

The `xint` packages source code

JEAN-FRANÇOIS BURNOL

jfbu (at) free (dot) fr

Package version: 1.4a (2020/02/19); documentation date: 2020/02/19.

From source file `xint.dtx`. Time-stamp: <19-02-2020 at 09:09:07 CET>.

Contents

1	History timeline	2
2	Package <code>xintkernel</code> implementation	4
3	Package <code>xinttools</code> implementation	20
4	Package <code>xintcore</code> implementation	60
5	Package <code>xint</code> implementation	118
6	Package <code>xintbinhex</code> implementation	162
7	Package <code>xintgcd</code> implementation	174
8	Package <code>xintfrac</code> implementation	185
9	Package <code>xintseries</code> implementation	274
10	Package <code>xintcfrc</code> implementation	283
11	Package <code>xintexpr</code> implementation	306
12	Package <code>xinttrig</code> implementation	414
13	Package <code>xintlog</code> implementation	432
14	Cumulative line count	437

1 History timeline

This is 1.4a of 2020/02/19.

Please refer [CHANGES.html](#) for a (much more) detailed history.

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

- Release 1.4a of 2020/02/19: fixes 1.4 known bugs. Improves documentation.
- Release 1.4 of 2020/01/31: [xintexpr](#) overhaul to use `\expanded` based expansion control. Many new features, in particular support for input and output of nested structures. Breaking changes, main ones being the (provisory) drop of `x*[a, b,...]`, `x+[a, b,...]` et al. syntax and the requirement of `\expanded` primitive (currently required only by [xintexpr](#)).
- Release 1.3f of 2019/09/10: starred variant `\xintDigits*`.
- Release 1.3e of 2019/04/05: packages [xinttrig](#), [xintlog](#); `\xintdefefunc` ``non-protected'` variant of [\xintdeffunc](#) (at 1.4 the two got merged and `\xintdefefunc` became a deprecated alias for [\xintdeffunc](#)). Indices removed from [sourcexint.pdf](#).
- Release 1.3d of 2019/01/06: fix of 1.2p bug for division with a zero dividend and a one-digit divisor, [\xinteval](#) et al. wrappers, `gcd()` and `lcm()` work with fractions.
- Release 1.3c of 2018/06/17: documentation better hyperlinked, indices added to [sourcexint.pdf](#). Colon in `:=` now optional for [\xintdefvar](#) and [\xintdeffunc](#).
- Release 1.3b of 2018/05/18: randomness related additions (still WIP).
- Release 1.3a of 2018/03/07: efficiency fix of the mechanism for recursive functions.
- Release 1.3 of 2018/03/01: addition and subtraction use systematically least common multiple of denominators. Extensive under-the-hood refactoring of [\xintNewExpr](#) and [\xintdeffunc](#) which now allow recursive definitions. Removal of 1.2o deprecated macros.
- Release 1.2q of 2018/02/06: fix of 1.2l subtraction bug in special situation; tacit multiplication extended to cases such as `10!20!30!`.
- Release 1.2p of 2017/12/05: maps `//` and `/:` to the floored, not truncated, division. Simultaneous assignments possible with [\xintdefvar](#). Efficiency improvements in [xinttools](#).
- Release 1.2o of 2017/08/29: massive deprecations of those macros from [xintcore](#) and [xint](#) which filtered their arguments via [\xintNum](#).
- Release 1.2n of 2017/08/06: improvements of [xintbinhex](#).
- Release 1.2m of 2017/07/31: rewrite of [xintbinhex](#) in the style of the 1.2 techniques.
- Release 1.2l of 2017/07/26: under the hood efficiency improvements in the style of the 1.2 techniques; subtraction refactored. Compatibility of most [xintfrac](#) macros with arguments using non-delimited `\the\numexpr` or `\the\mathcode` etc...
- Release 1.2i of 2016/12/13: under the hood efficiency improvements in the style of the 1.2 techniques.
- Release 1.2 of 2015/10/10: complete refactoring of the core arithmetic macros and faster [\xintexpr](#) parser.
- Release 1.1 of 2014/10/28: extensive changes in [xintexpr](#). Addition and subtraction do not multiply denominators blindly but sometimes produce smaller ones. Also with that release, packages [xintkernel](#) and [xintcore](#) got extracted from [xinttools](#) and [xint](#).

- Release 1.09g of 2013/11/22: the `xinttools` package is extracted from `xint`; addition of `\xintloop` and `\xintiloop`.
- Release 1.09c of 2013/10/09: `\xintFor`, `\xintNewNumExpr` (ancestor of `\xintNewExpr`/`\xintdeffunc` mechanism).
- Release 1.09a of 2013/09/24: support for functions by `xintexpr`.
- Release 1.08 of 2013/06/07: the `xintbinhex` package.
- Release 1.07 of 2013/05/25: support for floating point numbers added to `xintfrac` and first release of the `xintexpr` package (provided `\xintexpr` and `\xintfloatexpr`).
- Release 1.04 of 2013/04/25: the `xintcfrac` package.
- Release 1.03 of 2013/04/14: the `xintfrac` and `xintseries` packages.
- Release 1.0 of 2013/03/28: initial release of the `xint` and `xintgcd` packages.

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, ε -TeX and reload detection . .	4	.12	<code>\xintReverseOrder</code>	11
.1.1	<code>\XINT_setcatcodes</code> , <code>\XINT_storecatcodes</code> , <code>\XINT_restorecatcodes_endinput</code>	5	.13	<code>\xintLength</code>	11
.2	Package identification	7	.14	<code>\xintLastItem</code>	12
.3	Constants	7	.15	<code>\xintFirstItem</code>	12
.4	(WIP) <code>\xint_texuniformdeviate</code> and needed counts	7	.16	<code>\xintLastOne</code>	12
.5	Token management utilities	8	.17	<code>\xintFirstOne</code>	13
.6	“gob til” macros and UD style fork . . .	9	.18	<code>\xintLengthUpTo</code>	13
.7	<code>\xint_afterfi</code>	9	.19	<code>\xintreplicate</code> , <code>\xintReplicate</code> . . .	14
.8	<code>\xint_bye</code> , <code>\xint_Bye</code>	9	.20	<code>\xintgobble</code> , <code>\xintGobble</code>	15
.9	<code>\xintdothis</code> , <code>\xintorthat</code>	9	.21	(WIP) <code>\xintUniformDeviate</code>	18
.10	<code>\xint_zapspace</code>	10	.22	<code>\xintMessage</code> , <code>\ifxintverbose</code>	18
.11	<code>\odef</code> , <code>\oodef</code> , <code>\fdef</code>	10	.23	<code>\ifxintglobaldefs</code> , <code>\XINT_global</code> . . .	19
			.24	(WIP) Expandable error message	19

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](#).

1.4 (2020/01/11). [\xintReplicate](#), [\xintGobble](#), [\xintLastOne](#), [\xintFirstOne](#).

2.1 Catcodes, ε -TeX and reload detection

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

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

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2  \catcode13=5      % ^^M
3  \endlinechar=13 %
4  \catcode123=1     % {
5  \catcode125=2     % }
6  \catcode35=6      % #
7  \catcode44=12     % ,
8  \catcode45=12     % -
9  \catcode46=12     % .
10 \catcode58=12     % :
11 \catcode95=11     % _
12 \expandafter
13  \ifx\csname PackageInfo\endcsname\relax
14    \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
15  \else
16    \def\y#1#2{\PackageInfo{#1}{#2}}%
17  \fi
18 \let\z\relax
19 \expandafter
20  \ifx\csname numexpr\endcsname\relax
21    \y{xintkernel}{\numexpr not available, aborting input}%

```

```

22     \def\z{\endgroup\endinput}%
23     \else
24     \expandafter
25     \ifx\csname XINTsetupcatcodes\endcsname\relax
26     \else
27     \y{xintkernel}{I was already loaded, aborting input}%
28     \def\z{\endgroup\endinput}%
29     \fi
30     \fi
31     \ifx\z\relax\else\expandafter\z\fi%

```

2.1.1 \XINT_setcatcodes, \XINT_storecatcodes, \XINT_restorecatcodes_endinput

```

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

```

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

```

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 \catcode1=3 % for ultra-safe séparateur &&A
113 }%
114 \XINT_setcatcodes
115 }%
116 \PrepareCatcodes

```

Other modules could possibly be loaded under a different catcode regime.

```

117 \def\XINTsetupcatcodes {% for use by other modules
118 \edef\XINT_restorecatcodes_endinput
119 {%
120 \XINT_restorecatcodes\noexpand\endinput %
121 }%
122 \XINT_setcatcodes

```

123 }%

2.2 Package identification

Inspired from HEIKO OBERDIEK's packages. Modified in 1.09b to allow re-use in the other modules. Also I assume now that if `\ProvidesPackage` exists it then does define `\ver@<pkgname>.sty`, code of HO for some reason escaping me (compatibility with LaTeX 2.09 or other things ??) seems to set extra precautions.

1.09c uses e-TeX `\ifdefined`.

