fmin_secondneg .....	220
\XINT_flpower_checkB_d .....	242	\XINT_fmud .....	210
\XINT_flpower_chkopt .....	240, 241	\XINT_fmud_a .....	210, 211
\XINT_flpower_IItoIII .....	243, 244	\XINT_fmud_b .....	211
\XINT_flpower_ItoIII .....	243	\XINT_fmud_c .....	211
\XINT_flpower_loopI .....	243	\XINT_fmud_d .....	211
\XINT_flpower_loopI_even .....	243	\XINT_fmud_e .....	211
\XINT_flpower_loopI_odd .....	243	\XINT_fmud_zero .....	210, 211
\XINT_flpower_loopII .....	243	\XINT_fpow .....	211
\XINT_flpower_loopII_even .....	243	\XINT_fpow_fork .....	212
\XINT_flpower_loopII_odd .....	243, 244	\XINT_fpow_neg .....	212
\XINT_flpower_loopII_odda .....	244	\XINT_fpow_pos .....	212
\XINT_flpower_noopt .....	241, 242	\XINT_fpow_pos_A .....	212
\XINT_flpower_opt .....	241, 242	\XINT_fpow_pos_B .....	212
\XINT_flpower_opt_a .....	242	\XINT_fpow_zero .....	212
\XINT_flpower_toloopI .....	243	\XINT_fprdxpr .....	213
\XINT_flpower_toloopII .....	243, 244	\XINT_fprod_finished .....	213
\XINT_flpowerh_a .....	241	\XINT_fprod_loop_a .....	213
\XINT_flpowerh_b .....	241	\XINT_fprod_loop_b .....	213
\XINT_flpowerh_c .....	241	\XINT_fprod_loop_c .....	213
\XINT_flpowerh_d .....	241	\XINT_frac_gen .....	183, 184
\XINT_flpowerh_e .....	241	\XINT_frac_gen_A .....	184
\XINT_flpowerh_finish .....	241	\XINT_frac_gen_B .....	184
\XINT_flpowerh_i .....	241	\XINT_frac_gen_Ba .....	184
\XINT_flpowerh_int .....	241	\XINT_frac_gen_Bb .....	184
\XINT_flsqrt .....	255	\XINT_frac_gen_Bc .....	184
\XINT_flsqrt_a .....	255, 256	\XINT_frac_gen_Bd .....	184
\XINT_flsqrt_again .....	256	\XINT_frac_gen_C .....	184
\XINT_flsqrt_again_a .....	256	\XINT_frac_gen_Ca .....	185
\XINT_flsqrt_b .....	256	\XINT_frac_gen_Cb .....	185
\XINT_flsqrt_c .....	256	\XINT_frac_gen_Cc .....	185
\XINT_flsqrt_chkopt .....	255	\XINT_frac_gen_F .....	185
\XINT_flsqrt_d .....	256	\XINT_frac_gen_G .....	185
\XINT_flsqrt_f .....	256	\XINT_frac_gen_Ga .....	185, 186
\XINT_flsqrt_finish .....	256	\XINT_frac_gen_Gdivbyzero .....	185
\XINT_flsqrt_g .....	256	\XINT_frac_gen_Gdivbyzero_a .....	185
\XINT_flsqrt_h .....	256	\XINT_frac_gen_zero .....	186

\XINT_fracfrac_A .....	192, 193	\XINT_infloat_Rq .....	229
\XINT_fracfrac_B .....	192	\XINT_infloat_Sa .....	230
\XINT_fracfrac_C .....	192	\XINT_infloat_Sb .....	230
\XINT_fracfrac_D .....	192	\XINT_infloat_SEq .....	229, 230
\XINT_fracfrac_E .....	192	\XINT_infloat_sp .....	226
\XINT_fracisone .....	217	\XINT_infloat_sp_b .....	226, 227
\XINT_fsqr .....	211	\XINT_infloat_sp_c .....	227
\XINT_fsqr_a .....	211	\XINT_infloat_sp_needzeros .....	227
\XINT_fsqr_b .....	211	\XINT_infloat_sp_quick .....	227
\XINT_fsqr_zero .....	211	\XINT_infloat_sp_round .....	227
\XINT_fsub .....	210	\XINT_infloat_spneg .....	226
\XINT_fsub_a .....	210	\XINT_infloat_spneg_needzeros .....	226
\XINT_fsub_Azero .....	210	\XINT_infloat_spnegend .....	226
\XINT_fsub_b .....	210	\XINT_infloat_spos .....	226
\XINT_fsum_finished .....	210	\XINT_infloat_spzero .....	226
\XINT_fsum_loop_a .....	210	\XINT_infloat_SUp .....	229, 230
\XINT_fsum_loop_b .....	210	\XINT_infloat_SY .....	230
\XINT_fsum_loop_c .....	210	\XINT_infloat_X .....	230
\XINT_fsumexpr .....	210	\XINT_infloat_Y .....	227, 230, 231
\XINT_fwover_A .....	193, 194	\XINT_infloat_Z .....	231
\XINT_fwover_B .....	193	\XINT_infloat_ZZ .....	231
\XINT_fwover_C .....	193	\XINT_infloatdivmod .....	258
\XINT_fwover_D .....	193	\XINT_infloate .....	257
\XINT_geq_plusplus .....	218, 219	\XINT_infloate_end .....	257
\xint_gob_til_dot .....	204	\XINT_infloatfracdg_a .....	233
\xint_gob_til_exclam .....	248	\XINT_infloatS_clean .....	226
\xint_gob_til_R .....	203	\XINT_infloatS_clean_a .....	226
\xint_gob_til_sc .....	249	\XINT_inFrac .....	183
\xint_gob_til_xint: .....	210, 213, 219, 220	\XINT_infrac .....	
\xint_gob_til_zero .....	183, 186,		181, 183, 189--195, 199, 202, 211, 213, 226
	192, 193, 196, 198, 208--211, 217, 218, 234	\XINT_infrac_fork .....	183
\xint_gobble_i .....	205, 208	\XINT_infrac_res_a .....	183
\xint_gobble_ii .....	192	\XINT_infrac_res_b .....	183
\xint_gobble_vi .....	207	\XINT_infrac_res_ca .....	183
\XINT_half .....	243	\XINT_infrac_res_cb .....	183
\XINT_iffloatint .....	258, 259	\XINT_infrac_res_cc .....	183
\XINT_ifint .....	196, 197	\XINT_infrac_res_zero .....	183
\XINT_ifloor .....	191	\XINT_inrandomfloatS .....	259
\XINT_infloat .....	225, 226	\XINT_inrandomfloatS_a .....	259, 260
\XINT_infloat_a .....	226	\XINT_inrandomfloatS_b .....	259
\XINT_infloat_clean .....	225	\XINT_inrandomfloatS_zero .....	259
\XINT_infloat_clean_a .....	225	\XINT_inv .....	222
\XINT_infloat_done .....	225	\XINT_inv_a .....	222
\XINT_infloat_fork .....	226, 227	\XINT_inv_b .....	223
\XINT_infloat_J .....	227	\XINT_inv_expiszero .....	223
\XINT_infloat_K .....	227, 228	\XINT_inv_iszero .....	222
\XINT_infloat_L .....	228	\XINT_iround_A .....	201, 202
\XINT_infloat_Ma .....	228	\XINT_irr_D .....	195
\XINT_infloat_Mb .....	228	\XINT_irr_denomisone .....	195
\XINT_infloat_MtoN .....	228	\XINT_irr_divisionbyzero .....	196
\XINT_infloat_N .....	228	\XINT_irr_finish .....	196, 198
\XINT_infloat_Oa .....	228, 229	\XINT_irr_indeterminate .....	196
\XINT_infloat_Ob .....	228, 229	\XINT_irr_loop_a .....	196
\XINT_infloat_P .....	229	\XINT_irr_loop_d .....	196
\XINT_infloat_Q .....	229	\XINT_irr_loop_e .....	196
\XINT_infloat_Qq .....	229	\XINT_irr_loop_exit .....	196
\XINT_infloat_R .....	229, 230	\XINT_irr_loop_exitb .....	196



\XINT_irr_negative	195	\XINT_num_loop	185
\XINT_irr_nonneg	195	\XINT_numer	191
\XINT_irr_start	195	\XINT_numer_A	191
\XINT_irr_zero	196	\XINT_numer_B	191
\XINT_is_One	199, 202, 226	\XINT_opp	182, 222
\XINT_isOne	189, 190, 192, 193, 195, 197, 243	\xint_orthat	186, 190, 199--202, 214--216, 225, 226, 230, 232, 234, 238, 243, 245, 250, 251, 254--256, 258, 259
\XINT_itrunc_G	199, 201, 202	\XINT_outfrac	181, 211, 212, 214
\XINT_jrr_D	197	\XINT_outfrac_divisionbyzero	181, 182
\XINT_jrr_denomisone	197	\XINT_outfrac_N	181, 182
\XINT_jrr_divisionbyzero	197	\XINT_outfrac_N_a	182
\XINT_jrr_indeterminate	197	\XINT_outfrac_P	181, 182
\XINT_jrr_loop_a	197, 198	\XINT_outfrac_Zero	182
\XINT_jrr_loop_b	198	\XINT_pfac_fork	213
\XINT_jrr_loop_c	198	\XINT_pfloat_a	231, 232
\XINT_jrr_loop_d	198	\XINT_pfloat_chkopt	231
\XINT_jrr_loop_e	198	\XINT_pfloat_N	232
\XINT_jrr_loop_exit	198	\XINT_pfloat_N_i	232
\XINT_jrr_negative	197	\XINT_pfloat_N_ii	232
\XINT_jrr_nonneg	197	\XINT_pfloat_N_iii	232
\XINT_jrr_start	197	\XINT_pfloat_N_iv	232
\XINT_jrr_zero	197	\XINT_pfloat_N_v	232
\XINT_len_fork	181	\XINT_pfloat_neg	232
\XINT_length_loop	184, 185, 200	\XINT_pfloat_no	232
\XINT_maxof_a	219	\XINT_pfloat_noopt	231
\XINT_maxof_b	219	\XINT_pfloat_opt	231
\XINT_maxof_c	219	\XINT_pfloat_opt_a	231
\XINT_maxof_d	219	\XINT_pfloat_P	232
\XINT_maxof_e	219	\XINT_pfloat_P_	232
\XINT_minof_a	220	\XINT_pfloat_P_i	232
\XINT_minof_b	220	\XINT_pfloat_P_ii	232
\XINT_minof_c	220	\XINT_pfloat_P_iii	232
\XINT_minof_d	220	\XINT_pfloat_P_iv	232
\XINT_minof_e	220	\XINT_pfloat_P_v	232
\XINT_mod_a	216	\XINT_pfloat_pos	232
\XINT_mod_aiszero	216	\XINT_pfloat_Ps	232
\XINT_mod_b	216	\XINT_pfloat_Psi	233
\XINT_mod_bneg	216	\XINT_pfloat_Psii	233
\XINT_mod_bpos	216, 217	\XINT_pfloat_Psiii	233
\XINT_mod_bpos_a	217	\XINT_pfloat_Psiv	233
\XINT_mod_D_a	217	\XINT_pfloat_Psv	233
\XINT_mod_D_b	215--217	\XINT_pfloat_zero	232
\XINT_mod_D_c	217	\XINT_pirr_start	195
\XINT_mod_D_exit	217	\xint_pow	211
\XINT_mod_divbyzero	216	\XINT_praw	189
\XINT_mod_E	217	\XINT_praw_A	189
\XINT_mod_F	217	\XINT_praw_a	189
\XINT_modtrunc_a	214	\XINT_providespackage	180
\XINT_modtrunc_aiszero	214, 216	\XINT_raw	189
\XINT_modtrunc_b	214	\XINT_rawz_A	190
\XINT_modtrunc_bneg	214, 215	\XINT_rawz_Ba	190
\XINT_modtrunc_bpos	214, 215	\XINT_rawz_Bb	190
\XINT_modtrunc_divbyzero	214, 216	\XINT_rawz_fork	190
\XINT_modtrunc_pos	215	\XINT_rep	205, 207, 208, 238, 242
\XINT_modtrunc_pos_a	215	\XINT_restorecatcodes_endinput	260
\XINT_mul_fork	198	\XINT_rez_A	194
\XINT_mul_out	246		
\XINT_num_cleanup	185		



\XINT_rez_AB .....	194, 198	\XINT_trunc_Hb .....	201
\XINT_rez_B .....	194	\XINT_trunc_sp_B .....	199
\XINT_rez_C .....	194	\XINT_trunc_sp_b .....	199, 202
\XINT_rez_D .....	194	\XINT_trunc_sp_C .....	199, 200
\XINT_rez_E .....	194	\XINT_trunc_sp_Ca .....	200
\XINT_rez_neg .....	194	\XINT_trunc_sp_Cb .....	200
\XINT_rez_zero .....	194	\XINT_trunc_sp_Cc .....	200
\XINT_round .....	201	\XINT_trunc_sp_Cd .....	200
\XINT_round_A .....	201, 202	\XINT_trunc_sp_D .....	199, 200
\XINT_round_a .....	202	\XINT_trunc_sp_E .....	200
\XINT_round_aa .....	201, 202	\XINT_trunc_sp_F .....	200, 201
\XINT_round_B .....	202	\XINT_trunc_zero .....	199, 202
\XINT_rsepyviii_end_A .....	204	\xint_UDsignfork .....	192, 193, 195, 197, 199, 200, 205, 207, 215, 216, 218, 222, 227, 240
\XINT_rsepyviii_end_B .....	204	\xint_UDsignsfork .....	219--221
\xint_secondofthree .....	181	\xint_UDXINTWfork .....	183, 184
\xint_secondoftwo .....	189, 234, 249	\xint_UDzerominusfork .....	
\XINT_sepandrev_andcount .....	204	..... 185, 194, 198, 202, 209, 210, 212, 222, 225--227, 232, 237, 238, 242, 244, 255, 257	
\XINT_sepbyviii_Z_end .....	204	\xint_UDzerosfork .....	196, 197, 221
\XINT_Sgn .....	182, 199, 202	\XINT_W .....	183--185
\XINT_sgn .....	223	\XINT_xtrunc_a .....	202
\XINT_sgnfrac_a .....	192	\XINT_xtrunc_b .....	202
\XINT_sgnfrac_b .....	192	\XINT_xtrunc_BisEight .....	203
\XINT_sgnfrac_N .....	192, 193	\XINT_xtrunc_BisFive .....	203
\XINT_sgnfrac_P .....	192, 193	\XINT_xtrunc_BisFour .....	203
\XINT_sgnfwover_a .....	193	\XINT_xtrunc_BisOne .....	203
\XINT_sgnfwover_b .....	193	\XINT_xtrunc_BisSmall .....	203
\XINT_sgnfwover_N .....	193, 194	\XINT_xtrunc_BisTwo .....	203
\XINT_sgnfwover_P .....	193, 194	\XINT_xtrunc_c .....	202
\XINT_signalcondition .....	182, 185, 196, 197, 214, 215, 222, 238, 245, 250, 254, 255	\XINT_xtrunc_d .....	202
\XINT_split_fromleft .....	205, 208, 227--229, 238, 256	\XINT_xtrunc_e .....	203, 204
\XINT_split_fromright_a .....	200	\XINT_xtrunc_finish .....	207
\XINT_split_xfork .....	206, 208	\XINT_xtrunc_finish_a .....	207
\XINT_sqrt_start .....	256	\XINT_xtrunc_I .....	204, 205
\xint_stop_atfirstofthree .....	187, 188	\XINT_xtrunc_I_a .....	205
\xint_stop_atfirstoftwo .....	188, 189, 192, 196, 258	\XINT_xtrunc_I_b .....	205
\xint_stop_atsecondofthree .....	187	\XINT_xtrunc_IA_c .....	205
\xint_stop_atsecondoftwo .....	188, 189, 196, 258	\XINT_xtrunc_IA_d .....	205, 206
\xint_stop_atthirdofthree .....	187, 188	\XINT_xtrunc_IA_xd .....	205
\XINT_T .....	183, 185	\XINT_xtrunc_IA_xe .....	205, 206
\XINT_tfrac_fork .....	198	\XINT_xtrunc_IAA_e .....	205
\XINT_tfrac_P .....	198	\XINT_xtrunc_IAB_e .....	205
\XINT_tfrac_zero .....	198	\XINT_xtrunc_IB_c .....	205, 206
\xint_thirdofthree .....	181	\XINT_xtrunc_IB_d .....	206
\XINT_trunc .....	199	\XINT_xtrunc_II .....	204, 206
\XINT_trunc_a .....	199	\XINT_xtrunc_II_a .....	206
\XINT_trunc_B .....	199	\XINT_xtrunc_II_b .....	206
\XINT_trunc_b .....	199, 202	\XINT_xtrunc_II_c .....	206
\XINT_trunc_C .....	199	\XINT_xtrunc_loop .....	206
\XINT_trunc_CE .....	199	\XINT_xtrunc_loop_a .....	206, 207
\XINT_trunc_D .....	199, 200	\XINT_xtrunc_loop_b .....	206
\XINT_trunc_E .....	199, 200	\XINT_xtrunc_prepare_a .....	202, 203
\XINT_trunc_F .....	200, 201	\XINT_xtrunc_prepare_b .....	203, 204
\XINT_trunc_G .....	199, 201, 202	\XINT_xtrunc_prepare_c .....	204
\XINT_trunc_H .....	201	\XINT_xtrunc_prepare_d .....	204
\XINT_trunc_Ha .....	201	\XINT_xtrunc_prepare_e .....	204
\XINT_trunc_Haa .....	201		

\XINT_xtrunc_prepare_f .....	204	\xintfac .....	212
\XINT_xtrunc_prepare_g .....	204	\xintFloat .....	224
\XINT_xtrunc_prepare_small .....	203	\xintfloat . 224, 231--236, 240, 244, 249, 253, 255	
\XINT_xtrunc_small_a .....	203, 204	\xintFloatAdd .....	233
\XINT_xtrunc_sp_e .....	203, 207	\xintfloatadd .....	233
\XINT_xtrunc_sp_I .....	207	\xintFloatBinomial .....	253
\XINT_xtrunc_sp_I_a .....	207	\xintfloatbinomial .....	253
\XINT_xtrunc_sp_IA_b .....	207	\xintFloatDiv .....	236
\XINT_xtrunc_sp_IA_c .....	207, 208	\xintfloatdiv .....	236
\XINT_xtrunc_sp_IAA .....	207, 208	\xintFloatE .....	257
\XINT_xtrunc_sp_IAB .....	207, 208	\xintfloate .....	257
\XINT_xtrunc_sp_IB_b .....	207, 208	\xintFloatFac .....	244
\XINT_xtrunc_sp_IB_c .....	208	\xintfloatfac .....	244
\XINT_xtrunc_sp_II .....	207, 208	\xintFloatIsInt .....	259
\XINT_xtrunc_transition .....	206, 207	\xintfloatisint .....	259
\XINT_xtrunc_zero .....	202	\xintFloatMul .....	235
\XINT_Z .....	184	\xintfloatmul .....	235
\XINT_zeroes_forviii .....	204	\xintFloatPFactorial .....	249
\xintAbs .....	222	\xintfloatpfactorial .....	249
\xintabs .....	218, 222	\xintFloatPow .....	236
\xintAdd .....	208, 234	\xintfloatpow .....	236
\xintadd .....	208, 210	\xintFloatPower .....	240
\xintBinomial .....	212	\xintfloatpower .....	240
\xintbinomial .....	212	\xintFloatSqrt .....	255
\xintCeil .....	191	\xintfloatsqrt .....	255
\xintceil .....	191	\xintFloatSub .....	234
\xintCmp .....	187, 188, 221	\xintfloatsub .....	234
\xintcmp .....	221	\xintFloor .....	191
\xintdecsplit .....	201	\xintfloor .....	191
\xintDecToString .....	190	\xintFrac .....	192
\xintdectostring .....	190	\xintfrac .....	192
\xintDenominator .....	191	\xintFwOver .....	193
\xintdenominator .....	191	\xintfwover .....	193
\XINTdigits .....	208, 224, 231, 233-- -237, 241, 242, 244, 249, 253, 255, 257--259	\xintGeq .....	217
\xintDigits .....	208	\xintgeq .....	217, 218
\xintDiv .....	213, 214, 254	\xintGt .....	187
\xintdiv .....	213	\xintgt .....	187
\xintDivFloor .....	214	\xintGtorEq .....	187
\xintdivfloor .....	214, 258	\xinthalf .....	241
\xintDivMod .....	215	\xintiCeil .....	191
\xintdivmod .....	215, 258	\xinticeil .....	191
\xintDivRound .....	214	\xintifCmp .....	187
\xintdivround .....	214	\xintifcmp .....	187
\xintDivTrunc .....	214	\xintifEq .....	188
\xintdivtrunc .....	214	\xintifeq .....	187, 188
\xintDouble .....	231, 241	\xintifFloatInt .....	258
\xintdouble .....	203	\xintiffloatint .....	258
\xintDSH .....	231	\xintifGt .....	188
\xintdsrr .....	241	\xintifgt .....	187, 188
\xintE .....	194	\xintifInt .....	196
\xinte .....	194	\xintifint .....	196
\xintEq .....	187	\xintiFloor .....	191
\xinteq .....	187	\xintifloor .....	191, 214
\xintEven .....	187	\xintifLt .....	188
\xinteven .....	187	\xintiflt .....	187, 188
\xintFac .....	212	\xintifNotZero .....	188
		\xintifnotzero .....	188

<code>\xintifOdd</code> .....	189	<code>\XINTinFloatPower</code> .....	240
<code>\xintifodd</code> .....	189	<code>\XINTinfloatpower</code> .....	240
<code>\xintifOne</code> .....	188	<code>\XINTinFloatPowerH</code> .....	241
<code>\xintifone</code> .....	188	<code>\XINTinfloatpowerh</code> .....	241
<code>\xintifSgn</code> .....	187	<code>\XINTinFloatS</code> .....	226
<code>\xintifsgn</code> .....	187	<code>\XINTinfloatS</code> .....	226, 233--237, 240, 241
<code>\xintifZero</code> .....	188	<code>\XINTinFloatSqrt</code> .....	241, 255
<code>\xintifzero</code> .....	188	<code>\XINTinfloatsqrt</code> .....	255
<code>\xintiiadd</code> .....	198, 210	<code>\XINTinFloatSub</code> .....	234
<code>\xintiiCmp</code> .....	217	<code>\XINTinfloatsub</code> .....	234
<code>\xintiidivision</code> .....	216	<code>\XINTinRandomFloatS</code> .....	259
<code>\xintiiDivRound</code> .....	256	<code>\XINTinrandomfloatS</code> .....	259
<code>\xintiiE</code> .....	217	<code>\XINTinRandomFloatSdigits</code> .....	259
<code>\xintiie</code> .....	190	<code>\XINTinRandomFloatSixteen</code> .....	260
<code>\xintiiiflt</code> .....	231	<code>\xintinum</code> .....	256
<code>\xintiiifZero</code> .....	190	<code>\xintInv</code> .....	222
<code>\xintiiMul</code> .....	210, 217, 231, 235	<code>\xintinv</code> .....	222
<code>\xintiimul</code> .....	203, 209, 211, 213, 217, 218, 221, 239, 244	<code>\xintipfactorial</code> .....	213
<code>\xintiiOdd</code> .....	243	<code>\xintipow</code> .....	211
<code>\xintiiopp</code> .....	191, 198, 215, 216	<code>\xintiRound</code> .....	201
<code>\xintiiPow</code> .....	212	<code>\xintiround</code> .....	201, 214
<code>\xintiiQuo</code> .....	209, 217	<code>\xintIrr</code> .....	195
<code>\xintiiquo</code> .....	191, 196, 209, 217, 229	<code>\xintirr</code> .....	195
<code>\xintiiRem</code> .....	196	<code>\xintIsInt</code> .....	197
<code>\xintiiRem</code> .....	198, 215, 217	<code>\xintisint</code> .....	197
<code>\xintiiSqr</code> .....	211, 239, 243, 244	<code>\xintIsNotZero</code> .....	187
<code>\xintiiSub</code> .....	256	<code>\xintisnotzero</code> .....	187
<code>\xintInc</code> .....	231	<code>\xintIsOne</code> .....	188, 217
<code>\xintinc</code> .....	230, 231	<code>\xintisone</code> .....	217
<code>\XINTinFloat</code> .....	225, 258, 259	<code>\xintIsZero</code> .....	187
<code>\XINTinfloat</code> .....	224, 225, 233--235, 237, 242, 244, 249, 253--255, 257, 258	<code>\xintiszero</code> .....	187
<code>\XINTinFloatAdd</code> .....	233	<code>\xintiTrunc</code> .....	199
<code>\XINTinfloatadd</code> .....	233	<code>\xintitrunc</code> .....	199, 201, 241
<code>\XINTinFloatBinomial</code> .....	253	<code>\xintJrr</code> .....	197
<code>\XINTinfloatbinomial</code> .....	253	<code>\xintjrr</code> .....	197
<code>\XINTinFloatDiv</code> .....	236	<code>\xintLDg</code> .....	187, 241, 243
<code>\XINTinfloatdiv</code> .....	236	<code>\xintLen</code> .....	181
<code>\XINTinFloatDivFloor</code> .....	258	<code>\xintlen</code> .....	181
<code>\XINTinfloatdivfloor</code> .....	258	<code>\xintLength</code> .....	205--208, 218, 222, 226, 228, 237, 238, 242
<code>\XINTinFloatDivMod</code> .....	258	<code>\xintlength</code> .....	201
<code>\XINTinfloatdivmod</code> .....	258	<code>\xintLt</code> .....	187
<code>\XINTinFloatE</code> .....	257	<code>\xintlt</code> .....	187
<code>\XINTinfloate</code> .....	257	<code>\xintLtorEq</code> .....	187
<code>\XINTinFloatFac</code> .....	244	<code>\xintMax</code> .....	219
<code>\XINTinfloatfac</code> .....	244	<code>\xintmax</code> .....	219
<code>\XINTinFloatFracdigits</code> .....	233	<code>\xintMaxof</code> .....	219
<code>\XINTinfloatfracdigits</code> .....	233	<code>\xintmaxof</code> .....	219
<code>\XINTinFloatMod</code> .....	258	<code>\xintMin</code> .....	220
<code>\XINTinfloatmod</code> .....	258	<code>\xintmin</code> .....	220
<code>\XINTinFloatMul</code> .....	235	<code>\xintMinof</code> .....	220
<code>\XINTinfloatmul</code> .....	235	<code>\xintminof</code> .....	220
<code>\XINTinFloatPFactorial</code> .....	249	<code>\xintMod</code> .....	216, 258
<code>\XINTinfloatpfactorial</code> .....	249	<code>\xintmod</code> .....	216
<code>\XINTinFloatPow</code> .....	237	<code>\xintModTrunc</code> .....	214
<code>\XINTinfloatpow</code> .....	237	<code>\xintmodtrunc</code> .....	214
		<code>\xintMul</code> .....	210



## 12.8 Index of **xintseries**

<b>A</b>	
\aftergroup .....	261, 262
<b>C</b>	
\catcode .....	261
\csname .....	261
<b>E</b>	
\endcsname .....	261
\endinput .....	261, 262
\endlinechar .....	261
<b>I</b>	
\immediate .....	261
<b>N</b>	
\numexpr .....	261--269
<b>P</b>	
\ProvidesPackage .....	261, 262
<b>R</b>	
\romannumeral .....	262--269
<b>W</b>	
\w .....	261
\write .....	261
<b>X</b>	
\x .....	261
\xint: .....	267--269
\xint_afterfi .....	262--269
\xint_c_i .....	266--268
\XINT_flpowseries .....	268
\XINT_flpowseries_chkopt .....	267
\XINT_flpowseries_dont_i .....	268
\XINT_flpowseries_dont_ii .....	268
\XINT_flpowseries_exit_i .....	268, 269
\XINT_flpowseries_exit_ii .....	269
\XINT_flpowseries_loop_i .....	268
\XINT_flpowseries_loop_ii .....	268, 269
\XINT_flpowseries_loop_pre .....	268, 269
\XINT_flpowseries_noopt .....	268
\XINT_flpowseries_opt .....	267, 268
\XINT_flpowseriesx .....	269
\XINT_flpowseriesx_chkopt .....	269
\XINT_flpowseriesx_noopt .....	269
\XINT_flpowseriesx_opt .....	269
\XINT_flpowseriesx_pre .....	269
\XINT_fppowseries .....	266
\XINT_fppowseries_dont_i .....	266
\XINT_fppowseries_dont_ii .....	266
\XINT_fppowseries_exit_i .....	266, 267
\XINT_fppowseries_exit_ii .....	267
\XINT_fppowseries_loop_i .....	266
\XINT_fppowseries_loop_ii .....	266, 267
\XINT_fppowseries_loop_pre .....	266, 267
\XINT_fppowseriesx .....	267
\XINT_fppowseriesx_pre .....	267
\XINT_iseries .....	262
\XINT_iseries_exit .....	263
\XINT_iseries_loop .....	263
\XINT_powseries .....	263
\XINT_powseries_exit_i .....	263
\XINT_powseries_exit_ii .....	264
\XINT_powseries_loop_i .....	263, 264
\XINT_powseries_loop_ii .....	263
\XINT_powseriesx .....	264
\XINT_powseriesx_pre .....	264
\XINT_providespackage .....	262
\XINT_ratseries .....	264
\XINT_ratseries_exit_i .....	265
\XINT_ratseries_exit_ii .....	265
\XINT_ratseries_exit_iii .....	265
\XINT_ratseries_loop .....	265, 266
\XINT_ratseriesx .....	265
\XINT_ratseriesx_pre .....	265, 266
\XINT_restorecatcodes_endinput .....	269
\XINT_series .....	262
\XINT_series_exit .....	262
\XINT_series_loop .....	262
\xintadd .....	262, 263, 265
\XINTdigits .....	268, 269
\xintfloat .....	268
\xintfloatadd .....	269
\xintFloatPowerSeries .....	267
\xintfloatpowerseries .....	267
\xintFloatPowerSeriesX .....	269
\xintfloatpowerseriesx .....	269
\xintFxpPtPowerSeries .....	266
\xintfxptpowerseries .....	266
\xintFxpPtPowerSeriesX .....	267
\xintfxptpowerseriesx .....	267
\xintiiadd .....	263, 267
\XINTinfloatadd .....	268
\XINTinfloatmul .....	268, 269
\XINTinfloatpow .....	268, 269
\xintiSeries .....	262
\xintiseries .....	262
\xintiTrunc .....	267
\xintitrunc .....	266
\xintMul .....	265--267
\xintmul .....	263--265
\xintPow .....	264, 266, 267
\xintPowerSeries .....	263
\xintpowerseries .....	263
\xintPowerSeriesX .....	264
\xintpowerseriesx .....	264

xintseries

**Y**

**Z**

## 12.9 Index of [xintcfrac](#)

<b>A</b>	
<code>\aftergroup</code> .....	270, 271
<b>C</b>	
<code>\catcode</code> .....	270
<code>\cfrac</code> .....	272--275
<code>\csname</code> .....	270, 271, 273, 274
<b>E</b>	
<code>\endcsname</code> .....	270, 271, 273, 274
<code>\endinput</code> .....	270, 271
<code>\endlinechar</code> .....	270
<b>H</b>	
<code>\hfill</code> .....	271--274
<b>I</b>	
<code>\immediate</code> .....	270
<b>K</b>	
<code>\krof</code> .....	278, 279
<b>N</b>	
<code>\numexpr</code> .....	270, 287--291
<b>P</b>	
<code>\ProvidesPackage</code> .....	271
<b>R</b>	
<code>\romannumeral</code> .....	271--291
<b>W</b>	
<code>\w</code> .....	270, 271
<code>\write</code> .....	270
<b>X</b>	
<code>\x</code> .....	270, 271
<code>\xint:</code> .....	271--274, 280--286
<code>\xint_afterfi</code> .....	287--290
<code>\xint_c_</code> .....	287, 288
<code>\xint_c_i</code> .....	289--291
<code>\XINT_cfrac_A</code> .....	271, 272
<code>\XINT_cfrac_B</code> .....	272
<code>\XINT_cfrac_C</code> .....	272
<code>\XINT_cfrac_D</code> .....	272
<code>\XINT_cfrac_end</code> .....	272
<code>\XINT_cfrac_end_b</code> .....	272
<code>\XINT_cfrac_integer</code> .....	272
<code>\XINT_cfrac_loop_a</code> .....	272
<code>\XINT_cfrac_loop_d</code> .....	272
<code>\XINT_cfrac_loop_e</code> .....	272
<code>\xint_cfrac_loop_exit</code> .....	272
<code>\XINT_cfrac_loop_f</code> .....	272
<code>\XINT_cfrac_noopt</code> .....	271
<code>\XINT_cfrac_opt_a</code> .....	271
<code>\XINT_cfrac_opt_b</code> .....	271
<code>\XINT_cfrac_optc</code> .....	271
<code>\XINT_cfrac_optl</code> .....	271
<code>\XINT_cfrac_optr</code> .....	272
<code>\XINT_cfrac_T</code> .....	272
<code>\XINT_cntcs</code> .....	289
<code>\XINT_cntcs_exit</code> .....	289
<code>\XINT_cntcs_exit_b</code> .....	289
<code>\XINT_cntcs_loop</code> .....	289
<code>\XINT_cntf</code> .....	287
<code>\XINT_cntf_exit</code> .....	288
<code>\XINT_cntf_loop</code> .....	287, 288
<code>\XINT_cntgc</code> .....	290
<code>\XINT_cntgc_exit</code> .....	290
<code>\XINT_cntgc_exit_b</code> .....	290
<code>\XINT_cntgc_loop</code> .....	290
<code>\XINT_cstc_end</code> .....	291
<code>\XINT_cstc_loop_a</code> .....	291
<code>\XINT_cstc_loop_b</code> .....	291
<code>\XINT_cstc_prep</code> .....	291
<code>\XINT_ctcv_end</code> .....	283, 284
<code>\XINT_ctcv_loop_a</code> .....	283, 284
<code>\XINT_ctcv_loop_b</code> .....	283
<code>\XINT_ctcv_loop_c</code> .....	283
<code>\XINT_ctcv_loop_d</code> .....	283
<code>\XINT_ctcv_loop_e</code> .....	283
<code>\XINT_ctcv_loop_f</code> .....	283, 284
<code>\XINT_ctcv_loop_g</code> .....	284
<code>\XINT_ctcv_prep</code> .....	283
<code>\XINT_ctf_end</code> .....	279, 280
<code>\XINT_ctf_loop_a</code> .....	279, 280
<code>\XINT_ctf_loop_b</code> .....	279
<code>\XINT_ctf_loop_c</code> .....	280
<code>\XINT_ctf_loop_d</code> .....	280
<code>\XINT_ctf_loop_e</code> .....	280
<code>\XINT_ctf_prep</code> .....	279
<code>\XINT_div_prepare</code> .....	272, 275--278
<code>\XINT_fgtc_a</code> .....	277
<code>\XINT_fgtc_b</code> .....	277
<code>\XINT_fgtc_c</code> .....	277
<code>\XINT_fgtc_d</code> .....	277, 278
<code>\XINT_fgtc_da</code> .....	277
<code>\XINT_fgtc_e</code> .....	277
<code>\XINT_fgtc_f</code> .....	277
<code>\XINT_fgtc_g</code> .....	277
<code>\XINT_fgtc_h</code> .....	277
<code>\xint_firstofone</code> .....	277
<code>\XINT_ftc_A</code> .....	275
<code>\XINT_ftc_B</code> .....	275
<code>\XINT_ftc_C</code> .....	275
<code>\XINT_ftc_D</code> .....	275
<code>\XINT_ftc_integer</code> .....	275
<code>\XINT_ftc_loop_a</code> .....	275, 276
<code>\XINT_ftc_loop_d</code> .....	275
<code>\XINT_ftc_loop_e</code> .....	275