```
124 \ifdefined\ProvidesPackage
125   \let\XINT_providespackage\relax
126 \else
127   \def\XINT_providespackage #1#2[#3]%
128       {\immediate\write-1{Package: #2 #3}%
129        \expandafter\xdef\csname ver@#2.sty\endcsname{#3}}%
130 \fi
131 \XINT_providespackage
132 \ProvidesPackage {xintkernel}%
133 [2020/02/19 v1.4a Paraphernalia for the xint packages (JFB)]%
```

2.3 Constants

```
134 \chardef\xint_c_      0
135 \chardef\xint_c_i     1
136 \chardef\xint_c_ii    2
137 \chardef\xint_c_iii   3
138 \chardef\xint_c_iv    4
139 \chardef\xint_c_v     5
140 \chardef\xint_c_vi    6
141 \chardef\xint_c_vii   7
142 \chardef\xint_c_viii  8
143 \chardef\xint_c_ix    9
144 \chardef\xint_c_x     10
145 \chardef\xint_c_xii   12
146 \chardef\xint_c_xiv   14
147 \chardef\xint_c_xvi   16
148 \chardef\xint_c_xviii 18
149 \chardef\xint_c_xx    20
150 \chardef\xint_c_xxii  22
151 \chardef\xint_c_ii^v   32
152 \chardef\xint_c_ii^vi  64
153 \chardef\xint_c_ii^vii 128
154 \mathchardef\xint_c_ii^viii 256
155 \mathchardef\xint_c_ii^xii 4096
156 \mathchardef\xint_c_x^iv 10000
```

2.4 (WIP) `\xint_texuniformdeviate` and needed counts

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

```
163 \csname newcount\endcsname\xint_c_ii^xxi
164 \xint_c_ii^xxi 2097152 % "200000, 2**21
165 \fi
```

2.5 Token management utilities

1.3b. `\xint_gobandstop_...` macros because this is handy for `\xintRandomDigits`. 1.3g forces `\empty` and `\space` to have their standard meanings, rather than simply alerting user in the (theoretical) case they don't that nothing will work. If some ~~TeX~~ user has `\renewcommand`ed them they will be long and this will trigger xint redefinitions and warnings.

```
166 \def\xint_tmpa { }%
167 \ifx\xint_tmpa\space\else
168   \immediate\write-1{Package xintkernel Warning:}%
169   \immediate\write-1{\string\space\xint_tmpa macro does not have its normal
170     meaning from Plain or LaTeX, but:}%
171   \immediate\write-1{\meaning\space}%
172   \let\space\xint_tmpa
173   \immediate\write-1{\space\space\space\space
174     % an exclamation might let Emacs/AUCTeX think it is an error message, a fair
175     Forcing \string\space\space to be the usual one.}%
176 \fi
177 \def\xint_tmpa {}%
178 \ifx\xint_tmpa\empty\else
179   \immediate\write-1{Package xintkernel Warning:}%
180   \immediate\write-1{\string\empty\space macro does not have its normal
181     meaning from Plain or LaTeX, but:}%
182   \immediate\write-1{\meaning\empty}%
183   \let\empty\xint_tmpa
184   \immediate\write-1{\space\space\space\space
185     Forcing \string\empty\space to be the usual one.}%
186 \fi
187 \let\xint_tmpa\relax
188 \let\xint_gobble_\empty
189 \long\def\xint_gobble_i #1{%
190 \long\def\xint_gobble_ii #1#2{%
191 \long\def\xint_gobble_iii #1#2#3{%
192 \long\def\xint_gobble_iv #1#2#3#4{%
193 \long\def\xint_gobble_v #1#2#3#4#5{%
194 \long\def\xint_gobble_vi #1#2#3#4#5#6{%
195 \long\def\xint_gobble_vii #1#2#3#4#5#6#7{%
196 \long\def\xint_gobble_viii #1#2#3#4#5#6#7#8{%
197 \let\xint_gob_andstop_\space
198 \long\def\xint_gob_andstop_i #1{ }%
199 \long\def\xint_gob_andstop_ii #1#2{ }%
200 \long\def\xint_gob_andstop_iii #1#2#3{ }%
201 \long\def\xint_gob_andstop_iv #1#2#3#4{ }%
202 \long\def\xint_gob_andstop_v #1#2#3#4#5{ }%
203 \long\def\xint_gob_andstop_vi #1#2#3#4#5#6{ }%
204 \long\def\xint_gob_andstop_vii #1#2#3#4#5#6#7{ }%
205 \long\def\xint_gob_andstop_viii #1#2#3#4#5#6#7#8{ }%
206 \long\def\xint_firstofone #1{#1}%
207 \long\def\xint_firstoftwo #1#2{#1}%
208 \long\def\xint_secondoftwo #1#2{#2}%

```



```
209 \let\xint_stop_aftergobble\xint_gob_andstop_i
210 \long\def\xint_stop_atfirstofone #1{ #1}%
211 \long\def\xint_stop_atfirstoftwo #1#2{ #1}%
212 \long\def\xint_stop_atsecondoftwo #1#2{ #2}%
213 \long\def\xint_exchangetwo_keepbraces #1#2{{#2}{#1}}%
```

2.6 “gob til” macros and UD style fork

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

2.7 \xint_afterfi

```
239 \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](#).

```
240 \long\def\xint_bye #1\xint_bye {}%
241 \long\def\xint_Bye #1\xint_bye {}%
242 \long\def\xint_stop_afterbye #1\xint_bye { }%
```

2.9 \xintdothis, \xintorthat

1.1.

1.2. names without underscores.

To be used this way:

```
\if..\xint_dothis{..}\fi
\if..\xint_dothis{..}\fi
```

```
\if..\xint_dothis{..}\fi
...more such...
\xint_orthat{...}
```

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

```
243 \long\def\xint_dothis #1#2\xint_orthat #3{\fi #1}% 1.1
244 \let\xint_orthat \xint_firstofone
245 \long\def\xintdothis #1#2\xintorthat #3{\fi #1}%
246 \let\xintorthat \xint_firstofone
```

2.10 \xint_zapspaces

1.1.

This little (quite fragile in the normal sense i.e. non robust in the normal sense of programming lingua) utility zaps leading, intermediate, trailing, spaces in completely expanding context (`\e2def`, `\csname...\endcsname`).

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

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`.

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`.

But for example `\foo{aa} {bb} {cc}` where `\foo` is a macro with three non-delimited arguments breaks expansion, as expansion of `\foo` will happen with `\xint_zapspaces` still around, and even if it wasn't it would have stripped the braces around `{bb}`, certainly breaking other things.

Despite such obvious shortcomings it is enough for our purposes. It is currently used by `xintexpr` at various locations e.g. cleaning up optional argument of `\xintiexpr` and `\xintfloatexpr`; maybe in future internal usage will drop this in favour of a more robust utility.

1.2e. \xint_zapspaces_o.

1.2i. made \long.

ATTENTION THAT `xinttools` HAS AN WHICH SHOULD NOT GET CONFUSED WITH THIS ONE

```
247 \long\def\xint_zapspaces #1 #2{#1#2\xint_zapspaces }% 1.1
248 \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.

```
249 \def\xintodef #1{\expandafter\def\expandafter#1\expandafter }%
250 \def\xintoodef #1{\expandafter\expandafter\expandafter\def
251 \expandafter\expandafter\expandafter#1%
252 \expandafter\expandafter\expandafter }%
253 \def\xintfdef #1#2%
254 {\expandafter\def\expandafter#1\expandafter{\romannumeral`&&@#2}}%
255 \ifdefined\odef\else\let\odef\xintodef\fi
256 \ifdefined\oodef\else\let\oodef\xintoodef\fi
257 \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](#).

```
258 \def\xintReverseOrder {\romannumeral0\xintreverseorder }%
259 \long\def\xintreverseorder #1%
260 {%
261   \XINT_rord_main {}#1%
262   \xint:
263     \xint_bye\xint_bye\xint_bye\xint_bye
264     \xint_bye\xint_bye\xint_bye\xint_bye
265   \xint:
266 }%
267 \long\def\XINT_rord_main #1#2#3#4#5#6#7#8#9%
268 {%
269   \xint_bye #9\XINT_rord_cleanup\xint_bye
270   \XINT_rord_main {#9#8#7#6#5#4#3#2#1}%
271 }%
272 \def\XINT_rord_cleanup #1{%
273 \long\def\XINT_rord_cleanup\xint_bye\XINT_rord_main ##1##2\xint:
274 {%
275   \expandafter#1\xint_gob_til_xint: ##1%
276 }}\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. Syntax with `\romannumeral0` adds some slight (negligible) overhead; it is done to fit some general principles of structure of the xint package macros but maybe at some point I should drop it. And in fact it is often called directly via the `\numexpr` access point. (bad coding...)

```
277 \def\xintLength {\romannumeral0\xintlength }%
278 \def\xintlength #1{%
279 \long\def\xintlength ##1%
280 {%
281   \expandafter#1\the\numexpr\XINT_length_loop
282   ##1\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
283     \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
284     \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
285   \relax
286 }}\xintlength{ }%
287 \long\def\XINT_length_loop #1#2#3#4#5#6#7#8#9%
288 {%
289   \xint_gob_til_xint: #9\XINT_length_finish_a\xint:
290   \xint_c_ix+\XINT_length_loop
291 }%
292 \def\XINT_length_finish_a\xint:\xint_c_ix+\XINT_length_loop
293   #1#2#3#4#5#6#7#8#9%
```

```
294 {%
295     #9\xint_bye
296 }%
```

2.14 \xintLastItem

1.2i (2016/12/10). One level of braces removed in output. Output empty if input empty. Attention! This means that an empty input or an input ending with a empty brace pair both give same output. The `\xint:` token must not be among items. `\xintFirstItem` added at 1.4 for usage in `xintexpr`. It must contain neither `\xint:` nor `\xint_bye` in its first item.

```
297 \def\xintLastItem {\romannumeral0\xintlastitem }%
298 \long\def\xintlastitem #1%
299 {%
300     \XINT_last_loop {}. #1%
301     {\xint:\XINT_last_loop_enda}{\xint:\XINT_last_loop_endb}%
302     {\xint:\XINT_last_loop_endc}{\xint:\XINT_last_loop_endd}%
303     {\xint:\XINT_last_loop_ende}{\xint:\XINT_last_loop_endf}%
304     {\xint:\XINT_last_loop_endg}{\xint:\XINT_last_loop_endh}\xint_bye
305 }%
306 \long\def\XINT_last_loop #1.#2#3#4#5#6#7#8#9%
307 {%
308     \xint_gob_til_xint: #9%
309     {#8}{#7}{#6}{#5}{#4}{#3}{#2}{#1}\xint:
310     \XINT_last_loop {#9}.%
311 }%
312 \long\def\XINT_last_loop_enda #1#2\xint_bye{ #1}%
313 \long\def\XINT_last_loop_endb #1#2#3\xint_bye{ #2}%
314 \long\def\XINT_last_loop_endc #1#2#3#4\xint_bye{ #3}%
315 \long\def\XINT_last_loop_endd #1#2#3#4#5\xint_bye{ #4}%
316 \long\def\XINT_last_loop_ende #1#2#3#4#5#6\xint_bye{ #5}%
317 \long\def\XINT_last_loop_endf #1#2#3#4#5#6#7\xint_bye{ #6}%
318 \long\def\XINT_last_loop_endg #1#2#3#4#5#6#7#8\xint_bye{ #7}%
319 \long\def\XINT_last_loop_endh #1#2#3#4#5#6#7#8#9\xint_bye{ #8}%

```

2.15 \xintFirstItem

1.4. There must be neither `\xint:` nor `\xint_bye` in its first item.

```
320 \def\xintFirstItem {\romannumeral0\xintfirstitem }%
321 \long\def\xintfirstitem #1{\XINT_firstitem #1{\xint:\XINT_firstitem_end}\xint_bye}%
322 \long\def\XINT_firstitem #1#2\xint_bye{\xint_gob_til_xint: #1\xint:\space #1}%
323 \def\XINT_firstitem_end\xint:{ }%
```

2.16 \xintLastOne

As `xintexpr` 1.4 uses `{c1}{c2}...{cN}` storage when gathering comma separated values we need to not handle identically an empty list and a list with an empty item (as the above allows hierarchical structures). But `\xintLastItem` removed one level of brace pair so it is inadequate for the `last()` function.

By the way it is logical to interpret «item» as meaning `{cj}` inclusive of the braces; but `xint` user manual was not written in this spirit. And thus `\xintLastItem` did brace stripping, thus we

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

need another name for maintaining backwards compatibility (although the cardinality of users is small).

The `\xint:` token must not be found (visible) among the item contents.

```
324 \def\xintLastOne {\romannumeral0\xintlstone }%
325 \long\def\xintlstone #1%
326 {%
327   \XINT_lstone_loop {}.#1%
328   {\xint:\XINT_lstone_loop_enda}{\xint:\XINT_lstone_loop_endb}%
329   {\xint:\XINT_lstone_loop_endc}{\xint:\XINT_lstone_loop_endd}%
330   {\xint:\XINT_lstone_loop_ende}{\xint:\XINT_lstone_loop_endf}%
331   {\xint:\XINT_lstone_loop_endg}{\xint:\XINT_lstone_loop_endh}\xint_bye
332 }%
333 \long\def\XINT_lstone_loop #1.#2#3#4#5#6#7#8#9%
334 {%
335   \xint_gob_til_xint: #9%
336   {#8}{#7}{#6}{#5}{#4}{#3}{#2}{#1}\xint:
337   \XINT_lstone_loop {#9}%.%
338 }%
339 \long\def\XINT_lstone_loop_enda #1#2\xint_bye{#1}%
340 \long\def\XINT_lstone_loop_endb #1#2#3\xint_bye{#2}%
341 \long\def\XINT_lstone_loop_endc #1#2#3#4\xint_bye{#3}%
342 \long\def\XINT_lstone_loop_endd #1#2#3#4#5\xint_bye{#4}%
343 \long\def\XINT_lstone_loop_ende #1#2#3#4#5#6\xint_bye{#5}%
344 \long\def\XINT_lstone_loop_endf #1#2#3#4#5#6#7\xint_bye{#6}%
345 \long\def\XINT_lstone_loop_endg #1#2#3#4#5#6#7#8\xint_bye{#7}%
346 \long\def\XINT_lstone_loop_endh #1#2#3#4#5#6#7#8#9\xint_bye{ #8}%
```

2.17 `\xintFirstOne`

For `xintexpr` 1.4 too. Jan 3, 2020.

This is an experimental macro, don't use it. If input is nil (empty set) it expands to nil, if not it fetches first item and brace it. Fetching will have stripped one brace pair if item was braced to start with, which is the case in non-symbolic `xintexpr` data objects.

I have not given much thought to this (make it shorter, allow all tokens, (we could first test if empty via combination with `\detokenize`), etc...) as I need to get `xint` 1.4 out soon. So in particular attention that the macro assumes the `\xint:` token is absent from first item of input.

```
347 \def\xintFirstOne {\romannumeral0\xintfirstone }%
348 \long\def\xintfirstone #1{\XINT_firstone #1{\xint:\XINT_firstone_empty}\xint:}%
349 \long\def\XINT_firstone #1#2\xint:{\xint_gob_til_xint: #1\xint:{#1}}%
350 \def\XINT_firstone_empty\xint:#1{ }%
```

2.18 `\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.

```
351 \def\xintLengthUpTo {\romannumeral0\xintlenthupto}%
352 \long\def\xintlenthupto #1#2%
353 {%
```

```

354 \expandafter\XINT_lengthupto_loop
355 \the\numexpr#1.#2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
356 \xint_c_vii\xint_c_vi\xint_c_v\xint_c_iv
357 \xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye.%
358 }%
359 \def\XINT_lengthupto_loop_a #1%
360 {%
361 \xint_UDsignfork
362 #1\XINT_lengthupto_gt
363 -\XINT_lengthupto_loop
364 \krof #1%
365 }%
366 \long\def\XINT_lengthupto_gt #1\xint_bye.{-0}%
367 \long\def\XINT_lengthupto_loop #1.#2#3#4#5#6#7#8#9%
368 {%
369 \xint_gob_til_xint: #9\XINT_lengthupto_finish_a\xint:%
370 \expandafter\XINT_lengthupto_loop_a\the\numexpr #1-\xint_c_viii.%
371 }%
372 \def\XINT_lengthupto_finish_a\xint:\expandafter\XINT_lengthupto_loop_a
373 \the\numexpr #1-\xint_c_viii.#2#3#4#5#6#7#8#9%
374 {%
375 \expandafter\XINT_lengthupto_finish_b\the\numexpr #1-#9\xint_bye
376 }%
377 \def\XINT_lengthupto_finish_b #1#2.%
378 {%
379 \xint_UDsignfork
380 #1{-0}%
381 -{ #1#2}%
382 \krof
383 }%

```

2.19 \xintreplicate, \xintReplicate

1.2i.

This is cloned from LaTeX3's `\prg_replicate:nn`, 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`.

1.4 (2020/01/11). Added `\xintReplicate` ! The reason I did not before is that the prevailing habits in xint source code was to trigger with `\romannumeral0` not `\romannumeral` which is the lowercased named macros. Thus adding the camelcase one creates a couple `\xintReplicate`/`\xintreplicate` not obeying the general mold.

```

384 \def\xintReplicate{\romannumeral\xintreplicate}%
385 \def\xintreplicate#1%
386 {\expandafter\XINT_replicate\the\numexpr#1\endcsname}%
387 \def\XINT_replicate #1{\xint_UDsignfork

```


for playing with long decimal expansions.

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

1.4 (2020/01/11). Added `\xintGobble`.