<code>\xint_ftc_loop_exit</code>	276	<code>\XINT_gctcv_loop_i</code>	285
<code>\XINT_ftc_loop_f</code>	276	<code>\XINT_gctcv_loop_j</code>	285, 286
<code>\XINT_ftcc_A</code>	278	<code>\XINT_gctcv_loop_k</code>	286
<code>\XINT_ftcc_B</code>	278	<code>\XINT_gctcv_loop_l</code>	286
<code>\XINT_ftcc_C</code>	278	<code>\XINT_gctcv_loop_m</code>	286
<code>\XINT_ftcc_D</code>	278	<code>\XINT_gctcv_prep</code>	284
<code>\XINT_ftcc_En</code>	278	<code>\XINT_gctf_end</code>	281, 282
<code>\XINT_ftcc_end</code>	279	<code>\XINT_gctf_loop_a</code>	281, 282
<code>\XINT_ftcc_Ep</code>	278	<code>\XINT_gctf_loop_b</code>	281
<code>\XINT_ftcc_integer</code>	278	<code>\XINT_gctf_loop_c</code>	281
<code>\XINT_ftcc_loop_a</code>	278, 279	<code>\XINT_gctf_loop_d</code>	281
<code>\XINT_ftcc_loop_b</code>	278	<code>\XINT_gctf_loop_e</code>	281
<code>\XINT_ftcc_loop_c</code>	278, 279	<code>\XINT_gctf_loop_f</code>	281
<code>\XINT_ftcc_loop_d</code>	279	<code>\XINT_gctf_loop_g</code>	281
<code>\XINT_ftcc_loop_N</code>	279	<code>\XINT_gctf_loop_h</code>	281
<code>\XINT_ftcc_loop_P</code>	279	<code>\XINT_gctf_loop_i</code>	282
<code>\XINT_ftcx_A</code>	276	<code>\XINT_gctf_loop_j</code>	282
<code>\XINT_ftcx_B</code>	276	<code>\XINT_gctf_prep</code>	281
<code>\XINT_ftcx_C</code>	276	<code>\XINT_gctgc_end</code>	291, 292
<code>\XINT_ftcx_D</code>	276	<code>\XINT_gctgc_end_b</code>	292
<code>\XINT_ftcx_integer</code>	276	<code>\XINT_gctgc_loop_a</code>	291
<code>\XINT_ftcx_loop_a</code>	276	<code>\XINT_gctgc_loop_b</code>	291, 292
<code>\XINT_ftcx_loop_d</code>	276	<code>\XINT_gctgc_loop_c</code>	291
<code>\XINT_ftcx_loop_e</code>	276	<code>\XINT_gctgc_start</code>	291
<code>\xint_ftcx_loop_exit</code>	276	<code>\XINT_gctgcx_end</code>	275
<code>\XINT_ftcx_loop_f</code>	276	<code>\XINT_gctgcx_loop_a</code>	275
<code>\XINT_gcfrac</code>	273	<code>\XINT_gctgcx_loop_b</code>	275
<code>\XINT_gcfrac_end</code>	273	<code>\XINT_gctgcx_start</code>	275
<code>\XINT_gcfrac_end_b</code>	273	<code>\XINT_ggcfrac</code>	274
<code>\XINT_gcfrac_endloop</code>	273	<code>\XINT_ggcfrac_end</code>	274
<code>\XINT_gcfrac_enter</code>	273	<code>\XINT_ggcfrac_end_b</code>	274, 275
<code>\XINT_gcfrac_loop</code>	273	<code>\XINT_ggcfrac_endloop</code>	274
<code>\XINT_gcfrac_noopt</code>	273	<code>\XINT_ggcfrac_enter</code>	274
<code>\XINT_gcfrac_opt_a</code>	272	<code>\XINT_ggcfrac_loop</code>	274
<code>\XINT_gcfrac_opt_b</code>	273	<code>\XINT_ggcfrac_noopt</code>	274
<code>\XINT_gcfrac_optc</code>	273	<code>\XINT_ggcfrac_opt_a</code>	274
<code>\XINT_gcfrac_optl</code>	273	<code>\XINT_ggcfrac_opt_b</code>	274
<code>\XINT_gcfrac_optr</code>	273	<code>\XINT_ggcfrac_optc</code>	274
<code>\XINT_gcfrac_T</code>	273	<code>\XINT_ggcfrac_optl</code>	274
<code>\XINT_gcfrac_U</code>	273	<code>\XINT_ggcfrac_optr</code>	274
<code>\XINT_gcntf</code>	288	<code>\XINT_ggcfrac_T</code>	274
<code>\XINT_gcntf_exit</code>	288, 289	<code>\XINT_ggcfrac_U</code>	274
<code>\XINT_gcntf_loop</code>	288, 289	<code>\xint_gob_til_exclam</code>	273--275, 279--286, 291
<code>\XINT_gcntgc</code>	290	<code>\xint_gob_til_Z</code>	272
<code>\XINT_gcntgc_exit</code>	290, 291	<code>\xint_gob_til_zero</code>	272, 275, 276
<code>\XINT_gcntgc_exit_b</code>	291	<code>\xint_gobble_ii</code>	288, 289
<code>\XINT_gcntgc_loop</code>	290, 291	<code>\xint_gobble_iii</code>	277
<code>\XINT_gcntgc_loop_b</code>	291	<code>\XINT_icstcv_end</code>	284
<code>\XINT_gctcv_end</code>	285, 286	<code>\XINT_icstcv_loop_a</code>	284
<code>\XINT_gctcv_loop_a</code>	285, 286	<code>\XINT_icstcv_loop_b</code>	284
<code>\XINT_gctcv_loop_b</code>	285	<code>\XINT_icstcv_loop_c</code>	284
<code>\XINT_gctcv_loop_c</code>	285	<code>\XINT_icstcv_loop_d</code>	284
<code>\XINT_gctcv_loop_d</code>	285	<code>\XINT_icstcv_loop_e</code>	284
<code>\XINT_gctcv_loop_e</code>	285	<code>\XINT_icstcv_prep</code>	284
<code>\XINT_gctcv_loop_f</code>	285	<code>\XINT_icstf_end</code>	280
<code>\XINT_gctcv_loop_g</code>	285	<code>\XINT_icstf_loop_a</code>	280
<code>\XINT_gctcv_loop_h</code>	285	<code>\XINT_icstf_loop_b</code>	280



[illegible]

## 12.10 Index of xintexpr

Particularly for this package, there are quite a few macros which are defined via constructs such as `\expandafter \def \csname ... \endcsname`, and their indexing needs some extra mark-up which is yet to be added to the commented source code.

Symbols	
<code>\%</code> .....	378
A	
<code>\aftergroup</code> .....	295
C	
<code>\catcode</code> .....	295, 296, 304, 310--314, 321, 331, 333--339, 347, 350, 369, 371, 373, 378--380
<code>\count</code> .....	302
<code>\csname</code> .....	295, 296, 298, 299, 303, 304, 307, 311--321, 324, 327--338, 340--350, 355--366, 369--371, 377
D	
<code>\detokenize</code> .....	330, 332, 334, 373, 374
<code>\dimen</code> .....	302
<code>\dimexpr</code> .....	302
<code>\do</code> .....	312--314, 318--321, 329, 331, 369
<code>\dp</code> .....	302
E	
<code>\edef</code> .....	321, 333--335, 337, 369, 370, 378, 379
<code>\endcsname</code> .....	295, 296, 298, 299, 303--308, 311--321, 324, 327--338, 340--350, 355--366, 369--371, 377
<code>\endinput</code> .....	295
<code>\endlinechar</code> .....	295, 378
<code>\escapechar</code> .....	378
<code>\everyeof</code> .....	378
F	
<code>\fontchardp</code> .....	302
<code>\fontcharht</code> .....	302
<code>\fontcharic</code> .....	302
<code>\fontcharwd</code> .....	302
<code>\fontdimen</code> .....	302
G	
<code>\glueexpr</code> .....	302
H	
<code>\ht</code> .....	302
I	
<code>\if</code> .....	296, 301--311, 316, 321--323, 330, 340--342, 344--355, 362, 366--368, 373--375, 377
<code>\ifcat</code> .....	302, 304--311
<code>\ifcsname</code> .....	311, 331, 333, 335
<code>\ifxintexprsafecatcodes</code> .....	379
<code>\ifxintglobaldefs</code> ..	334, 335, 337, 369--371, 379
<code>\ifxintverbose</code> .....	334, 335, 337, 369--371, 379
<code>\immediate</code> .....	295
K	
<code>\krof</code> ....	312, 314, 317, 320--323, 327--329, 373
L	
<code>\lccode</code> .....	303, 334
<code>\lowercase</code> .....	303, 334
M	
<code>\meaning</code> .....	369, 371, 378, 379
N	
<code>\newif</code> .....	379
<code>\noexpand</code> .....	297, 321, 323, 334, 337, 377--379
<code>\numexpr</code> .....	295, 296, 299, 301, 302, 305, 306, 322--326, 334, 341, 344, 346, 348--350, 369
P	
<code>\protected</code> .....	297
<code>\ProvidesPackage</code> .....	295, 296
R	
<code>\repeat</code> .....	334
<code>\romannumeral</code> .....	296--302, 304--312, 314--318, 320, 321, 323--329, 331, 332, 334, 338--355, 358--362, 365--369, 371--374, 377, 378
S	
<code>\scantokens</code> .....	378
T	
<code>\t</code> .....	295
<code>\thexintboolexpr</code> .....	300
<code>\thexintexpr</code> .....	298
<code>\thexintfloatexpr</code> .....	298
<code>\thexintiexpr</code> .....	298
<code>\thexintiexpr</code> .....	298
<code>\to</code> .....	334
<code>\toks</code> .....	370, 378
U	
<code>\unexpanded</code> .....	369
<code>\unless</code> .....	322, 323, 379
W	
<code>\W</code> .....	301
<code>\w</code> .....	295
<code>\write</code> .....	295
X	
<code>\x</code> .....	295
<code>\xint</code> .....	369, 370
<code>\xint:</code> .....	302, 352--354
<code>\XINT:expr:macrofunc</code> .....	371, 376

\XINT:expr:one:and:opt	354, 358--360, 376	\XINT_allexpr_subx	343
\XINT:expr:one:or:two:nums	376	\XINT_andof:_a	350
\XINT:expr:randrange	361, 362	\XINT_andof:_c	350
\XINT:expr:tacitzeroifonearg	354, 359, 376	\XINT_andof:_e	350
\XINT:expr:totwo	355, 374	\XINT_andof:_no	350
\XINT:expr:two:to:one	355, 360--362, 376	\XINT_applyon::_a	373
\XINT:expr:two:to:two	355, 360, 376	\XINT_applyon::_b	373
\XINT:expr:userfunc	369, 376	\XINT_applyon::_c	373
\XINT:flexpr:two:to:one	355, 360, 361, 376	\XINT_applyon::_d	373
\XINT:flexpr:two:to:two	355, 360, 376	\XINT_applyon::_e	373
\XINT:iiexpr:one:or:two:	376	\XINT_applyon::_end	373
\XINT:iiexpr:randrange	361, 362	\XINT_applyon::_a	372, 373
\XINT:iiexpr:tacitzeroifonearg	355, 359, 376	\XINT_applyon::_b	373
\XINT:NE:csv	374, 376	\XINT_applyon::_c	373
\XINT:NE:LApply::csv	373	\XINT_applyon::_d	373
\XINT:NE:one	373, 374, 376	\XINT_applyon::_e	373
\XINT:NE:one_a	373	\XINT_applyon::_end	373
\XINT:NE:oneopt	374	\XINT_boolexpr_done	300
\XINT:NE:oneopt_a	374	\XINT_boolexpr_print	297, 300, 377
\XINT:NE:oneopt_b	374	\xint_Bye	324
\XINT:NE:oneopt_fork	374	\xint_bye	324, 330, 339, 340
\XINT:NE:RApply::csv	373	\xint_c	301, 311, 313, 322, 323, 340
\XINT:NE:RLApply::csv	373	\xint_c_i	306, 313, 321, 325, 326, 340, 341, 344, 346, 348--350, 369
\XINT:NE:two	373--376	\xint_c_ii	313, 320, 321, 329
\XINT:NE:two_	373, 374, 376	\xint_c_iii	321
\XINT:NE:two_a	373	\xint_c_ix	304, 306, 307, 309, 310
\XINT:NE:two_b	373	\xint_c_mone	339
\XINT:NE:two_fork_dd	373	\xint_c_vii	331
\XINT:NE:two_fork_nn	373	\xint_c_viii	314, 324
\XINT:NEhook:csv	300, 362--365, 376	\xint_c_x	305, 330
\XINT:NEhook:one	300, 314, 321, 330, 355--359, 362, 376	\xint_c_xviii	303, 304, 310, 313, 339, 341, 342
\XINT:NEhook:two	300, 315, 316, 342, 376	\XINT_Cmp	296, 328, 367, 368
\XINT:newexpr:insertdollar	374	\XINT_csv::_a	301
\XINT:newexpr:macrofunc	371, 376	\XINT_csv::_b	301
\XINT:newexpr:macrofunc:a	371	\XINT_csv::_c	301
\XINT:newexpr:one:and:opt	374, 376	\XINT_csv::_d	301
\XINT:newexpr:one:or:two:nums	376	\xint_ddfork	373
\XINT:newexpr:tacitzeroifonearg	374, 376	\xint_dothis	301--311, 316, 321, 323, 331, 340--342, 344--350, 373--375
\XINT:newexpr:two:to:one	374--376	\xint_exchangetwo_keepbraces	317
\XINT:newexpr:two:to:two	374, 376	\XINT_expr:_unlock	345, 346, 349, 350
\XINT:newexpr:userfunc	369, 376	\XINT_expr_binop_inline	316--318
\XINT:newflexpr:two:to:one	375, 376	\XINT_expr_binop_inline_a	316
\XINT:newflexpr:two:to:two	374, 376	\XINT_expr_binop_inline_b	316
\XINT:newiiexpr:one:or:two:	376	\XINT_expr_binop_inline_c	316
\XINT:newiiexpr:tacitzeroifonearg	374, 376	\XINT_expr_binop_inline_d	316
\XINT_	369	\XINT_expr_binop_inline_e	316
\XINT::_:end	301, 302	\XINT_expr_binop_inline_end	316
\xint::_:end	301	\XINT_expr_binopwrđ	311
\xint_afterfi	308, 311, 312, 315, 317, 318, 320, 321, 323, 329, 351, 366--368	\XINT_expr_countetc	302
\XINT_allexpr_iter	345	\XINT_expr_defbin_b	315, 316
\XINT_allexpr_iterr	348	\XINT_expr_defbin_c	314--316
\XINT_allexpr_opx	342	\XINT_expr_deflistopl_b	318
\XINT_allexpr_rrseq	346	\XINT_expr_deflistopl_c	318
\XINT_allexpr_rseq	343	\XINT_expr_deflistopr_b	317
\XINT_allexpr_seqx	340	\XINT_expr_deflistopr_c	317

\XINT_expr_defuserfunc .....	369	\XINT_expr_func_rrseq .....	346
\XINT_expr_defvar .....	334, 335	\XINT_expr_func_rseq .....	343
\XINT_expr_defvar_getname .....	334	\XINT_expr_func_seqx .....	340
\XINT_expr_defvar_one .....	333, 334	\XINT_expr_func_sgn .....	356
\XINT_expr_done .....	297, 312, 313	\XINT_expr_func_sqr .....	357
\XINT_expr_extra_) .....	312	\XINT_expr_func_sqrt .....	359
\XINT_expr_foundend .....	311, 314, 337	\XINT_expr_func_subx .....	343
\XINT_expr_foundop .....	311, 312	\XINT_expr_func_trunc .....	359
\XINT_expr_foundop_a .....	311	\XINT_expr_func_unknown .....	331, 332
\XINT_expr_func .....	309, 310	\XINT_expr_func_xor .....	364
\XINT_expr_func_! .....	357	\XINT_expr_getnext .....	298, 302, 303, 312, 314, 317, 318, 320, 328--330
\XINT_expr_func_? .....	357	\XINT_expr_getnext_a .....	302
\XINT_expr_func_@@ .....	338	\XINT_expr_getnextfork .....	302, 303
\XINT_expr_func_@@@ .....	338	\XINT_expr_getop .....	302--304, 307, 310--312, 314, 330--333, 340, 342, 343, 345, 347, 349
\XINT_expr_func_@@@ .....	338	\XINT_expr_getop_a .....	310
\XINT_expr_func_abs .....	355, 356	\XINT_expr_getop_b .....	311
\XINT_expr_func_all .....	364	\XINT_expr_gobz_a .....	304
\XINT_expr_func_any .....	364	\XINT_expr_gobz_scandec_a .....	305
\XINT_expr_func_binomial .....	360	\XINT_expr_gobz_scandec_b .....	305, 306
\XINT_expr_func_break .....	332	\XINT_expr_gobz_scandec_c .....	306
\XINT_expr_func_ceil .....	356	\XINT_expr_gobz_scandec_endbycs .....	306
\XINT_expr_func_divmod .....	360	\XINT_expr_gobz_scanint_b .....	304, 305
\XINT_expr_func_even .....	357, 358	\XINT_expr_gobz_scanint_c .....	305
\XINT_expr_func_factorial .....	358	\XINT_expr_gobz_scanint_d .....	305
\XINT_expr_func_first .....	365	\XINT_expr_gobz_scanint_endbycs .....	305
\XINT_expr_func_float .....	359, 360	\XINT_expr_gobz_startdec_a .....	305
\XINT_expr_func_floor .....	356	\XINT_expr_gotnil .....	304
\XINT_expr_func_frac .....	356	\XINT_expr_inhex .....	296
\XINT_expr_func_gcd .....	362	\XINT_expr_iter:_A .....	345, 346
\XINT_expr_func_if .....	365	\XINT_expr_iter:_a .....	345
\XINT_expr_func_ifint .....	365	\XINT_expr_iter:_aa .....	345
\XINT_expr_func_ifone .....	366	\XINT_expr_iter:_Abort .....	346
\XINT_expr_func_ifsgn .....	366	\XINT_expr_iter:_abort .....	346
\XINT_expr_func_isint .....	358	\XINT_expr_iter:_b .....	345, 346
\XINT_expr_func_ison .....	358	\XINT_expr_iter:_Break .....	346
\XINT_expr_func_iter .....	345	\XINT_expr_iter:_break .....	346
\XINT_expr_func_iterr .....	348	\XINT_expr_iter:_c .....	345
\XINT_expr_func_last .....	365	\XINT_expr_iter:_D .....	346
\XINT_expr_func_lcm .....	363	\XINT_expr_iter:_d .....	345
\XINT_expr_func_len .....	364, 365	\XINT_expr_iter:_E .....	346
\XINT_expr_func_max .....	363	\XINT_expr_iter:_end .....	345
\XINT_expr_func_min .....	363	\XINT_expr_iter:_Goon .....	346
\XINT_expr_func_mod .....	360	\XINT_expr_iter:_goon .....	346
\XINT_expr_func_not .....	357	\XINT_expr_iter:_noop .....	345
\XINT_expr_func_num .....	355	\XINT_expr_iter:_Omit .....	346
\XINT_expr_func_nuple .....	358	\XINT_expr_iter:_omit .....	346
\XINT_expr_func_odd .....	357	\XINT_expr_iterr:_A .....	349
\XINT_expr_func_opxadd .....	342	\XINT_expr_iterr:_aa .....	349
\XINT_expr_func_opxmud .....	342	\XINT_expr_iterr:_Abort .....	350
\XINT_expr_func_pfactorial .....	361	\XINT_expr_iterr:_abort .....	349
\XINT_expr_func_preduce .....	355	\XINT_expr_iterr:_b .....	349
\XINT_expr_func_quo .....	362	\XINT_expr_iterr:_Break .....	350
\XINT_expr_func_randrange .....	361	\XINT_expr_iterr:_break .....	349
\XINT_expr_func_reduce .....	355	\XINT_expr_iterr:_c .....	349
\XINT_expr_func_rem .....	362		
\XINT_expr_func_reversed .....	365		
\XINT_expr_func_round .....	359		

\XINT_expr_iterr:_D .....	349, 350	\XINT_expr_op_? .....	330
\XINT_expr_iterr:_d .....	349	\XINT_expr_op_?? .....	330
\XINT_expr_iterr:_E .....	349, 350	\XINT_expr_op_?a .....	330
\XINT_expr_iterr:_end .....	349	\XINT_expr_op_?checka .....	330
\XINT_expr_iterr:_Goon .....	349, 350	\XINT_expr_op_?checkb .....	330
\XINT_expr_iterr:_goon .....	349	\XINT_expr_op_?checkc .....	330
\XINT_expr_iterr:_noop .....	349	\XINT_expr_op_[ .....	331
\XINT_expr_iterr:_Omit .....	349, 350	\XINT_expr_op_[[: .....	321
\XINT_expr_iterr:_omit .....	349	\XINT_expr_op__ .....	310, 333
\XINT_expr_iterrx .....	348	\XINT_expr_precedence_ .....	311, 334, 337, 338
\XINT_expr_iterrry .....	348, 349	\XINT_expr_precedence_! .....	330
\XINT_expr_itory .....	345	\XINT_expr_precedence_!? .....	313
\XINT_expr_itery .....	345	\XINT_expr_precedence_:] .....	321
\XINT_expr_itself_] [ .....	321	\XINT_expr_precedence_? .....	311, 330
\XINT_expr_lockit .....	296, 334, 344, 346--349	\XINT_expr_precedence_[ .....	331
\XINT_expr_lockscan .....	296	\XINT_expr_precedence_] [: .....	321
\XINT_expr_macrofunc_ .....	371	\XINT_expr_precedence_a .....	313
\XINT_expr_makedummy .....	336, 337	\XINT_expr_print .....	297, 377
\XINT_expr_missing_) .....	312, 313	\XINT_expr_redefinemacros .....	376, 378
\XINT_expr_newfunction .....	370, 371	\XINT_expr_redefineprints .....	377
\XINT_expr_onliteral_@ .....	338	\XINT_expr_rrseq:_A .....	347, 348
\XINT_expr_onliteral_` .....	304	\XINT_expr_rrseq:_a .....	347
\XINT_expr_onliteral_add .....	341	\XINT_expr_rrseq:_aa .....	347
\XINT_expr_onliteral_add_f .....	341	\XINT_expr_rrseq:_Abort .....	348
\XINT_expr_onliteral_bool .....	332	\XINT_expr_rrseq:_abort .....	347
\XINT_expr_onliteral_mul .....	341	\XINT_expr_rrseq:_b .....	347
\XINT_expr_onliteral_mul_f .....	342	\XINT_expr_rrseq:_Break .....	348
\XINT_expr_onliteral_nil .....	337	\XINT_expr_rrseq:_break .....	347
\XINT_expr_onliteral_protect .....	332	\XINT_expr_rrseq:_c .....	347
\XINT_expr_onliteral_qfloat .....	332	\XINT_expr_rrseq:_D .....	348
\XINT_expr_onliteral_qfrac .....	332	\XINT_expr_rrseq:_d .....	347
\XINT_expr_onliteral_qint .....	332	\XINT_expr_rrseq:_E .....	348
\XINT_expr_onliteral_qrand .....	333	\XINT_expr_rrseq:_end .....	347
\XINT_expr_onliteral_qraw .....	332	\XINT_expr_rrseq:_Goon .....	348
\XINT_expr_onliteral_random .....	332	\XINT_expr_rrseq:_goon .....	347
\XINT_expr_onliteral_seq .....	339	\XINT_expr_rrseq:_noop .....	347
\XINT_expr_onliteral_seq_a .....	339, 341--343, 345, 346, 348	\XINT_expr_rrseq:_Omit .....	348
\XINT_expr_onliteral_seq_b .....	339	\XINT_expr_rrseq:_omit .....	347
\XINT_expr_onliteral_seq_c .....	339	\XINT_expr_rrseqx .....	346, 347
\XINT_expr_onliteral_seq_d .....	339	\XINT_expr_rrseqy .....	347
\XINT_expr_onliteral_seq_e .....	339	\XINT_expr_rseq:_A .....	344
\XINT_expr_onliteral_seq_f .....	339	\XINT_expr_rseq:_a .....	344
\XINT_expr_onliteral_subs .....	342	\XINT_expr_rseq:_aa .....	343, 344
\XINT_expr_onliteral_subs_f .....	342	\XINT_expr_rseq:_Abort .....	344
\XINT_expr_onliteral_togl .....	332	\XINT_expr_rseq:_abort .....	344
\XINT_expr_op:_a .....	342	\XINT_expr_rseq:_b .....	344
\XINT_expr_op:_b .....	342	\XINT_expr_rseq:_Break .....	344
\XINT_expr_op:_c .....	342	\XINT_expr_rseq:_break .....	344
\XINT_expr_op:_d .....	342	\XINT_expr_rseq:_c .....	344
\XINT_expr_op:_end .....	342	\XINT_expr_rseq:_D .....	344
\XINT_expr_op:_noop .....	342	\XINT_expr_rseq:_d .....	344
\XINT_expr_op_ .....	314, 317, 320, 328, 329	\XINT_expr_rseq:_E .....	344
\XINT_expr_op_! .....	330	\XINT_expr_rseq:_end .....	344
\XINT_expr_op_!? .....	337	\XINT_expr_rseq:_Goon .....	344
\XINT_expr_op:_ .....	320	\XINT_expr_rseq:_goon .....	344
\XINT_expr_op_:] .....	321	\XINT_expr_rseq:_noop .....	344
		\XINT_expr_rseq:_Omit .....	344

\XINT_expr_rseq:_omit	344	\XINT_expr_seq:_noop	340
\XINT_expr_rseqx	343	\XINT_expr_seq:_Omit	341
\XINT_expr_rseqy	343	\XINT_expr_seq:_omit	341
\XINT_expr_scan_macropar	303	\XINT_expr_seq_empty?	327, 328, 340, 341
\XINT_expr_scan_nbr_or_func	303, 304	\XINT_expr_startdec	303, 304
\XINT_expr_scandec_a	304, 305	\XINT_expr_startdec_a	305
\XINT_expr_scandec_b	305, 306	\XINT_expr_startint	304
\XINT_expr_scandec_c	306	\XINT_expr_subexpr	302
\XINT_expr_scandec_d	306	\XINT_expr_subx:_end	343
\XINT_expr_scandec_endbycs	306	\XINT_expr_tmpa	334, 335, 369--371
\XINT_expr_scanexp_a	305--307	\XINT_expr_tmpb	334, 369, 370
\XINT_expr_scanexp_b	307	\XINT_expr_tmpc	334, 335, 369
\XINT_expr_scanexp_bb	307	\XINT_expr_tmpd	334, 335
\XINT_expr_scanexp_c	307	\XINT_expr_tmpval	334
\XINT_expr_scanexp_cb	307	\XINT_expr_tmpvar	334
\XINT_expr_scanexp_d	307	\XINT_expr_unexpectedtoken	296
\XINT_expr_scanexp_db	307	\XINT_expr_unknown_function	331
\XINT_expr_scanexp_endbycs	307	\XINT_expr_unknown_operator	311, 312
\XINT_expr_scanexp_endbycs_b	307	\XINT_expr_unknown_variable	333
\XINT_expr_scanexpr_endbycs	307	\XINT_expr_unlock	296--299, 314, 315, 317, 318, 320, 321, 328--332, 334, 338--350, 355--366, 369, 371, 377
\XINT_expr_scanfunc	304, 309	\XINT_expr_unlock_a	296
\XINT_expr_scanfunc_a	309	\XINT_expr_unlock_hex_in	296, 307
\XINT_expr_scanfunc_b	309	\XINT_expr_unlock_sp	299, 300, 377
\XINT_expr_scanhex_I	304, 307	\XINT_expr_unpackvar	302, 303
\XINT_expr_scanhex_transition	308	\XINT_expr_until_	314, 317, 320, 328, 329
\XINT_expr_scanhexI_a	307, 308	\XINT_expr_until_:a	320
\XINT_expr_scanhexI_aa	308	\XINT_expr_until_:b	321
\XINT_expr_scanhexI_b	308	\XINT_expr_until_end_a	298
\XINT_expr_scanhexI_bgob	308	\XINT_expr_usethe	297, 299, 300
\XINT_expr_scanhexII_a	308, 309	\XINT_expr_var_@	338
\XINT_expr_scanhexII_aa	308	\XINT_expr_var_abort	337
\XINT_expr_scanhexII_b	308, 309	\XINT_expr_var_nil	337
\XINT_expr_scanhexII_bgob	308, 309	\XINT_expr_var_omit	337
\XINT_expr_scanint_a	304	\XINT_expr_wrap	297--299, 370
\XINT_expr_scanint_b	304	\XINT_expr_wrapit	370
\XINT_expr_scanint_c	304, 305	\xint_firstofone	351, 352, 354, 377
\XINT_expr_scanint_d	304, 305	\xint_firstofthree	310
\XINT_expr_scanint_endbycs	304	\xint_firstoftwo	306, 309, 333, 354, 355, 362, 374, 377
\XINT_expr_scanop_a	311, 314	\XINT_flexpr_func_!	357
\XINT_expr_scanop_b	311	\XINT_flexpr_func_?	357
\XINT_expr_scanop_c	311	\XINT_flexpr_func_@@	338
\XINT_expr_scanop_d	311	\XINT_flexpr_func_@@@	338
\XINT_expr_seq:_A	340, 341	\XINT_flexpr_func_@@@	338
\XINT_expr_seq:_a	340	\XINT_flexpr_func_abs	356
\XINT_expr_seq:_aa	340	\XINT_flexpr_func_all	364
\XINT_expr_seq:_Abort	341	\XINT_flexpr_func_any	364
\XINT_expr_seq:_abort	341	\XINT_flexpr_func_binomial	361
\XINT_expr_seq:_b	340, 341	\XINT_flexpr_func_break	332
\XINT_expr_seq:_Break	341	\XINT_flexpr_func_ceil	356
\XINT_expr_seq:_break	341	\XINT_flexpr_func_divmod	360
\XINT_expr_seq:_c	340, 341	\XINT_flexpr_func_even	358
\XINT_expr_seq:_D	341	\XINT_flexpr_func_factorial	358
\XINT_expr_seq:_d	341	\XINT_flexpr_func_first	365
\XINT_expr_seq:_E	341	\XINT_flexpr_func_float	360
\XINT_expr_seq:_end	340, 341		
\XINT_expr_seq:_Goon	341		
\XINT_expr_seq:_goon	341		



\XINT_flexpr_func_floor .....	356	\XINT_flseqb::csv_p .....	328
\XINT_flexpr_func_frac .....	356	\XINT_flseqb::csv_z .....	328
\XINT_flexpr_func_gcd .....	362	\XINT_flseqb:f:csv .....	368
\XINT_flexpr_func_if .....	365	\XINT_flseqb:f:csv_a .....	368
\XINT_flexpr_func_ifint .....	366	\XINT_flseqb:f:csv_bg .....	368
\XINT_flexpr_func_ifone .....	366	\XINT_flseqb:f:csv_bl .....	368
\XINT_flexpr_func_ifsgn .....	366	\XINT_flseqb:f:csv_n .....	368
\XINT_flexpr_func_isint .....	358	\XINT_flseqb:f:csv_na .....	368
\XINT_flexpr_func_isonone .....	358	\XINT_flseqb:f:csv_p .....	368
\XINT_flexpr_func_iter .....	345	\XINT_flseqb:f:csv_pa .....	368
\XINT_flexpr_func_iterr .....	348	\XINT_flseqb:f:csv_pb .....	368
\XINT_flexpr_func_last .....	365	\XINT_flseqb:f:csv_pc .....	368
\XINT_flexpr_func_lcm .....	363	\XINT_gcdof:_a .....	352
\XINT_flexpr_func_len .....	365	\XINT_gcdof:_b .....	352
\XINT_flexpr_func_max .....	363	\XINT_gcdof:_c .....	352
\XINT_flexpr_func_min .....	363	\XINT_gcdof:_d .....	352, 353
\XINT_flexpr_func_mod .....	360	\XINT_gcdof:_e .....	352, 353
\XINT_flexpr_func_not .....	357	\XINT_gcdof:_end .....	352, 353
\XINT_flexpr_func_num .....	355	\XINT_gcdof:_f .....	352, 353
\XINT_flexpr_func_nuple .....	358	\XINT_global .....	333--337, 369--371, 378
\XINT_flexpr_func_odd .....	357	\xint_gob_til_! .....	296, 298, 302, 330
\XINT_flexpr_func_opxadd .....	342	\xint_gob_til_minus .....	324
\XINT_flexpr_func_opxmulo .....	342	\xint_gob_til_W .....	301
\XINT_flexpr_func_pfactorial .....	361	\xint_gobble_i ....	299, 301, 311, 312, 316, 323, 331, 333, 341, 349, 350, 366--368, 377, 378
\XINT_flexpr_func_preduce .....	355	\xint_gobble_iii .....	297, 298, 302, 322, 323, 327, 328
\XINT_flexpr_func_quo .....	362	\xint_gobble_iv .....	300, 349, 350, 366--368
\XINT_flexpr_func_randrange .....	361	\XINT_iexpr_noopt .....	299
\XINT_flexpr_func_reduce .....	355	\XINT_iexpr_withopt .....	299
\XINT_flexpr_func_rem .....	362	\XINT_iexpr_wrap .....	299
\XINT_flexpr_func_reversed .....	365	\XINT_iiexpr_func_! .....	357
\XINT_flexpr_func_round .....	359	\XINT_iiexpr_func_? .....	357
\XINT_flexpr_func_rrseq .....	346	\XINT_iiexpr_func_@@ .....	338
\XINT_flexpr_func_rseq .....	343	\XINT_iiexpr_func_@@@ .....	338
\XINT_flexpr_func_seqx .....	340	\XINT_iiexpr_func_@@@ .....	338
\XINT_flexpr_func_sgn .....	356	\XINT_iiexpr_func_abs .....	356
\XINT_flexpr_func_sqr .....	357	\XINT_iiexpr_func_all .....	364
\XINT_flexpr_func_sqrt .....	359	\XINT_iiexpr_func_any .....	364
\XINT_flexpr_func_subx .....	343	\XINT_iiexpr_func_binomial .....	361
\XINT_flexpr_func_trunc .....	359	\XINT_iiexpr_func_break .....	332
\XINT_flexpr_func_xor .....	364	\XINT_iiexpr_func_ceil .....	356
\XINT_flexpr_noopt .....	299, 377	\XINT_iiexpr_func_divmod .....	360
\XINT_flexpr_op_! .....	330	\XINT_iiexpr_func_even .....	358
\XINT_flexpr_op_!? .....	337	\XINT_iiexpr_func_factorial .....	358
\XINT_flexpr_op_:] .....	321	\XINT_iiexpr_func_first .....	365
\XINT_flexpr_op_? .....	330	\XINT_iiexpr_func_float .....	360
\XINT_flexpr_op_[ .....	331	\XINT_iiexpr_func_floor .....	356
\XINT_flexpr_op_][: .....	321	\XINT_iiexpr_func_gcd .....	363
\XINT_flexpr_op_~ .....	333	\XINT_iiexpr_func_if .....	365
\XINT_flexpr_print .....	299	\XINT_iiexpr_func_ifint .....	365
\XINT_flexpr_until_end_a .....	298	\XINT_iiexpr_func_ifone .....	366
\XINT_flexpr_withopt_a .....	299	\XINT_iiexpr_func_ifsgn .....	366
\XINT_flexpr_withopt_b .....	299, 377	\XINT_iiexpr_func_isint .....	358
\XINT_flexpr_wrap .....	299, 377	\XINT_iiexpr_func_isonone .....	358
\XINT_flseqa::csv .....	327	\XINT_iiexpr_func_iter .....	345
\XINT_flseqb::csv .....	328	\XINT_iiexpr_func_iterr .....	348
\XINT_flseqb::csv_a .....	328	\XINT_iiexpr_func_last .....	365
\XINT_flseqb::csv_n .....	328		

\XINT_iiexpr_func_lcm .....	363	\XINT_infloat::_b .....	302
\XINT_iiexpr_func_len .....	365	\XINT_infloat::_c .....	302
\XINT_iiexpr_func_max .....	363	\XINT_infloat::_d .....	302
\XINT_iiexpr_func_min .....	363	\XINT_infloat::_e .....	302
\XINT_iiexpr_func_mod .....	360	\XINT_inv .....	353, 354
\XINT_iiexpr_func_not .....	357	\XINT_isbalanced_a .....	339, 340
\XINT_iiexpr_func_num .....	355	\XINT_isbalanced_b .....	339
\XINT_iiexpr_func_nuple .....	358	\XINT_isbalanced_c .....	339
\XINT_iiexpr_func_odd .....	357	\XINT_isbalanced_d .....	339, 340
\XINT_iiexpr_func_opxadd .....	342	\XINT_isbalanced_error .....	339
\XINT_iiexpr_func_opxmuls .....	342	\XINT_isbalanced_no .....	340
\XINT_iiexpr_func_pfactorial .....	361	\XINT_isbalanced_yes .....	339, 340
\XINT_iiexpr_func_quo .....	362	\XINT_istrue::_a .....	301
\XINT_iiexpr_func_randrange .....	361	\XINT_istrue::_b .....	301
\XINT_iiexpr_func_rem .....	362	\XINT_istrue::_c .....	301
\XINT_iiexpr_func_reversed .....	365	\XINT_istrue::_d .....	301
\XINT_iiexpr_func_round .....	359	\XINT_istrue::_e .....	301
\XINT_iiexpr_func_rrseq .....	346	\XINT_keep:x:csv_finish .....	324
\XINT_iiexpr_func_rseq .....	343	\XINT_keep:x:csv_loop .....	324
\XINT_iiexpr_func_seqx .....	340	\XINT_keep:x:csv_loop_pickeight .....	324
\XINT_iiexpr_func_sgn .....	356	\XINT_keep:x:csv_pos .....	324
\XINT_iiexpr_func_sqr .....	357	\XINT_lcmof:_a .....	353
\XINT_iiexpr_func_sqrt .....	359	\XINT_lcmof:_b .....	353
\XINT_iiexpr_func_sqrttr .....	359	\XINT_lcmof:_c .....	353
\XINT_iiexpr_func_subx .....	343	\XINT_lcmof:_d .....	353
\XINT_iiexpr_func_trunc .....	359	\XINT_lcmof:_e .....	353
\XINT_iiexpr_func_xor .....	364	\XINT_lcmof:_end .....	353, 354
\XINT_iiexpr_op_! .....	330	\XINT_lcmof:_f .....	353
\XINT_iiexpr_op_! ? .....	337	\XINT_lcmof:_g .....	353
\XINT_iiexpr_op_:] .....	321	\XINT_lcmof:_h .....	353
\XINT_iiexpr_op_? .....	330	\XINT_lcmof:_zero .....	353
\XINT_iiexpr_op_[ .....	331	\XINT_listsel:_: .....	323
\XINT_iiexpr_op_][: .....	321	\XINT_listsel:_:a .....	323
\XINT_iiexpr_op__ .....	333	\XINT_listsel:_N:N .....	323
\XINT_iiexpr_print .....	297, 377	\XINT_listsel:_N:N_a .....	323
\XINT_iiexpr_until_end_a .....	298	\XINT_listsel:_N:N_abort .....	323
\XINT_iiexpr_wrap .....	297, 298	\XINT_listsel:_N:P .....	323
\XINT_iiseq::csv .....	326	\XINT_listsel:_N:P_a .....	323
\XINT_iiseq::csv_n .....	326	\XINT_listsel:_nth .....	322, 323
\XINT_iiseq::csv_p .....	326	\XINT_listsel:_0:P .....	323
\XINT_iiseq::csv_z .....	326	\XINT_listsel:_P:N .....	323
\XINT_iiseqa::csv .....	327	\XINT_listsel:_P:N_a .....	323
\XINT_iiseqb::csv .....	327	\XINT_listsel:_P:0 .....	323
\XINT_iiseqb::csv_a .....	328	\XINT_listsel:_P:P .....	323
\XINT_iiseqb::csv_n .....	328	\XINT_listsel:_s .....	321--323
\XINT_iiseqb::csv_p .....	328	\XINT_listxsel:_: .....	321, 322
\XINT_iiseqb::csv_z .....	328	\XINT_listxsel:_N:N .....	322
\XINT_iiseqb:f:csv .....	367	\XINT_listxsel:_N:N_a .....	322
\XINT_iiseqb:f:csv_a .....	367	\XINT_listxsel:_N:P .....	322
\XINT_iiseqb:f:csv_bg .....	367	\XINT_listxsel:_N:P_a .....	322
\XINT_iiseqb:f:csv_bl .....	367	\XINT_listxsel:_0:P .....	322
\XINT_iiseqb:f:csv_n .....	367	\XINT_listxsel:_P:N .....	322
\XINT_iiseqb:f:csv_na .....	367, 368	\XINT_listxsel:_P:N_a .....	322
\XINT_iiseqb:f:csv_p .....	367	\XINT_listxsel:_P:0 .....	322
\XINT_iiseqb:f:csv_pa .....	367	\XINT_listxsel:_P:P .....	322
\XINT_iiseqb:f:csv_pb .....	367	\XINT_NewExpr .....	377, 378
\XINT_iiseqb:f:csv_pc .....	367	\XINT_NewExpr_a .....	378



\XINT_newexpr_clean	377	\XINT_seqb:f:csv_pc	367
\XINT_NewFloatFunc	369, 377	\XINT_spraw	301
\XINT_NewFunc	369, 377	\XINT_spraw::_a	301
\XINT_newfunc_clean	377, 378	\XINT_spraw::_b	301
\XINT_NewIIFunc	369, 377	\XINT_spraw::_c	301
\XINT_oncsv:_a	351, 352, 354	\XINT_spraw::_d	301
\XINT_oncsv:_b	351	\XINT_spraw::_e	301
\XINT_oncsv:_c	351	\XINT_spraw_a	301
\XINT_oncsv:_d	351	\XINT_spraw_p	301
\XINT_oncsv:_e	351	\XINT_thecoords_a	298
\XINT_oncsv:_empty	351--353	\XINT_thecoords_b	298
\XINT_oncsv:_end	351	\XINT_thecoords_c	298
\XINT_orof:_a	351	\xint_thirdofthree	311
\XINT_orof:_c	351	\XINT_tmpa	312,
\XINT_orof:_e	351	313, 320, 321, 328--331, 368--370, 378, 380	
\XINT_orof:_yes	351	\XINT_tmpb	329, 378, 380
\xint_orthat	301--311, 316,	\XINT_tmpc	380
322, 323, 331, 340--342, 344--350, 373--375		\xint_UDsignfork	312, 314, 317, 320, 328, 329
\XINT_pfloat::_b	302	\xint_UDsignsfork	322, 323
\XINT_pfloat::_c	302	\xint_UDzerominusfork	327
\XINT_pfloat::_d	302	\xint_undefined	335, 370
\XINT_pfloat::_e	302	\XINT_xorof:_a	351
\XINT_pfloat_opt	302	\XINT_xorof:_b	351
\XINT_protectii	297, 299, 300	\XINT_xorof:_c	351
\XINT_providespackage	296	\XINT_xorof:_e	351
\XINT_restorecatcodes_endinput	380	\xint_zapspaces	299, 311, 331
\XINT_round::_b	301	\xint_zapspaces_o	334, 335, 369, 370
\XINT_round::_c	301	\xintAbs	355
\XINT_round::_d	301	\xintAdd	342
\XINT_round::_e	301	\xintadd	327, 352, 367
\xint_secondofthree	310, 311	\xintANDof:csv	350, 364
\xint_secondoftwo		\xintApply	334
.... 306, 333, 354, 355, 362, 373, 374, 377		\xintapply	347, 348
\XINT_seq::csv	325	\xintApply:::csv	373
\XINT_seq::csv_e	325, 326	\xintApply:::csv	372, 373
\XINT_seq::csv_n	325, 326	\xintApplyInline	330
\XINT_seq::csv_p	325	\xintApplyUnbraced	337
\XINT_seq::csv_z	325	\xintapplyunbraced	349, 350
\XINT_seqa::csv	327	\xintAssignArray	334
\XINT_seqa::csv_a	327	\xintbareeval	
\XINT_seqb::csv	327	297--299, 335, 342, 343, 345, 346, 348, 370	
\XINT_seqb::csv_a	327	\xintbarefloateval	298,
\XINT_seqb::csv_n	327	299, 335, 342, 343, 345, 346, 348, 371, 377	
\XINT_seqb::csv_p	327	\xintbareiieval	
\XINT_seqb::csv_z	327	297, 298, 335, 342, 343, 345, 346, 348, 370	
\XINT_seqb:f:csv	366	\xintBinomial	361
\XINT_seqb:f:csv_a	366	\xintBool	332
\XINT_seqb:f:csv_be	366--368	\xintboolexpr	300
\XINT_seqb:f:csv_bg	366, 367	\xintCeil	356
\XINT_seqb:f:csv_bl	366	\xintCSV::csv	297, 301, 377
\XINT_seqb:f:csv_n	367	\xintCSVLength	334
\XINT_seqb:f:csv_na	367	\xintCSVtoList	334
\XINT_seqb:f:csv_nb	367, 368	\xintCSVtoListNonStripped	347, 348, 369, 371
\XINT_seqb:f:csv_nc	367, 368	\xintcsvtolistnonstripped	369, 371
\XINT_seqb:f:csv_p	367	\xintdeffloatfunc	369
\XINT_seqb:f:csv_pa	366, 367	\xintdeffloatfunc_a	369
\XINT_seqb:f:csv_pb	367	\xintdeffloatvar	335

<code>\xintdeffloatvar_a</code> .....	335	<code>\xintiiabs</code> .....	352, 353
<code>\xintdeffunc</code> .....	369	<code>\xintiiAdd</code> .....	342
<code>\xintdeffunc_a</code> .....	369	<code>\xintiiadd</code> .....	328, 352, 367
<code>\xintdefiifunc</code> .....	369	<code>\xintiiBinomial</code> .....	361
<code>\xintdefiifunc_a</code> .....	369	<code>\xintiiCmp</code> .....	296
<code>\xintdefiivar</code> .....	335	<code>\xintiiDivMod</code> .....	360
<code>\xintdefiivar_a</code> .....	335	<code>\xintiiE</code> .....	331
<code>\xintdefvar</code> .....	335	<code>\xintiieval</code> .....	297
<code>\xintdefvar_a</code> .....	335	<code>\xintiiEven</code> .....	358
<code>\XINTdigits</code> .....	327, 328, 354, 368, 377	<code>\xintiiexpr</code> .....	297
<code>\xintDivMod</code> .....	360	<code>\xintiiexpro</code> .....	297, 298
<code>\xintE</code> .....	331	<code>\xintiiFac</code> .....	330, 359
<code>\xintError:ignored</code> .....	296	<code>\xintiigcd</code> .....	354
<code>\xintError:inserted</code> .....	313	<code>\xintiiGCDof:csv</code> .....	354, 363
<code>\xintError:missing_xintthe!</code> .....	297	<code>\xintiiifNotZero</code> .....	330, 350, 351, 365, 375
<code>\xintError:removed</code> .....	312, 331, 333	<code>\xintiiifnotzero</code> .....	300
<code>\xintError:we_are_doomed</code> .....	339	<code>\xintiiifNotZero:</code> .....	365, 376
<code>\xinteval</code> .....	297	<code>\xintiiifNotZeroNE:</code> .....	375, 376
<code>\xintEven</code> .....	358	<code>\xintiiifOne</code> .....	366, 375
<code>\xintexpr</code> .....	297	<code>\xintiiifOne:</code> .....	366, 376
<code>\xintexpro</code> .....	297, 298, 300	<code>\xintiiifOneNE:</code> .....	375, 376
<code>\xintexprRestoreCatcodes</code> .....	334, 369, 379	<code>\xintiiifSgn</code> .....	330, 366, 375
<code>\xintexprSafeCatcodes</code> .....	335, 369, 378, 379	<code>\xintiiifsgn</code> .....	300
<code>\xintexprsafecatcodesfalse</code> .....	379	<code>\xintiiifSgn:</code> .....	366, 376
<code>\xintexprsafecatcodestrue</code> .....	379	<code>\xintiiifSgnNE:</code> .....	375, 376
<code>\xintFac</code> .....	330, 358	<code>\xintiiIsNotZero</code> .....	357
<code>\xintFirstItem:f:csv</code> .....	365	<code>\xintiiIsOne</code> .....	358
<code>\xintfloateval</code> .....	297	<code>\xintiiIsZero</code> .....	357
<code>\xintfloatexpr</code> .....	297	<code>\xintiilcm</code> .....	354
<code>\xintfloatexpro</code> .....	297, 299	<code>\xintiiLCMof:csv</code> .....	354, 363
<code>\xintFloatIsInt</code> .....	358	<code>\xintiiimax</code> .....	351
<code>\xintFloor</code> .....	356	<code>\xintiiMaxof:csv</code> .....	351, 363
<code>\xintFor</code> .....	312--314, 318--321, 329, 331, 369	<code>\xintiimin</code> .....	352
<code>\xintGCDof:csv</code> .....	352, 362	<code>\xintiiMinof:csv</code> .....	352, 363
<code>\xintHexToDec</code> .....	296	<code>\xintiiMod</code> .....	360
<code>\xintiCeil</code> .....	314, 325, 356	<code>\xintiiMul</code> .....	296, 342
<code>\xintieval</code> .....	297	<code>\xintiiMul</code> .....	352
<code>\xintiiexpr</code> .....	297	<code>\xintiiOdd</code> .....	357
<code>\xintiiexpro</code> .....	297, 299	<code>\xintiiOpp</code> .....	330
<code>\xintifboolexpr</code> .....	300	<code>\xintiiPFactorial</code> .....	361
<code>\xintifboolfloatexpr</code> .....	300	<code>\xintiiPow</code> .....	296
<code>\xintifboolliexpr</code> .....	300	<code>\xintiiPrd:csv</code> .....	352, 364
<code>\xintifCmp</code> .....	327, 328, 366--368	<code>\xintiiQuo</code> .....	362
<code>\xintifFloatInt</code> .....	366, 375	<code>\xintiiRandRange</code> .....	362, 376
<code>\xintifFloatInt:</code> .....	366, 376	<code>\xintiiRandRangeAtoB</code> .....	362, 376
<code>\xintifFloatIntNE:</code> .....	375, 376	<code>\xintiiRem</code> .....	362
<code>\xintifInt</code> .....	365, 375	<code>\xintiiSeq:csv</code> .....	326, 376
<code>\xintifInt:</code> .....	365, 376	<code>\xintiiiseq:csv</code> .....	326
<code>\xintifIntNE:</code> .....	375, 376	<code>\xintiiSeqA:csv</code> .....	327
<code>\xintiFloor</code> .....	325, 356	<code>\xintiiSeqB:csv</code> .....	327, 376
<code>\xintifOne</code> .....	366, 375	<code>\xintiiSeqB:f:csv</code> .....	367
<code>\xintifOne:</code> .....	366, 376	<code>\xintiiSeqBNumeric:csv</code> .....	376
<code>\xintifOneNE:</code> .....	375, 376	<code>\xintiiSeqNumeric:csv</code> .....	376
<code>\xintifsgnexpr</code> .....	300	<code>\xintiiSgn</code> .....	356
<code>\xintifsgnfloatexpr</code> .....	300	<code>\xintiiSqr</code> .....	357
<code>\xintifsgnliexpr</code> .....	300	<code>\xintiiSqrt</code> .....	359
<code>\xintiiAbs</code> .....	356	<code>\xintiiSqrtR</code> .....	359

\xintiiSum:csv	352, 364	\xintMod	360
\XINTinFloat	354, 360	\xintmod	353
\XINTinfloat	302, 327, 328, 368	\xintMul	342
\XINTinFloat::csv	299, 302	\xintmul	352
\XINTinFloatAdd	342	\xintNewBoolExpr	377
\XINTinfloatadd	328, 354, 368	\xintnewdummy	335, 337
\XINTinFloatBinomial	361	\xintNewExpr	320, 377
\XINTinFloatdigits	332, 354, 360	\xintNewFloatExpr	377
\XINTinFloatDivMod	360	\xintNewFunction	370
\XINTinFloatE	331	\xintNewIExpr	377
\XINTinFloatFac	330, 354, 358	\xintNewIIExpr	377
\XINTinFloatFacdigits	354, 358	\xintntheltnoexpand	338, 339
\XINTinFloatFracdigits	356	\xintNthEltPy:f:csv	322
\XINTinFloatMaxof:csv	354, 363	\xintNum	321, 322, 338, 354, 355, 362, 374
\XINTinFloatMinof:csv	354, 363	\xintOdd	357
\XINTinFloatMod	360	\xintOpp	330
\XINTinFloatMul	342, 357	\xintORof:csv	351, 364
\XINTinfloatmul	354	\xintPFactorial	361
\XINTinFloatPFactorial	361	\xintPFloat::csv	299, 302, 377
\XINTinFloatPrd:csv	354, 364	\xintPIrr	355
\XINTinFloatSeq::csv	376	\xintpraw	301
\XINTinFloatSeqA::csv	327	\xintPrd:csv	352, 364
\XINTinFloatSeqB::csv	328, 376	\xintRaw	332, 353
\XINTinFloatSeqB:f:csv	368	\xintraw	327, 353, 366
\XINTinFloatSeqBNumeric::csv	376	\xintRelaxArray	334
\XINTinFloatSeqNumeric::csv	376	\xintReverse:f:csv	365
\XINTinFloatSqr	357	\xintReverseOrder	349, 350
\XINTinFloatSqrt	354, 359	\xintRevWithBraces	347, 348
\XINTinFloatSqrtdigits	354, 359	\xintRound	359
\XINTinFloatSum:csv	354, 364	\xintround	301
\XINTinRandomFloatSdigits	333, 376	\xintRound::csv	299, 301, 377
\XINTinRandomFloatSixteen	333, 376	\xintSeq::csv	325, 376
\xintiNum	332	\xintseq::csv	325
\xintiQuo	362	\xintSeqA::csv	327
\xintiRem	362	\xintSeqB::csv	327, 376
\xintiRound	359	\xintSeqB:f:csv	366
\xintiround	301	\xintSeqBNumeric::csv	376
\xintIrr	355	\xintSeqNumeric::csv	376
\xintIsInt	358	\XINTsetupcatcodes	296
\xintisnotzero	301	\xintSgn	356
\xintIsOne	358	\xintSgnFork	328, 367, 368
\xintIsTrue::csv	297, 301, 377	\xintSPRaw	301
\xintiTrunc	359	\xintspraw	301
\xintKeep:f:csv	322, 323	\xintSPRaw::csv	297, 301, 377
\xintKeep:x:csv	322, 323	\xintSqr	357
\xintLastItem:f:csv	365	\xintSum:csv	352, 363
\xintLCMof:csv	353, 363	\xintTFrac	356
\xintLength	296, 335	\xintthe	297
\xintLength:f:csv	322, 323, 365	\xintthe_o	297
\xintListSel:f:csv	320, 322	\xintthebareeval	298, 340, 343, 345, 346, 348, 377
\xintListSel:x:csv	320, 321, 376	\xintthebarefloateval	
\xintloop	334		298, 340, 343, 345, 346, 348, 377
\xintmax	351, 354	\xintthebareiieval	
\xintMaxof:csv	351, 363		298, 340, 343, 345, 346, 348, 378
\xintMessage	334, 335, 337, 369–371, 379	\xinttheboolexpr	300, 377
\xintmin	352, 354	\xintthecoords	298
\xintMinof:csv	352, 363	\xinttheDigits	299

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

<code>\xinttheexpr</code> .....	297, 298, 300, 377	<code>\xintunassignvar</code> .....	335
<code>\xintthefloatexpr</code> .....	297, 298, 300, 377	<code>\xintXORof:csv</code> .....	351, 364
<code>\xinttheiexpr</code> .....	297, 298, 377		
<code>\xinttheiexpr</code> .....	297, 298, 300, 377	<b>Y</b>	
<code>\xintToggle</code> .....	332	<code>\y</code> .....	295
<code>\xinttrim</code> .....	347--349		
<code>\xintTrim:f:csv</code> .....	322, 323	<b>Z</b>	
<code>\xintTrunc</code> .....	359	<code>\z</code> .....	295, 296

## 13 Cumulative index

Currently there are some macros which did not make it to the indices because they are defined via constructs such as `\expandafter\def\csname ... \endcsname`, thus their indexing needs some extra mark-up which is yet to be added. This is particularly true for some definitions done by `xintexpr`.

The first indicated page number is the one where the macro is first encountered, hence this is most likely also the page where it gets defined (if it is not one of those which are only used as delimiters).

Symbols	
<code>\%</code> .....	378
<code>\@backslashchar</code> .....	112
<code>\_!_/</code> .....	18
A	
<code>\A</code> .....	176--178
<code>\afterassignment</code> .....	208
<code>\aftergroup</code> .....	20, 56, 114, 154, 155, 166, 167, 180, 261, 262, 270, 271, 295
B	
<code>\B</code> .....	176--178
<code>\BEZ</code> .....	177, 178
<code>\bgroup</code> .....	43
<code>\break</code> .....	177
C	
<code>\catcode</code> .....	5--7, 18, 19, 37--40, 42, 43, 55, 57, 58, 113, 154, 166, 180, 192, 193, 261, 270, 295, 296, 304, 310--314, 321, 331, 333--339, 347, 350, 369, 371, 373, 378--380
<code>\cfrac</code> .....	272--275
<code>\char</code> .....	10
<code>\chardef</code> .....	8
<code>\count</code> .....	176--178, 302
<code>\cr</code> .....	176, 177
<code>\csname</code> .....	5, 6, 8, 14--17, 19, 20, 28, 29, 31, 38--46, 49--52, 56--59, 92, 111--114, 117, 118, 135, 148, 150, 151, 154--156, 158--160, 163--166, 180, 229, 232, 233, 247, 261, 270, 271, 273, 274, 295, 296, 298, 299, 303, 304, 307, 311--321, 324, 327--338, 340--350, 355--366, 369--371, 377
D	
<code>\D</code> .....	178
<code>\detokenize</code> .....	330, 332, 334, 373, 374
<code>\dimen</code> .....	302
<code>\dimexpr</code> .....	40, 41, 302
<code>\do</code> .....	312--314, 318--321, 329, 331, 369
<code>\dp</code> .....	302
E	
<code>\edef</code> ....	6, 7, 15--17, 38, 39, 41--45, 155, 176--178, 321, 333--335, 337, 369, 370, 378, 379
<code>\endcsname</code> ...	5, 6, 8, 14--17, 19, 20, 28, 29, 31, 38--46, 49--52, 56--59, 92, 112--114, 116--118, 120, 135, 148, 150, 151, 154--156, 158--160, 163--166, 180, 205, 207, 208, 229, 232, 233, 238, 242, 247, 261, 270, 271, 273, 274, 295, 296, 298, 299, 303--308, 311--321, 324, 327--338, 340--350, 355--366, 369--371, 377
<code>\endinput</code> .....	5--7, 20, 56, 114, 154, 155, 166, 167, 180, 261, 262, 270, 271, 295
<code>\endlinechar</code> .....	5--7, 19, 55, 113, 154, 166, 180, 261, 270, 295, 378
<code>\errmessage</code> .....	176, 177
<code>\escapechar</code> .....	44, 45, 378
<code>\everyeof</code> .....	378
F	
<code>\fdef</code> .....	11
<code>\fontchardp</code> .....	302
<code>\fontcharht</code> .....	302
<code>\fontcharic</code> .....	302
<code>\fontcharwd</code> .....	302
<code>\fontdimen</code> .....	302
<code>\frac</code> .....	192
<code>\futurelet</code> .....	36, 38, 39
G	
<code>\glueexpr</code> .....	302
H	
<code>\halign</code> .....	176, 177
<code>\hfill</code> .....	177, 271--274
<code>\ht</code> .....	302
I	
<code>\if</code> .....	17, 18, 52, 63, 64, 75, 87, 88, 91, 93, 98, 100, 102--104, 122, 123, 125--131, 134, 138--140, 146, 159, 161, 162, 171, 172, 182, 186--190, 192, 193, 195--197, 199--202, 210, 214--217, 219, 220, 225, 226, 229--231, 238, 240, 241, 243, 248, 250, 254, 256, 258, 259, 296, 301--311, 316, 321--323, 330, 340--342, 344--355, 362, 366--368, 373--375, 377
<code>\ifcat</code> .....	302, 304--311
<code>\ifcsname</code> ....	43, 44, 57, 58, 311, 331, 333, 335
<code>\ifdefined</code> ..	8, 9, 11, 17, 58, 59, 114, 153, 176, 177
<code>\iffalse</code> .....	238, 243
<code>\iftoggle</code> .....	148
<code>\iftrue</code> .....	238, 242

430

\XINT:expr:one:and:opt	354, 358--360, 376	\XINT_?expr_Vb	40
\XINT:expr:one:or:two:nums	376	\XINT_?expr_Vc	40
\XINT:expr:randrange	361, 362	\XINT_?expr_Vd	40, 41
\XINT:expr:tacitzeroifonearg	354, 359, 376	\XINT_?expr_Ve	41
\XINT:expr:totwo	355, 374	\XINT_?expr_Vf	40, 41
\XINT:expr:two:to:one	355, 360--362, 376	\XINT_?expr_Vi	41
\XINT:expr:two:to:two	355, 360, 376	\XINT_?expr_Vx	41
\XINT:expr:userfunc	369, 376	\XINT_?expr_Vy	41
\XINT:flexpr:two:to:one	355, 360, 361, 376	\XINT_abs	61, 181, 222
\XINT:flexpr:two:to:two	355, 360, 376	\XINT_add_A	73
\XINT:iiexpr:one:or:two:	376	\XINT_add_a	73--75, 85
\XINT:iiexpr:randrange	361, 362	\XINT_add_aa	73
\XINT:iiexpr:tacitzeroifonearg	355, 359, 376	\XINT_add_aa_small	73
\XINT:NE:csv	374, 376	\XINT_add_b	74
\XINT:NE:LApply::csv	373	\XINT_add_bi	74
\XINT:NE:one	373, 374, 376	\XINT_add_c	74
\XINT:NE:one_a	373	\XINT_add_checklengths	73
\XINT:NE:oneopt	374	\XINT_add_d	74
\XINT:NE:oneopt_a	374	\XINT_add_di	74
\XINT:NE:oneopt_b	374	\XINT_add_e	74
\XINT:NE:oneopt_fork	374	\XINT_add_exchange	73
\XINT:NE:RApply::csv	373	\XINT_add_f	74
\XINT:NE:RLApply::csv	373	\XINT_add_fi	74
\XINT:NE:two	373--376	\XINT_add_firstiszero	72
\XINT:NE:two_	373, 374, 376	\XINT_add_fork	72, 132
\XINT:NE:two_a	373	\XINT_add_g	74
\XINT:NE:two_b	373	\XINT_add_h	74
\XINT:NE:two_fork_dd	373	\XINT_add_hi	74, 75
\XINT:NE:two_fork_nn	373	\XINT_add_i	75
\XINT:NEhook:csv	300, 362--365, 376	\XINT_add_k	74, 75
\XINT:NEhook:one	300, 314, 321, 330, 355--359, 362, 376	\XINT_add_ke	75
\XINT:NEhook:two	300, 315, 316, 342, 376	\XINT_add_kf	75
\XINT:newexpr:insertdollar	374	\XINT_add_l	75
\XINT:newexpr:macrofunc	371, 376	\XINT_add_lf	75
\XINT:newexpr:macrofunc:a	371	\XINT_add_m	75
\XINT:newexpr:one:and:opt	374, 376	\XINT_add_minusminus	72
\XINT:newexpr:one:or:two:nums	376	\XINT_add_minusplus	72
\XINT:newexpr:tacitzeroifonearg	374, 376	\XINT_add_n	75
\XINT:newexpr:two:to:one	374--376	\XINT_add_nfork	72
\XINT:newexpr:two:to:two	374, 376	\XINT_add_o	75
\XINT:newexpr:userfunc	369, 376	\XINT_add_out	73
\XINT:newflexpr:two:to:one	375, 376	\XINT_add_plusminus	72
\XINT:newflexpr:two:to:two	374, 376	\XINT_add_plusplus	72, 73
\XINT:newiiexpr:one:or:two:	376	\XINT_add_pp_a	72, 73, 78
\XINT:newiiexpr:tacitzeroifonearg	374, 376	\XINT_add_pp_b	72, 73
\XINT_	369	\XINT_add_secondiszero	72
\XINT::_:end	301, 302	\xint_afterfi	10, 34, 38, 45, 100, 107, 123, 124, 128, 129, 146, 158, 187, 192, 193, 195, 197, 201, 217, 219, 220, 240, 248, 250, 262--269, 287--290, 308, 311, 312, 315, 317, 318, 320, 321, 323, 329, 351, 366--368
\XINT::_:end	301	\XINT_allexpr_iter	345
\XINT_?expr_D	41	\XINT_allexpr_iterr	348
\XINT_?expr_Da	40	\XINT_allexpr_opx	342
\XINT_?expr_Di	41	\XINT_allexpr_rrseq	346
\XINT_?expr_U	41	\XINT_allexpr_rseq	343
\XINT_?expr_Ua	40	\XINT_allexpr_seqx	340
\XINT_?expr_Ui	41		
\XINT_?expr_V	41		
\XINT_?expr_Va	40		

\XINT_allexpr_subx .....	343	\XINT_bezout .....	169
\XINT_andof:_a .....	350	\XINT_bezout_botharezero .....	169
\XINT_andof:_c .....	350	\XINT_bezout_exitA .....	171, 172
\XINT_andof:_e .....	350	\XINT_bezout_exita .....	172
\XINT_andof:_no .....	350	\XINT_bezout_firstiszero .....	169
\XINT_andof_a .....	128	\XINT_bezout_fork .....	169
\XINT_andof_b .....	128	\XINT_bezout_loop_B .....	171, 172
\XINT_andof_c .....	128	\XINT_bezout_loop_b .....	172
\XINT_andof_e .....	128	\XINT_bezout_loop_C .....	171
\XINT_andof_no .....	128	\XINT_bezout_loop_c .....	172
\XINT_apply .....	32	\XINT_bezout_loop_D .....	171, 172
\XINT_apply_end .....	32	\XINT_bezout_loop_d .....	172
\XINT_apply_loop_a .....	32	\XINT_bezout_loop_E .....	172
\XINT_apply_loop_b .....	32	\XINT_bezout_loop_e .....	172
\XINT_applyon::_a .....	373	\XINT_bezout_minusminus .....	169, 170
\XINT_applyon::_b .....	373	\XINT_bezout_minusplus .....	169, 170
\XINT_applyon::_c .....	373	\XINT_bezout_mm_post .....	170
\XINT_applyon::_d .....	373	\XINT_bezout_mm_postb .....	170
\XINT_applyon::_e .....	373	\XINT_bezout_mp_post .....	170
\XINT_applyon::_end .....	373	\XINT_bezout_plusminus .....	169, 170
\XINT_applyon::_a .....	372, 373	\XINT_bezout_plusplus .....	169, 170
\XINT_applyon::_b .....	373	\XINT_bezout_pm_post .....	170
\XINT_applyon::_c .....	373	\XINT_bezout_preloop_A .....	171
\XINT_applyon::_d .....	373	\XINT_bezout_preloop_a .....	170, 171
\XINT_applyon::_e .....	373	\XINT_bezout_preloop_exchange .....	171
\XINT_applyon::_end .....	373	\XINT_bezout_preloop_exit .....	171
\XINT_applyunbr .....	32	\XINT_bezout_secondiszero .....	169
\XINT_applyunbr_end .....	32, 33	\XINT_binom_a .....	140
\XINT_applyunbr_loop_a .....	32, 33	\XINT_binom_b .....	140
\XINT_applyunbr_loop_b .....	32, 33	\XINT_binom_bigloop .....	140, 142
\xint_arrayname .....	44, 45	\XINT_binom_bigloop_a .....	142
\XINT_assign_a .....	43	\XINT_binom_bigloop_b .....	142
\XINT_assign_b .....	43	\XINT_binom_div .....	141--144
\XINT_assign_c .....	43, 44	\XINT_binom_end_ .....	141, 142, 144, 145
\XINT_assign_d .....	44	\XINT_binom_end_i .....	141, 142, 144, 145
\XINT_assign_def .....	43, 44	\XINT_binom_end_ii .....	141, 144, 145
\XINT_assign_e .....	44	\XINT_binom_end_iii .....	141, 144
\XINT_assign_f .....	43, 44	\XINT_binom_finish .....	144
\XINT_assign_fork .....	43	\XINT_binom_fork .....	140
\XINT_assign_opt .....	43	\XINT_binom_medloop .....	140--142
\XINT_assignarray .....	44, 45	\XINT_binom_medloop_a .....	141
\XINT_assignarray_def .....	44, 45	\XINT_binom_medloop_b .....	141, 142
\XINT_assignarray_end .....	45	\XINT_binom_mul .....	141--144
\XINT_assignarray_fork .....	44	\XINT_binom_pre .....	140, 212
\XINT_assignarray_loop .....	45	\XINT_binom_smallloop .....	140, 141
\XINT_assignarray_opt .....	44	\XINT_binom_smallloop_a .....	141, 143
\XINT_bezalg .....	174	\XINT_binom_smallloop_b .....	141
\XINT_bezalg_a .....	174, 175	\XINT_binom_vbigloop .....	140, 142
\XINT_bezalg_AisZero .....	174	\XINT_binom_vbigloop_a .....	142
\XINT_bezalg_b .....	174	\XINT_binom_vsmallend_ .....	143, 145
\XINT_bezalg_BisZero .....	174	\XINT_binom_vsmallend_b .....	145
\XINT_bezalg_c .....	174, 175	\XINT_binom_vsmallend_i .....	143, 145
\XINT_bezalg_d .....	175	\XINT_binom_vsmallend_ib .....	145
\XINT_bezalg_e .....	175	\XINT_binom_vsmallend_ii .....	143, 144
\XINT_bezalg_end .....	175	\XINT_binom_vsmallend_iib .....	145
\XINT_bezalg_end_a .....	175	\XINT_binom_vsmallend_iii .....	143, 144
\XINT_bezalg_fork .....	174	\XINT_binom_vsmallend_iiib .....	144