```

435 \def\xintGobble{\romannumeral\xintgobble}%
436 \def\xintgobble #1%
437   {\csname xint_c_\expandafter\XINT_gobble_a\the\numexpr#1.0}%
438 \def\XINT_gobble #1.{\csname xint_c_\XINT_gobble_a #1.0}%
439 \def\XINT_gobble_a #1{\xint_gob_til_zero#1\XINT_gobble_d0\XINT_gobble_b#1}%
440 \def\XINT_gobble_b #1.#2%
441   {\expandafter\XINT_gobble_c
442     \the\numexpr (#1+\xint_c_v)/\xint_c_ix-\xint_c_i\expandafter.%
443     \the\numexpr #2+\xint_c_i.#1}%
444 \def\XINT_gobble_c #1.#2.#3%
445   {\csname XINT_g#2\the\numexpr#3-\xint_c_ix*#1\relax\XINT_gobble_a #1.#2}%
446 \def\XINT_gobble_d0\XINT_gobble_b0.#1{\endcsname}%
447 \expandafter\let\csname XINT_g10\endcsname\endcsname
448 \long\expandafter\def\csname XINT_g11\endcsname#1{\endcsname}%
449 \long\expandafter\def\csname XINT_g12\endcsname#1#2{\endcsname}%
450 \long\expandafter\def\csname XINT_g13\endcsname#1#2#3{\endcsname}%
451 \long\expandafter\def\csname XINT_g14\endcsname#1#2#3#4{\endcsname}%
452 \long\expandafter\def\csname XINT_g15\endcsname#1#2#3#4#5{\endcsname}%
453 \long\expandafter\def\csname XINT_g16\endcsname#1#2#3#4#5#6{\endcsname}%
454 \long\expandafter\def\csname XINT_g17\endcsname#1#2#3#4#5#6#7{\endcsname}%
455 \long\expandafter\def\csname XINT_g18\endcsname#1#2#3#4#5#6#7#8{\endcsname}%
456 \expandafter\let\csname XINT_g20\endcsname\endcsname
457 \long\expandafter\def\csname XINT_g21\endcsname #1#2#3#4#5#6#7#8#9%
458   {\endcsname}%
459 \long\expandafter\edef\csname XINT_g22\endcsname #1#2#3#4#5#6#7#8#9%
460   {\expandafter\noexpand\csname XINT_g21\endcsname}%
461 \long\expandafter\edef\csname XINT_g23\endcsname #1#2#3#4#5#6#7#8#9%
462   {\expandafter\noexpand\csname XINT_g22\endcsname}%
463 \long\expandafter\edef\csname XINT_g24\endcsname #1#2#3#4#5#6#7#8#9%
464   {\expandafter\noexpand\csname XINT_g23\endcsname}%
465 \long\expandafter\edef\csname XINT_g25\endcsname #1#2#3#4#5#6#7#8#9%
466   {\expandafter\noexpand\csname XINT_g24\endcsname}%
467 \long\expandafter\edef\csname XINT_g26\endcsname #1#2#3#4#5#6#7#8#9%
468   {\expandafter\noexpand\csname XINT_g25\endcsname}%
469 \long\expandafter\edef\csname XINT_g27\endcsname #1#2#3#4#5#6#7#8#9%
470   {\expandafter\noexpand\csname XINT_g26\endcsname}%
471 \long\expandafter\edef\csname XINT_g28\endcsname #1#2#3#4#5#6#7#8#9%
472   {\expandafter\noexpand\csname XINT_g27\endcsname}%
473 \expandafter\let\csname XINT_g30\endcsname\endcsname
474 \long\expandafter\edef\csname XINT_g31\endcsname #1#2#3#4#5#6#7#8#9%
475   {\expandafter\noexpand\csname XINT_g28\endcsname}%
476 \long\expandafter\edef\csname XINT_g32\endcsname #1#2#3#4#5#6#7#8#9%
477   {\noexpand\csname XINT_g31\expandafter\noexpand\csname XINT_g28\endcsname}%
478 \long\expandafter\edef\csname XINT_g33\endcsname #1#2#3#4#5#6#7#8#9%
479   {\noexpand\csname XINT_g32\expandafter\noexpand\csname XINT_g28\endcsname}%
480 \long\expandafter\edef\csname XINT_g34\endcsname #1#2#3#4#5#6#7#8#9%
481   {\noexpand\csname XINT_g33\expandafter\noexpand\csname XINT_g28\endcsname}%
482 \long\expandafter\edef\csname XINT_g35\endcsname #1#2#3#4#5#6#7#8#9%
483   {\noexpand\csname XINT_g34\expandafter\noexpand\csname XINT_g28\endcsname}%

```



```

484 \long\expandafter\edef\csname XINT_g36\endcsname #1#2#3#4#5#6#7#8#9%
485 {\noexpand\csname XINT_g35\expandafter\noexpand\csname XINT_g28\endcsname}%
486 \long\expandafter\edef\csname XINT_g37\endcsname #1#2#3#4#5#6#7#8#9%
487 {\noexpand\csname XINT_g36\expandafter\noexpand\csname XINT_g28\endcsname}%
488 \long\expandafter\edef\csname XINT_g38\endcsname #1#2#3#4#5#6#7#8#9%
489 {\noexpand\csname XINT_g37\expandafter\noexpand\csname XINT_g28\endcsname}%
490 \expandafter\let\csname XINT_g40\endcsname\endcsname
491 \expandafter\edef\csname XINT_g41\endcsname
492 {\noexpand\csname XINT_g38\expandafter\noexpand\csname XINT_g31\endcsname}%
493 \expandafter\edef\csname XINT_g42\endcsname
494 {\noexpand\csname XINT_g41\expandafter\noexpand\csname XINT_g41\endcsname}%
495 \expandafter\edef\csname XINT_g43\endcsname
496 {\noexpand\csname XINT_g42\expandafter\noexpand\csname XINT_g41\endcsname}%
497 \expandafter\edef\csname XINT_g44\endcsname
498 {\noexpand\csname XINT_g43\expandafter\noexpand\csname XINT_g41\endcsname}%
499 \expandafter\edef\csname XINT_g45\endcsname
500 {\noexpand\csname XINT_g44\expandafter\noexpand\csname XINT_g41\endcsname}%
501 \expandafter\edef\csname XINT_g46\endcsname
502 {\noexpand\csname XINT_g45\expandafter\noexpand\csname XINT_g41\endcsname}%
503 \expandafter\edef\csname XINT_g47\endcsname
504 {\noexpand\csname XINT_g46\expandafter\noexpand\csname XINT_g41\endcsname}%
505 \expandafter\edef\csname XINT_g48\endcsname
506 {\noexpand\csname XINT_g47\expandafter\noexpand\csname XINT_g41\endcsname}%
507 \expandafter\let\csname XINT_g50\endcsname\endcsname
508 \expandafter\edef\csname XINT_g51\endcsname
509 {\noexpand\csname XINT_g48\expandafter\noexpand\csname XINT_g41\endcsname}%
510 \expandafter\edef\csname XINT_g52\endcsname
511 {\noexpand\csname XINT_g51\expandafter\noexpand\csname XINT_g51\endcsname}%
512 \expandafter\edef\csname XINT_g53\endcsname
513 {\noexpand\csname XINT_g52\expandafter\noexpand\csname XINT_g51\endcsname}%
514 \expandafter\edef\csname XINT_g54\endcsname
515 {\noexpand\csname XINT_g53\expandafter\noexpand\csname XINT_g51\endcsname}%
516 \expandafter\edef\csname XINT_g55\endcsname
517 {\noexpand\csname XINT_g54\expandafter\noexpand\csname XINT_g51\endcsname}%
518 \expandafter\edef\csname XINT_g56\endcsname
519 {\noexpand\csname XINT_g55\expandafter\noexpand\csname XINT_g51\endcsname}%
520 \expandafter\edef\csname XINT_g57\endcsname
521 {\noexpand\csname XINT_g56\expandafter\noexpand\csname XINT_g51\endcsname}%
522 \expandafter\edef\csname XINT_g58\endcsname
523 {\noexpand\csname XINT_g57\expandafter\noexpand\csname XINT_g51\endcsname}%
524 \expandafter\let\csname XINT_g60\endcsname\endcsname
525 \expandafter\edef\csname XINT_g61\endcsname
526 {\noexpand\csname XINT_g58\expandafter\noexpand\csname XINT_g51\endcsname}%
527 \expandafter\edef\csname XINT_g62\endcsname
528 {\noexpand\csname XINT_g61\expandafter\noexpand\csname XINT_g61\endcsname}%
529 \expandafter\edef\csname XINT_g63\endcsname
530 {\noexpand\csname XINT_g62\expandafter\noexpand\csname XINT_g61\endcsname}%
531 \expandafter\edef\csname XINT_g64\endcsname
532 {\noexpand\csname XINT_g63\expandafter\noexpand\csname XINT_g61\endcsname}%
533 \expandafter\edef\csname XINT_g65\endcsname
534 {\noexpand\csname XINT_g64\expandafter\noexpand\csname XINT_g61\endcsname}%
535 \expandafter\edef\csname XINT_g66\endcsname

```

```

536 {\noexpand\csname XINT_g65\expandafter\noexpand\csname XINT_g61\endcsname}%
537 \expandafter\edef\csname XINT_g67\endcsname
538 {\noexpand\csname XINT_g66\expandafter\noexpand\csname XINT_g61\endcsname}%
539 \expandafter\edef\csname XINT_g68\endcsname
540 {\noexpand\csname XINT_g67\expandafter\noexpand\csname XINT_g61\endcsname}%

```

2.21 (WIP) \xintUniformDeviate

1.3b. See user manual for related information.

```

541 \ifdefined\xint_texuniformdeviate
542     \expandafter\xint_firstoftwo
543 \else\expandafter\xint_secondoftwo
544 \fi
545 {%
546     \def\xintUniformDeviate#1%
547         {\the\numexpr\expandafter\XINT_uniformdeviate_sgnfork\the\numexpr#1\xint:}%
548     \def\XINT_uniformdeviate_sgnfork#1%
549         {%
550             \if-#1\XINT_uniformdeviate_neg\fi \XINT_uniformdeviate{ }#1%
551         }%
552     \def\XINT_uniformdeviate_neg\fi\XINT_uniformdeviate#1-%
553         {%
554             \fi-\numexpr\XINT_uniformdeviate\relax
555         }%
556     \def\XINT_uniformdeviate#1#2\xint:
557         {%
558             \expandafter\XINT_uniformdeviate_a\the\numexpr%
559                 -\xint_texuniformdeviate\xint_c_ii^vii%
560                 -\xint_c_ii^vii*\xint_texuniformdeviate\xint_c_ii^vii%
561                 -\xint_c_ii^xiv*\xint_texuniformdeviate\xint_c_ii^vii%
562                 -\xint_c_ii^xxi*\xint_texuniformdeviate\xint_c_ii^vii%
563                 +\xint_texuniformdeviate#2\xint:/#2)*#2\xint:+#2\fi\relax#1%
564         }%
565     \def\XINT_uniformdeviate_a #1\xint:
566         {%
567             \expandafter\XINT_uniformdeviate_b\the\numexpr#1-(#1%
568         }%
569     \def\XINT_uniformdeviate_b#1#2\xint:{#1#2\if-#1}%
570 }%
571 {%
572     \def\xintUniformDeviate#1%
573         {%
574             \the\numexpr
575             \XINT_expandableerror{No uniformdeviate at engine level, returning 0.}%
576             0\relax
577         }%
578 }%

```

2.22 \xintMessage, \ifxintverbose

1.2c. for use by [\xintdefvar](#) and [\xintdeffunc](#) of [xintexpr](#).

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

1.2e. uses `\write128` rather than `\write16` for compatibility with future extended range of output streams, in LuaTeX in particular.

1.3e. set the `\newlinechar`.

```
579 \def\xintMessage #1#2#3{%
580   \edef\xINT_newlinechar{\the\xintnewlinechar}%
581   \newlinechar10
582   \immediate\write128{Package #1 #2: (on line \the\inputlineno)}%
583   \immediate\write128{\space\space\space\space#3}%
584   \newlinechar\xINT_newlinechar\space
585 }%
586 \newif\ifxintverbose
```

2.23 `\ifxintglobaldefs`, `\XINT_global`

1.3c.

```
587 \newif\ifxintglobaldefs
588 \def\xINT_global{\ifxintglobaldefs\global\fi}%
```

2.24 (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.

1.4 (2020/01/25). Finally rather than `\ ! /` I use `\xint/`.

```
589 \def\xINT_expandableerror #1#2{%
590   \def\xINT_expandableerror ##1{%
591     \expandafter\expandafter\expandafter
592     \xINT_expandableerror_continue\xint_firstofone{#2#1##1}}%
593   \def\xINT_expandableerror_continue ##1#1##2#1{##1}%
594 }%
595 \begingroup\lccode`$ 32 \catcode`/ 11 % $
596 \lowercase{\endgroup\xINT_expandableerror$\xint/\let\xint/\xint_undefined}% $
597 \XINT_restorecatcodes_endinput%
```

3 Package [xinttools](#) implementation

.1	Catcodes, ε -T _E X and reload detection . .	20		
.2	Package identification	21		
.3	\xintgodef, \xintgoodef, \xintgfdef .	21	.24	\XINT_xflet 40
.4	\xintRevWithBraces	21	.25	\xintApplyInline 40
.5	\xintZapFirstSpaces	22	.26	\xintFor, \xintFor*, \xintBreakFor, \xintBreakForAndDo 41
.6	\xintZapLastSpaces	23	.27	\XINT_forever, \xintintegers, \xintdi- mensions, \xintrationals 43
.7	\xintZapSpaces	24	.28	\xintForpair, \xintForthree, \xintFor- four 45
.8	\xintZapSpacesB	24	.29	\xintAssign, \xintAssignArray, \xint- DigitsOf 47
.9	\xintCSVtoList, \xintCSVtoListNon- Stripped	24	.30	CSV (non user documented) variants of Length, Keep, Trim, NthElt, Reverse . . 49
.10	\xintListWithSep	26	.30.1	\xintLength:f:csv 50
.11	\xintNthElt	27	.30.2	\xintLengthUpTo:f:csv 51
.12	\xintNthOnePy	28	.30.3	\xintKeep:f:csv 52
.13	\xintKeep	29	.30.4	\xintTrim:f:csv 54
.14	\xintKeepUnbraced	30	.30.5	\xintNthEltPy:f:csv 55
.15	\xintTrim	31	.30.6	\xintReverse:f:csv 56
.16	\xintTrimUnbraced	33	.30.7	\xintFirstItem:f:csv 57
.17	\xintApply	34	.30.8	\xintLastItem:f:csv 57
.18	\xintApply:x (not public)	34	.30.9	\xintKeep:x:csv 58
.19	\xintApplyUnbraced	35	.30.10	Public names for the undocumented csv macros: \xintCSVLength, \xintCSVKeep, \xintCSVKeepx, \xintCSVTrim, \xintCSVNthEltPy, \xintCSVRe- verse, \xintCSVFirstItem, \xintCSVLastItem 58
.20	\xintApplyUnbraced:x (not public) . .	35		
.21	\xintSeq	36		
.22	\xintloop, \xintbreakloop, \xintbreak- loopanddo, \xintloopskiptonext . . .	39		
.23	\xintilooop, \xintilooopindex, \xint- bracedilooopindex, \xintouterilooopindex, \xintbracedouterilooopindex, \xintbreak-			

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, ε -T_EX and reload detection

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

```

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

```

```

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

```

3.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xinttools}%
46 [2020/02/19 v1.4a Expandable and non-expandable utilities (JFB)]%

```

`\XINT_toks` is used in macros such as `\xintFor`. It is not used elsewhere in the xint bundle.

```

47 \newtoks\XINT_toks
48 \xint_firstofone{\let\XINT_sptoken= } %<- space here!

```

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

1.09i. For use in `\xintAssign`.

```

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

```

3.4 `\xintRevWithBraces`

New with 1.06. Makes the expansion of its argument and then reverses the resulting tokens or braced tokens, adding a pair of braces to each (thus, maintaining it when it was already there.) The reason for `\xint:`, here and in other locations, is in case #1 expands to nothing, the `\romannumeral-`0` must be stopped

```

52 \def\xintRevWithBraces          {\romannumeral0\xintrevwithbraces }%
53 \def\xintRevWithBracesNoExpand {\romannumeral0\xintrevwithbracesnoexpand }%
54 \long\def\xintrevwithbraces #1%
55 {%
56     \expandafter\XINT_revwbr_loop\expandafter{\expandafter}%
57     \romannumeral`&&#1\xint:\xint:\xint:\xint:%
58         \xint:\xint:\xint:\xint:\xint_bye
59 }%
60 \long\def\xintrevwithbracesnoexpand #1%
61 {%
62     \XINT_revwbr_loop }%
63     #1\xint:\xint:\xint:\xint:%
64     \xint:\xint:\xint:\xint:\xint_bye
65 }%
66 \long\def\XINT_revwbr_loop #1#2#3#4#5#6#7#8#9%
67 {%
68     \xint_gob_til_xint: #9\XINT_revwbr_finish_a\xint:%
69     \XINT_revwbr_loop {{#9}{#8}{#7}{#6}{#5}{#4}{#3}{#2}{#1}}%
70 }%
71 \long\def\XINT_revwbr_finish_a\xint:\XINT_revwbr_loop #1#2\xint_bye
72 {%
73     \XINT_revwbr_finish_b #2\R\R\R\R\R\R\R\R\Z #1%
74 }%
75 \def\XINT_revwbr_finish_b #1#2#3#4#5#6#7#8\Z
76 {%
77     \xint_gob_til_R
78         #1\XINT_revwbr_finish_c \xint_gobble_viii
79         #2\XINT_revwbr_finish_c \xint_gobble_vii
80         #3\XINT_revwbr_finish_c \xint_gobble_vi
81         #4\XINT_revwbr_finish_c \xint_gobble_v
82         #5\XINT_revwbr_finish_c \xint_gobble_iv
83         #6\XINT_revwbr_finish_c \xint_gobble_iii
84         #7\XINT_revwbr_finish_c \xint_gobble_ii
85         \R\XINT_revwbr_finish_c \xint_gobble_i\Z
86 }%

```

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

```

87 \def\XINT_revwbr_finish_c#1{%
88 \def\XINT_revwbr_finish_c##1##2\Z{\expandafter#1##1}%
89 }\XINT_revwbr_finish_c{ }%

```

3.5 \xintZapFirstSpaces

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

```

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

```

If the original #1 started with a space, the grabbed #1 is empty. Thus `_again?` will see `#1=\xint_bye`, and hand over control to `_again` which will loop back into `\XINT_zapbsp_a`, with one initial space less. If the original #1 did not start with a space, or was empty, then the #1 below will be a `<sptoken>`, then an extract of the original #1, not empty and not starting with a space, which contains what was up to the first `<sp><sp>` present in original #1, or, if none preexisted, `<sptoken>` and all of #1 (possibly empty) plus an ending `\xint:`. The added initial space will stop later the `\romannumeral0`. No brace stripping is possible. Control is handed over to `\XINT_zapbsp_b` which strips out the ending `\xint:<sp><sp>\xint:`

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

```

3.6 \xintZapLastSpaces

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

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

```

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

```
104 \xint_firstofone {\long\def\xint_zapbsp_a #1#2 } %<- second space here
105   {\expandafter\xint_zapbsp_b\expandafter{#2}{#1}}%

```

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

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

```

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

```
108 \long\def\xint_zapbsp_end? #1{\xint_bye #1\xint_zapbsp_end }%

```

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

```
109 \long\def\xint_zapbsp_end\xint_zapbsp_e #1#2\xint:{ #1}%

```

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

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

```

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

```

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

```

3.9 \xintCSVtoList, \xintCSVtoListNonStripped

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

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

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

```

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

```



```

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

```

1.1c revisits this old code and improves upon the earlier endings. But as the `_d..` macros have already nine parameters, I needed the `\expandafter` and `\xint_gob_til_Z` in `finish_b` (compare `\XINT_keep_endb`, or also `\XINT_RQ_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:\xint:
258   \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
259   \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
260   -#1.#2\xint_bye
261 }%
262 \def\XINT_nthelt_neg_a #1%
263 {%
264   \xint_UDzerominusfork
265     #1-\xint_stop_afterbye
266     0#1\xint_stop_afterbye
267     0-{}%
268   \krof
269   \expandafter\XINT_nthelt_neg_b
270   \romannumeral\expandafter\XINT_gobble\the\numexpr-\xint_c_i+#1%

```

```

271 }%
272 \long\def\XINT_nthelt_neg_b #1#2\xint_bye{ #1}%
273 \long\def\XINT_nthelt_pos #1.#2%
274 {%
275     \expandafter\XINT_nthelt_pos_done
276     \romannumeral0\expandafter\XINT_trim_loop\the\numexpr#1-\xint_c_x.%
277     #2\xint:\xint:\xint:\xint:\xint:%
278     \xint:\xint:\xint:\xint:\xint:%
279     \xint_bye
280 }%
281 \def\XINT_nthelt_pos_done #1{%
282 \long\def\XINT_nthelt_pos_done ##1##2\xint_bye{%
283     \xint_gob_til_xint:##1\expandafter#1\xint_gobble_ii\xint:#1##1}%
284 }\XINT_nthelt_pos_done{ }%

```

3.12 \xintNthOnePy

First included in release 1.4. See relevant code comments in `xintexpr`.