\XINT_binom_vsmallfinish .....	144, 145	\xint_c_iv .....	8, 12, 13, 26--31, 47, 49, 51, 52, 73, 76, 78, 84, 90, 105, 112, 115, 118, 136, 141, 143, 146, 181, 184, 185, 200, 204, 228, 246--248, 250, 251, 254
\XINT_binom_vsmallloop .....	140, 143	\xint_c_ix .....	8, 12, 15, 30, 31, 47, 49, 51, 52, 93, 135, 186, 304, 306, 307, 309, 310
\XINT_binom_vsmallloop_a .....	143	\xint_c_mone .....	58, 60, 339
\XINT_binom_vsmallloop_b .....	143	\xint_c_nine_x^viii .....	114, 150
\XINT_binom_vsmallmuldiv .....	143--145	\xint_c_v .....	8, 12, 13, 15, 26--31, 47, 49, 51, 52, 63, 64, 76, 89, 90, 115, 118, 181, 184, 185, 200, 229, 232, 243, 256
\XINT_boolexpr_done .....	300	\xint_c_vi .....	8, 12, 13, 26--31, 47, 49, 51, 52, 72, 73, 76, 78, 84, 90, 92, 105, 115, 118, 133, 181, 184, 185, 200, 204
\XINT_boolexpr_print .....	297, 300, 377	\xint_c_vii ...	8, 12, 13, 26--31, 47, 49, 51, 52, 76, 90, 92, 115, 118, 181, 184, 185, 200, 331
\xint_bracedstopper .....	10, 44, 45	\xint_c_viii .....	8, 12, 13, 26--32, 47--52, 69, 72, 73, 78, 84, 90, 105, 115, 117, 118, 136, 137, 150--153, 158--160, 181, 184, 185, 200, 204, 246, 248--251, 256, 314, 324
\XINT_btd_checkin .....	162	\xint_c_x 8, 27, 63--65, 72, 73, 78, 84, 89, 90, 103, 105, 133, 152, 202, 204, 227, 243, 305, 330	
\XINT_btd_htd .....	163	\xint_c_x^iv .....	8, 110, 245, 246, 250
\XINT_btd_main .....	163	\xint_c_x^ix .....	59, 85, 86, 97, 248
\XINT_btd_N .....	162, 163	\xint_c_x^viii 58, 86, 87, 89, 97, 100, 101, 111, 112, 136, 143, 146--148, 150, 203, 246--253	
\XINT_bth_checkin .....	163	\xint_c_x^viii_mone .....	59, 101, 102, 140, 146, 245, 250, 254
\XINT_bth_loop .....	163, 164	\xint_c_xi_e_viii_mone .....	59, 83, 100, 101
\XINT_bth_main .....	163	\xint_c_xii .....	8, 72, 73, 78, 84, 90, 105, 204
\XINT_bth_N .....	163	\xint_c_xii_e_viii .....	59
\xint_Bye 10, 63--65, 89, 103, 202, 227, 243, 324		\xint_c_xiv .....	8, 71
\xint_bye ..... 10--13, 21--34, 47--54, 62--65, 67, 82, 89, 91, 94, 99, 103, 115, 117, 118, 121, 137--139, 144, 157, 159, 161--165, 174, 175, 181, 184, 185, 200, 202, 205, 206, 208, 227--229, 238, 243, 256, 324, 330, 339, 340		\xint_c_xvi .....	8, 137, 158--160
\xint_c_ ..... 8, 12--15, 26--31, 34, 44, 45, 47, 51, 60, 62, 66, 69, 73, 76, 78, 84, 90, 93, 99, 100, 105, 115, 118, 126, 152, 158, 181, 184, 185, 188--191, 200, 201, 204, 206, 207, 229, 230, 232, 234, 238, 239, 243, 244, 249, 258, 259, 287, 288, 301, 311, 313, 322, 323, 340		\xint_c_xviii 8, 303, 304, 310, 313, 339, 341, 342	
\xint_c_i ..... 8, 12, 13, 15, 26--31, 33, 42, 45, 47, 49, 51, 52, 60, 63, 64, 67, 70--73, 75, 76, 78--81, 83, 84, 89--91, 93, 94, 96, 98--101, 105, 106, 108, 109, 111, 112, 115, 118, 134--136, 140--148, 152, 157--160, 173, 174, 181, 184--186, 200--206, 225, 227, 229--232, 238, 239, 243, 244, 246--257, 266--268, 289--291, 306, 313, 321, 325, 326, 340, 341, 344, 346, 348--350, 369		\xint_c_xxii .....	8, 84
\xint_c_ii ..... 8, 12, 13, 26--31, 42, 47, 49, 51, 52, 62--64, 72--76, 78, 83--85, 89, 90, 105, 108, 109, 111, 112, 115, 118, 133--139, 141--145, 147, 148, 181, 184, 185, 200, 203, 204, 229, 239, 243, 244, 246--248, 250--253, 255, 256, 313, 320, 321, 329		\XINT_cfrac_A .....	271, 272
\xint_c_ii^v ..... 8, 205, 206		\XINT_cfrac_B .....	272
\xint_c_ii^vi ..... 8, 205, 206		\XINT_cfrac_C .....	272
\xint_c_ii^vii ..... 8, 17, 150, 158, 159		\XINT_cfrac_D .....	272
\xint_c_ii^viii ..... 8, 158, 159		\XINT_cfrac_end .....	272
\xint_c_ii^xii ..... 8		\XINT_cfrac_end_b .....	272
\xint_c_ii^xiv ..... 8, 17, 150		\XINT_cfrac_integer .....	272
\xint_c_ii^xv ..... 155, 157		\XINT_cfrac_loop_a .....	272
\xint_c_ii^xvi ..... 155, 157, 161		\XINT_cfrac_loop_d .....	272
\xint_c_ii^xxi ..... 8, 17, 150		\XINT_cfrac_loop_e .....	272
\xint_c_iii ..... 8, 12, 13, 26--31, 43, 47, 49, 51, 52, 76, 90, 111, 112, 115, 118, 141, 143, 144, 147, 148, 150, 151, 181, 184, 185, 200, 203, 229, 237, 241, 242, 246--248, 251, 252, 254--256, 321		\xint_cfrac_loop_exit .....	272
		\XINT_cfrac_loop_f .....	272
		\XINT_cfrac_noopt .....	271
		\XINT_cfrac_opt_a .....	271
		\XINT_cfrac_opt_b .....	271
		\XINT_cfrac_optc .....	271
		\XINT_cfrac_optl .....	271
		\XINT_cfrac_optr .....	272
		\XINT_cfrac_T .....	272
		\XINT_chtb_checkin .....	165
		\XINT_chtb_main .....	165
		\XINT_chtb_N .....	165

\XINT_Clamped.handler	58	\XINT_csvtol_loop_a	24
\XINT_Cmp	296, 328, 367, 368	\XINT_csvtol_loop_b	24
\XINT_cmp_a	76, 77	\XINT_ctcv_end	283, 284
\XINT_cmp_checklengths	76	\XINT_ctcv_loop_a	283, 284
\XINT_cmp_distinctlengths	76	\XINT_ctcv_loop_b	283
\XINT_cmp_equal	77	\XINT_ctcv_loop_c	283
\XINT_cmp_firstiszero	75, 76	\XINT_ctcv_loop_d	283
\XINT_cmp_gt	77	\XINT_ctcv_loop_e	283
\XINT_cmp_lt	77	\XINT_ctcv_loop_f	283, 284
\XINT_cmp_minusminus	75, 76	\XINT_ctcv_loop_g	284
\XINT_cmp_minusplus	75, 76	\XINT_ctcv_prep	283
\XINT_cmp_nfork	75	\XINT_ctf_end	279, 280
\XINT_cmp_plusminus	75, 76	\XINT_ctf_loop_a	279, 280
\XINT_cmp_plusplus	75, 76, 122, 222	\XINT_ctf_loop_b	279
\XINT_cmp_pp	76	\XINT_ctf_loop_c	280
\XINT_cmp_secondiszero	75, 76	\XINT_ctf_loop_d	280
\XINT_cntcs	289	\XINT_ctf_loop_e	280
\XINT_cntcs_exit	289	\XINT_ctf_prep	279
\XINT_cntcs_exit_b	289	\XINT_cuz	66, 106
\XINT_cntcs_loop	289	\XINT_cuz_byviii	66
\XINT_cntf	287	\XINT_cuz_byviii_done	66
\XINT_cntf_exit	288	\XINT_cuz_byviii_e	66
\XINT_cntf_loop	287, 288	\XINT_cuz_byviii_z	66
\XINT_cntgc	290	\XINT_cuz_cleantoeend	66
\XINT_cntgc_exit	290	\XINT_cuz_hitend	66
\XINT_cntgc_exit_b	290	\XINT_cuz_loop	66, 68
\XINT_cntgc_loop	290	\XINT_cuz_small	59, 71, 73, 82, 85
\XINT_cntSgn	60, 107, 181, 191, 218, 222	\XINT_dbl	62, 137--139, 256
\XINT_cntSgnFork	181, 218, 222	\XINT_dbl_a	62
\XINT_ConversionSyntax-signal	57	\XINT_dbl_e	62, 63
\XINT_ConversionSyntax.handler	58	\XINT_dbl_fork	62
\XINT_csbth_none	156	\XINT_dbl_neg	62
\XINT_cshtb_A	156	\xint_ddfork	373
\XINT_cshtb_B	156	\XINT_dec	63, 64, 139
\XINT_cshtb_C	156	\XINT_dec_a	64
\XINT_cshtb_D	156	\XINT_dec_bye	63, 64, 139
\XINT_cshtb_E	156	\XINT_dec_e	64
\XINT_cshtb_F	156	\XINT_dec_fork	64
\XINT_cshtb_none	156	\XINT_dec_neg	64
\XINT_cstc_end	291	\XINT_dectostr	190
\XINT_cstc_loop_a	291	\XINT_dectostr_a	190
\XINT_cstc_loop_b	291	\XINT_dectostr_b	190
\XINT_cstc_prep	291	\XINT_dectostr_z	190
\XINT_csv::_a	301	\XINT_denom_A	191, 192
\XINT_csv::_b	301	\XINT_denom_B	191, 192
\XINT_csv::_c	301	\XINT_denom_fork	191
\XINT_csv::_d	301	\XINT_div_BisOne	88
\XINT_csvtol_finish_a	24	\XINT_div_BisTwo	88
\XINT_csvtol_finish_b	24	\XINT_div_BisTwo_a	89
\XINT_csvtol_finish_di	24, 25	\XINT_div_cleanR	91, 92
\XINT_csvtol_finish_dii	24, 25	\XINT_div_dosmalldiv	89, 90
\XINT_csvtol_finish_diii	24	\XINT_div_dosmallsmall	89
\XINT_csvtol_finish_div	24	\XINT_div_exittofinish	94
\XINT_csvtol_finish_dv	24	\XINT_div_finish	91
\XINT_csvtol_finish_dvi	24	\XINT_div_finish_a	91
\XINT_csvtol_finish_dvii	24	\XINT_div_finish_b	91
\XINT_csvtol_finish_dviii	24	\XINT_div_finish_bRpos	91

\XINT_div_finish_bRzero .....	91	\XINT_div_start_c .....	91, 92
\XINT_div_I_a .....	92, 97	\XINT_div_start_c_ .....	92
\XINT_div_I_b .....	92, 93	\XINT_div_start_c_i .....	92
\XINT_div_I_c .....	92, 93	\XINT_div_start_c_ii .....	92
\XINT_div_I_czero .....	92, 93	\XINT_div_start_c_iii .....	92
\XINT_div_I_da .....	93	\XINT_div_start_c_iv .....	92
\XINT_div_I_db .....	93	\XINT_div_start_c_v .....	92
\XINT_div_I_dc .....	93	\XINT_div_start_c_vi .....	92
\XINT_div_I_dd .....	93	\XINT_div_start_ca .....	92
\XINT_div_I_de .....	93, 94	\XINT_div_start_cb .....	92
\XINT_div_I_dN .....	93	\XINT_div_sub .....	93--96, 98
\XINT_div_I_dP .....	93, 94	\XINT_div_sub_a .....	98
\XINT_div_I_dz .....	93	\XINT_div_sub_b .....	98
\XINT_div_I_g .....	93, 94	\XINT_div_sub_bi .....	98, 99
\XINT_div_I_h .....	94	\XINT_div_sub_c .....	98, 99
\XINT_div_II_b .....	94	\XINT_div_sub_clean .....	98
\XINT_div_II_c .....	95	\XINT_div_sub_d .....	98
\XINT_div_II_d .....	95	\XINT_div_sub_di .....	98, 99
\XINT_div_II_e .....	95	\XINT_div_sub_e .....	98, 99
\XINT_div_II_f .....	95, 96	\XINT_div_sub_f .....	98
\XINT_div_II_fa .....	96	\XINT_div_sub_fi .....	98, 99
\XINT_div_II_g .....	96	\XINT_div_sub_g .....	98, 99
\XINT_div_II_h .....	96	\XINT_div_sub_h .....	98
\XINT_div_II_k .....	95, 96	\XINT_div_sub_hi .....	98, 99
\XINT_div_II_l .....	96	\XINT_div_sub_i .....	98, 99
\XINT_div_II_m .....	96, 97	\XINT_div_sub_l .....	99
\XINT_div_II_skipc .....	95	\XINT_div_sub_neg .....	98
\XINT_div_II_skipf .....	95, 96	\XINT_div_sub_r .....	99
\XINT_div_mini .....	95, 101	\XINT_div_unsepQ .....	67, 91
\XINT_div_mini_a .....	101	\XINT_div_unsepQ_delim .....	67, 94
\XINT_div_mini_b .....	101	\XINT_div_unsepQ_one .....	67, 68
\XINT_div_mini_c .....	101	\XINT_div_unsepQ_x .....	67
\XINT_div_mini_d .....	101, 102	\XINT_div_unsepQ_y .....	67, 68
\XINT_div_mini_w .....	101	\XINT_div_unsepR .....	68, 94
\XINT_div_minimulwc_a .....	97	\XINT_div_unsepR_x .....	68
\XINT_div_minimulwc_b .....	97	\XINT_div_verysmallisone .....	97
\XINT_div_minimulwc_c .....	97	\XINT_div_verysmallmul .....	93, 94, 97
\XINT_div_prepare 88, 103, 104, 149, 168, 170--174, 196, 198, 201, 209, 215--217, 272, 275--278		\XINT_div_verysmallmul_a .....	97
\XINT_div_prepare_a .....	88	\XINT_div_verysmallmul_b .....	97
\XINT_div_prepare_b .....	88, 90	\XINT_div_verysmallmul_e .....	97
\XINT_div_prepare_c .....	90	\XINT_div_xmini .....	95, 96
\XINT_div_prepare_d .....	90	\XINT_div_xmini_a .....	95
\XINT_div_prepare_e .....	90	\XINT_div_xmini_b .....	95
\XINT_div_prepare_f .....	90	\XINT_div_xmini_c .....	95
\XINT_div_prepare_g .....	90, 204	\XINT_div_zeroQ .....	91
\XINT_div_prepare_h .....	90, 91	\XINT_div_zeroQ_end .....	91
\XINT_div_prepare_small .....	88	\XINT_DivisionByZero.handler .....	58
\XINT_div_small_a .....	88, 89	\XINT_DivisionImpossible-signal .....	57
\XINT_div_small_b .....	89, 204	\XINT_DivisionImpossible.handler .....	58
\XINT_div_small_ba .....	89, 204	\XINT_DivisionUndefined-signal .....	57
\XINT_div_smallmul_a .....	95--97	\XINT_DivisionUndefined.handler .....	58
\XINT_div_smallmul_e .....	97	\XINT_divmod_a .....	215
\XINT_div_smallsmall .....	89	\XINT_divmod_aiszero .....	215
\XINT_div_smallsmallend .....	89	\XINT_divmod_b .....	215
\XINT_div_start_a .....	91	\XINT_divmod_bneg .....	215
\XINT_div_start_b .....	91	\XINT_divmod_bneg_finish .....	216
		\XINT_divmod_bpos .....	215, 216

<a href="#">\XINT_divmod_bpos_a</a>	216	<a href="#">\XINT_dth_tohex</a>	157--159
<a href="#">\XINT_divmod_bpos_finish</a>	216	<a href="#">\XINT_dth_tohex_a</a>	158
<a href="#">\XINT_divmod_divbyzero</a>	215	<a href="#">\XINT_dthb_again</a>	157, 158
<a href="#">\xint_dothis</a>	10, 57, 58, 64, 87, 102--104, 110, 123, 133--135, 137, 138, 140, 146, 152, 158, 159, 171, 186, 190, 199--202, 214--216, 225, 226, 230, 232, 234, 238, 243, 245, 250, 251, 254--256, 258, 259, 301--311, 316, 321, 323, 331, 340--342, 344--350, 373--375	<a href="#">\XINT_dthb_exclam</a>	157
<a href="#">\XINT_dsh_fork</a>	119	<a href="#">\XINT_dthb_lastpass</a>	157
<a href="#">\XINT_dsh_xisPos</a>	119	<a href="#">\XINT_dthb_nextfour</a>	157
<a href="#">\XINT_dsh_xiszero</a>	119	<a href="#">\XINT_dthb_small</a>	157
<a href="#">\XINT_dshr_fork</a>	119	<a href="#">\XINT_dthb_start</a>	157, 159
<a href="#">\XINT_dshr_xpositive</a>	119	<a href="#">\XINT_dthb_start_a</a>	157
<a href="#">\XINT_dshr_xzeroorneg</a>	119	<a href="#">\XINT_dthb_update</a>	157, 158
<a href="#">\XINT_dsl</a>	64	<a href="#">\XINT_dthb_update_a</a>	157
<a href="#">\xint_dsl_zero</a>	64	<a href="#">\XINT_e</a>	195
<a href="#">\XINT_dsr</a>	65	<a href="#">\XINT_e_end</a>	195
<a href="#">\XINT_dsr_a</a>	65	<a href="#">\XINT_eightrandomdigits</a>	150, 151, 260
<a href="#">\XINT_dsr_b</a>	65	<a href="#">\XINT_euc</a>	172, 173
<a href="#">\XINT_dsr_e</a>	65	<a href="#">\XINT_euc_a</a>	173
<a href="#">\XINT_dsr_fork</a>	64	<a href="#">\XINT_euc_AisZero</a>	173
<a href="#">\XINT_dsr_neg</a>	65	<a href="#">\XINT_euc_b</a>	173
<a href="#">\XINT_dsrr</a>	65, 103, 202, 227	<a href="#">\XINT_euc_BisZero</a>	173
<a href="#">\XINT_dsrr_a</a>	65	<a href="#">\XINT_euc_c</a>	173
<a href="#">\XINT_dsrr_b</a>	65	<a href="#">\XINT_euc_end</a>	173
<a href="#">\XINT_dsrr_e</a>	65	<a href="#">\XINT_euc_end_a</a>	173, 174
<a href="#">\XINT_dsrr_fork</a>	65	<a href="#">\XINT_euc_fork</a>	173
<a href="#">\XINT_dsrr_neg</a>	65	<a href="#">\xint_exchangetwo_keepbraces</a>	9, 88, 92, 105, 170, 171, 216, 317
<a href="#">\XINT_dsx_addzeros</a>	120, 138, 190--192, 199, 200, 207, 209, 218, 222, 225, 256	<a href="#">\XINT_expandableerror</a>	18, 45, 58, 152, 153
<a href="#">\XINT_dsx_addzerosnofuss</a>	120, 138, 229	<a href="#">\XINT_expandableerror_continue</a>	18
<a href="#">\XINT_dsx_AisNeg</a>	121	<a href="#">\XINT_expr:_unlock</a>	345, 346, 349, 350
<a href="#">\XINT_dsx_AisNeg_checkiffirstempty</a>	121	<a href="#">\XINT_expr_binop_inline</a>	316--318
<a href="#">\XINT_dsx_AisNeg_finish_notzero</a>	121	<a href="#">\XINT_expr_binop_inline_a</a>	316
<a href="#">\XINT_dsx_AisNeg_finish_zero</a>	121	<a href="#">\XINT_expr_binop_inline_b</a>	316
<a href="#">\XINT_dsx_AisPos</a>	121	<a href="#">\XINT_expr_binop_inline_c</a>	316
<a href="#">\XINT_dsx_AisPos_finish</a>	121	<a href="#">\XINT_expr_binop_inline_d</a>	316
<a href="#">\XINT_dsx_AisZero</a>	120, 121	<a href="#">\XINT_expr_binop_inline_e</a>	316
<a href="#">\XINT_dsx_append</a>	116, 120	<a href="#">\XINT_expr_binop_inline_end</a>	316
<a href="#">\XINT_dsx_end</a>	121	<a href="#">\XINT_expr_binop_pwd</a>	311
<a href="#">\XINT_dsx_fork</a>	120	<a href="#">\XINT_expr_countetc</a>	302
<a href="#">\XINT_dsx_xisNeg_Azero</a>	120	<a href="#">\XINT_expr_defbin_b</a>	315, 316
<a href="#">\XINT_dsx_xisNeg_checkA</a>	119, 120	<a href="#">\XINT_expr_defbin_c</a>	314--316
<a href="#">\XINT_dsx_xisPos</a>	119, 120	<a href="#">\XINT_expr_deflistopl_b</a>	318
<a href="#">\XINT_dsx_xisZero</a>	120	<a href="#">\XINT_expr_deflistopl_c</a>	318
<a href="#">\XINT_dtb_checkin</a>	159	<a href="#">\XINT_expr_deflistopr_b</a>	317
<a href="#">\XINT_dtb_finish</a>	159, 160	<a href="#">\XINT_expr_deflistopr_c</a>	317
<a href="#">\XINT_dtb_finish_a</a>	160	<a href="#">\XINT_expr_defuserfunc</a>	369
<a href="#">\XINT_dtb_main</a>	159	<a href="#">\XINT_expr_defvar</a>	334, 335
<a href="#">\XINT_dtb_neg</a>	159	<a href="#">\XINT_expr_defvar_getname</a>	334
<a href="#">\XINT_dtb_tobin</a>	159, 160	<a href="#">\XINT_expr_defvar_one</a>	333, 334
<a href="#">\XINT_dtb_tobin_a</a>	159	<a href="#">\XINT_expr_done</a>	297, 312, 313
<a href="#">\XINT_dth_checkin</a>	156	<a href="#">\XINT_expr_extra_)</a>	312
<a href="#">\XINT_dth_finish</a>	157, 159	<a href="#">\XINT_expr_foundend</a>	311, 314, 337
<a href="#">\XINT_dth_main</a>	157	<a href="#">\XINT_expr_foundop</a>	311, 312
<a href="#">\XINT_dth_neg</a>	156, 157	<a href="#">\XINT_expr_foundop_a</a>	311
		<a href="#">\XINT_expr_func</a>	309, 310
		<a href="#">\XINT_expr_func_!</a>	357
		<a href="#">\XINT_expr_func_?</a>	357
		<a href="#">\XINT_expr_func_@@</a>	338
		<a href="#">\XINT_expr_func_@@@</a>	338

\XINT_expr_func_@@@	338	\XINT_expr_getop	302--304, 307, 310--312, 314, 330--333, 340, 342, 343, 345, 347, 349
\XINT_expr_func_abs	355, 356	\XINT_expr_getop_a	310
\XINT_expr_func_all	364	\XINT_expr_getop_b	311
\XINT_expr_func_any	364	\XINT_expr_gobz_a	304
\XINT_expr_func_binomial	360	\XINT_expr_gobz_scandec_a	305
\XINT_expr_func_break	332	\XINT_expr_gobz_scandec_b	305, 306
\XINT_expr_func_ceil	356	\XINT_expr_gobz_scandec_c	306
\XINT_expr_func_divmod	360	\XINT_expr_gobz_scandec_endbycs	306
\XINT_expr_func_even	357, 358	\XINT_expr_gobz_scanint_b	304, 305
\XINT_expr_func_factorial	358	\XINT_expr_gobz_scanint_c	305
\XINT_expr_func_first	365	\XINT_expr_gobz_scanint_d	305
\XINT_expr_func_float	359, 360	\XINT_expr_gobz_scanint_endbycs	305
\XINT_expr_func_floor	356	\XINT_expr_gobz_startdec_a	305
\XINT_expr_func_frac	356	\XINT_expr_gotnil	304
\XINT_expr_func_gcd	362	\XINT_expr_inhex	296
\XINT_expr_func_if	365	\XINT_expr_iter:_A	345, 346
\XINT_expr_func_ifint	365	\XINT_expr_iter:_a	345
\XINT_expr_func_ifone	366	\XINT_expr_iter:_aa	345
\XINT_expr_func_ifsgn	366	\XINT_expr_iter:_Abort	346
\XINT_expr_func_isint	358	\XINT_expr_iter:_abort	346
\XINT_expr_func_isonone	358	\XINT_expr_iter:_b	345, 346
\XINT_expr_func_iter	345	\XINT_expr_iter:_Break	346
\XINT_expr_func_iterr	348	\XINT_expr_iter:_break	346
\XINT_expr_func_last	365	\XINT_expr_iter:_c	345
\XINT_expr_func_lcm	363	\XINT_expr_iter:_D	346
\XINT_expr_func_len	364, 365	\XINT_expr_iter:_d	345
\XINT_expr_func_max	363	\XINT_expr_iter:_E	346
\XINT_expr_func_min	363	\XINT_expr_iter:_end	345
\XINT_expr_func_mod	360	\XINT_expr_iter:_Goon	346
\XINT_expr_func_not	357	\XINT_expr_iter:_goon	346
\XINT_expr_func_num	355	\XINT_expr_iter:_noop	345
\XINT_expr_func_nuple	358	\XINT_expr_iter:_Omit	346
\XINT_expr_func_odd	357	\XINT_expr_iter:_omit	346
\XINT_expr_func_opxadd	342	\XINT_expr_iterr:_A	349
\XINT_expr_func_opxmud	342	\XINT_expr_iterr:_a	349
\XINT_expr_func_pfactorial	361	\XINT_expr_iterr:_aa	349
\XINT_expr_func_reduce	355	\XINT_expr_iterr:_Abort	350
\XINT_expr_func_quo	362	\XINT_expr_iterr:_abort	349
\XINT_expr_func_randrange	361	\XINT_expr_iterr:_b	349
\XINT_expr_func_reduce	355	\XINT_expr_iterr:_Break	350
\XINT_expr_func_rem	362	\XINT_expr_iterr:_break	349
\XINT_expr_func_reversed	365	\XINT_expr_iterr:_c	349
\XINT_expr_func_round	359	\XINT_expr_iterr:_D	349, 350
\XINT_expr_func_rrseq	346	\XINT_expr_iterr:_d	349
\XINT_expr_func_rseq	343	\XINT_expr_iterr:_E	349, 350
\XINT_expr_func_seqx	340	\XINT_expr_iterr:_end	349
\XINT_expr_func_sgn	356	\XINT_expr_iterr:_Goon	349, 350
\XINT_expr_func_sqr	357	\XINT_expr_iterr:_goon	349
\XINT_expr_func_sqrt	359	\XINT_expr_iterr:_noop	349
\XINT_expr_func_subx	343	\XINT_expr_iterr:_Omit	349, 350
\XINT_expr_func_trunc	359	\XINT_expr_iterr:_omit	349
\XINT_expr_func_unknown	331, 332	\XINT_expr_iterrx	348
\XINT_expr_func_xor	364	\XINT_expr_iterry	348, 349
\XINT_expr_getnext	298, 302, 303, 312, 314, 317, 318, 320, 328--330	\XINT_expr_iterx	345
\XINT_expr_getnext_a	302	\XINT_expr_itory	345
\XINT_expr_getnextfork	302, 303	\XINT_expr_itself_[]	321

\XINT_expr_lockit .....	296, 334, 344, 346--349	\XINT_expr_precedence_[ .....	331
\XINT_expr_lockscan .....	296	\XINT_expr_precedence_[[: .....	321
\XINT_expr_macrofunc_ .....	371	\XINT_expr_precedence_a .....	313
\XINT_expr_makedummy .....	336, 337	\XINT_expr_print .....	297, 377
\XINT_expr_missing_) .....	312, 313	\XINT_expr_redefinemacros .....	376, 378
\XINT_expr_newfunction .....	370, 371	\XINT_expr_redefineprints .....	377
\XINT_expr_onliteral_@ .....	338	\XINT_expr_rrseq:_A .....	347, 348
\XINT_expr_onliteral_` .....	304	\XINT_expr_rrseq:_a .....	347
\XINT_expr_onliteral_add .....	341	\XINT_expr_rrseq:_aa .....	347
\XINT_expr_onliteral_add_f .....	341	\XINT_expr_rrseq:_Abort .....	348
\XINT_expr_onliteral_bool .....	332	\XINT_expr_rrseq:_abort .....	347
\XINT_expr_onliteral_mul .....	341	\XINT_expr_rrseq:_b .....	347
\XINT_expr_onliteral_mul_f .....	342	\XINT_expr_rrseq:_Break .....	348
\XINT_expr_onliteral_nil .....	337	\XINT_expr_rrseq:_break .....	347
\XINT_expr_onliteral_protect .....	332	\XINT_expr_rrseq:_c .....	347
\XINT_expr_onliteral_qfloat .....	332	\XINT_expr_rrseq:_D .....	348
\XINT_expr_onliteral_qfrac .....	332	\XINT_expr_rrseq:_d .....	347
\XINT_expr_onliteral_qint .....	332	\XINT_expr_rrseq:_E .....	348
\XINT_expr_onliteral_qrand .....	333	\XINT_expr_rrseq:_end .....	347
\XINT_expr_onliteral_qraw .....	332	\XINT_expr_rrseq:_Goon .....	348
\XINT_expr_onliteral_random .....	332	\XINT_expr_rrseq:_goon .....	347
\XINT_expr_onliteral_seq .....	339	\XINT_expr_rrseq:_noop .....	347
\XINT_expr_onliteral_seq_a .....	339, 341--343, 345, 346, 348	\XINT_expr_rrseq:_Omit .....	348
\XINT_expr_onliteral_seq_b .....	339	\XINT_expr_rrseq:_omit .....	347
\XINT_expr_onliteral_seq_c .....	339	\XINT_expr_rrseqx .....	346, 347
\XINT_expr_onliteral_seq_d .....	339	\XINT_expr_rrseqy .....	347
\XINT_expr_onliteral_seq_e .....	339	\XINT_expr_rseq:_A .....	344
\XINT_expr_onliteral_seq_f .....	339	\XINT_expr_rseq:_a .....	344
\XINT_expr_onliteral_subs .....	342	\XINT_expr_rseq:_aa .....	343, 344
\XINT_expr_onliteral_subs_f .....	342	\XINT_expr_rseq:_Abort .....	344
\XINT_expr_onliteral_togl .....	332	\XINT_expr_rseq:_abort .....	344
\XINT_expr_op:_a .....	342	\XINT_expr_rseq:_b .....	344
\XINT_expr_op:_b .....	342	\XINT_expr_rseq:_Break .....	344
\XINT_expr_op:_c .....	342	\XINT_expr_rseq:_break .....	344
\XINT_expr_op:_d .....	342	\XINT_expr_rseq:_c .....	344
\XINT_expr_op:_end .....	342	\XINT_expr_rseq:_D .....	344
\XINT_expr_op:_noop .....	342	\XINT_expr_rseq:_d .....	344
\XINT_expr_op_ .....	314, 317, 320, 328, 329	\XINT_expr_rseq:_E .....	344
\XINT_expr_op_! .....	330	\XINT_expr_rseq:_end .....	344
\XINT_expr_op_!? .....	337	\XINT_expr_rseq:_Goon .....	344
\XINT_expr_op:_ .....	320	\XINT_expr_rseq:_goon .....	344
\XINT_expr_op:_] .....	321	\XINT_expr_rseq:_noop .....	344
\XINT_expr_op_? .....	330	\XINT_expr_rseq:_Omit .....	344
\XINT_expr_op_?? .....	330	\XINT_expr_rseq:_omit .....	344
\XINT_expr_op_?a .....	330	\XINT_expr_rseqx .....	343
\XINT_expr_op_?checka .....	330	\XINT_expr_rseqy .....	343
\XINT_expr_op_?checkb .....	330	\XINT_expr_scan_macropar .....	303
\XINT_expr_op_?checkc .....	330	\XINT_expr_scan_nbr_or_func .....	303, 304
\XINT_expr_op_[ .....	331	\XINT_expr_scandec_a .....	304, 305
\XINT_expr_op_[[: .....	321	\XINT_expr_scandec_b .....	305, 306
\XINT_expr_op__ .....	310, 333	\XINT_expr_scandec_c .....	306
\XINT_expr_precedence_ .....	311, 334, 337, 338	\XINT_expr_scandec_d .....	306
\XINT_expr_precedence_! .....	330	\XINT_expr_scandec_endbycs .....	306
\XINT_expr_precedence_!? .....	313	\XINT_expr_scanexp_a .....	305--307
\XINT_expr_precedence:_] .....	321	\XINT_expr_scanexp_b .....	307
\XINT_expr_precedence_? .....	311, 330	\XINT_expr_scanexp_bb .....	307
		\XINT_expr_scanexp_c .....	307