```

285 \def\xintNthOnePy          {\romannumeral0\xintnthonepy }%
286 \def\xintNthOnePyNoExpand {\romannumeral0\xintnthonepynoexpand }%
287 \long\def\xintnthonepy #1#2{\expandafter\XINT_nthonepy_a\the\numexpr #1\expandafter.%
288     \expandafter{\romannumeral`&&@#2}}%
289 \def\xintnthonepynoexpand #1{\expandafter\XINT_nthonepy_a\the\numexpr #1.}%
290 \def\XINT_nthonepy_a #1%
291 {%
292     \xint_UDsignfork
293     #1\XINT_nthonepy_neg
294     -{\XINT_nthonepy_nonneg #1}%
295     \krof
296 }%
297 \long\def\XINT_nthonepy_neg #1.#2%
298 {%
299     \expandafter\XINT_nthonepy_neg_a\the\numexpr\xint_c_i+\XINT_length_loop
300     #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
301     \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
302     \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
303     -#1.#2\xint_bye
304 }%
305 \def\XINT_nthonepy_neg_a #1%
306 {%
307     \xint_UDzerominusfork
308     #1-\xint_stop_afterbye
309     0#1\xint_stop_afterbye
310     0-{}%
311     \krof
312     \expandafter\XINT_nthonepy_neg_b
313     \romannumeral\expandafter\XINT_gobble\the\numexpr-\xint_c_i+#1%
314 }%
315 \long\def\XINT_nthonepy_neg_b #1#2\xint_bye{{#1}}%
316 \long\def\XINT_nthonepy_nonneg #1.#2%
317 {%
318     \expandafter\XINT_nthonepy_nonneg_done

```

```

319 \romannumeral0\expandafter\XINT_trim_loop\the\numexpr#1-\xint_c_ix.%
320 #2\xint:\xint:\xint:\xint:\xint:%
321 \xint:\xint:\xint:\xint:\xint:%
322 \xint_bye
323 }%
324 \def\XINT_nthoney_nonneg_done #1{%
325 \long\def\XINT_nthoney_nonneg_done ##1##2\xint_bye{%
326 \xint_gob_til_xint:##1\expandafter#1\xint_gobble_ii\xint:{##1}}%
327 }\XINT_nthoney_nonneg_done{ }%

```

3.13 \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`.

```

328 \def\xintKeep          {\romannumeral0\xintkeep }%
329 \def\xintKeepNoExpand {\romannumeral0\xintkeepnoexpand }%
330 \long\def\xintkeep #1#2{\expandafter\XINT_keep_a\the\numexpr #1\expandafter.%
331 \expandafter{\romannumeral`&&@#2}}%
332 \def\xintkeepnoexpand #1{\expandafter\XINT_keep_a\the\numexpr #1.}%
333 \def\XINT_keep_a #1%
334 {%
335 \xint_UDzerominusfork
336 #1-\XINT_keep_keeptime
337 0#1\XINT_keep_neg
338 0-{\XINT_keep_pos #1}%
339 \krof
340 }%
341 \long\def\XINT_keep_keeptime .#1{ }%
342 \long\def\XINT_keep_neg #1.#2%
343 {%
344 \expandafter\XINT_keep_neg_a\the\numexpr
345 #1-\numexpr\XINT_length_loop
346 #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
347 \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
348 \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye.#2%
349 }%
350 \def\XINT_keep_neg_a #1%
351 {%
352 \xint_UDsignfork
353 #1{\expandafter\space\romannumeral\XINT_gobble}%
354 -\XINT_keep_keeptime
355 \krof

```

```

356 }%
357 \def\XINT_keep_keepall #1.{ }%
358 \long\def\XINT_keep_pos #1.#2%
359 {%
360     \expandafter\XINT_keep_loop
361     \the\numexpr#1-\XINT_lengthupto_loop
362     #1.#2\xint:\xint:\xint:\xint:\xint:\xint:\xint:
363     \xint_c_vii\xint_c_vi\xint_c_v\xint_c_iv
364     \xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye.%
365     -\xint_c_viii.{ }#2\xint_bye%
366 }%
367 \def\XINT_keep_loop #1#2.%
368 {%
369     \xint_gob_til_minus#1\XINT_keep_loop_end-%
370     \expandafter\XINT_keep_loop
371     \the\numexpr#1#2-\xint_c_viii\expandafter.\XINT_keep_loop_pickeight
372 }%
373 \long\def\XINT_keep_loop_pickeight
374     #1#2#3#4#5#6#7#8#9{{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}}}%
375 \def\XINT_keep_loop_end-\expandafter\XINT_keep_loop
376     \the\numexpr-#1-\xint_c_viii\expandafter.\XINT_keep_loop_pickeight
377     {\csname XINT_keep_end#1\endcsname}%
378 \long\expandafter\def\csname XINT_keep_end1\endcsname
379     #1#2#3#4#5#6#7#8#9\xint_bye { #1{#2}{#3}{#4}{#5}{#6}{#7}{#8}}}%
380 \long\expandafter\def\csname XINT_keep_end2\endcsname
381     #1#2#3#4#5#6#7#8\xint_bye { #1{#2}{#3}{#4}{#5}{#6}{#7}}}%
382 \long\expandafter\def\csname XINT_keep_end3\endcsname
383     #1#2#3#4#5#6#7\xint_bye { #1{#2}{#3}{#4}{#5}{#6}}}%
384 \long\expandafter\def\csname XINT_keep_end4\endcsname
385     #1#2#3#4#5#6\xint_bye { #1{#2}{#3}{#4}{#5}}}%
386 \long\expandafter\def\csname XINT_keep_end5\endcsname
387     #1#2#3#4#5\xint_bye { #1{#2}{#3}{#4}}}%
388 \long\expandafter\def\csname XINT_keep_end6\endcsname
389     #1#2#3#4\xint_bye { #1{#2}{#3}}}%
390 \long\expandafter\def\csname XINT_keep_end7\endcsname
391     #1#2#3\xint_bye { #1{#2}}}%
392 \long\expandafter\def\csname XINT_keep_end8\endcsname
393     #1#2\xint_bye { #1}%

```

3.14 \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.

```

394 \def\xintKeepUnbraced          {\romannumeral0\xintkeepunbraced }%
395 \def\xintKeepUnbracedNoExpand {\romannumeral0\xintkeepunbracednoexpand }%
396 \long\def\xintkeepunbraced #1#2%
397     {\expandafter\XINT_keepunbr_a\the\numexpr #1\expandafter.%
398     \expandafter{\romannumeral`&&@#2}}}%
399 \def\xintkeepunbracednoexpand #1%

```

```

400   {\expandafter\XINT_keepunbr_a\the\numexpr #1.}%
401 \def\XINT_keepunbr_a #1%
402 {%
403   \xint_UDzerominusfork
404     #1-\XINT_keep_keeptime
405     0#1\XINT_keep_neg
406     0-{\XINT_keepunbr_pos #1}%
407   \krof
408 }%
409 \long\def\XINT_keepunbr_pos #1.#2%
410 {%
411   \expandafter\XINT_keepunbr_loop
412   \the\numexpr#1-\XINT_lengthupto_loop
413   #1.#2\xint:\xint:\xint:\xint:\xint:\xint:\xint:
414     \xint_c_vii\xint_c_vi\xint_c_v\xint_c_iv
415     \xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye.%
416   -\xint_c_viii.{}#2\xint_bye%
417 }%
418 \def\XINT_keepunbr_loop #1#2.%
419 {%
420   \xint_gob_til_minus#1\XINT_keepunbr_loop_end-%
421   \expandafter\XINT_keepunbr_loop
422   \the\numexpr#1#2-\xint_c_viii\expandafter.\XINT_keepunbr_loop_pickeight
423 }%
424 \long\def\XINT_keepunbr_loop_pickeight
425   #1#2#3#4#5#6#7#8#9{{#1#2#3#4#5#6#7#8#9}}%
426 \def\XINT_keepunbr_loop_end-\expandafter\XINT_keepunbr_loop
427   \the\numexpr-#1-\xint_c_viii\expandafter.\XINT_keepunbr_loop_pickeight
428   {\csname XINT_keepunbr_end#1\endcsname}%
429 \long\expandafter\def\csname XINT_keepunbr_end1\endcsname
430   #1#2#3#4#5#6#7#8#9\xint_bye { #1#2#3#4#5#6#7#8}%
431 \long\expandafter\def\csname XINT_keepunbr_end2\endcsname
432   #1#2#3#4#5#6#7#8\xint_bye { #1#2#3#4#5#6#7}%
433 \long\expandafter\def\csname XINT_keepunbr_end3\endcsname
434   #1#2#3#4#5#6#7\xint_bye { #1#2#3#4#5#6}%
435 \long\expandafter\def\csname XINT_keepunbr_end4\endcsname
436   #1#2#3#4#5#6\xint_bye { #1#2#3#4#5}%
437 \long\expandafter\def\csname XINT_keepunbr_end5\endcsname
438   #1#2#3#4#5\xint_bye { #1#2#3#4}%
439 \long\expandafter\def\csname XINT_keepunbr_end6\endcsname
440   #1#2#3#4\xint_bye { #1#2#3}%
441 \long\expandafter\def\csname XINT_keepunbr_end7\endcsname
442   #1#2#3\xint_bye { #1#2}%
443 \long\expandafter\def\csname XINT_keepunbr_end8\endcsname
444   #1#2\xint_bye { #1}%

```

3.15 \xintTrim

First included in release 1.09m.

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

the tail. It thus removes the last $|i|$ items, i.e. it keeps the first $N-|i|$ items. For $|i| \geq N$, the empty list is returned.

`\xintTrimNoExpand` does not expand the L argument.

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