\XINT_expr_scanexp_cb	307	\XINT_expr_tmpvar	334
\XINT_expr_scanexp_d	307	\XINT_expr_unexpectedtoken	296
\XINT_expr_scanexp_db	307	\XINT_expr_unknown_function	331
\XINT_expr_scanexp_endbycs	307	\XINT_expr_unknown_operator	311, 312
\XINT_expr_scanexp_endbycs_b	307	\XINT_expr_unknown_variable	333
\XINT_expr_scanexpr_endbycs	307	\XINT_expr_unlock	
\XINT_expr_scanfunc	304, 309		296--299, 314, 315, 317, 318, 320, 321, 328--332, 334, 338--350, 355--366, 369, 371, 377
\XINT_expr_scanfunc_a	309	\XINT_expr_unlock_a	296
\XINT_expr_scanfunc_b	309	\XINT_expr_unlock_hex_in	296, 307
\XINT_expr_scanhex_I	304, 307	\XINT_expr_unlock_sp	299, 300, 377
\XINT_expr_scanhex_transition	308	\XINT_expr_unpackvar	302, 303
\XINT_expr_scanhexI_a	307, 308	\XINT_expr_until	314, 317, 320, 328, 329
\XINT_expr_scanhexI_aa	308	\XINT_expr_until:_a	320
\XINT_expr_scanhexI_b	308	\XINT_expr_until:_b	321
\XINT_expr_scanhexI_bgob	308	\XINT_expr_until_end_a	298
\XINT_expr_scanhexII_a	308, 309	\XINT_expr_usethe	297, 299, 300
\XINT_expr_scanhexII_aa	308	\XINT_expr_var_@	338
\XINT_expr_scanhexII_b	308, 309	\XINT_expr_var_abort	337
\XINT_expr_scanhexII_bgob	308, 309	\XINT_expr_var_nil	337
\XINT_expr_scanint_a	304	\XINT_expr_var_omit	337
\XINT_expr_scanint_b	304	\XINT_expr_wrap	297--299, 370
\XINT_expr_scanint_c	304, 305	\XINT_expr_wrapit	370
\XINT_expr_scanint_d	304, 305	\XINT_fac_bigloop_a	110, 111
\XINT_expr_scanint_endbycs	304	\XINT_fac_bigloop_b	111
\XINT_expr_scanop_a	311, 314	\XINT_fac_bigloop_exit	111
\XINT_expr_scanop_b	311	\XINT_fac_bigloop_loop	111
\XINT_expr_scanop_c	311	\XINT_fac_bigloop_mul	111
\XINT_expr_scanop_d	311	\XINT_fac_checksize	110
\XINT_expr_seq:_A	340, 341	\XINT_fac_fork	110, 212
\XINT_expr_seq:_a	340	\XINT_fac_loop_exit	111, 112
\XINT_expr_seq:_aa	340	\XINT_fac_medloop_a	110, 111
\XINT_expr_seq:_Abort	341	\XINT_fac_medloop_b	111
\XINT_expr_seq:_abort	341	\XINT_fac_medloop_loop	111
\XINT_expr_seq:_b	340, 341	\XINT_fac_medloop_mul	111
\XINT_expr_seq:_Break	341	\XINT_fac_neg	110
\XINT_expr_seq:_break	341	\XINT_fac_smallloop_a	110, 111
\XINT_expr_seq:_c	340, 341	\XINT_fac_smallloop_loop	112
\XINT_expr_seq:_D	341	\XINT_fac_smallloop_mul	112
\XINT_expr_seq:_d	341	\XINT_fac_toobig	110, 111
\XINT_expr_seq:_E	341	\XINT_fac_zero	110
\XINT_expr_seq:_end	340, 341	\XINT_factortens	186, 194
\XINT_expr_seq:_Goon	341	\XINT_factortens_a	186
\XINT_expr_seq:_goon	341	\XINT_factortens_b	186
\XINT_expr_seq:_noop	340	\XINT_factortens_c	186
\XINT_expr_seq:_Omit	341	\XINT_factortens_cc	186
\XINT_expr_seq:_omit	341	\XINT_factortens_d	186
\XINT_expr_seq:_empty?	327, 328, 340, 341	\XINT_factortens_e	186
\XINT_expr_startdec	303, 304	\XINT_factortens_f	186
\XINT_expr_startdec_a	305	\XINT_factortens_g	186
\XINT_expr_startint	304	\XINT_factortens_x	186
\XINT_expr_subexpr	302	\XINT_factortens_y	186
\XINT_expr_subx:_end	343	\XINT_factortens_yy	186
\XINT_expr_tmpa	334, 335, 369--371	\XINT_factortens_z	186
\XINT_expr_tmpb	334, 369, 370	\XINT_fadd	208
\XINT_expr_tmpc	334, 335, 369	\XINT_fadd_a	208
\XINT_expr_tmpd	334, 335	\XINT_fadd_Aa	209
\XINT_expr_tmpval	334		

\XINT_fadd_Azero .....	208	\XINT_fgtd_e .....	277
\XINT_fadd_B .....	209	\XINT_fgtd_f .....	277
\XINT_fadd_b .....	208	\XINT_fgtd_g .....	277
\XINT_fadd_Ba .....	209	\XINT_fgtd_h .....	277
\XINT_fadd_Bb .....	209	\XINT_first:f:csv_a .....	53
\XINT_fadd_Bzero .....	208, 210	\xint_firstofone .....	9, 10, 18, 20, 22, 23, 46, 53, 58, 231, 277, 351, 352, 354, 377
\XINT_fadd_C .....	209	\xint_firstofthree .....	114, 181, 310
\XINT_fadd_c .....	208, 210	\xint_firstoftwo .....	9, 17, 38, 39, 42, 58, 76, 77, 103, 128, 158, 189, 201, 234, 249, 306, 309, 333, 354, 355, 362, 374, 377
\XINT_fadd_D_a .....	209	\XINT_FL_add_a .....	233--235
\XINT_fadd_D_b .....	209	\XINT_FL_add_b .....	234
\XINT_fadd_D_c .....	209	\XINT_FL_add_c .....	234
\XINT_fadd_D_exit .....	209	\XINT_FL_add_d .....	234
\XINT_fadd_E .....	209	\XINT_FL_add_zero .....	234
\XINT_fadd_F .....	209, 210	\XINT_FL_binom_a .....	253
\XINT_fadd_G .....	210	\XINT_FL_binom_aa .....	254
\XINT_fadd_iszero .....	209, 210	\XINT_FL_binom_ab .....	254
\XINT_fcmp .....	221	\XINT_FL_binom_fork .....	253
\XINT_fcmp_A .....	221	\XINT_FL_binom_neg .....	254
\XINT_fcmp_B .....	221	\XINT_FL_binom_one .....	254
\XINT_fcmp_C .....	221	\XINT_FL_binom_toobig .....	254
\XINT_fcmp_D .....	221, 222	\XINT_FL_binom_zero .....	254
\XINT_fcmp_E .....	222	\XINT_FL_div_a .....	236
\XINT_fcmp_Fd .....	222	\XINT_FL_div_b .....	236
\XINT_fcmp_Fe .....	222	\XINT_FL_fac_addzeros .....	247, 248
\XINT_fcmp_firstneg .....	221	\XINT_FL_fac_addzeros_exit .....	248
\XINT_fcmp_firstzero .....	221	\XINT_FL_fac_big .....	245, 246
\XINT_fcmp_Fn .....	222	\XINT_FL_fac_bigloop_a .....	246
\XINT_fcmp_Fo .....	222	\XINT_FL_fac_bigloop_b .....	246
\XINT_fcmp_minusminus .....	221	\XINT_FL_fac_bigloop_loop .....	246, 247
\XINT_fcmp_nonneg_a .....	221	\XINT_FL_fac_bigloop_mul .....	247
\XINT_fcmp_pos .....	221	\XINT_FL_fac_countdigits .....	246, 250
\XINT_fcmp_secondneg .....	221	\XINT_FL_fac_countdone .....	246
\XINT_fcmp_secondzero .....	221	\XINT_FL_fac_fork_a .....	244
\XINT_fcmp_zerozero .....	221	\XINT_FL_fac_fork_b .....	245, 254
\XINT_FDg .....	61	\XINT_FL_fac_increaseP .....	246
\XINT_fdg .....	61	\XINT_FL_fac_isneg .....	244, 245
\XINT_fdiv .....	213	\XINT_FL_fac_iszero .....	244, 245
\XINT_fdiv_A .....	213	\XINT_FL_fac_loop_exit .....	246--248
\XINT_fdiv_B .....	213, 214	\XINT_FL_fac_med .....	245, 246
\XINT_fdiv_C .....	214	\XINT_FL_fac_medloop_a .....	246, 247
\XINT_fgeq .....	218	\XINT_FL_fac_medloop_b .....	247
\XINT_fgeq_A .....	218--220	\XINT_FL_fac_medloop_loop .....	247
\XINT_fgeq_B .....	218	\XINT_FL_fac_medloop_mul .....	247
\XINT_fgeq_C .....	218	\XINT_FL_fac_minimulwc_a .....	248, 249
\XINT_fgeq_D .....	218	\XINT_FL_fac_minimulwc_b .....	248
\XINT_fgeq_E .....	218	\XINT_FL_fac_minimulwc_c .....	248
\XINT_fgeq_Fd .....	218	\XINT_FL_fac_minimulwc_d .....	248
\XINT_fgeq_Fe .....	218	\XINT_FL_fac_minimulwc_e .....	248
\XINT_fgeq_Fn .....	218	\XINT_FL_fac_mul .....	246--248, 251--253
\XINT_fgeq_Fo .....	218, 219	\XINT_FL_fac_mul_a .....	248
\XINT_fgeq_Zi .....	218	\XINT_FL_fac_out .....	244, 246, 252--254
\XINT_fgeq_Zii .....	218	\XINT_FL_fac_small .....	245, 246
\XINT_fgtd_a .....	277	\XINT_FL_fac_smallloop_a .....	246, 247
\XINT_fgtd_b .....	277	\XINT_FL_fac_smallloop_loop .....	248
\XINT_fgtd_c .....	277		
\XINT_fgtd_d .....	277, 278		
\XINT_fgtd_da .....	277		



\XINT_FL_fac_smallloop_mul .....	248	\XINT_flexpr_func_@@@ .....	338
\XINT_FL_fac_smallmul .....	248, 249	\XINT_flexpr_func_abs .....	356
\XINT_FL_fac_smallmul_end .....	249	\XINT_flexpr_func_all .....	364
\XINT_FL_fac_toobig .....	245	\XINT_flexpr_func_any .....	364
\XINT_FL_fac_vbig .....	245, 246	\XINT_flexpr_func_binomial .....	361
\XINT_FL_fac_vbigloop_a .....	246	\XINT_flexpr_func_break .....	332
\XINT_FL_fac_vbigloop_loop .....	246	\XINT_flexpr_func_ceil .....	356
\XINT_FL_mul_a .....	235	\XINT_flexpr_func_divmod .....	360
\XINT_FL_mul_b .....	235	\XINT_flexpr_func_even .....	358
\XINT_FL_pfac_a .....	250	\XINT_flexpr_func_factorial .....	358
\XINT_FL_pfac_addzeroes .....	250, 251	\XINT_flexpr_func_first .....	365
\XINT_FL_pfac_addzeroes_exit .....	251	\XINT_flexpr_func_float .....	360
\XINT_FL_pfac_b .....	250, 251	\XINT_flexpr_func_floor .....	356
\XINT_FL_pfac_bigloop .....	251, 252	\XINT_flexpr_func_frac .....	356
\XINT_FL_pfac_bigloop_a .....	252	\XINT_flexpr_func_gcd .....	362
\XINT_FL_pfac_bigloop_b .....	252	\XINT_flexpr_func_if .....	365
\XINT_FL_pfac_end_ .....	251--253	\XINT_flexpr_func_ifint .....	366
\XINT_FL_pfac_end_i .....	251--253	\XINT_flexpr_func_ifone .....	366
\XINT_FL_pfac_end_iii .....	251, 252	\XINT_flexpr_func_ifsgn .....	366
\XINT_FL_pfac_end_iii .....	251, 252	\XINT_flexpr_func_isint .....	358
\XINT_FL_pfac_fork .....	249, 250	\XINT_flexpr_func_isonone .....	358
\XINT_FL_pfac_increaseP .....	250, 254	\XINT_flexpr_func_iter .....	345
\XINT_FL_pfac_medloop .....	251, 252	\XINT_flexpr_func_iterr .....	348
\XINT_FL_pfac_medloop_a .....	251	\XINT_flexpr_func_last .....	365
\XINT_FL_pfac_medloop_b .....	251, 252	\XINT_flexpr_func_lcm .....	363
\XINT_FL_pfac_neg .....	250	\XINT_flexpr_func_len .....	365
\XINT_FL_pfac_one .....	250	\XINT_flexpr_func_max .....	363
\XINT_FL_pfac_outofrange .....	250	\XINT_flexpr_func_min .....	363
\XINT_FL_pfac_smallloop .....	251	\XINT_flexpr_func_mod .....	360
\XINT_FL_pfac_smallloop_a .....	251	\XINT_flexpr_func_not .....	357
\XINT_FL_pfac_smallloop_b .....	251	\XINT_flexpr_func_num .....	355
\XINT_FL_pfac_vbigloop .....	251, 252	\XINT_flexpr_func_nuple .....	358
\XINT_FL_pfac_vbigloop_a .....	252	\XINT_flexpr_func_odd .....	357
\XINT_FL_pfac_zero .....	250	\XINT_flexpr_func_opxadd .....	342
\XINT_FL_sqrt_a .....	255	\XINT_flexpr_func_opxmulo .....	342
\XINT_FL_sqrt_isneg .....	255	\XINT_flexpr_func_pfactorial .....	361
\XINT_FL_sqrt_iszero .....	255	\XINT_flexpr_func_preduce .....	355
\XINT_FL_sqrt_pos .....	255	\XINT_flexpr_func_quo .....	362
\XINT_fladd_chkopt .....	233	\XINT_flexpr_func_randrange .....	361
\XINT_fladd_noopt .....	233	\XINT_flexpr_func_reduce .....	355
\XINT_fladd_opt .....	233	\XINT_flexpr_func_rem .....	362
\XINT_fladd_opt_a .....	233, 234	\XINT_flexpr_func_reversed .....	365
\XINT_flbinom_chkopt .....	253	\XINT_flexpr_func_round .....	359
\XINT_flbinom_noopt .....	253	\XINT_flexpr_func_rrseq .....	346
\XINT_flbinom_opt .....	253	\XINT_flexpr_func_rseq .....	343
\XINT_fldiv_chkopt .....	236	\XINT_flexpr_func_seqx .....	340
\XINT_fldiv_noopt .....	236	\XINT_flexpr_func_sgn .....	356
\XINT_fldiv_opt .....	236	\XINT_flexpr_func_sqr .....	357
\XINT_fldiv_opt_a .....	236	\XINT_flexpr_func_sqrt .....	359
\XINT_flen .....	181	\XINT_flexpr_func_subx .....	343
\XINT_flet_macro .....	38, 43, 44	\XINT_flexpr_func_trunc .....	359
\XINT_flet_sp? .....	38	\XINT_flexpr_func_xor .....	364
\XINT_flet_zapsp .....	38, 43, 44	\XINT_flexpr_noopt .....	299, 377
\XINT_flexpr_func_! .....	357	\XINT_flexpr_op_! .....	330
\XINT_flexpr_func_? .....	357	\XINT_flexpr_op_!? .....	337
\XINT_flexpr_func_@@ .....	338	\XINT_flexpr_op_:] .....	321
\XINT_flexpr_func_@@@ .....	338	\XINT_flexpr_op_? .....	330

\XINT_flexpr_op_	331	\XINT_flpow_loopII_odda	239
\XINT_flexpr_op_][:	321	\XINT_flpow_noopt	237
\XINT_flexpr_op__	333	\XINT_flpow_opt	237
\XINT_flexpr_print	299	\XINT_flpow_opt_a	237
\XINT_flexpr_until_end_a	298	\XINT_flpow_truncate	238, 239, 243, 244
\XINT_flexpr_withopt_a	299	\XINT_flpow_truncate_a	238
\XINT_flexpr_withopt_b	299, 377	\XINT_flpow_zero	238, 242
\XINT_flexpr_wrap	299, 377	\XINT_flpower_a	242
\XINT_flfac_chkopt	244	\XINT_flpower_aa	242
\XINT_flfac_noopt	244	\XINT_flpower_ab	242
\XINT_flfac_opt	244	\XINT_flpower_b	242, 243
\XINT_flfac_opt_a	244	\XINT_flpower_BisZero	241, 242
\XINT_flmul_chkopt	235	\XINT_flpower_checkB_a	241, 242
\XINT_flmul_noopt	235	\XINT_flpower_checkB_b	242
\XINT_flmul_opt	235	\XINT_flpower_checkB_c	242
\XINT_flmul_opt_a	235	\XINT_flpower_checkB_d	242
\XINT_float_chkopt	224	\XINT_flpower_chkopt	240, 241
\XINT_float_neg	225	\XINT_flpower_IItoIII	243, 244
\XINT_float_noopt	224	\XINT_flpower_ItoIII	243
\XINT_float_opt	224	\XINT_flpower_loopI	243
\XINT_float_opt_a	224	\XINT_flpower_loopI_even	243
\XINT_float_pos	225	\XINT_flpower_loopI_odd	243
\XINT_float_pos_done	225, 257	\XINT_flpower_loopII	243
\XINT_float_post	224, 225	\XINT_flpower_loopII_even	243
\XINT_float_zero	225	\XINT_flpower_loopII_odd	243, 244
\XINT_floate_chkopt	257	\XINT_flpower_loopII_odda	244
\XINT_floate_neg	257	\XINT_flpower_noopt	241, 242
\XINT_floate_noopt	257	\XINT_flpower_opt	241, 242
\XINT_floate_opt	257	\XINT_flpower_opt_a	242
\XINT_floate_opt_a	257	\XINT_flpower_toloopI	243
\XINT_floate_pos	257	\XINT_flpower_toloopII	243, 244
\XINT_floate_post	257	\XINT_flpowerh_a	241
\XINT_floate_zero	257	\XINT_flpowerh_b	241
\XINT_flpfac_chkopt	249	\XINT_flpowerh_c	241
\XINT_flpfac_noopt	249	\XINT_flpowerh_d	241
\XINT_flpfac_opt	249	\XINT_flpowerh_e	241
\XINT_flpfac_opt_b	249, 250	\XINT_flpowerh_finish	241
\XINT_flpow_a	238	\XINT_flpowerh_i	241
\XINT_flpow_aa	237	\XINT_flpowerh_int	241
\XINT_flpow_ab	238	\XINT_flpowseries	268
\XINT_flpow_b	238	\XINT_flpowseries_chkopt	267
\XINT_flpow_BisZero	237	\XINT_flpowseries_dont_i	268
\XINT_flpow_checkB_a	237	\XINT_flpowseries_dont_ii	268
\XINT_flpow_checkB_b	237	\XINT_flpowseries_exit_i	268, 269
\XINT_flpow_checkB_c	237	\XINT_flpowseries_exit_ii	269
\XINT_flpow_checkB_d	237	\XINT_flpowseries_loop_i	268
\XINT_flpow_chkopt	236, 237	\XINT_flpowseries_loop_ii	268, 269
\XINT_flpow_III	238--240, 243, 244	\XINT_flpowseries_loop_pre	268, 269
\XINT_flpow_IIIend	240	\XINT_flpowseries_noopt	268
\XINT_flpow_IItoIII	239	\XINT_flpowseries_opt	267, 268
\XINT_flpow_ItoIII	238	\XINT_flpowseriesx	269
\XINT_flpow_loopI	238, 239	\XINT_flpowseriesx_chkopt	269
\XINT_flpow_loopI_even	238, 239	\XINT_flpowseriesx_noopt	269
\XINT_flpow_loopI_odd	238, 239	\XINT_flpowseriesx_opt	269
\XINT_flpow_loopII	239	\XINT_flpowseriesx_pre	269
\XINT_flpow_loopII_even	239	\XINT_flseqa::csv	327
\XINT_flpow_loopII_odd	239	\XINT_flseqb::csv	328

<a href="#">\XINT_flseqb::csv_a</a>	<a href="#">328</a>	<a href="#">\XINT_for_ifstar</a>	<a href="#">38</a>
<a href="#">\XINT_flseqb::csv_n</a>	<a href="#">328</a>	<a href="#">\XINT_for_last?</a>	<a href="#">39</a>
<a href="#">\XINT_flseqb::csv_p</a>	<a href="#">328</a>	<a href="#">\XINT_for_last?yes</a>	<a href="#">39, 42, 43</a>
<a href="#">\XINT_flseqb::csv_z</a>	<a href="#">328</a>	<a href="#">\XINT_forever</a>	<a href="#">39, 40</a>
<a href="#">\XINT_flseqb:f:csv</a>	<a href="#">368</a>	<a href="#">\XINT_forever_a</a>	<a href="#">40, 41</a>
<a href="#">\XINT_flseqb:f:csv_a</a>	<a href="#">368</a>	<a href="#">\XINT_forever_b</a>	<a href="#">41</a>
<a href="#">\XINT_flseqb:f:csv_bg</a>	<a href="#">368</a>	<a href="#">\XINT_forever_c</a>	<a href="#">41</a>
<a href="#">\XINT_flseqb:f:csv_bl</a>	<a href="#">368</a>	<a href="#">\XINT_forever_d</a>	<a href="#">41</a>
<a href="#">\XINT_flseqb:f:csv_n</a>	<a href="#">368</a>	<a href="#">\XINT_forever_opt_a</a>	<a href="#">41</a>
<a href="#">\XINT_flseqb:f:csv_na</a>	<a href="#">368</a>	<a href="#">\XINT_forever_opt_c</a>	<a href="#">41</a>
<a href="#">\XINT_flseqb:f:csv_p</a>	<a href="#">368</a>	<a href="#">\XINT_forfour_d</a>	<a href="#">42, 43</a>
<a href="#">\XINT_flseqb:f:csv_pa</a>	<a href="#">368</a>	<a href="#">\XINT_forpair_d</a>	<a href="#">42</a>
<a href="#">\XINT_flseqb:f:csv_pb</a>	<a href="#">368</a>	<a href="#">\XINT_forthree_d</a>	<a href="#">42</a>
<a href="#">\XINT_flseqb:f:csv_pc</a>	<a href="#">368</a>	<a href="#">\XINT_forx</a>	<a href="#">38, 39</a>
<a href="#">\XINT_flsqrt</a>	<a href="#">255</a>	<a href="#">\XINT_forx_d</a>	<a href="#">39</a>
<a href="#">\XINT_flsqrt_a</a>	<a href="#">255, 256</a>	<a href="#">\XINT_forx_empty?</a>	<a href="#">39</a>
<a href="#">\XINT_flsqrt_again</a>	<a href="#">256</a>	<a href="#">\XINT_forx_forever?</a>	<a href="#">39</a>
<a href="#">\XINT_flsqrt_again_a</a>	<a href="#">256</a>	<a href="#">\XINT_fpow</a>	<a href="#">211</a>
<a href="#">\XINT_flsqrt_b</a>	<a href="#">256</a>	<a href="#">\XINT_fpow_fork</a>	<a href="#">212</a>
<a href="#">\XINT_flsqrt_c</a>	<a href="#">256</a>	<a href="#">\XINT_fpow_neg</a>	<a href="#">212</a>
<a href="#">\XINT_flsqrt_chkopt</a>	<a href="#">255</a>	<a href="#">\XINT_fpow_pos</a>	<a href="#">212</a>
<a href="#">\XINT_flsqrt_d</a>	<a href="#">256</a>	<a href="#">\XINT_fpow_pos_A</a>	<a href="#">212</a>
<a href="#">\XINT_flsqrt_f</a>	<a href="#">256</a>	<a href="#">\XINT_fpow_pos_B</a>	<a href="#">212</a>
<a href="#">\XINT_flsqrt_finish</a>	<a href="#">256</a>	<a href="#">\XINT_fpow_zero</a>	<a href="#">212</a>
<a href="#">\XINT_flsqrt_g</a>	<a href="#">256</a>	<a href="#">\XINT_fppowseries</a>	<a href="#">266</a>
<a href="#">\XINT_flsqrt_h</a>	<a href="#">256</a>	<a href="#">\XINT_fppowseries_dont_i</a>	<a href="#">266</a>
<a href="#">\XINT_flsqrt_noopt</a>	<a href="#">255</a>	<a href="#">\XINT_fppowseries_dont_ii</a>	<a href="#">266</a>
<a href="#">\XINT_flsqrt_opt</a>	<a href="#">255</a>	<a href="#">\XINT_fppowseries_exit_i</a>	<a href="#">266, 267</a>
<a href="#">\XINT_flsqrt_opt_a</a>	<a href="#">255</a>	<a href="#">\XINT_fppowseries_exit_ii</a>	<a href="#">267</a>
<a href="#">\XINT_flsub_chkopt</a>	<a href="#">234</a>	<a href="#">\XINT_fppowseries_loop_i</a>	<a href="#">266</a>
<a href="#">\XINT_flsub_noopt</a>	<a href="#">234</a>	<a href="#">\XINT_fppowseries_loop_ii</a>	<a href="#">266, 267</a>
<a href="#">\XINT_flsub_opt</a>	<a href="#">234</a>	<a href="#">\XINT_fppowseries_loop_pre</a>	<a href="#">266, 267</a>
<a href="#">\XINT_flsub_opt_a</a>	<a href="#">235</a>	<a href="#">\XINT_fppowseriesx</a>	<a href="#">267</a>
<a href="#">\XINT_fmax</a>	<a href="#">219</a>	<a href="#">\XINT_fppowseriesx_pre</a>	<a href="#">267</a>
<a href="#">\XINT_fmax_A</a>	<a href="#">219</a>	<a href="#">\XINT_fprdxpr</a>	<a href="#">213</a>
<a href="#">\XINT_fmax_firstneg</a>	<a href="#">219</a>	<a href="#">\XINT_fprod_finished</a>	<a href="#">213</a>
<a href="#">\XINT_fmax_minusminus</a>	<a href="#">219</a>	<a href="#">\XINT_fprod_loop_a</a>	<a href="#">213</a>
<a href="#">\XINT_fmax_nonneg_a</a>	<a href="#">219</a>	<a href="#">\XINT_fprod_loop_b</a>	<a href="#">213</a>
<a href="#">\XINT_fmax_nonneg_b</a>	<a href="#">219, 220</a>	<a href="#">\XINT_fprod_loop_c</a>	<a href="#">213</a>
<a href="#">\XINT_fmax_secondneg</a>	<a href="#">219</a>	<a href="#">\XINT_frac_gen</a>	<a href="#">183, 184</a>
<a href="#">\XINT_fmin</a>	<a href="#">220</a>	<a href="#">\XINT_frac_gen_A</a>	<a href="#">184</a>
<a href="#">\XINT_fmin_A</a>	<a href="#">220</a>	<a href="#">\XINT_frac_gen_B</a>	<a href="#">184</a>
<a href="#">\XINT_fmin_firstneg</a>	<a href="#">220</a>	<a href="#">\XINT_frac_gen_Ba</a>	<a href="#">184</a>
<a href="#">\XINT_fmin_minusminus</a>	<a href="#">220</a>	<a href="#">\XINT_frac_gen_Bb</a>	<a href="#">184</a>
<a href="#">\XINT_fmin_nonneg_a</a>	<a href="#">220</a>	<a href="#">\XINT_frac_gen_Bc</a>	<a href="#">184</a>
<a href="#">\XINT_fmin_nonneg_b</a>	<a href="#">219, 220</a>	<a href="#">\XINT_frac_gen_Bd</a>	<a href="#">184</a>
<a href="#">\XINT_fmin_secondneg</a>	<a href="#">220</a>	<a href="#">\XINT_frac_gen_C</a>	<a href="#">184</a>
<a href="#">\XINT_fmud</a>	<a href="#">210</a>	<a href="#">\XINT_frac_gen_Ca</a>	<a href="#">185</a>
<a href="#">\XINT_fmud_a</a>	<a href="#">210, 211</a>	<a href="#">\XINT_frac_gen_Cb</a>	<a href="#">185</a>
<a href="#">\XINT_fmud_b</a>	<a href="#">211</a>	<a href="#">\XINT_frac_gen_Cc</a>	<a href="#">185</a>
<a href="#">\XINT_fmud_c</a>	<a href="#">211</a>	<a href="#">\XINT_frac_gen_F</a>	<a href="#">185</a>
<a href="#">\XINT_fmud_d</a>	<a href="#">211</a>	<a href="#">\XINT_frac_gen_G</a>	<a href="#">185</a>
<a href="#">\XINT_fmud_e</a>	<a href="#">211</a>	<a href="#">\XINT_frac_gen_Ga</a>	<a href="#">185, 186</a>
<a href="#">\XINT_fmud_zero</a>	<a href="#">210, 211</a>	<a href="#">\XINT_frac_gen_Gdivbyzero</a>	<a href="#">185</a>
<a href="#">\XINT_for</a>	<a href="#">38</a>	<a href="#">\XINT_frac_gen_Gdivbyzero_a</a>	<a href="#">185</a>
<a href="#">\XINT_for_d</a>	<a href="#">38, 39</a>	<a href="#">\XINT_frac_gen_zero</a>	<a href="#">186</a>
<a href="#">\XINT_for_forever?</a>	<a href="#">38</a>	<a href="#">\XINT_fracfrac_A</a>	<a href="#">192, 193</a>

\XINT_fracfrac_B .....	192	\XINT_gcd_Aiszero .....	149, 167
\XINT_fracfrac_C .....	192	\XINT_gcd_BisZero .....	149, 167
\XINT_fracfrac_D .....	192	\XINT_gcd_Biszero .....	149, 167
\XINT_fracfrac_E .....	192	\XINT_gcd_CheckRem .....	149, 168
\XINT_fracisone .....	217	\XINT_gcd_end .....	149, 168
\XINT_fsqr .....	211	\XINT_gcd_fork .....	148, 149, 167
\XINT_fsqr_a .....	211	\XINT_gcd_loop .....	149, 167, 168
\XINT_fsqr_b .....	211	\XINT_gcdof:_a .....	352
\XINT_fsqr_zero .....	211	\XINT_gcdof:_b .....	352
\XINT_fsub .....	210	\XINT_gcdof:_c .....	352
\XINT_fsub_a .....	210	\XINT_gcdof:_d .....	352, 353
\XINT_fsub_Azero .....	210	\XINT_gcdof:_e .....	352, 353
\XINT_fsub_b .....	210	\XINT_gcdof:_end .....	352, 353
\XINT_fsum_finished .....	210	\XINT_gcdof:_f .....	352, 353
\XINT_fsum_loop_a .....	210	\XINT_gcdof_a .....	176
\XINT_fsum_loop_b .....	210	\XINT_gcdof_b .....	176
\XINT_fsum_loop_c .....	210	\XINT_gcdof_c .....	176
\XINT_fsumexpr .....	210	\XINT_gcdof_d .....	176
\XINT_ftc_A .....	275	\XINT_gcdof_e .....	176
\XINT_ftc_B .....	275	\XINT_gcfrac .....	273
\XINT_ftc_C .....	275	\XINT_gcfrac_end .....	273
\XINT_ftc_D .....	275	\XINT_gcfrac_end_b .....	273
\XINT_ftc_integer .....	275	\XINT_gcfrac_endloop .....	273
\XINT_ftc_loop_a .....	275, 276	\XINT_gcfrac_enter .....	273
\XINT_ftc_loop_d .....	275	\XINT_gcfrac_loop .....	273
\XINT_ftc_loop_e .....	275	\XINT_gcfrac_noopt .....	273
\xint_ftc_loop_exit .....	276	\XINT_gcfrac_opt_a .....	272
\XINT_ftc_loop_f .....	276	\XINT_gcfrac_opt_b .....	273
\XINT_ftcc_A .....	278	\XINT_gcfrac_optc .....	273
\XINT_ftcc_B .....	278	\XINT_gcfrac_optl .....	273
\XINT_ftcc_C .....	278	\XINT_gcfrac_optr .....	273
\XINT_ftcc_D .....	278	\XINT_gcfrac_T .....	273
\XINT_ftcc_En .....	278	\XINT_gcfrac_U .....	273
\XINT_ftcc_end .....	279	\XINT_gcntf .....	288
\XINT_ftcc_Ep .....	278	\XINT_gcntf_exit .....	288, 289
\XINT_ftcc_integer .....	278	\XINT_gcntf_loop .....	288, 289
\XINT_ftcc_loop_a .....	278, 279	\XINT_gcntgc .....	290
\XINT_ftcc_loop_b .....	278	\XINT_gcntgc_exit .....	290, 291
\XINT_ftcc_loop_c .....	278, 279	\XINT_gcntgc_exit_b .....	291
\XINT_ftcc_loop_d .....	279	\XINT_gcntgc_loop .....	290, 291
\XINT_ftcc_loop_N .....	279	\XINT_gcntgc_loop_b .....	291
\XINT_ftcc_loop_P .....	279	\XINT_gctcv_end .....	285, 286
\XINT_ftcx_A .....	276	\XINT_gctcv_loop_a .....	285, 286
\XINT_ftcx_B .....	276	\XINT_gctcv_loop_b .....	285
\XINT_ftcx_C .....	276	\XINT_gctcv_loop_c .....	285
\XINT_ftcx_D .....	276	\XINT_gctcv_loop_d .....	285
\XINT_ftcx_integer .....	276	\XINT_gctcv_loop_e .....	285
\XINT_ftcx_loop_a .....	276	\XINT_gctcv_loop_f .....	285
\XINT_ftcx_loop_d .....	276	\XINT_gctcv_loop_g .....	285
\XINT_ftcx_loop_e .....	276	\XINT_gctcv_loop_h .....	285
\xint_ftcx_loop_exit .....	276	\XINT_gctcv_loop_i .....	285
\XINT_ftcx_loop_f .....	276	\XINT_gctcv_loop_j .....	285, 286
\XINT_fwover_A .....	193, 194	\XINT_gctcv_loop_k .....	286
\XINT_fwover_B .....	193	\XINT_gctcv_loop_l .....	286
\XINT_fwover_C .....	193	\XINT_gctcv_loop_m .....	286
\XINT_fwover_D .....	193	\XINT_gctcv_prep .....	284
\XINT_gcd_AisZero .....	149, 167	\XINT_gctf_end .....	281, 282