```

445 \def\xintTrim          {\romannumeral0\xinttrim }%
446 \def\xintTrimNoExpand {\romannumeral0\xinttrimnoexpand }%
447 \long\def\xinttrim #1#2{\expandafter\xint_trim_a\the\numexpr #1\expandafter.%
448                      \expandafter{\romannumeral`&&@#2}}%
449 \def\xinttrimnoexpand #1{\expandafter\xint_trim_a\the\numexpr #1.}%
450 \def\xint_trim_a #1%
451 {%
452   \xint_UDzerominusfork
453   #1-\xint_trim_trimnone
454   0#1\xint_trim_neg
455   0-{\xint_trim_pos #1}%
456   \krof
457 }%
458 \long\def\xint_trim_trimnone .#1{ #1}%
459 \long\def\xint_trim_neg #1.#2%
460 {%
461   \expandafter\xint_trim_neg_a\the\numexpr
462   #1-\numexpr\xint_length_loop
463   #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
464   \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
465   \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
466   .}{#2\xint_bye
467 }%
468 \def\xint_trim_neg_a #1%
469 {%
470   \xint_UDsignfork
471   #1{\expandafter\xint_keep_loop\the\numexpr-\xint_c_viii+}%
472   -\xint_trim_trimall
473   \krof
474 }%
475 \def\xint_trim_trimall#1{%
476 \def\xint_trim_trimall {\expandafter#1\xint_bye}%
477 }\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`.

```

478 \long\def\xint_trim_pos #1.#2%
479 {%
480   \expandafter\xint_trim_pos_done\expandafter\space
481   \romannumeral0\expandafter\xint_trim_loop\the\numexpr#1-\xint_c_ix.%
482   #2\xint:\xint:\xint:\xint:\xint:
483   \xint:\xint:\xint:\xint:\xint:
484   \xint_bye
485 }%

```



```

486 \def\XINT_trim_loop #1#2.%
487 {%
488     \xint_gob_til_minus#1\XINT_trim_finish-%
489     \expandafter\XINT_trim_loop\the\numexpr#1#2\XINT_trim_loop_trimnine
490 }%
491 \long\def\XINT_trim_loop_trimnine #1#2#3#4#5#6#7#8#9%
492 {%
493     \xint_gob_til_xint: #9\XINT_trim_toofew\xint:-\xint_c_ix.%
494 }%
495 \def\XINT_trim_toofew\xint:{*\xint_c_}%
496 \def\XINT_trim_finish#1{%
497 \def\XINT_trim_finish-%
498     \expandafter\XINT_trim_loop\the\numexpr-##1\XINT_trim_loop_trimnine
499 {%
500     \expandafter\expandafter\expandafter#1%
501     \csname xint_gobble_\romannumeral\numexpr\xint_c_ix-##1\endcsname
502 }}\XINT_trim_finish{ }%
503 \long\def\XINT_trim_pos_done #1\xint:#2\xint_bye {#1}%

```

3.16 \xintTrimUnbraced

1.2a. Modified in 1.2i like \xintTrim

```

504 \def\xintTrimUnbraced          {\romannumeral0\xinttrimunbraced }%
505 \def\xintTrimUnbracedNoExpand {\romannumeral0\xinttrimunbracednoexpand }%
506 \long\def\xinttrimunbraced #1#2%
507     {\expandafter\XINT_trimunbr_a\the\numexpr #1\expandafter.%
508         \expandafter{\romannumeral`&&@#2}}%
509 \def\xinttrimunbracednoexpand #1%
510     {\expandafter\XINT_trimunbr_a\the\numexpr #1.}%
511 \def\XINT_trimunbr_a #1%
512 {%
513     \xint_UDzerominusfork
514         #1-\XINT_trim_trimnone
515         0#1\XINT_trimunbr_neg
516         0-{\XINT_trim_pos #1}%
517     \krof
518 }%
519 \long\def\XINT_trimunbr_neg #1.#2%
520 {%
521     \expandafter\XINT_trimunbr_neg_a\the\numexpr
522     #1-\numexpr\XINT_length_loop
523     #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
524     \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
525     \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
526     .}{#2\xint_bye
527 }%
528 \def\XINT_trimunbr_neg_a #1%
529 {%
530     \xint_UDsignfork
531         #1{\expandafter\XINT_keeputnbr_loop\the\numexpr-\xint_c_viii+}%
532         -\XINT_trim_trimall
533     \krof

```

534 }%

3.17 \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.

```
535 \def\xintApply          {\romannumeral0\xintapply }%
536 \def\xintApplyNoExpand {\romannumeral0\xintapplynoexpand }%
537 \long\def\xintapply #1#2%
538 {%
539   \expandafter\XINT_apply\expandafter {\romannumeral`&&@#2}%
540   {#1}%
541 }%
542 \long\def\XINT_apply #1#2{\XINT_apply_loop_a {}{#2}#1\xint_bye }%
543 \long\def\xintapplynoexpand #1#2{\XINT_apply_loop_a {}{#1}#2\xint_bye }%
544 \long\def\XINT_apply_loop_a #1#2#3%
545 {%
546   \xint_bye #3\XINT_apply_end\xint_bye
547   \expandafter
548   \XINT_apply_loop_b
549   \expandafter {\romannumeral`&&@#2{#3}}{#1}{#2}%
550 }%
551 \long\def\XINT_apply_loop_b #1#2{\XINT_apply_loop_a {#2{#1}}}%
552 \long\def\XINT_apply_end\xint_bye\expandafter\XINT_apply_loop_b
553   \expandafter #1#2#3{ #2}%

```

3.18 \xintApply:x (not public)

Done for 1.4, 2020/01/27. For usage in the NumPy-like slicing routines.

Supposed to expand in an `\expanded` context, does not need to do any expansion of its second argument.

Uses techniques I had developed for 1.2i/1.2j `Keep`, `Trim`, `Length`, `LastItem` like macros, and I should revamp venerable `\xintApply` probably too. But the latter f-expandability (if it does not have `\expanded` at disposal) complicates significantly matters as it has to store material and release at very end.

Here it is simpler and I am doing it quickly as I really want to release 1.4. The `\xint:` token should not be located in looped over items. I could use something more exotic like the null char with catcode 3...

Could be however that picking one by one would be better for small number of items.

And anyhow for small number of items gain with respect to `\xintApply` is little if any (might even be a loss).

```
554 \long\def\xintApply:x #1#2%
555 {%
556   \XINT_apply:x_loop {#1}#2%
557   {\xint:\XINT_apply:x_loop_enda}{\xint:\XINT_apply:x_loop_endb}%
558   {\xint:\XINT_apply:x_loop_endc}{\xint:\XINT_apply:x_loop_endd}%
559   {\xint:\XINT_apply:x_loop_ende}{\xint:\XINT_apply:x_loop_endf}%
560   {\xint:\XINT_apply:x_loop_endg}{\xint:\XINT_apply:x_loop_endh}\xint_bye
561 }%
562 \long\def\XINT_apply:x_loop #1#2#3#4#5#6#7#8#9%

```

```

563 {%
564   \xint_gob_til_xint: #9\xint:
565   {#1{#2}}{#1{#3}}{#1{#4}}{#1{#5}}{#1{#6}}{#1{#7}}{#1{#8}}{#1{#9}}%
566   \XINT_apply:x_loop {#1}%
567 }%
568 \long\def\XINT_apply:x_loop_endh\xint: #1\xint_bye{}%
569 \long\def\XINT_apply:x_loop_endg\xint: #1#2\xint_bye{#1}%
570 \long\def\XINT_apply:x_loop_endf\xint: #1#2#3\xint_bye{#1{#2}}%
571 \long\def\XINT_apply:x_loop_ende\xint: #1#2#3#4\xint_bye{#1{#2}{#3}}%
572 \long\def\XINT_apply:x_loop_endd\xint: #1#2#3#4#5\xint_bye{#1{#2}{#3}{#4}}%
573 \long\def\XINT_apply:x_loop_endc\xint: #1#2#3#4#5#6\xint_bye{#1{#2}{#3}{#4}{#5}}%
574 \long\def\XINT_apply:x_loop_endb\xint: #1#2#3#4#5#6#7\xint_bye{#1{#2}{#3}{#4}{#5}{#6}}%
575 \long\def\XINT_apply:x_loop_enda\xint: #1#2#3#4#5#6#7#8\xint_bye{#1{#2}{#3}{#4}{#5}{#6}{#7}}%