\XINT_gctf_loop_a .....	281, 282	\xint_gob_til_dot .....	10, 90, 121, 204
\XINT_gctf_loop_b .....	281	\xint_gob_til_eightzeroes .....	10, 66, 67, 86, 95
\XINT_gctf_loop_c .....	281	\xint_gob_til_exclam .....	10, 96, 248, 273--275, 279--286, 291
\XINT_gctf_loop_d .....	281	\xint_gob_til_G .....	10
\XINT_gctf_loop_e .....	281	\xint_gob_til_minus .....	10, 28, 29, 31, 49--51, 324
\XINT_gctf_loop_f .....	281	\xint_gob_til_one .....	10, 68, 95, 97
\XINT_gctf_loop_g .....	281	\xint_gob_til_R .....	9, 21, 24, 66, 68, 70, 71, 88, 89, 116, 156, 203
\XINT_gctf_loop_h .....	281	\xint_gob_til_sc .....	10, 71, 74, 75, 77, 80--82, 85, 86, 97, 98, 100, 108, 249
\XINT_gctf_loop_i .....	282	\xint_gob_til_W .....	9, 301
\XINT_gctf_loop_j .....	282	\xint_gob_til_xint: .....	10, 12, 13, 21, 23, 27, 31, 47, 48, 51, 52, 54, 60, 107, 128, 129, 131, 132, 176, 210, 213, 219, 220
\XINT_gctf_prep .....	281	\xint_gob_til_Z .....	10, 24, 67, 116, 272
\XINT_gctgc_end .....	291, 292	\xint_gob_til_zero .....	10, 15, 64, 86, 92, 95, 100, 101, 120, 149, 168, 173, 175, 183, 186, 192, 193, 196, 198, 208--211, 217, 218, 234, 272, 275, 276
\XINT_gctgc_end_b .....	292	\xint_gob_til_zeros_iii .....	10
\XINT_gctgc_loop_a .....	291	\xint_gob_til_zeros_iv .....	10
\XINT_gctgc_loop_b .....	291, 292	\XINT_gobble .....	15, 26, 28
\XINT_gctgc_loop_c .....	291	\XINT_gobble_a .....	15
\XINT_gctgc_start .....	291	\XINT_gobble_b .....	15
\XINT_gctgcx_end .....	275	\XINT_gobble_c .....	15
\XINT_gctgcx_loop_a .....	275	\XINT_gobble_d .....	15
\XINT_gctgcx_loop_b .....	275	\xint_gobble_i .....	9, 11, 21, 35, 37, 40, 57, 58, 150, 205, 208, 299, 301, 311, 312, 316, 323, 331, 333, 341, 349, 350, 366--368, 377, 378
\XINT_gctgcx_start .....	275	\xint_gobble_ii .....	9, 21, 27, 192, 262, 263, 288, 289
\XINT_geq .....	122	\xint_gobble_iii .....	9, 21, 35, 36, 176, 177, 277, 297, 298, 302, 322, 323, 327, 328
\XINT_geq_finish .....	122	\xint_gobble_iv .....	9, 21, 34, 112, 300, 349, 350, 366--368
\XINT_geq_firstiszero .....	122	\xint_gobble_v .....	9, 21
\XINT_geq_fork .....	122	\xint_gobble_vi .....	9, 21, 207
\XINT_geq_minusminus .....	122	\xint_gobble_vii .....	9, 21
\XINT_geq_minusplus .....	122	\xint_gobble_viii .....	9, 21
\XINT_geq_no .....	122	\XINT_half .....	63, 89, 243
\XINT_geq_plusminus .....	122	\XINT_half_a .....	63
\XINT_geq_plusplus .....	122, 130, 131, 218, 219	\XINT_half_b .....	63
\XINT_geq_secondiszero .....	122	\XINT_half_e .....	63
\XINT_geq_yes .....	122	\XINT_half_fork .....	63
\XINT_ggcfrac .....	274	\XINT_half_neg .....	63
\XINT_ggcfrac_end .....	274	\XINT_htb_checkin .....	164
\XINT_ggcfrac_end_b .....	274, 275	\XINT_htb_cuz .....	164
\XINT_ggcfrac_endloop .....	274	\XINT_htb_loop .....	164, 165
\XINT_ggcfrac_enter .....	274	\XINT_htb_main .....	164
\XINT_ggcfrac_loop .....	274	\XINT_htb_N .....	164
\XINT_ggcfrac_noopt .....	274	\XINT_htd_A .....	161, 162
\XINT_ggcfrac_opt_a .....	274	\XINT_htd_a .....	161, 162
\XINT_ggcfrac_opt_b .....	274	\XINT_htd_Aa .....	162
\XINT_ggcfrac_optc .....	274	\XINT_htd_Ab .....	162
\XINT_ggcfrac_optl .....	274	\XINT_htd_again .....	161, 162
\XINT_ggcfrac_optr .....	274	\XINT_htd_checkin .....	160
\XINT_ggcfrac_T .....	274		
\XINT_ggcfrac_U .....	274		
\XINT_global .....	18, 333--337, 369--371, 378		
\xint_gob_andstop_ .....	9		
\xint_gob_andstop_i .....	9		
\xint_gob_andstop_ii .....	9		
\xint_gob_andstop_iii .....	9		
\xint_gob_andstop_iv .....	9		
\xint_gob_andstop_v .....	9		
\xint_gob_andstop_vi .....	9		
\xint_gob_andstop_vii .....	9		
\xint_gob_andstop_viii .....	9		
\xint_gob_til_! .....	296, 298, 302, 330		

\XINT_htd_finish .....	161, 162	\XINT_iidivision .....	87
\XINT_htd_finish_cuz .....	162	\XINT_iidivision_a .....	87
\XINT_htd_main .....	160	\XINT_iidivision_aiszero .....	87
\XINT_htd_neg .....	160	\XINT_iidivision_aneg .....	87, 88
\XINT_htd_nextfour .....	161, 162	\XINT_iidivision_aneg_b .....	88
\XINT_htd_starta .....	161, 163	\XINT_iidivision_aneg_end .....	88
\XINT_htd_startb .....	161, 163	\XINT_iidivision_aneg_rpos .....	88
\XINT_htd_startba .....	161	\XINT_iidivision_aneg_rzero .....	88
\XINT_htd_startbb .....	161	\XINT_iidivision_apos .....	87, 88
\XINT_htd_unsep_loop .....	162	\XINT_iidivision_bneg .....	87
\XINT_htd_unsep_loop_a .....	162	\XINT_iidivision_bpos .....	87
\XINT_htd_update .....	161	\XINT_iidivision_divbyzero .....	87
\XINT_iadd .....	72	\XINT_iidivmod .....	104
\XINT_icmp .....	75	\XINT_iidivmod_a .....	104
\XINT_icstcv_end .....	284	\XINT_iidivmod_aiszero .....	104
\XINT_icstcv_loop_a .....	284	\XINT_iidivmod_bneg .....	104, 105
\XINT_icstcv_loop_b .....	284	\XINT_iidivmod_bneg_finish .....	105
\XINT_icstcv_loop_c .....	284	\XINT_iidivmod_bpos .....	104, 105
\XINT_icstcv_loop_d .....	284	\XINT_iidivmod_divbyzero .....	104
\XINT_icstcv_loop_e .....	284	\XINT_iidivround .....	102
\XINT_icstcv_prep .....	284	\XINT_iidivround_a .....	102
\XINT_icstf_end .....	280	\XINT_iidivround_aiszero .....	102--104
\XINT_icstf_loop_a .....	280	\XINT_iidivround_bneg .....	102
\XINT_icstf_loop_b .....	280	\XINT_iidivround_bpos .....	102
\XINT_icstf_loop_c .....	280	\XINT_iidivround_divbyzero .....	102--104
\XINT_icstf_prep .....	280	\XINT_iidivround_pos .....	102, 103
\XINT_idivround .....	102	\XINT_iidivtrunc .....	103
\XINT_iexpr_noopt .....	299	\XINT_iidivtrunc_a .....	103
\XINT_iexpr_withopt .....	299	\XINT_iidivtrunc_aiszero .....	103
\XINT_iexpr_wrap .....	299	\XINT_iidivtrunc_bneg .....	103
\XINT_ifflagRaised .....	58, 112	\XINT_iidivtrunc_bpos .....	103
\XINT_iffloatint .....	258, 259	\XINT_iidivtrunc_divbyzero .....	103
\XINT_ifint .....	196, 197	\XINT_iidivtrunc_pos .....	103
\XINT_ifloor .....	191	\XINT_ii_e_a .....	116
\XINT_igctcv_end_a .....	286, 287	\XINT_ii_e_fork .....	116
\XINT_igctcv_end_b .....	287	\XINT_ii_e_neg .....	116
\XINT_igctcv_loop_a .....	286, 287	\XINT_iiexpr_func! .....	357
\XINT_igctcv_loop_b .....	286	\XINT_iiexpr_func? .....	357
\XINT_igctcv_loop_c .....	286	\XINT_iiexpr_func_@@ .....	338
\XINT_igctcv_loop_f .....	286	\XINT_iiexpr_func_@@@ .....	338
\XINT_igctcv_loop_g .....	286	\XINT_iiexpr_func_@@@ .....	338
\XINT_igctcv_loop_h .....	286, 287	\XINT_iiexpr_func_abs .....	356
\XINT_igctcv_loop_i .....	287	\XINT_iiexpr_func_all .....	364
\XINT_igctcv_loop_k .....	287	\XINT_iiexpr_func_any .....	364
\XINT_igctcv_loop_l .....	287	\XINT_iiexpr_func_binomial .....	361
\XINT_igctcv_prep .....	286	\XINT_iiexpr_func_break .....	332
\XINT_igctf_end .....	282, 283	\XINT_iiexpr_func_ceil .....	356
\XINT_igctf_loop_a .....	282, 283	\XINT_iiexpr_func_divmod .....	360
\XINT_igctf_loop_b .....	282	\XINT_iiexpr_func_even .....	358
\XINT_igctf_loop_c .....	282	\XINT_iiexpr_func_factorial .....	358
\XINT_igctf_loop_f .....	282	\XINT_iiexpr_func_first .....	365
\XINT_igctf_loop_g .....	282	\XINT_iiexpr_func_float .....	360
\XINT_igctf_loop_h .....	282	\XINT_iiexpr_func_floor .....	356
\XINT_igctf_loop_i .....	282, 283	\XINT_iiexpr_func_gcd .....	363
\XINT_igctf_prep .....	282	\XINT_iiexpr_func_if .....	365
\XINT_iiadd .....	72	\XINT_iiexpr_func_ifint .....	365
\XINT_iicmp .....	75	\XINT_iiexpr_func_ifone .....	366



\XINT_iiexpr_func_ifsgn .....	366	\XINT_iiminof_e .....	131
\XINT_iiexpr_func_isint .....	358	\XINT_iimodtrunc .....	103
\XINT_iiexpr_func_ison .....	358	\XINT_iimodtrunc_a .....	103, 104
\XINT_iiexpr_func_iter .....	345	\XINT_iimodtrunc_aiszero .....	104
\XINT_iiexpr_funciterr .....	348	\XINT_iimodtrunc_bneg .....	104
\XINT_iiexpr_func_last .....	365	\XINT_iimodtrunc_bpos .....	104
\XINT_iiexpr_func_lcm .....	363	\XINT_iimodtrunc_divbyzero .....	104
\XINT_iiexpr_func_len .....	365	\XINT_iimodtrunc_pos .....	104
\XINT_iiexpr_func_max .....	363	\XINT_iimul .....	83
\XINT_iiexpr_func_min .....	363	\XINT_iiseq::csv .....	326
\XINT_iiexpr_func_mod .....	360	\XINT_iiseq::csv_n .....	326
\XINT_iiexpr_func_not .....	357	\XINT_iiseq::csv_p .....	326
\XINT_iiexpr_func_num .....	355	\XINT_iiseq::csv_z .....	326
\XINT_iiexpr_func_nuple .....	358	\XINT_iiseqa::csv .....	327
\XINT_iiexpr_func_odd .....	357	\XINT_iiseqb::csv .....	327
\XINT_iiexpr_func_opxadd .....	342	\XINT_iiseqb::csv_a .....	328
\XINT_iiexpr_func_opxmul .....	342	\XINT_iiseqb::csv_n .....	328
\XINT_iiexpr_func_pfactorial .....	361	\XINT_iiseqb::csv_p .....	328
\XINT_iiexpr_func_quo .....	362	\XINT_iiseqb::csv_z .....	328
\XINT_iiexpr_func_randrange .....	361	\XINT_iiseqb:f:csv .....	367
\XINT_iiexpr_func_rem .....	362	\XINT_iiseqb:f:csv_a .....	367
\XINT_iiexpr_func_reversed .....	365	\XINT_iiseqb:f:csv_bg .....	367
\XINT_iiexpr_func_round .....	359	\XINT_iiseqb:f:csv_bl .....	367
\XINT_iiexpr_func_rrseq .....	346	\XINT_iiseqb:f:csv_n .....	367
\XINT_iiexpr_func_rseq .....	343	\XINT_iiseqb:f:csv_na .....	367, 368
\XINT_iiexpr_func_seqx .....	340	\XINT_iiseqb:f:csv_p .....	367
\XINT_iiexpr_func_sgn .....	356	\XINT_iiseqb:f:csv_pa .....	367
\XINT_iiexpr_func_sqr .....	357	\XINT_iiseqb:f:csv_pb .....	367
\XINT_iiexpr_func_sqrt .....	359	\XINT_iiseqb:f:csv_pc .....	367
\XINT_iiexpr_func_sqrttr .....	359	\XINT_iisub .....	77
\XINT_iiexpr_func_subx .....	343	\XINT_inc .....	63, 64
\XINT_iiexpr_func_trunc .....	359	\XINT_inc_a .....	63
\XINT_iiexpr_func_xor .....	364	\XINT_inc_e .....	63, 65
\XINT_iiexpr_op_! .....	330	\XINT_inc_fork .....	63
\XINT_iiexpr_op_!? .....	337	\XINT_inc_neg .....	63
\XINT_iiexpr_op:] .....	321	\XINT_Inexact.handler .....	58
\XINT_iiexpr_op_? .....	330	\XINT_infloat .....	225, 226
\XINT_iiexpr_op_[ .....	331	\XINT_infloat::_b .....	302
\XINT_iiexpr_op_][: .....	321	\XINT_infloat::_c .....	302
\XINT_iiexpr_op_ .....	333	\XINT_infloat::_d .....	302
\XINT_iiexpr_print .....	297, 377	\XINT_infloat::_e .....	302
\XINT_iiexpr_until_end_a .....	298	\XINT_infloat_a .....	226
\XINT_iiexpr_wrap .....	297, 298	\XINT_infloat_clean .....	225
\XINT_iigcd .....	148, 167	\XINT_infloat_clean_a .....	225
\XINT_iigeq .....	122	\XINT_infloat_done .....	225
\XINT_iilcm .....	149, 168	\XINT_infloat_fork .....	226, 227
\xint_iimax .....	129	\XINT_infloat_J .....	227
\XINT_iimaxof_a .....	131	\XINT_infloat_K .....	227, 228
\XINT_iimaxof_b .....	131	\XINT_infloat_L .....	228
\XINT_iimaxof_c .....	131	\XINT_infloat_Ma .....	228
\XINT_iimaxof_d .....	131	\XINT_infloat_Mb .....	228
\XINT_iimaxof_e .....	131	\XINT_infloat_MtoN .....	228
\xint_iimin .....	130	\XINT_infloat_N .....	228
\XINT_iiminof_a .....	131	\XINT_infloat_Oa .....	228, 229
\XINT_iiminof_b .....	131	\XINT_infloat_Ob .....	228, 229
\XINT_iiminof_c .....	131	\XINT_infloat_P .....	229
\XINT_iiminof_d .....	131	\XINT_infloat_Q .....	229

\XINT_infloat_Qq .....	229	\XINT_InvalidOperation.handler .....	58
\XINT_infloat_R .....	229, 230	\XINT_iround_A .....	201, 202
\XINT_infloat_Rq .....	229	\XINT_irr_D .....	195
\XINT_infloat_Sa .....	230	\XINT_irr_denomisone .....	195
\XINT_infloat_Sb .....	230	\XINT_irr_divisionbyzero .....	196
\XINT_infloat_SEq .....	229, 230	\XINT_irr_finish .....	196, 198
\XINT_infloat_sp .....	226	\XINT_irr_indeterminate .....	196
\XINT_infloat_sp_b .....	226, 227	\XINT_irr_loop_a .....	196
\XINT_infloat_sp_c .....	227	\XINT_irr_loop_d .....	196
\XINT_infloat_sp_needzeros .....	227	\XINT_irr_loop_e .....	196
\XINT_infloat_sp_quick .....	227	\XINT_irr_loop_exit .....	196
\XINT_infloat_sp_round .....	227	\XINT_irr_loop_exitb .....	196
\XINT_infloat_spneg .....	226	\XINT_irr_negative .....	195
\XINT_infloat_spneg_needzeros .....	226	\XINT_irr_nonneg .....	195
\XINT_infloat_spnegend .....	226	\XINT_irr_start .....	195
\XINT_infloat_sppos .....	226	\XINT_irr_zero .....	196
\XINT_infloat_spzero .....	226	\XINT_is_One .....	123, 199, 202, 226
\XINT_infloat_SUp .....	229, 230	\XINT_isbalanced_a .....	339, 340
\XINT_infloat_SY .....	230	\XINT_isbalanced_b .....	339
\XINT_infloat_X .....	230	\XINT_isbalanced_c .....	339
\XINT_infloat_Y .....	227, 230, 231	\XINT_isbalanced_d .....	339, 340
\XINT_infloat_Z .....	231	\XINT_isbalanced_error .....	339
\XINT_infloat_ZZ .....	231	\XINT_isbalanced_no .....	340
\XINT_infloatdivmod .....	258	\XINT_isbalanced_yes .....	339, 340
\XINT_infloate .....	257	\XINT_iseries .....	262
\XINT_infloate_end .....	257	\XINT_iseries_exit .....	263
\XINT_infloatfracdg_a .....	233	\XINT_iseries_loop .....	263
\XINT_infloatS_clean .....	226	\XINT_isOne 123, 189, 190, 192, 193, 195, 197, 243	
\XINT_infloatS_clean_a .....	226	\XINT_isone .....	123
\XINT_infrac .....	183	\XINT_istrue::a .....	301
\XINT_infrac .....	181, 183, 189--195, 199, 202, 211, 213, 226	\XINT_istrue::b .....	301
\XINT_infrac_fork .....	183	\XINT_istrue::c .....	301
\XINT_infrac_res_a .....	183	\XINT_istrue::d .....	301
\XINT_infrac_res_b .....	183	\XINT_istrue::e .....	301
\XINT_infrac_res_ca .....	183	\XINT_isub .....	78
\XINT_infrac_res_cb .....	183	\XINT_item .....	37
\XINT_infrac_res_cc .....	183	\xint_itemcount .....	45
\XINT_infrac_res_zero .....	183	\XINT_itrunc_G .....	199, 201, 202
\XINT_inline_b .....	37	\XINT_jrr_D .....	197
\XINT_inline_d .....	37	\XINT_jrr_denomisone .....	197
\XINT_inline_e .....	37	\XINT_jrr_divisionbyzero .....	197
\XINT_inline_f .....	37	\XINT_jrr_indeterminate .....	197
\XINT_inline_g .....	37	\XINT_jrr_loop_a .....	197, 198
\XINT_inline_macro .....	37	\XINT_jrr_loop_b .....	198
\XINT_inline_w .....	37	\XINT_jrr_loop_c .....	198
\XINT_inrandomfloatS .....	259	\XINT_jrr_loop_d .....	198
\XINT_inrandomfloatS_a .....	259, 260	\XINT_jrr_loop_e .....	198
\XINT_inrandomfloatS_b .....	259	\XINT_jrr_loop_exit .....	198
\XINT_inrandomfloatS_zero .....	259	\XINT_jrr_negative .....	197
\XINT_inv .....	222, 353, 354	\XINT_jrr_nonneg .....	197
\XINT_inv_a .....	222	\XINT_jrr_start .....	197
\XINT_inv_b .....	223	\XINT_jrr_zero .....	197
\XINT_inv_expiszero .....	223	\XINT_keep:f:csv_a .....	48
\XINT_inv_iszero .....	222	\XINT_keep:f:csv_keepall .....	49
\XINT_InvalidContext-signal .....	57	\XINT_keep:f:csv_keepnone .....	48, 49
\XINT_InvalidContext.handler .....	58	\XINT_keep:f:csv_loop .....	49--51
		\XINT_keep:f:csv_loop_end .....	50



\XINT_keep:f:csv_loop_pickeight	50	\XINT_ldg	62
\XINT_keep:f:csv_neg	49	\XINT_ldg_a	62
\XINT_keep:f:csv_neg_a	49	\XINT_ldg_b	62
\XINT_keep:f:csv_neg_done	49	\XINT_ldg_c	62
\XINT_keep:f:csv_pos	49	\XINT_ldg_cbye	62
\XINT_keep:f:csv_pos_fork	49	\XINT_ldg_d	62
\XINT_keep:f:csv_pos_keepall	49, 50	\XINT_ldg_fork	62
\XINT_keep:f:csv_trimloop	49, 52	\XINT_len_fork	115, 181
\XINT_keep:f:csv_trimloop_finish	49	\XINT_length:f:csv_a	47, 49, 51, 52
\XINT_keep:f:csv_trimloop_trimmine	49	\XINT_length:f:csv_finish	47
\XINT_keep:x:csv_finish	324	\XINT_length:f:csv_loop	47
\XINT_keep:x:csv_loop	324	\XINT_length_finish_a	12
\XINT_keep:x:csv_loop_pickeight	324	\XINT_length_loop	
\XINT_keep:x:csv_pos	324	. 12, 26, 27, 30, 31, 115, 118, 184, 185, 200	
\XINT_keep_a	27	\XINT_lengthupto:f:csv_a	47--49
\XINT_keep_keepall	28	\XINT_lengthupto:f:csv_empty	48
\XINT_keep_keepnone	27, 29	\XINT_lengthupto:f:csv_finish_a	48
\XINT_keep_loop	28, 30	\XINT_lengthupto:f:csv_finish_b	48
\XINT_keep_loop_end	28	\XINT_lengthupto:f:csv_gt	48
\XINT_keep_loop_pickeight	28	\XINT_lengthupto:f:csv_loop_a	48
\XINT_keep_neg	27, 29	\XINT_lengthupto:f:csv_loop_b	48
\XINT_keep_neg_a	27	\XINT_lengthupto_finish_a	13
\XINT_keep_pos	27, 28	\XINT_lengthupto_finish_b	13
\XINT_keepunbr_a	29	\XINT_lengthupto_gt	13
\XINT_keepunbr_loop	29, 32	\XINT_lengthupto_loop	13, 28, 29
\XINT_keepunbr_loop_end	29	\XINT_lengthupto_loop_a	13
\XINT_keepunbr_loop_pickeight	29	\XINT_listsel:_:	323
\XINT_keepunbr_pos	29	\XINT_listsel:_:a	323
\XINT_last:f:csv_loop	54	\XINT_listsel:_:N:N	323
\XINT_last_loop	12, 13	\XINT_listsel:_:N:N_a	323
\XINT_last_loop_enda	12, 13, 54	\XINT_listsel:_:N:N_abort	323
\XINT_last_loop_endb	12, 13, 54	\XINT_listsel:_:N:P	323
\XINT_last_loop_endc	12, 13, 54	\XINT_listsel:_:N:P_a	323
\XINT_last_loop_endd	12, 13, 54	\XINT_listsel:_:nth	322, 323
\XINT_last_loop_ende	12, 13, 54	\XINT_listsel:_:O:P	323
\XINT_last_loop_endf	12, 13, 54	\XINT_listsel:_:P:N	323
\XINT_last_loop_endg	12, 13, 54	\XINT_listsel:_:P:N_a	323
\XINT_last_loop_endh	12, 13, 54	\XINT_listsel:_:P:O	323
\XINT_lcm_end	149, 168	\XINT_listsel:_:P:P	323
\XINT_lcm_fork	149, 168	\XINT_listsel:_:s	321--323
\XINT_lcm_iszero	149, 168	\XINT_listxsel:_:	321, 322
\XINT_lcm_notzero	149, 168	\XINT_listxsel:_:N:N	322
\XINT_lcmof:_a	353	\XINT_listxsel:_:N:N_a	322
\XINT_lcmof:_b	353	\XINT_listxsel:_:N:P	322
\XINT_lcmof:_c	353	\XINT_listxsel:_:N:P_a	322
\XINT_lcmof:_d	353	\XINT_listxsel:_:O:P	322
\XINT_lcmof:_e	353	\XINT_listxsel:_:P:N	322
\XINT_lcmof:_end	353, 354	\XINT_listxsel:_:P:N_a	322
\XINT_lcmof:_f	353	\XINT_listxsel:_:P:O	322
\XINT_lcmof:_g	353	\XINT_listxsel:_:P:P	322
\XINT_lcmof:_h	353	\XINT_lws	25
\XINT_lcmof:_zero	353	\XINT_lws_e	25, 26
\XINT_lcmof_a	176	\XINT_lws_e_i	25, 26
\XINT_lcmof_b	176	\XINT_lws_e_ii	25, 26
\XINT_lcmof_c	176	\XINT_lws_e_iii	25, 26
\XINT_lcmof_d	176	\XINT_lws_e_iv	25
\XINT_lcmof_e	176	\XINT_lws_e_v	25

<a href="#">\XINT_lws_e_vi</a>	25	<a href="#">\XINT_modtrunc_pos_a</a>	215
<a href="#">\XINT_lws_loop_a</a>	25	<a href="#">\XINT_mul_a</a>	85
<a href="#">\XINT_lws_loop_b</a>	25, 26	<a href="#">\XINT_mul_b</a>	85
<a href="#">\XINT_max_fork</a>	129	<a href="#">\XINT_mul_checklengths</a>	84
<a href="#">\XINT_max_minusminus</a>	129, 130	<a href="#">\XINT_mul_e</a>	85
<a href="#">\XINT_max_minusplus</a>	129, 130	<a href="#">\XINT_mul_exchange</a>	84
<a href="#">\XINT_max_plusminus</a>	129, 130	<a href="#">\XINT_mul_fork</a>	83, 132, 198, 280--286
<a href="#">\XINT_max_plusplus</a>	129, 130	<a href="#">\XINT_mul_loop</a>	85, 106, 108
<a href="#">\XINT_max_pluszero</a>	129, 130	<a href="#">\XINT_mul_minusminus</a>	83
<a href="#">\XINT_max_zeroplus</a>	129, 130	<a href="#">\XINT_mul_minusplus</a>	83
<a href="#">\XINT_max_zerozero</a>	129, 130	<a href="#">\XINT_mul_nfork</a>	83
<a href="#">\XINT_maxof_a</a>	219	<a href="#">\XINT_mul_oneisone</a>	84
<a href="#">\XINT_maxof_b</a>	219	<a href="#">\XINT_mul_out</a>	84, 85, 106, 108--111, 148, 246
<a href="#">\XINT_maxof_c</a>	219	<a href="#">\XINT_mul_plusminus</a>	83
<a href="#">\XINT_maxof_d</a>	219	<a href="#">\XINT_mul_plusplus</a>	83, 171, 172
<a href="#">\XINT_maxof_e</a>	219	<a href="#">\XINT_mul_pre_b</a>	83, 84
<a href="#">\XINT_microrevsep</a>	116	<a href="#">\XINT_mul_smallbyfirst</a>	84
<a href="#">\XINT_microrevsep_end</a>	115, 116	<a href="#">\XINT_mul_smallbysecond</a>	84
<a href="#">\XINT_min_fork</a>	130	<a href="#">\XINT_mul_start</a>	84, 85
<a href="#">\XINT_min_minusminus</a>	130, 131	<a href="#">\XINT_mul_verysmall</a>	84
<a href="#">\XINT_min_minusplus</a>	130, 131	<a href="#">\XINT_mul_zero</a>	83
<a href="#">\XINT_min_plusminus</a>	130, 131	<a href="#">\XINT_NewExpr</a>	377, 378
<a href="#">\XINT_min_plusplus</a>	130, 131	<a href="#">\XINT_NewExpr_a</a>	378
<a href="#">\XINT_min_pluszero</a>	130, 131	<a href="#">\XINT_newexpr_clean</a>	377
<a href="#">\XINT_min_zeroplus</a>	130, 131	<a href="#">\XINT_NewFloatFunc</a>	369, 377
<a href="#">\XINT_min_zerozero</a>	130, 131	<a href="#">\XINT_NewFunc</a>	369, 377
<a href="#">\XINT_minimul_a</a>	86, 100, 101, 106	<a href="#">\XINT_newfunc_clean</a>	377, 378
<a href="#">\XINT_minimul_b</a>	86	<a href="#">\XINT_NewIIFunc</a>	369, 377
<a href="#">\XINT_minimul_c</a>	86, 87	<a href="#">\XINT_nthelt:f:csv_a</a>	52
<a href="#">\XINT_minimulwc_a</a>	85, 86	<a href="#">\XINT_nthelt:f:csv_neg</a>	52
<a href="#">\XINT_minimulwc_b</a>	85	<a href="#">\XINT_nthelt:f:csv_neg_done</a>	52
<a href="#">\XINT_minimulwc_c</a>	85, 86	<a href="#">\XINT_nthelt:f:csv_neg_fork</a>	52
<a href="#">\XINT_minof_a</a>	220	<a href="#">\XINT_nthelt:f:csv_pos</a>	52
<a href="#">\XINT_minof_b</a>	220	<a href="#">\XINT_nthelt:f:csv_pos_cleanup</a>	52, 53
<a href="#">\XINT_minof_c</a>	220	<a href="#">\XINT_nthelt:f:csv_pos_done</a>	52, 53
<a href="#">\XINT_minof_d</a>	220	<a href="#">\XINT_nthelt_a</a>	26
<a href="#">\XINT_minof_e</a>	220	<a href="#">\XINT_nthelt_neg</a>	26
<a href="#">\XINT_mod_a</a>	216	<a href="#">\XINT_nthelt_neg_a</a>	26
<a href="#">\XINT_mod_aiszero</a>	216	<a href="#">\XINT_nthelt_neg_b</a>	26
<a href="#">\XINT_mod_b</a>	216	<a href="#">\XINT_nthelt_pos</a>	26, 27
<a href="#">\XINT_mod_bneg</a>	216	<a href="#">\XINT_nthelt_pos_done</a>	27
<a href="#">\XINT_mod_bpos</a>	216, 217	<a href="#">\XINT_nthelt_zero</a>	26
<a href="#">\XINT_mod_bpos_a</a>	217	<a href="#">\XINT_num</a>	59, 121
<a href="#">\XINT_mod_D_a</a>	217	<a href="#">\XINT_num_cleanup</a>	59, 60, 82, 185
<a href="#">\XINT_mod_D_b</a>	215--217	<a href="#">\XINT_num_end</a>	60
<a href="#">\XINT_mod_D_c</a>	217	<a href="#">\XINT_num_loop</a>	59, 60, 82, 185
<a href="#">\XINT_mod_D_exit</a>	217	<a href="#">\XINT_numer</a>	191
<a href="#">\XINT_mod_divbyzero</a>	216	<a href="#">\XINT_numer_A</a>	191
<a href="#">\XINT_mod_E</a>	217	<a href="#">\XINT_numer_B</a>	191
<a href="#">\XINT_mod_F</a>	217	<a href="#">\XINT_oncsv:_a</a>	351, 352, 354
<a href="#">\XINT_modtrunc_a</a>	214	<a href="#">\XINT_oncsv:_b</a>	351
<a href="#">\XINT_modtrunc_aiszero</a>	214, 216	<a href="#">\XINT_oncsv:_c</a>	351
<a href="#">\XINT_modtrunc_b</a>	214	<a href="#">\XINT_oncsv:_d</a>	351
<a href="#">\XINT_modtrunc_bneg</a>	214, 215	<a href="#">\XINT_oncsv:_e</a>	351
<a href="#">\XINT_modtrunc_bpos</a>	214, 215	<a href="#">\XINT_oncsv:_empty</a>	351--353
<a href="#">\XINT_modtrunc_divbyzero</a>	214, 216	<a href="#">\XINT_oncsv:_end</a>	351
<a href="#">\XINT_modtrunc_pos</a>	215	<a href="#">\XINT_Opp</a>	61

\XINT_opp ..	61, 72, 76, 78, 79, 87, 107, 182, 222	\XINT_pfloat_neg .....	232
\XINT_orof:_a .....	351	\XINT_pfloat_no .....	232
\XINT_orof:_c .....	351	\XINT_pfloat_noopt .....	231
\XINT_orof:_e .....	351	\XINT_pfloat_opt .....	231, 302
\XINT_orof:_yes .....	351	\XINT_pfloat_opt_a .....	231
\XINT_orof_a .....	128	\XINT_pfloat_P .....	232
\XINT_orof_b .....	128	\XINT_pfloat_P_ .....	232
\XINT_orof_c .....	128	\XINT_pfloat_P_i .....	232
\XINT_orof_e .....	128	\XINT_pfloat_P_ii .....	232
\XINT_orof_yes .....	128	\XINT_pfloat_P_iii .....	232
\xint_orthat .....	10, 57, 58, 64, 87, 102--104, 110, 123, 133--135, 137, 138, 140, 146, 152, 158, 159, 171, 186, 190, 199--202, 214--216, 225, 226, 230, 232, 234, 238, 243, 245, 250, 251, 254--256, 258, 259, 301--311, 316, 322, 323, 331, 340--342, 344--350, 373--375	\XINT_pfloat_P_iv .....	232
\XINT_outfrac .....	181, 211, 212, 214	\XINT_pfloat_P_v .....	232
\XINT_outfrac_divisionbyzero .....	181, 182	\XINT_pfloat_pos .....	232
\XINT_outfrac_N .....	181, 182	\XINT_pfloat_Ps .....	232
\XINT_outfrac_N_a .....	182	\XINT_pfloat_Psi .....	233
\XINT_outfrac_P .....	181, 182	\XINT_pfloat_Psii .....	233
\XINT_outfrac_Zero .....	182	\XINT_pfloat_Psiii .....	233
\XINT_Overflow.handler .....	58	\XINT_pfloat_Psiv .....	233
\XINT_pfac_a .....	146	\XINT_pfloat_Psv .....	233
\XINT_pfac_b .....	146	\XINT_pfloat_zero .....	232
\XINT_pfac_bigloop .....	146, 147	\XINT_pirr_start .....	195
\XINT_pfac_bigloop_a .....	147	\xint_pow .....	106, 211
\XINT_pfac_bigloop_b .....	147	\XINT_pow_AatleastTwo .....	107
\XINT_pfac_end_ .....	146--148	\XINT_pow_AisOne .....	107
\XINT_pfac_end_i .....	146--148	\XINT_pow_AisZero .....	107
\XINT_pfac_end_ii .....	146--148	\XINT_pow_Aneg .....	107
\XINT_pfac_end_iii .....	146, 148	\XINT_pow_Apos .....	107
\XINT_pfac_fork .....	145, 146, 213	\XINT_pow_Apos_a .....	107
\XINT_pfac_medloop .....	146, 147	\XINT_pow_Apos_short .....	107
\XINT_pfac_medloop_a .....	147	\XINT_pow_BisNegative .....	107
\XINT_pfac_medloop_b .....	147	\XINT_pow_BisZero .....	107, 108
\XINT_pfac_neg .....	146	\XINT_pow_I_exit .....	108
\XINT_pfac_one .....	146	\XINT_pow_I_in .....	107, 108
\XINT_pfac_outofrange .....	146	\XINT_pow_I_loop .....	108
\XINT_pfac_smallloop .....	146, 147	\XINT_pow_I_squareit .....	108
\XINT_pfac_smallloop_a .....	146	\XINT_pow_II_even .....	109
\XINT_pfac_smallloop_b .....	146, 147	\XINT_pow_II_exit .....	109
\XINT_pfac_vbigloop .....	146--148	\XINT_pow_II_in .....	108
\XINT_pfac_vbigloop_a .....	148	\XINT_pow_II_loop .....	108, 109
\XINT_pfac_zero .....	146	\XINT_pow_II_odda .....	109
\XINT_pfloat::_b .....	302	\XINT_pow_II_oddb .....	109
\XINT_pfloat::_c .....	302	\XINT_pow_mul_small .....	108
\XINT_pfloat::_d .....	302	\XINT_pow_mulbutcheckifsmall .....	108, 109
\XINT_pfloat::_e .....	302	\XINT_powseries .....	263
\XINT_pfloat_a .....	231, 232	\XINT_powseries_exit_i .....	263
\XINT_pfloat_chkopt .....	231	\XINT_powseries_exit_ii .....	264
\XINT_pfloat_N .....	232	\XINT_powseries_loop_i .....	263, 264
\XINT_pfloat_N_i .....	232	\XINT_powseries_loop_ii .....	263
\XINT_pfloat_N_ii .....	232	\XINT_powseriesx .....	264
\XINT_pfloat_N_iii .....	232	\XINT_powseriesx_pre .....	264
\XINT_pfloat_N_iv .....	232	\XINT_praw .....	189
\XINT_pfloat_N_v .....	232	\XINT_praw_A .....	189
		\XINT_praw_a .....	189
		\XINT_prdexpr .....	132
		\XINT_prod_finished .....	132
		\XINT_prod_loop_a .....	132
		\XINT_prod_loop_b .....	132

\XINT_prod_loop_c .....	132	\XINT_revwbr_finish_c .....	21
\XINT_protectii .....	297, 299, 300	\XINT_revwbr_loop .....	21
\XINT_providespackage .....	8,	\XINT_rez_A .....	194
20, 56, 114, 155, 167, 180, 262, 271, 296		\XINT_rez_AB .....	194, 198
\XINT_RaiseFlag .....	58	\XINT_rez_B .....	194
\XINT_randomdigits .....	150	\XINT_rez_C .....	194
\XINT_randomdigits_a .....	150	\XINT_rez_D .....	194
\XINT_randrange .....	151, 153	\XINT_rez_E .....	194
\XINT_randrange_A .....	152, 153	\XINT_rez_neg .....	194
\XINT_randrange_a .....	152	\XINT_rez_zero .....	194
\XINT_randrange_again .....	153	\XINT_rord_cleanup .....	12
\XINT_randrange_B .....	153	\XINT_rord_main .....	11, 12, 174, 175
\XINT_randrange_b .....	152	\XINT_round .....	201
\XINT_randrange_c .....	152, 153	\XINT_round::b .....	301
\XINT_randrange_d .....	152	\XINT_round::c .....	301
\XINT_randrange_E .....	153	\XINT_round::d .....	301
\XINT_randrange_e .....	152	\XINT_round::e .....	301
\XINT_randrange_err:empty .....	151, 152	\XINT_round_A .....	201, 202
\XINT_randrange_Z .....	152	\XINT_round_a .....	202
\XINT_randrangeAtoB_a .....	151	\XINT_round_aa .....	201, 202
\XINT_ratseries .....	264	\XINT_round_B .....	202
\XINT_ratseries_exit_i .....	265	\XINT_Rounded.handler .....	58
\XINT_ratseries_exit_ii .....	265	\XINT_rsepybviii .....	69, 70
\XINT_ratseries_exit_iii .....	265	\XINT_rsepybviii_b .....	69
\XINT_ratseries_loop .....	265, 266	\XINT_rsepybviii_end_A .....	
\XINT_ratseriesx .....	265	.. 70, 72, 73, 78, 83, 84, 90, 105, 108, 204	
\XINT_ratseriesx_pre .....	265, 266	\XINT_rsepybviii_end_B .....	
\XINT_raw .....	189	.. 70, 72, 73, 78, 83, 84, 90, 105, 108, 204	
\XINT_rawz_A .....	190	\XINT_sdiv_out .....	90, 99
\XINT_rawz_Ba .....	190	\xint_secondofthree .....	114, 181, 310, 311
\XINT_rawz_Bb .....	190	\xint_secondoftwo .....	9, 17, 38, 39, 41--
\XINT_rawz_fork .....	190	43, 58, 76, 77, 128, 149, 158, 168, 189, 234,	
\XINT_rdg .....	150	249, 306, 333, 354, 355, 362, 373, 374, 377	
\XINT_rdg_aux .....	150	\XINT_sepandrev .....	70, 108
\XINT_rep .. 14, 116, 120, 205, 207, 208, 238, 242		\XINT_sepandrev_a .....	70
\XINT_rep_ .....	14	\XINT_sepandrev_andcount .....	
\XINT_rep_a .....	14	..... 70, 72, 73, 78, 83, 84, 90, 105, 204	
\XINT_rep_neg .....	14	\XINT_sepandrev_andcount_a .....	70
\XINT_replicate .....	14, 150, 151	\XINT_sepandrev_andcount_b .....	70, 71
\XINT_resetFlag .....	58	\XINT_sepandrev_andcount_done .....	71
\XINT_resetFlags .....	58	\XINT_sepandrev_andcount_end .....	70, 71
\XINT_restorecatcodes .....	6, 7	\XINT_sepandrev_b .....	70
\XINT_restorecatcodes_endinput .....	6, 7,	\XINT_sepandrev_done .....	70
18, 54, 112, 153, 165, 178, 260, 269, 292, 380		\XINT_sepandrev_end .....	70
\XINT_restoreescapechar .....	44, 45	\XINT_sepbyviii .....	69
\XINT_rev_nounsep .....	71, 93--96, 98, 143	\XINT_sepbyviii_andcount .....	69, 76, 90
\XINT_rev_nounsep_done .....	71	\XINT_sepbyviii_andcount_a .....	69
\XINT_rev_nounsep_end .....	71	\XINT_sepbyviii_andcount_b .....	69
\XINT_revdigits .....	115	\XINT_sepbyviii_andcount_end .....	69
\XINT_revdigits_a .....	115	\XINT_sepbyviii_end .....	69, 76, 90
\XINT_revdigits_b .....	116	\XINT_sepbyviii_Z .....	68, 89
\XINT_revdigits_end .....	116	\XINT_sepbyviii_Z_end .....	68, 89, 204
\XINT_reverse:f:csv_cleanup .....	53	\XINT_seq .....	33
\XINT_reverse:f:csv_finish .....	53	\XINT_seq::csv .....	325
\XINT_reverse:f:csv_loop .....	53	\XINT_seq::csv_e .....	325, 326
\XINT_revwbr_finish_a .....	21	\XINT_seq::csv_n .....	325, 326
\XINT_revwbr_finish_b .....	21	\XINT_seq::csv_p .....	325

\XINT_seq::csv_z .....	325	\XINT_smallldiv_d .....	100, 101
\XINT_seq_chkopt .....	33	\XINT_smallldiv_e .....	100
\XINT_seq_e .....	33	\XINT_smallldiv_end .....	100
\XINT_seq_n .....	33	\XINT_smallldiv_f .....	100
\XINT_seq_noopt .....	33	\XINT_smallldiv_fz .....	100
\XINT_seq_opt .....	33, 34	\XINT_smallldiv_g .....	100
\XINT_seq_p .....	33	\XINT_smallldiv_h .....	100
\XINT_seqa::csv .....	327	\XINT_smallldiv_i .....	100
\XINT_seqa::csv_a .....	327	\XINT_smallldiv_j .....	100, 101
\XINT_seqb::csv .....	327	\XINT_smallldiv_k .....	101
\XINT_seqb::csv_a .....	327	\XINT_smallldivx_a .....	90, 99, 143
\XINT_seqb::csv_n .....	327	\XINT_smallldivx_b .....	99
\XINT_seqb::csv_p .....	327	\XINT_smallmul 84--86, 108, 111, 112, 143, 147, 148	
\XINT_seqb::csv_z .....	327	\XINT_smallmul_a .....	86
\XINT_seqb:f:csv .....	366	\XINT_smallmul_e .....	86
\XINT_seqb:f:csv_a .....	366	\XINT_smallmul_f .....	86
\XINT_seqb:f:csv_be .....	366--368	\XINT_smallunrevbyviii .....	71, 73
\XINT_seqb:f:csv_bg .....	366, 367	\XINT_split_finish .....	117
\XINT_seqb:f:csv_bl .....	366	\XINT_split_fromleft .....	117, 118, 138, 205, 208, 227--229, 238, 256
\XINT_seqb:f:csv_n .....	367	\XINT_split_fromleft_a .....	117
\XINT_seqb:f:csv_na .....	367	\XINT_split_fromleft_b .....	117
\XINT_seqb:f:csv_nb .....	367, 368	\XINT_split_fromleft_clean .....	117
\XINT_seqb:f:csv_nc .....	367, 368	\XINT_split_fromleft_end_a .....	117
\XINT_seqb:f:csv_p .....	367	\XINT_split_fromleft_end_b .....	117, 118
\XINT_seqb:f:csv_pa .....	366, 367	\XINT_split_fromright .....	117, 118
\XINT_seqb:f:csv_pb .....	367	\XINT_split_fromright_a .....	118, 200
\XINT_seqb:f:csv_pc .....	367	\XINT_split_fromright_Lempty .....	118
\XINT_seqo .....	34	\XINT_split_xfork .....	117, 118, 121, 206, 208
\XINT_seqo_a .....	34	\XINT_split_zerosplit .....	117
\XINT_seqo_na .....	34	\XINT_splitl_finish .....	118
\XINT_seqo_nb .....	35	\XINT_splitr_finish .....	118
\XINT_seqo_nc .....	35	\XINT_spraw .....	301
\XINT_seqo_nd .....	35	\XINT_spraw::a .....	301
\XINT_seqo_o .....	34, 35	\XINT_spraw::b .....	301
\XINT_seqo_pa .....	34	\XINT_spraw::c .....	301
\XINT_seqo_pb .....	34	\XINT_spraw::d .....	301
\XINT_seqo_pc .....	34	\XINT_spraw::e .....	301
\XINT_seqo_pd .....	34	\XINT_spraw_a .....	301
\XINT_series .....	262	\XINT_spraw_p .....	301
\XINT_series_exit .....	262	\XINT_sptoken .....	20, 36, 38
\XINT_series_loop .....	262	\XINT_sqr .....	105
\XINT_setcatcodes .....	7	\XINT_sqr_a .....	105, 106
\XINT_Sgn .....	60, 88, 139, 182, 199, 202	\XINT_sqr_small .....	106
\XINT_sgn .....	60, 223	\XINT_sqr_small_out .....	106
\XINT_sgnfrac_a .....	192	\XINT_sqr_start .....	106
\XINT_sgnfrac_b .....	192	\XINT_sqr_verysmall .....	106
\XINT_sgnfrac_N .....	192, 193	\XINT_sqrt .....	133
\XINT_sgnfrac_P .....	192, 193	\XINT_sqrt_a .....	133
\XINT_sgnfwover_a .....	193	\XINT_sqrt_bE .....	133
\XINT_sgnfwover_b .....	193	\XINT_sqrt_big_a .....	133
\XINT_sgnfwover_N .....	193, 194	\XINT_sqrt_big_d .....	133, 135
\XINT_sgnfwover_P .....	193, 194	\XINT_sqrt_big_eE .....	135
\XINT_signalcondition .....	57, 87, 102, 104, 107, 110, 111, 133, 140, 146, 182, 185, 196, 197, 214, 215, 222, 238, 245, 250, 254, 255	\XINT_sqrt_big_eE_a .....	135
\XINT_signalcondition_a .....	57	\XINT_sqrt_big_end .....	139
\XINT_smallldiv_c .....	100	\XINT_sqrt_big_e0 .....	135
		\XINT_sqrt_big_e0_a .....	135

\XINT_sqrt_big_f .....	135, 136	\xint_stop_atfirstoftwo ...	9, 33, 102, 103, 105, 125--127, 129--131, 188, 189, 192, 196, 258
\XINT_sqrt_big_fa .....	136	\xint_stop_atsecondofthree ...	114, 124, 125, 187
\XINT_sqrt_big_ga .....	136	\xint_stop_atsecondoftwo .....	9, 102, 104, 105, 119, 125--127, 129--131, 188, 189, 196, 258
\XINT_sqrt_big_gb .....	136	\xint_stop_atthirdofthree	114, 124, 125, 187, 188
\XINT_sqrt_big_gc .....	136	\XINT_sub_a .....	79, 80
\XINT_sqrt_big_gd .....	136, 137	\XINT_sub_aa .....	79
\XINT_sqrt_big_ge .....	137	\XINT_sub_b .....	80
\XINT_sqrt_big_gf .....	137	\XINT_sub_bi .....	80
\XINT_sqrt_big_gg .....	137	\XINT_sub_c .....	80
\XINT_sqrt_big_gi .....	137	\XINT_sub_checklengths .....	78, 79
\XINT_sqrt_big_gj .....	137	\XINT_sub_comp_clean .....	83
\XINT_sqrt_big_gk .....	137	\XINT_sub_comp_finish .....	82
\XINT_sqrt_big_gl .....	137, 138	\XINT_sub_comp_loop .....	82, 83
\XINT_sqrt_big_gloop .....	137, 138	\XINT_sub_d .....	80
\XINT_sqrt_big_gm .....	138	\XINT_sub_di .....	80, 81
\XINT_sqrt_big_gn .....	138	\XINT_sub_e .....	80, 81
\XINT_sqrt_big_ka .....	136--138	\XINT_sub_exchange .....	79
\XINT_sqrt_big_kb .....	138	\XINT_sub_f .....	80
\XINT_sqrt_big_kc .....	138	\XINT_sub_fi .....	80, 81
\XINT_sqrt_big_ke .....	139	\XINT_sub_firstiszero .....	78
\XINT_sqrt_big_kend .....	138	\XINT_sub_fix_cuz .....	82
\XINT_sqrt_big_kf .....	139	\XINT_sub_fix_neg .....	82
\XINT_sqrt_big_kg .....	139	\XINT_sub_fix_none .....	81, 82
\XINT_sqrt_big_kloop .....	138, 139	\XINT_sub_g .....	80, 81
\XINT_sqrt_big_kz .....	138	\XINT_sub_h .....	80
\XINT_sqrt_bigormed_f .....	135	\XINT_sub_hi .....	80, 81
\XINT_sqrt_b0 .....	133	\XINT_sub_i .....	80, 81
\XINT_sqrt_c .....	133	\XINT_sub_k .....	81
\XINT_sqrt_checkin .....	133	\XINT_sub_l .....	81, 82
\XINT_sqrt_isneg .....	133	\XINT_sub_l_carry .....	81
\XINT_sqrt_iszero .....	133	\XINT_sub_l_Ia .....	81
\XINT_sqrt_med_fa .....	136	\XINT_sub_l_Ib .....	81, 82
\XINT_sqrt_med_fb .....	136	\XINT_sub_l_Ic .....	81
\XINT_sqrt_med_fv .....	136	\XINT_sub_l_Ica .....	81
\XINT_sqrt_med_fvi .....	136	\XINT_sub_l_Id .....	81, 82
\XINT_sqrt_med_fvii .....	136	\XINT_sub_l_Id_a .....	82
\XINT_sqrt_med_fviii .....	136	\XINT_sub_l_Id_b .....	82
\XINT_sqrt_post .....	139	\XINT_sub_l_Ida .....	82
\XINT_sqrt_small_a .....	133	\XINT_sub_minusminus .....	78
\XINT_sqrt_small_d .....	133, 134	\XINT_sub_minusplus .....	78
\XINT_sqrt_small_e .....	134, 135	\XINT_sub_mm_a .....	72, 78, 88
\XINT_sqrt_small_ea .....	134, 136	\XINT_sub_mm_b .....	78
\XINT_sqrt_small_eb .....	134	\XINT_sub_nfork .....	77, 78
\XINT_sqrt_small_ec .....	134	\XINT_sub_out .....	79
\XINT_sqrt_small_end .....	134, 135	\XINT_sub_p .....	81, 82
\XINT_sqrt_small_ez .....	134	\XINT_sub_plusminus .....	78
\XINT_sqrt_small_f .....	134, 135	\XINT_sub_plusplus .....	78
\XINT_sqrt_small_g .....	134	\XINT_sub_secondiszero .....	78
\XINT_sqrt_small_h .....	135	\XINT_Subnormal.handler .....	58
\XINT_sqrt_start .....	133, 256	\XINT_sum_finished .....	132
\XINT_sqrtr_post .....	139	\XINT_sum_loop_a .....	132
\xint_stop_afterbye .....	10, 26, 52	\XINT_sum_loop_b .....	132
\xint_stop_aftergobble .....	9, 48, 50	\XINT_sum_loop_c .....	132
\xint_stop_atfirstofone .....	9, 58	\XINT_sumexpr .....	132
\xint_stop_atfirstofthree .....	58, 114, 124, 125, 187, 188	\XINT_T .....	183, 185



<code>\xint_temp</code> .....	44, 45	<code>\XINT_trunc_C</code> .....	199
<code>\xint_texuniformdeviate</code> ....	8, 17, 114, 150, 153	<code>\XINT_trunc_CE</code> .....	199
<code>\XINT_tfrac_fork</code> .....	198	<code>\XINT_trunc_D</code> .....	199, 200
<code>\XINT_tfrac_P</code> .....	198	<code>\XINT_trunc_E</code> .....	199, 200
<code>\XINT_tfrac_zero</code> .....	198	<code>\XINT_trunc_F</code> .....	200, 201
<code>\XINT_thecoords_a</code> .....	298	<code>\XINT_trunc_G</code> .....	199, 201, 202
<code>\XINT_thecoords_b</code> .....	298	<code>\XINT_trunc_H</code> .....	201
<code>\XINT_thecoords_c</code> .....	298	<code>\XINT_trunc_Ha</code> .....	201
<code>\xint_thirdofthree</code> .....	114, 181, 277, 311	<code>\XINT_trunc_Haa</code> .....	201
<code>\XINT_tmpa</code> .....	9, 38, 54, 155, 312, 313, 320, 321, 328--331, 368--370, 378, 380	<code>\XINT_trunc_Hb</code> .....	201
<code>\XINT_tmppb</code> .....	9, 38, 54, 329, 378, 380	<code>\XINT_trunc_sp_B</code> .....	199
<code>\XINT_tmppc</code> .....	38, 54, 380	<code>\XINT_trunc_sp_b</code> .....	199, 202
<code>\XINT_to_forever</code> .....	38, 39	<code>\XINT_trunc_sp_C</code> .....	199, 200
<code>\XINT_to_forxever</code> .....	39	<code>\XINT_trunc_sp_Ca</code> .....	200
<code>\XINT_tofourhex</code> .....	158	<code>\XINT_trunc_sp_Cb</code> .....	200
<code>\XINT_tofourhex_a</code> .....	158	<code>\XINT_trunc_sp_Cc</code> .....	200
<code>\XINT_tofourhex_c</code> .....	158	<code>\XINT_trunc_sp_Cd</code> .....	200
<code>\XINT_tofourhex_d</code> .....	158	<code>\XINT_trunc_sp_D</code> .....	199, 200
<code>\XINT_tofourhex_e</code> .....	158	<code>\XINT_trunc_sp_E</code> .....	200
<code>\XINT_token</code> .....	36--39, 43, 44	<code>\XINT_trunc_sp_F</code> .....	200, 201
<code>\XINT_tokenB</code> .....	36	<code>\XINT_trunc_zero</code> .....	199, 202
<code>\XINT_toks</code> .....	20, 38, 39, 41--43	<code>\XINT_TypesetBezoutAlgorithm</code> .....	177
<code>\XINT_tosixteenbits</code> .....	159	<code>\XINT_TypesetEuclideanAlgorithm</code> .....	176
<code>\XINT_tosixteenbits_a</code> .....	159	<code>\xint_UDonezerofork</code> .....	10
<code>\XINT_tosixteenbits_c</code> .....	159, 160	<code>\xint_UDsignfork</code> .....	10, 13, 14, 27, 30, 31, 48, 49, 51, 52, 61--65, 87, 102--105, 115--118, 156, 159, 160, 162--165, 169, 192, 193, 195, 197, 199, 200, 205, 207, 215, 216, 218, 222, 227, 240, 312, 314, 317, 320, 328, 329
<code>\XINT_tosixteenbits_d</code> .....	160	<code>\xint_UDsignsfork</code> .....	10, 72, 75, 78, 83, 122, 129, 130, 169, 219--221, 322, 323
<code>\XINT_tosixteenbits_e</code> .....	160	<code>\xint_UDwfork</code> .....	10
<code>\XINT_trim:f:csv_a</code> .....	50	<code>\xint_UDXINTWfork</code> .....	10, 183, 184
<code>\XINT_trim:f:csv_finish</code> .....	51	<code>\xint_UDzerofork</code> .....	10, 72, 75, 78, 81--83, 99, 122, 149, 167, 168, 173, 174
<code>\XINT_trim:f:csv_loop</code> .....	51, 52	<code>\xint_UDzerominusfork</code> .....	10, 26, 27, 29--31, 48, 50, 60, 61, 76, 107, 110, 117, 119, 120, 133, 151, 185, 194, 198, 202, 209, 210, 212, 222, 225--227, 232, 237, 238, 242, 244, 255, 257, 278, 279, 327
<code>\XINT_trim:f:csv_loop_trimnine</code> .....	51	<code>\xint_UDzerosfork</code> .....	10, 129, 130, 169, 196, 197, 221
<code>\XINT_trim:f:csv_neg</code> .....	50	<code>\XINT_undefined</code> .....	58
<code>\XINT_trim:f:csv_neg_a</code> .....	51	<code>\xint_undefined</code> .....	8, 335, 370
<code>\XINT_trim:f:csv_pos</code> .....	50, 51	<code>\XINT_Underflow.handler</code> .....	58
<code>\XINT_trim:f:csv_pos_done</code> .....	51, 52	<code>\XINT_uniformdeviate</code> .....	17, 152
<code>\XINT_trim:f:csv_toofew</code> .....	51	<code>\XINT_uniformdeviate_a</code> .....	17
<code>\XINT_trim:f:csv_trimall</code> .....	51	<code>\XINT_uniformdeviate_b</code> .....	17, 18
<code>\XINT_trim:f:csv_trimnone</code> .....	50	<code>\XINT_uniformdeviate_neg</code> .....	17
<code>\XINT_trim_a</code> .....	30	<code>\XINT_uniformdeviate_sgnfork</code> .....	17
<code>\XINT_trim_finish</code> .....	31	<code>\XINT_unrevbyviii</code> .....	71, 73, 79, 85
<code>\XINT_trim_loop</code> .....	27, 30, 31	<code>\XINT_unrevbyviii_a</code> .....	71
<code>\XINT_trim_loop_trimnine</code> .....	31	<code>\XINT_unsep_clean</code> .....	66, 67, 162
<code>\XINT_trim_neg</code> .....	30	<code>\XINT_unsep_cuzsmall</code> .....	67, 91, 99, 144
<code>\XINT_trim_neg_a</code> .....	30	<code>\XINT_unsep_cuzsmall_x</code> .....	67
<code>\XINT_trim_pos</code> .....	30, 31	<code>\XINT_unsep_loop</code> .....	66--68
<code>\XINT_trim_pos_done</code> .....	30, 31	<code>\XINT_useiimessage</code> .....	112
<code>\XINT_trim_toofew</code> .....	31	<code>\XINT_verysmallmul</code> .....	84, 86
<code>\XINT_trim_trimall</code> .....	30, 32		
<code>\XINT_trim_trimnone</code> .....	30, 31		
<code>\XINT_trimunbr_a</code> .....	31		
<code>\XINT_trimunbr_neg</code> .....	31		
<code>\XINT_trimunbr_neg_a</code> .....	31		
<code>\XINT_trunc</code> .....	199		
<code>\XINT_trunc_a</code> .....	199		
<code>\XINT_trunc_B</code> .....	199		
<code>\XINT_trunc_b</code> .....	199, 202		

\XINT_verysmallmul_a	86	\XINT_xtrunc_prepare_a	202, 203
\XINT_verysmallmul_bi	86	\XINT_xtrunc_prepare_b	203, 204
\XINT_verysmallmul_bj	86	\XINT_xtrunc_prepare_c	204
\XINT_verysmallmul_cj	86	\XINT_xtrunc_prepare_d	204
\XINT_verysmallmul_e	86	\XINT_xtrunc_prepare_e	204
\XINT_verysmallmul_f	86	\XINT_xtrunc_prepare_f	204
\XINT_W	10, 183--185	\XINT_xtrunc_prepare_g	204
\XINT_x	39--43	\XINT_xtrunc_prepare_small	203
\XINT_xflet	36, 37, 39	\XINT_xtrunc_small_a	203, 204
\XINT_xflet_eq?	36	\XINT_xtrunc_sp_e	203, 207
\XINT_xflet_macro	36	\XINT_xtrunc_sp_I	207
\XINT_xflet_sp?	36	\XINT_xtrunc_sp_I_a	207
\XINT_xflet_spB?	36	\XINT_xtrunc_sp_IA_b	207
\XINT_xflet_zapsp	36	\XINT_xtrunc_sp_IA_c	207, 208
\XINT_xflet_zapspB	36	\XINT_xtrunc_sp_IAA	207, 208
\XINT_XINT_rdg	150	\XINT_xtrunc_sp_IAB	207, 208
\XINT_xorof:_a	351	\XINT_xtrunc_sp_IB_b	207, 208
\XINT_xorof:_b	351	\XINT_xtrunc_sp_IB_c	208
\XINT_xorof:_c	351	\XINT_xtrunc_sp_II	207, 208
\XINT_xorof:_e	351	\XINT_xtrunc_transition	206, 207
\XINT_xorof_a	128, 129	\XINT_xtrunc_zero	202
\XINT_xorof_b	129	\XINT_y	39, 41--43
\XINT_xorof_c	129	\XINT_Z	184
\XINT_xorof_e	129	\XINT_zapbsp_a	21, 22
\XINT_xrandomdigits	151	\XINT_zapbsp_again	22
\XINT_xrandomdigits_a	151	\XINT_zapbsp_again?	22
\XINT_xtrunc_a	202	\XINT_zapbsp_b	22
\XINT_xtrunc_b	202	\XINT_zapesp_a	22, 23
\XINT_xtrunc_BisEight	203	\XINT_zapesp_b	22
\XINT_xtrunc_BisFive	203	\XINT_zapesp_e	22
\XINT_xtrunc_BisFour	203	\XINT_zapesp_end	22
\XINT_xtrunc_BisOne	203	\XINT_zapesp_end?	22
\XINT_xtrunc_BisSmall	203	\XINT_zapsp_a	23
\XINT_xtrunc_BisTwo	203	\XINT_zapsp_again	23
\XINT_xtrunc_c	202	\XINT_zapsp_again?	23
\XINT_xtrunc_d	202	\XINT_zapsp_b	23
\XINT_xtrunc_e	203, 204	\XINT_zapsp_c	23
\XINT_xtrunc_finish	207	\xint_zapspaces	11, 299, 311, 331
\XINT_xtrunc_finish_a	207	\xint_zapspaces_o	11, 334, 335, 369, 370
\XINT_xtrunc_I	204, 205	\XINT_zapspb_bracedorone	23
\XINT_xtrunc_I_a	205	\XINT_zapspb_one?	23
\XINT_xtrunc_I_b	205	\XINT_zapspb_onlyspaces	23
\XINT_xtrunc_IA_c	205	\XINT_zeroes_foriv	156, 157, 159, 161, 163
\XINT_xtrunc_IA_d	205, 206	\XINT_zeroes_foriv_done	156
\XINT_xtrunc_IA_xd	205	\XINT_zeroes_foriv_end	156
\XINT_xtrunc_IA_xe	205, 206	\XINT_zeroes_forviii	68, 72, 73, 76, 78, 83, 84, 89, 90, 105, 108, 204
\XINT_xtrunc_IAA_e	205	\XINT_zeroes_forviii_end	68
\XINT_xtrunc_IAB_e	205	\xintAbs	222, 273, 355
\XINT_xtrunc_IB_c	205, 206	\xintabs	218, 222
\XINT_xtrunc_IB_d	206	\xintAdd	208, 234, 278, 342
\XINT_xtrunc_II	204, 206	\xintadd	208, 210, 262, 263, 265, 288, 327, 352, 367
\XINT_xtrunc_II_a	206	\xintAND	128
\XINT_xtrunc_II_b	206	\xintand	128
\XINT_xtrunc_II_c	206	\xintANDof	128
\XINT_xtrunc_loop	206	\xintandof	128
\XINT_xtrunc_loop_a	206, 207	\xintANDof:csv	350, 364
\XINT_xtrunc_loop_b	206		



<code>\xintApply</code> .....	23, 24, 32, 334	<code>\xintCstoGC</code> .....	291
<code>\xintapply</code> .....	32, 46, 347, 348	<code>\xintcstogc</code> .....	291
<code>\xintApply::csv</code> .....	373	<code>\xintCSV::csv</code> .....	297, 301, 377
<code>\xintApply::csv</code> .....	372, 373	<code>\xintCSVFirstItem</code> .....	54
<code>\xintApplyInline</code> .....	37, 38, 330	<code>\xintCSVKeep</code> .....	54
<code>\xintApplyNoExpand</code> .....	32	<code>\xintCSVLastItem</code> .....	54
<code>\xintapplynoexpand</code> .....	32	<code>\xintCSVLength</code> .....	54, 334
<code>\xintApplyUnbraced</code> .....	32, 38, 337	<code>\xintCSVNthEltPy</code> .....	54
<code>\xintapplyunbraced</code> .....	32, 349, 350	<code>\xintCSVReverse</code> .....	54
<code>\xintApplyUnbracedNoExpand</code> .....	32	<code>\xintCSVtoList</code> .....	23, 334
<code>\xintapplyunbracednoexpand</code> .....	32	<code>\xintcsvtolist</code> .....	23, 38, 39, 42, 43, 279, 283
<code>\xintAssign</code> .....	43	<code>\xintCSVtoListNoExpand</code> .....	24
<code>\xintAssignArray</code> .....	44, 45, 176, 177, 334	<code>\xintcsvtolistnoexpand</code> .....	24
<code>\xintbareeval</code> .....	297--299, 335, 342, 343, 345, 346, 348, 370	<code>\xintCSVtoListNonStripped</code> .....	24, 347, 348, 369, 371
<code>\xintbarefloateval</code> .....	298, 299, 335, 342, 343, 345, 346, 348, 371, 377	<code>\xintcsvtolistnonstripped</code> .....	23, 24, 369, 371
<code>\xintbareiieval</code> .....	297, 298, 335, 342, 343, 345, 346, 348, 370	<code>\xintCSVtoListNonStrippedNoExpand</code> .....	24
<code>\xintBezout</code> .....	169	<code>\xintcsvtolistnonstrippednoexpand</code> .....	24
<code>\xintbezout</code> .....	169	<code>\xintCSVTrim</code> .....	54
<code>\xintBezoutAlgorithm</code> .....	174, 177	<code>\xintCtoCv</code> .....	283
<code>\xintbezoutalgorithm</code> .....	174	<code>\xintctocv</code> .....	283
<code>\xintBinomial</code> .....	212, 361	<code>\xintCtoF</code> .....	279
<code>\xintbinomial</code> .....	212	<code>\xintctof</code> .....	279
<code>\xintBinToDec</code> .....	162	<code>\xintCTRLC</code> .....	58
<code>\xintbintodec</code> .....	162	<code>\xintDec</code> .....	64
<code>\xintBinToHex</code> .....	163	<code>\xintdec</code> .....	64
<code>\xintbintohehex</code> .....	163	<code>\xintDecSplit</code> .....	117
<code>\xintBool</code> .....	148, 332	<code>\xintdecsplit</code> .....	117, 201
<code>\xintboolexpr</code> .....	300	<code>\xintDecSplitL</code> .....	118
<code>\xintbracediloopindex</code> .....	35	<code>\xintdecsplitl</code> .....	118
<code>\xintbracedouteriloopindex</code> .....	36	<code>\xintDecSplitR</code> .....	118
<code>\xintBreakFor</code> .....	38, 39	<code>\xintdecsplitr</code> .....	118
<code>\xintBreakForAndDo</code> .....	38, 40	<code>\xintDecToBin</code> .....	159
<code>\xintbreakiloop</code> .....	35	<code>\xintdectobin</code> .....	159
<code>\xintbreakiloopanddo</code> .....	35	<code>\xintDecToHex</code> .....	156
<code>\xintbreakloop</code> .....	35	<code>\xintdectohehex</code> .....	156
<code>\xintbreakloopanddo</code> .....	35	<code>\xintDecToString</code> .....	190
<code>\xintCeil</code> .....	191, 356	<code>\xintdectosttring</code> .....	190
<code>\xintceil</code> .....	191	<code>\xintdeffloatfunc</code> .....	369
<code>\xintCFrac</code> .....	271	<code>\xintdeffloatfunc_a</code> .....	369
<code>\xintcfrac</code> .....	271	<code>\xintdeffloatvar</code> .....	335
<code>\xintCHexToBin</code> .....	164	<code>\xintdeffloatvar_a</code> .....	335
<code>\xintchextobin</code> .....	164, 165	<code>\xintdeffunc</code> .....	369
<code>\xintCmp</code> .....	187, 188, 221	<code>\xintdeffunc_a</code> .....	369
<code>\xintcmp</code> .....	221	<code>\xintdefiifunc</code> .....	369
<code>\xintCnToCs</code> .....	289	<code>\xintdefiifunc_a</code> .....	369
<code>\xintcntocs</code> .....	289	<code>\xintdefiivar</code> .....	335
<code>\xintCnToF</code> .....	287	<code>\xintdefiivar_a</code> .....	335
<code>\xintcntof</code> .....	287	<code>\xintdefvar</code> .....	335
<code>\xintCnToGC</code> .....	290	<code>\xintdefvar_a</code> .....	335
<code>\xintcntogc</code> .....	290	<code>\xintDenominator</code> .....	191
<code>\xintCstoCv</code> .....	283	<code>\xintdenominator</code> .....	191
<code>\xintcstocv</code> .....	283	<code>\XINTdigits</code> .....	208, 224, 231, 233--237, 241, 242, 244, 249, 253, 255, 257--259, 268, 269, 327, 328, 354, 368, 377
<code>\xintCstoF</code> .....	279	<code>\xintDigits</code> .....	208
<code>\xintcstof</code> .....	279	<code>\xintDigitsOf</code> .....	45
		<code>\xintdimensions</code> .....	40

<a href="#">\xintDiv</a>	213, 214, 254, 288	<a href="#">\xintFloat</a>	224
<a href="#">\xintdiv</a>	213, 278, 279	<a href="#">\xintfloat</a>	224, 231--236, 240, 244, 249, 253, 255, 268
<a href="#">\xintDivFloor</a>	214	<a href="#">\xintFloatAdd</a>	233
<a href="#">\xintdivfloor</a>	214, 258	<a href="#">\xintfloatadd</a>	233, 269
<a href="#">\xintDivMod</a>	215, 360	<a href="#">\xintFloatBinomial</a>	253
<a href="#">\xintdivmod</a>	215, 258	<a href="#">\xintfloatbinomial</a>	253
<a href="#">\xintDivRound</a>	214	<a href="#">\xintFloatDiv</a>	236
<a href="#">\xintdivround</a>	214	<a href="#">\xintfloatdiv</a>	236
<a href="#">\xintDivTrunc</a>	214	<a href="#">\xintFloatE</a>	257
<a href="#">\xintdivtrunc</a>	214	<a href="#">\xintfloate</a>	257
<a href="#">\xintdothis</a>	10	<a href="#">\xintfloateval</a>	297
<a href="#">\xintDouble</a>	62, 137, 231, 241	<a href="#">\xintfloatexpr</a>	297
<a href="#">\xintdouble</a>	62, 203	<a href="#">\xintfloatexpro</a>	297, 299
<a href="#">\xintDSH</a>	119, 231	<a href="#">\xintFloatFac</a>	244
<a href="#">\xintdsh</a>	119	<a href="#">\xintfloatfac</a>	244
<a href="#">\xintDSHr</a>	119	<a href="#">\xintFloatIsInt</a>	259, 358
<a href="#">\xintdshr</a>	119	<a href="#">\xintfloatisint</a>	259
<a href="#">\xintDSL</a>	64	<a href="#">\xintFloatMul</a>	235
<a href="#">\xintdsl</a>	64	<a href="#">\xintfloatmul</a>	235
<a href="#">\xintDSR</a>	64	<a href="#">\xintFloatPFactorial</a>	249
<a href="#">\xintdsr</a>	64	<a href="#">\xintfloatpfactorial</a>	249
<a href="#">\xintDSRr</a>	65	<a href="#">\xintFloatPow</a>	236
<a href="#">\xintdsrr</a>	65, 241	<a href="#">\xintfloatpow</a>	236
<a href="#">\xintDSx</a>	120	<a href="#">\xintFloatPower</a>	240
<a href="#">\xintdsx</a>	120	<a href="#">\xintfloatpower</a>	240
<a href="#">\xintE</a>	194, 331	<a href="#">\xintFloatPowerSeries</a>	267
<a href="#">\xinte</a>	194	<a href="#">\xintfloatpowerseries</a>	267
<a href="#">\xintegers</a>	40	<a href="#">\xintFloatPowerSeriesX</a>	269
<a href="#">\xintEq</a>	187	<a href="#">\xintfloatpowerseriesx</a>	269
<a href="#">\xinteq</a>	187	<a href="#">\xintFloatSqrt</a>	255
<a href="#">\xintError:ignored</a>	296	<a href="#">\xintfloatsqrt</a>	255
<a href="#">\xintError:inserted</a>	313	<a href="#">\xintFloatSub</a>	234
<a href="#">\xintError:missing_xintthe!</a>	297	<a href="#">\xintfloatsub</a>	234
<a href="#">\xintError:removed</a>	312, 331, 333	<a href="#">\xintFloor</a>	191, 356
<a href="#">\xintError:thiscannothappen</a>	107	<a href="#">\xintfloor</a>	191
<a href="#">\xintError:we_are_doomed</a>	339	<a href="#">\xintFor</a>	38, 312--314, 318--321, 329, 331, 369
<a href="#">\xintEuclideanAlgorithm</a>	172, 176	<a href="#">\xintForfour</a>	42
<a href="#">\xinteuclideanalgorithm</a>	172	<a href="#">\xintForpair</a>	42
<a href="#">\xinteval</a>	297	<a href="#">\xintForthree</a>	42
<a href="#">\xintEven</a>	187, 358	<a href="#">\xintFrac</a>	192, 273
<a href="#">\xinteven</a>	187	<a href="#">\xintfrac</a>	192
<a href="#">\xintExpandArgs</a>	46	<a href="#">\xintFtoC</a>	277
<a href="#">\xintexpr</a>	297	<a href="#">\xintftoc</a>	277
<a href="#">\xintexpro</a>	297, 298, 300	<a href="#">\xintFtoCC</a>	278, 287
<a href="#">\xintexprRestoreCatcodes</a>	334, 369, 379	<a href="#">\xintftocc</a>	278
<a href="#">\xintexprSafeCatcodes</a>	335, 369, 378, 379	<a href="#">\xintFtoCCv</a>	287
<a href="#">\xintexprsafecatcodesfalse</a>	379	<a href="#">\xintftoccv</a>	287
<a href="#">\xintexprsafecatcodestrue</a>	379	<a href="#">\xintFtoCs</a>	275, 287
<a href="#">\xintFac</a>	212, 330, 358	<a href="#">\xintftocs</a>	275
<a href="#">\xintfac</a>	212	<a href="#">\xintFtoCv</a>	287
<a href="#">\xintfdef</a>	11, 20	<a href="#">\xintftocv</a>	287
<a href="#">\xintFDg</a>	61	<a href="#">\xintFtoCx</a>	276
<a href="#">\xintfdg</a>	61	<a href="#">\xintftocx</a>	276, 277
<a href="#">\xintFGtoC</a>	277	<a href="#">\xintFtoGC</a>	277
<a href="#">\xintfgtoc</a>	277	<a href="#">\xintftogc</a>	277
<a href="#">\xintFirstItem:f:csv</a>	53, 54, 365	<a href="#">\xintFwOver</a>	193
<a href="#">\xintfirstitem:f:csv</a>	53		

<code>\xintfwover</code>	193	<code>\xintifFalseAelseB</code>	127
<code>\xintFxFtPowerSeries</code>	266	<code>\xintifFloatInt</code>	258, 366, 375
<code>\xintfxptpowerseries</code>	266	<code>\xintiffloatint</code>	258
<code>\xintFxFtPowerSeriesX</code>	267	<code>\xintifFloatInt:</code>	366, 376
<code>\xintfxptpowerseriesx</code>	267	<code>\xintifFloatIntNE:</code>	375, 376
<code>\xintGCD</code>	167	<code>\xintifForFirst</code>	38, 39, 41--43
<code>\xintgcd</code>	167, 176	<code>\xintifForLast</code>	38, 39, 41--43
<code>\xintGCDof</code>	176	<code>\xintifGt</code>	188
<code>\xintgcdof</code>	176	<code>\xintifgt</code>	187, 188
<code>\xintGCDof:csv</code>	352, 362	<code>\xintifInt</code>	196, 365, 375
<code>\xintGCFrac</code>	272	<code>\xintifint</code>	196
<code>\xintgcfrac</code>	272	<code>\xintifInt:</code>	365, 376
<code>\xintGCntoF</code>	288	<code>\xintifIntNE:</code>	375, 376
<code>\xintgcntof</code>	288	<code>\xintiFloor</code>	191, 325, 356
<code>\xintGCntoGC</code>	290	<code>\xintifloor</code>	191, 214
<code>\xintgcntogc</code>	290	<code>\xintifLt</code>	188
<code>\xintGctoCv</code>	284	<code>\xintiflt</code>	187, 188
<code>\xintgctocv</code>	284	<code>\xintifNotZero</code>	188
<code>\xintGctoF</code>	281	<code>\xintifnotzero</code>	188
<code>\xintgctof</code>	281	<code>\xintifOdd</code>	189
<code>\xintGctoGC</code>	291	<code>\xintifodd</code>	189
<code>\xintgctogc</code>	291	<code>\xintifOne</code>	188, 366, 375
<code>\xintGctoGCx</code>	275	<code>\xintifone</code>	188
<code>\xintgctogcx</code>	275	<code>\xintifOne:</code>	366, 376
<code>\xintGeq</code>	217	<code>\xintifOneNE:</code>	375, 376
<code>\xintgeq</code>	217, 218	<code>\xintifSgn</code>	187
<code>\xintgfdef</code>	20	<code>\xintifsgn</code>	187
<code>\xintGGCFrac</code>	274	<code>\xintifsgnexpr</code>	300
<code>\xintggcfrac</code>	274	<code>\xintifsgnfloatexpr</code>	300
<code>\xintgobble</code>	15	<code>\xintifsgniexpr</code>	300
<code>\xintgodef</code>	20	<code>\xintifTrueAelseB</code>	127--129
<code>\xintgoodef</code>	20	<code>\xintifZero</code>	188
<code>\xintGt</code>	187	<code>\xintifzero</code>	188
<code>\xintgt</code>	187	<code>\xintiGctoCv</code>	286
<code>\xintGtorEq</code>	187	<code>\xintigctocv</code>	286, 287
<code>\xintHalf</code>	63	<code>\xintiGctoF</code>	282
<code>\xinthalf</code>	63, 241	<code>\xintigctof</code>	282
<code>\xintHexToBin</code>	164	<code>\xintiiAbs</code>	61, 356
<code>\xinthextobin</code>	164	<code>\xintiiabs</code>	61,
<code>\xintHexToDec</code>	160, 296		105, 148, 149, 167, 168, 172--174, 352, 353
<code>\xinthextodec</code>	160	<code>\xintiiAdd</code>	72, 342
<code>\xintiCeil</code>	191, 314, 325, 356	<code>\xintiiadd</code>	
<code>\xinticeil</code>	191		41, 72, 136, 137, 139, 151, 171, 172, 175,
<code>\xintiCstoCv</code>	284		198, 210, 263, 267, 280--286, 328, 352, 367
<code>\xinticstocv</code>	284, 287	<code>\xintiiBinomial</code>	139, 361
<code>\xintiCstoF</code>	280	<code>\xintiiBinomial</code>	139, 140
<code>\xinticstof</code>	280	<code>\xintiiCmp</code>	75, 125, 126, 217, 296
<code>\xintieval</code>	297	<code>\xintiiCmp</code>	75
<code>\xintiexpr</code>	297	<code>\xintiiDivFloor</code>	105
<code>\xintiexpro</code>	297, 299	<code>\xintiidivfloor</code>	105
<code>\xintifboolexpr</code>	300	<code>\xintiiDivision</code>	87
<code>\xintifboolfloatexpr</code>	300	<code>\xintiidivision</code>	
<code>\xintifbooliiexpr</code>	300		87, 102, 105, 137, 139, 216, 272, 275--277
<code>\xintifCmp</code>	187, 327, 328, 366--368	<code>\xintiiDivMod</code>	104, 360
<code>\xintifcmp</code>	187	<code>\xintiidivmod</code>	104, 105
<code>\xintifEq</code>	40, 188, 277	<code>\xintiiDivRound</code>	102, 256
<code>\xintifeq</code>	187, 188	<code>\xintiidivround</code>	102

<code>\xintiDivTrunc</code>	103	<code>\xintiimaxof</code>	131
<code>\xintiDivtrunc</code>	103	<code>\xintiMaxof:csv</code>	351, 363
<code>\xintiiE</code>	116, 217, 331	<code>\xintiMin</code>	130
<code>\xintiiE</code>	116, 190	<code>\xintiimin</code>	130, 131, 352
<code>\xintiiEq</code>	121	<code>\xintiMinof</code>	131
<code>\xintiiEval</code>	297	<code>\xintiiminof</code>	131
<code>\xintiiEven</code>	124, 358	<code>\xintiMinof:csv</code>	352, 363
<code>\xintiiEven</code>	124	<code>\xintiMMON</code>	124
<code>\xintiiExpr</code>	297	<code>\xintiimmon</code>	124
<code>\xintiiExpro</code>	297, 298	<code>\xintiMod</code>	105, 360
<code>\xintiFac</code>	110, 330, 359	<code>\xintiimod</code>	105
<code>\xintiifac</code>	110	<code>\xintiModTrunc</code>	103
<code>\xintiGCD</code>	148, 167	<code>\xintiimodtrunc</code>	103
<code>\xintiigcd</code>	148, 167, 354	<code>\xintiMON</code>	124
<code>\xintiGCDof:csv</code>	354, 363	<code>\xintiimon</code>	124
<code>\xintiGeq</code>	122	<code>\xintiMul</code>	
<code>\xintiigeq</code>	122		83, 137, 175, 210, 217, 231, 235, 296, 342
<code>\xintiGt</code>	122	<code>\xintiimul</code>	40, 41, 83, 149, 168, 203,
<code>\xintiGtorEq</code>	123		209, 211, 213, 217, 218, 221, 239, 244, 352
<code>\xintiifCmp</code>	125	<code>\xintiNotEq</code>	121
<code>\xintiifcmp</code>	125	<code>\xintiOdd</code>	123, 127, 243, 357
<code>\xintiifEq</code>	125	<code>\xintiodd</code>	123
<code>\xintiifeq</code>	121, 125	<code>\xintiOpp</code>	61, 330
<code>\xintiifGt</code>	125	<code>\xintiopp</code>	
<code>\xintiifgt</code>	122, 123, 125		61, 63, 65, 102--105, 170, 191, 198, 215, 216
<code>\xintiifLt</code>	126, 139, 153	<code>\xintiPFactorial</code>	145, 361
<code>\xintiiflt</code>	122, 123, 126, 231	<code>\xintiipfactorial</code>	145, 146
<code>\xintiifNotZero</code>	126, 330, 350, 351, 365, 375	<code>\xintiPow</code>	58, 106, 296
<code>\xintiifnotzero</code>	126, 127, 300	<code>\xintiipow</code>	106, 212
<code>\xintiifNotZero:</code>	365, 376	<code>\xintiPrd</code>	132
<code>\xintiifNotZeroNE:</code>	375, 376	<code>\xintiiprd</code>	132
<code>\xintiifOdd</code>	127	<code>\xintiPrd:csv</code>	352, 364
<code>\xintiifodd</code>	127	<code>\xintiQuo</code>	102, 149, 168, 209, 217, 362
<code>\xintiifOne</code>	126, 366, 375	<code>\xintiquo</code>	102, 191, 196, 209, 217, 229, 278
<code>\xintiifone</code>	126	<code>\xintiRandRange</code>	151, 362, 376
<code>\xintiifOne:</code>	366, 376	<code>\xintiirandrange</code>	151
<code>\xintiifOneNE:</code>	375, 376	<code>\xintiRandRangeAtoB</code>	151, 362, 376
<code>\xintiifSgn</code>	125, 330, 366, 375	<code>\xintiirandrangeAtoB</code>	151
<code>\xintiifsgn</code>	125, 300	<code>\xintiRem</code>	102, 196, 362
<code>\xintiifSgn:</code>	366, 376	<code>\xintiirem</code>	102, 198, 215, 217
<code>\xintiifSgnNE:</code>	375, 376	<code>\xintiSeq:csv</code>	326, 376
<code>\xintiifZero</code>	126, 190, 277	<code>\xintiiseq:csv</code>	326
<code>\xintiifzero</code>	126, 127	<code>\xintiSeqA:csv</code>	327
<code>\xintiiIsNotZero</code>	123, 357	<code>\xintiSeqB:csv</code>	327, 376
<code>\xintiiisnotzero</code>	123, 128	<code>\xintiSeqB:f:csv</code>	367
<code>\xintiiIsOne</code>	123, 127, 358	<code>\xintiSeqBNumeric:csv</code>	376
<code>\xintiiisone</code>	123	<code>\xintiSeqNumeric:csv</code>	376
<code>\xintiiIsZero</code>	123, 128, 357	<code>\xintiSgn</code>	60, 123, 125, 126, 128, 356
<code>\xintiiiszero</code>	123, 127	<code>\xintiisgn</code>	60
<code>\xintiLCM</code>	149, 168	<code>\xintiSqr</code>	105, 137, 139, 357
<code>\xinti lcm</code>	149, 168, 354	<code>\xintiisqr</code>	105, 211, 239, 243, 244
<code>\xintiLCMof:csv</code>	354, 363	<code>\xintiSqrt</code>	139, 359
<code>\xintiLt</code>	122	<code>\xintiisqrt</code>	139
<code>\xintiLtorEq</code>	123	<code>\xintiSqrtR</code>	139, 359
<code>\xintiMax</code>	129	<code>\xintiisqrtr</code>	139
<code>\xintiimax</code>	129, 131, 351	<code>\xintiSquareRoot</code>	133
<code>\xintiMaxof</code>	131	<code>\xintiisquareroot</code>	133, 139

\xintiiSub .....	77, 137	\XINTinFloatSeqB::csv .....	328, 376
\xintiisub .....	77, 137--139, 151, 256	\XINTinFloatSeqB:f:csv .....	368
\xintiiSum .....	132	\XINTinFloatSeqBNumeric::csv .....	376
\xintiisum .....	132	\XINTinFloatSeqNumeric::csv .....	376
\xintiiSum:csv .....	352, 364	\XINTinFloatSqr .....	357
\xintiLen .....	115	\XINTinFloatSqrt .....	241, 255, 354, 359
\xintilen .....	115	\XINTinfloatsqr .....	255
\xintiloop .....	35, 44	\XINTinFloatSqrtdigits .....	354, 359
\xintiloop_a .....	35	\XINTinFloatSub .....	234
\xintiloop_again .....	35, 36	\XINTinfloatsub .....	234
\xintiloop_again_b .....	35, 36	\XINTinFloatSum:csv .....	354, 364
\xintiloopindex .....	35, 44	\XINTinRandomFloatS .....	259
\xintiloopskipandredo .....	36	\XINTinrandomfloatS .....	259
\xintiloopskiptonext .....	36	\XINTinRandomFloatSdigits .....	259, 333, 376
\xintInc .....	63, 231	\XINTinRandomFloatSixteen .....	260, 333, 376
\xintinc .....	63, 88, 138, 230, 231	\xintintegers .....	40
\XINTinFloat .....	225, 258, 259, 354, 360	\xintiNum .....	59, 137, 138, 332
\XINTinfloat .. 224, 225, 233--235, 237, 242, 244, 249, 253--255, 257, 258, 302, 327, 328, 368		\xintinum .....	59, 115, 152, 256
\XINTinFloat::csv .....	299, 302	\xintInv .....	222
\XINTinFloatAdd .....	233, 342	\xintinv .....	222
\XINTinfloatadd .....	233, 268, 328, 354, 368	\xintipfactorial .....	213
\XINTinFloatBinomial .....	253, 361	\xintipow .....	211
\XINTinfloatbinomial .....	253	\xintiQuo .....	362
\XINTinFloatdigits .....	332, 354, 360	\xintiRem .....	362
\XINTinFloatDiv .....	236	\xintiRound .....	201, 359
\XINTinfloatdiv .....	236	\xintiround .....	201, 214, 301
\XINTinFloatDivFloor .....	258	\xintIrr .....	195, 355
\XINTinfloatdivfloor .....	258	\xintirr .....	195
\XINTinFloatDivMod .....	258, 360	\xintiSeries .....	262
\XINTinfloatdivmod .....	258	\xintiseries .....	262
\XINTinFloatE .....	257, 331	\xintIsFalse .....	127
\XINTinfloate .....	257	\xintIsInt .....	197, 358
\XINTinFloatFac .....	244, 330, 354, 358	\xintisint .....	197
\XINTinfloatfac .....	244	\xintIsNotZero .....	127, 187
\XINTinFloatFacdigits .....	354, 358	\xintisnotzero .....	187, 301
\XINTinFloatFracdigits .....	233, 356	\xintIsOne .....	188, 217, 358
\XINTinfloatfracdigits .....	233	\xintisone .....	217
\XINTinFloatMaxof:csv .....	354, 363	\xintIsTrue .....	127
\XINTinFloatMinof:csv .....	354, 363	\xintIsTrue::csv .....	297, 301, 377
\XINTinFloatMod .....	258, 360	\xintIsZero .....	127, 187
\XINTinfloatmod .....	258	\xintiszero .....	187
\XINTinFloatMul .....	235, 342, 357	\xintiTrunc .....	199, 267, 359
\XINTinfloatmul .....	235, 268, 269, 354	\xintitrunc .....	199, 201, 241, 266
\XINTinFloatPFactorial .....	249, 361	\xintJrr .....	197
\XINTinfloatpfactorial .....	249	\xintjrr .....	197
\XINTinFloatPow .....	237	\xintKeep .....	27
\XINTinfloatpow .....	237, 268, 269	\xintkeep .....	27
\XINTinFloatPower .....	240	\xintKeep:f:csv .....	48, 54, 322, 323
\XINTinfloatpower .....	240	\xintkeep:f:csv .....	48
\XINTinFloatPowerH .....	241	\xintKeep:x:csv .....	322, 323
\XINTinfloatpowerh .....	241	\xintKeepNoExpand .....	27
\XINTinFloatPrd:csv .....	354, 364	\xintkeepnoexpand .....	27
\XINTinFloatS .....	226	\xintKeepUnbraced .....	28
\XINTinfloatS .....	226, 233--237, 240, 241	\xintkeepunbraced .....	28
\XINTinFloatSeq::csv .....	376	\xintKeepUnbracedNoExpand .....	28
\XINTinFloatSeqA::csv .....	327	\xintkeepunbracednoexpand .....	28, 29
		\xintLastItem .....	12

<code>\xintlastitem</code> .....	12	<code>\xintNOT</code> .....	127
<code>\xintLastItem:f:csv</code> .....	54, 365	<code>\xintNotEq</code> .....	187
<code>\xintlastitem:f:csv</code> .....	54	<code>\xintNthElt</code> .....	26
<code>\xintLCM</code> .....	168	<code>\xintnthelt</code> .....	26
<code>\xintlcm</code> .....	168, 176	<code>\xintNthEltNoExpand</code> .....	26
<code>\xintLCMof</code> .....	176	<code>\xintntheltnoexpand</code> .....	26, 338, 339
<code>\xintlcmof</code> .....	176	<code>\xintNthEltPy:f:csv</code> .....	52, 54, 322
<code>\xintLCMof:csv</code> .....	353, 363	<code>\xintntheltpy:f:csv</code> .....	52
<code>\xintLDg</code> .....	62, 89, 123, 124, 187, 241, 243	<code>\xintNum</code> .....	59,
<code>\xintldg</code> .....	62		167, 168, 172--174, 187, 211--213, 244, 249,
<code>\xintLen</code> .....	115, 181		250, 253, 321, 322, 338, 354, 355, 362, 374
<code>\xintlen</code> .....	115, 181	<code>\xintnum</code> .....	59,
<code>\xintLength</code> .....	12, 205--208,		61, 72, 75, 78, 102, 122, 169, 201, 211, 242
	218, 222, 226, 228, 237, 238, 242, 296, 335	<code>\xintNumerator</code> .....	191
<code>\xintlength</code> .....	12, 26, 133, 152, 201	<code>\xintnumerator</code> .....	191
<code>\xintLength:f:csv</code> .....	47, 54, 322, 323, 365	<code>\xintOdd</code> .....	187, 189, 357
<code>\xintlength:f:csv</code> .....	47	<code>\xintodd</code> .....	187
<code>\xintLengthUpTo</code> .....	13	<code>\xintodef</code> .....	11, 20
<code>\xintlengthupto</code> .....	13	<code>\xintoodef</code> .....	11, 20
<code>\xintLengthUpTo:f:csv</code> .....	47	<code>\xintOpp</code> .....	191, 222, 234, 235, 330
<code>\xintlengthupto:f:csv</code> .....	47	<code>\xintopp</code> .....	210, 222
<code>\xintListSel:f:csv</code> .....	320, 322	<code>\xintOR</code> .....	128
<code>\xintListSel:x:csv</code> .....	320, 321, 376	<code>\xintor</code> .....	128
<code>\xintListWithSep</code> .....	25	<code>\xintORof</code> .....	128
<code>\xintlistwithsep</code> .....	25	<code>\xintorof</code> .....	128
<code>\xintListWithSepNoExpand</code> .....	25	<code>\xintORof:csv</code> .....	351, 364
<code>\xintlistwithseпноexpand</code> .....	25	<code>\xintorthat</code> .....	10
<code>\xintloop</code> .....	35, 177, 334	<code>\xintouteriloopindex</code> .....	36
<code>\xintloop_again</code> .....	35	<code>\xintPFactorial</code> .....	145, 213, 361
<code>\xintloopskiptonext</code> .....	35	<code>\xintpfactorial</code> .....	145, 146, 213
<code>\xintLt</code> .....	187	<code>\xintPFloat</code> .....	231
<code>\xintlt</code> .....	187	<code>\xintpfloat</code> .....	231
<code>\xintLtorEq</code> .....	187	<code>\xintPFloat::csv</code> .....	299, 302, 377
<code>\xintMax</code> .....	219	<code>\xintPIrr</code> .....	195, 355
<code>\xintmax</code> .....	219, 351, 354	<code>\xintpirr</code> .....	195
<code>\xintMaxof</code> .....	219	<code>\xintPow</code> .....	211, 264, 266, 267
<code>\xintmaxof</code> .....	219	<code>\xintpow</code> .....	211
<code>\xintMaxof:csv</code> .....	351, 363	<code>\xintPowerSeries</code> .....	263
<code>\xintMessage</code> ...	18, 334, 335, 337, 369--371, 379	<code>\xintpowerseries</code> .....	263
<code>\xintMin</code> .....	220	<code>\xintPowerSeriesX</code> .....	264
<code>\xintmin</code> .....	220, 352, 354	<code>\xintpowerseriesx</code> .....	264
<code>\xintMinof</code> .....	220	<code>\xintPRaw</code> .....	189
<code>\xintminof</code> .....	220	<code>\xintpraw</code> .....	189, 301
<code>\xintMinof:csv</code> .....	352, 363	<code>\xintPrd</code> .....	213
<code>\xintMod</code> .....	216, 258, 360	<code>\xintprd</code> .....	213
<code>\xintmod</code> .....	216, 353	<code>\xintPrd:csv</code> .....	352, 364
<code>\xintModTrunc</code> .....	214	<code>\xintRandomDigits</code> .....	150, 152, 153
<code>\xintmodtrunc</code> .....	214	<code>\xintrandomdigits</code> .....	150, 152, 153, 259
<code>\xintMul</code> .....	210, 265--267, 342	<code>\xintrationals</code> .....	40
<code>\xintmul</code> .....	210, 213, 263--265, 352	<code>\xintRationalSeries</code> .....	264
<code>\xintNewBoolExpr</code> .....	377	<code>\xintRationalSeriesX</code> .....	265
<code>\xintnewdummy</code> .....	335, 337	<code>\xintratseries</code> .....	264
<code>\xintNewExpr</code> .....	320, 377	<code>\xintratseriesx</code> .....	265
<code>\xintNewFloatExpr</code> .....	377	<code>\xintRaw</code> .....	189, 332, 353
<code>\xintNewFunction</code> .....	370	<code>\xintraw</code> .....	189, 190, 195, 202, 208,
<code>\xintNewIExpr</code> .....	377		210, 211, 214--216, 219--223, 327, 353, 366
<code>\xintNewIIExpr</code> .....	377	<code>\xintRawWithZeros</code> .....	190

<code>\xintdrawwithzeros</code>	40, 190, 191, 195--198, 217, 271, 272, 275--285, 287
<code>\xintRelaxArray</code>	44, 334
<code>\xintreplicate</code>	14, 120, 137, 201, 202, 206, 207
<code>\xintRev</code>	116
<code>\xintReverse:f:csv</code>	53, 54, 365
<code>\xintreverse:f:csv</code>	53
<code>\xintReverseDigits</code>	115, 116
<code>\xintreversedigits</code>	115
<code>\xintReverseOrder</code>	11, 349, 350
<code>\xintreverseorder</code>	11
<code>\xintRevWithBraces</code>	20, 347, 348
<code>\xintrevwithbraces</code>	20, 21
<code>\xintRevWithBracesNoExpand</code>	21
<code>\xintrevwithbracesnoexpand</code>	21
<code>\xintREZ</code>	194
<code>\xintrez</code>	194, 258, 259
<code>\xintREZ?</code>	191
<code>\xintRound</code>	201, 359
<code>\xintround</code>	201, 301
<code>\xintRound::csv</code>	299, 301, 377
<code>\xintSeq</code>	33
<code>\xintseq</code>	33
<code>\xintSeq::csv</code>	325, 376
<code>\xintseq::csv</code>	325
<code>\xintSeqA::csv</code>	327
<code>\xintSeqB::csv</code>	327, 376
<code>\xintSeqB:f:csv</code>	366
<code>\xintSeqBNumeric::csv</code>	376
<code>\xintSeqNumeric::csv</code>	376
<code>\xintSeries</code>	262
<code>\xintseries</code>	262
<code>\XINTsetupcatcodes</code>	7, 20, 56, 114, 155, 167, 180, 262, 271, 296
<code>\xintSgn</code>	187, 188, 223, 273, 356
<code>\xintsgn</code>	223
<code>\xintSgnFork</code>	124, 328, 367, 368
<code>\xintsgnfork</code>	124
<code>\xintSignedFrac</code>	192
<code>\xintsignedfrac</code>	192
<code>\xintSignedFwOver</code>	193
<code>\xintsignedfwover</code>	193
<code>\xintSPRaw</code>	301
<code>\xintspraw</code>	301
<code>\xintSPRaw::csv</code>	297, 301, 377
<code>\xintSqr</code>	211, 357
<code>\xintsqr</code>	211
<code>\xintSub</code>	210
<code>\xintsub</code>	210, 278, 279
<code>\xintSum</code>	210
<code>\xintsum</code>	210
<code>\xintSum:csv</code>	352, 363
<code>\xintTFrac</code>	198, 356
<code>\xinttfrac</code>	198, 233
<code>\xintthe</code>	297
<code>\xintthe_o</code>	297
<code>\xintthebareeval</code>	298, 340, 343, 345, 346, 348, 377

<code>\xintthebarefloateval</code>	298, 340, 343, 345, 346, 348, 377
<code>\xintthebareiieval</code>	298, 340, 343, 345, 346, 348, 378
<code>\xinttheboolexpr</code>	300, 377
<code>\xintthecoords</code>	298
<code>\xinttheDigits</code>	208, 299
<code>\xinttheexpr</code>	297, 298, 300, 377
<code>\xintthefloatexpr</code>	297, 298, 300, 377
<code>\xinttheiexpr</code>	297, 298, 377
<code>\xinttheiieval</code>	297, 298, 300, 377
<code>\xintToggle</code>	148, 332
<code>\xintTrim</code>	30
<code>\xinttrim</code>	30, 347--349
<code>\xintTrim:f:csv</code>	50, 54, 322, 323
<code>\xinttrim:f:csv</code>	50
<code>\xintTrimNoExpand</code>	30
<code>\xinttrimnoexpand</code>	30
<code>\xintTrimUnbraced</code>	31
<code>\xinttrimunbraced</code>	31
<code>\xintTrimUnbracedNoExpand</code>	31
<code>\xinttrimunbracednoexpand</code>	31
<code>\xintTrunc</code>	199, 359
<code>\xinttrunc</code>	190, 199, 266, 267
<code>\xintTTrunc</code>	201
<code>\xintttrunc</code>	201, 214
<code>\xintTypesetBezoutAlgorithm</code>	177
<code>\xintTypesetEuclideanAlgorithm</code>	176
<code>\xintunassignvar</code>	335
<code>\xintUniformDeviate</code>	17, 18
<code>\xintUse</code>	58
<code>\xintXOR</code>	128
<code>\xintxor</code>	128
<code>\xintXORof</code>	128
<code>\xintxorof</code>	128
<code>\xintXORof:csv</code>	351, 364
<code>\xintXRandomDigits</code>	151, 153
<code>\xintXTrunc</code>	202
<code>\xintZapFirstSpaces</code>	21, 23
<code>\xintzapfirstspaces</code>	21
<code>\xintZapLastSpaces</code>	22
<code>\xintzaplastspaces</code>	22
<code>\xintZapSpaces</code>	23
<code>\xintzapspaces</code>	23
<code>\xintZapSpacesB</code>	23, 40
<code>\xintzapspacesb</code>	23, 24

## Y

<code>\y</code>	5, 6, 19, 20, 56, 114, 154, 166, 180, 261, 270, 295
-----------------	---

## Z

<code>\Z</code>	10, 21, 24, 25, 59--61, 67, 82, 91, 93, 94, 115, 116, 132, 169, 173--175, 185, 192--198, 209, 212, 217, 218, 222, 246, 250, 271, 272, 275--279
<code>\z</code>	5, 6, 19, 20, 56, 113, 114, 154, 155, 166, 167, 180, 261, 262, 270, 271, 295, 296

## 14 Cumulative line count

xintkernel: 554. Total number of code lines: 14567. (but 3359 lines among them  
xinttools:1454. start either with {% or with }%.)  
xintcore:2182. Each package starts with circa 50 lines dealing with cat-  
xint:1551. codes, package identification and reloading management,  
xintbinhex: 472. also for Plain TeX. Version 1.3d of 2019/01/06.  
xintgcd: 434.  
xintfrac:3248.  
xintseries: 386.  
xintcfrac:1029.  
xintexpr:3257.