```

3.19 \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.

```

576 \def\xintApplyUnbraced {\romannumeral0\xintapplyunbraced}%
577 \def\xintApplyUnbracedNoExpand {\romannumeral0\xintapplyunbracednoexpand}%
578 \long\def\xintapplyunbraced #1#2%
579 {%
580   \expandafter\XINT_applyunbr\expandafter {\romannumeral`&&@#2}%
581   {#1}%
582 }%
583 \long\def\XINT_applyunbr #1#2{\XINT_applyunbr_loop_a {}{#2}#1\xint_bye}%
584 \long\def\xintapplyunbracednoexpand #1#2%
585   {\XINT_applyunbr_loop_a {}{#1}#2\xint_bye}%
586 \long\def\XINT_applyunbr_loop_a #1#2#3%
587 {%
588   \xint_bye #3\XINT_applyunbr_end\xint_bye
589   \expandafter\XINT_applyunbr_loop_b
590   \expandafter {\romannumeral`&&@#2{#3}}{#1}{#2}%
591 }%
592 \long\def\XINT_applyunbr_loop_b #1#2{\XINT_applyunbr_loop_a {#2}#1}%
593 \long\def\XINT_applyunbr_end\xint_bye\expandafter\XINT_applyunbr_loop_b
594   \expandafter #1#2#3{ #2}%

```

3.20 \xintApplyUnbraced:x (not public)

Done for 1.4, 2020/01/27. For usage in the NumPy-like slicing routines.

The items should not contain `\xint:` and the applied macro should not contain `\empty`.

```

595 \long\def\xintApplyUnbraced:x #1#2%
596 {%
597   \XINT_applyunbraced:x_loop {#1}#2%
598   {\xint:\XINT_applyunbraced:x_loop_enda}{\xint:\XINT_applyunbraced:x_loop_endb}%
599   {\xint:\XINT_applyunbraced:x_loop_endc}{\xint:\XINT_applyunbraced:x_loop_endd}%
600   {\xint:\XINT_applyunbraced:x_loop_ende}{\xint:\XINT_applyunbraced:x_loop_endf}%
601   {\xint:\XINT_applyunbraced:x_loop_endg}{\xint:\XINT_applyunbraced:x_loop_endh}\xint_bye

```

```

602 }%
603 \long\def\XINT_applyunbraced:x_loop #1#2#3#4#5#6#7#8#9%
604 {%
605     \xint_gob_til_xint: #9\xint:
606         #1{#2}%
607     \empty#1{#3}%
608     \empty#1{#4}%
609     \empty#1{#5}%
610     \empty#1{#6}%
611     \empty#1{#7}%
612     \empty#1{#8}%
613     \empty#1{#9}%
614     \XINT_applyunbraced:x_loop {#1}%
615 }%
616 \long\def\XINT_applyunbraced:x_loop_endh\xint: #1\xint_bye{%
617 \long\def\XINT_applyunbraced:x_loop_endg\xint: #1\empty#2\xint_bye{#1}%
618 \long\def\XINT_applyunbraced:x_loop_endf\xint: #1\empty
619     #2\empty#3\xint_bye{#1#2}%
620 \long\def\XINT_applyunbraced:x_loop_ende\xint: #1\empty
621     #2\empty
622     #3\empty#4\xint_bye{#1#2#3}%
623 \long\def\XINT_applyunbraced:x_loop_endd\xint: #1\empty
624     #2\empty
625     #3\empty
626     #4\empty#5\xint_bye{#1#2#3#4}%
627 \long\def\XINT_applyunbraced:x_loop_endc\xint: #1\empty
628     #2\empty
629     #3\empty
630     #4\empty
631     #5\empty#6\xint_bye{#1#2#3#4#5}%
632 \long\def\XINT_applyunbraced:x_loop_endb\xint: #1\empty
633     #2\empty
634     #3\empty
635     #4\empty
636     #5\empty
637     #6\empty#7\xint_bye{#1#2#3#4#5#6}%
638 \long\def\XINT_applyunbraced:x_loop_enda\xint: #1\empty
639     #2\empty
640     #3\empty
641     #4\empty
642     #5\empty
643     #6\empty
644     #7\empty#8\xint_bye{#1#2#3#4#5#6#7}%

```

3.21 \xintSeq

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

```

645 \def\xintSeq {\romannumeral0\xintseq}%
646 \def\xintseq #1{\XINT_seq_chkopt #1\xint_bye}%
647 \def\XINT_seq_chkopt #1%
648 {%

```

```

649 \ifx [#1\expandafter\XINT_seq_opt
650 \else\expandafter\XINT_seq_noopt
651 \fi #1%
652 }%
653 \def\XINT_seq_noopt #1\xint_bye #2%
654 {%
655 \expandafter\XINT_seq\expandafter
656 {\the\numexpr#1\expandafter}\expandafter{\the\numexpr #2}%
657 }%
658 \def\XINT_seq #1#2%
659 {%
660 \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
661 \expandafter\xint_stop_atfirstoftwo
662 \or
663 \expandafter\XINT_seq_p
664 \else
665 \expandafter\XINT_seq_n
666 \fi
667 {#2}{#1}%
668 }%
669 \def\XINT_seq_p #1#2%
670 {%
671 \ifnum #1>#2
672 \expandafter\expandafter\expandafter\XINT_seq_p
673 \else
674 \expandafter\XINT_seq_e
675 \fi
676 \expandafter{\the\numexpr #1-\xint_c_i}{#2}{#1}%
677 }%
678 \def\XINT_seq_n #1#2%
679 {%
680 \ifnum #1<#2
681 \expandafter\expandafter\expandafter\XINT_seq_n
682 \else
683 \expandafter\XINT_seq_e
684 \fi
685 \expandafter{\the\numexpr #1+\xint_c_i}{#2}{#1}%
686 }%
687 \def\XINT_seq_e #1#2#3{ }%
688 \def\XINT_seq_opt [\xint_bye #1]#2#3%
689 {%
690 \expandafter\XINT_seqo\expandafter
691 {\the\numexpr #2\expandafter}\expandafter
692 {\the\numexpr #3\expandafter}\expandafter
693 {\the\numexpr #1}%
694 }%
695 \def\XINT_seqo #1#2%
696 {%
697 \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
698 \expandafter\XINT_seqo_a
699 \or
700 \expandafter\XINT_seqo_pa

```

```

701 \else
702 \expandafter\XINT_seqo_na
703 \fi
704 {#1}{#2}%
705 }%
706 \def\XINT_seqo_a #1#2#3{ {#1}}%
707 \def\XINT_seqo_o #1#2#3#4{ #4}%
708 \def\XINT_seqo_pa #1#2#3%
709 {%
710 \ifcase\ifnum #3=\xint_c_ 0\else\ifnum #3>\xint_c_ 1\else -1\fi\fi\space
711 \expandafter\XINT_seqo_o
712 \or
713 \expandafter\XINT_seqo_pb
714 \else
715 \xint_afterfi{\expandafter\space\xint_gobble_iv}%
716 \fi
717 {#1}{#2}{#3}{#1}}%
718 }%
719 \def\XINT_seqo_pb #1#2#3%
720 {%
721 \expandafter\XINT_seqo_pc\expandafter{\the\numexpr #1+#3}{#2}{#3}%
722 }%
723 \def\XINT_seqo_pc #1#2%
724 {%
725 \ifnum #1>#2
726 \expandafter\XINT_seqo_o
727 \else
728 \expandafter\XINT_seqo_pd
729 \fi
730 {#1}{#2}%
731 }%
732 \def\XINT_seqo_pd #1#2#3#4{\XINT_seqo_pb {#1}{#2}{#3}{#4{#1}}}%
733 \def\XINT_seqo_na #1#2#3%
734 {%
735 \ifcase\ifnum #3=\xint_c_ 0\else\ifnum #3>\xint_c_ 1\else -1\fi\fi\space
736 \expandafter\XINT_seqo_o
737 \or
738 \xint_afterfi{\expandafter\space\xint_gobble_iv}%
739 \else
740 \expandafter\XINT_seqo_nb
741 \fi
742 {#1}{#2}{#3}{#1}}%
743 }%
744 \def\XINT_seqo_nb #1#2#3%
745 {%
746 \expandafter\XINT_seqo_nc\expandafter{\the\numexpr #1+#3}{#2}{#3}%
747 }%
748 \def\XINT_seqo_nc #1#2%
749 {%
750 \ifnum #1<#2
751 \expandafter\XINT_seqo_o
752 \else

```

```

753      \expandafter\XINT_sequo_nd
754      \fi
755      {#1}{#2}%
756 }%
757 \def\XINT_sequo_nd #1#2#3#4{\XINT_sequo_nb {#1}{#2}{#3}{#4{#1}}}%

```

3.22 \xintloop, \xintbreakloop, \xintbreakloopanddo, \xintloopskiptonext

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

```

758 \long\def\xintloop #1#2\repeat {#1#2\xintloop_again\fi\xint_gobble_i {#1#2}}%
759 \long\def\xintloop_again\fi\xint_gobble_i #1{\fi
760      #1\xintloop_again\fi\xint_gobble_i {#1}}%
761 \long\def\xintbreakloop #1\xintloop_again\fi\xint_gobble_i #2{%
762 \long\def\xintbreakloopanddo #1#2\xintloop_again\fi\xint_gobble_i #3{#1}%
763 \long\def\xintloopskiptonext #1\xintloop_again\fi\xint_gobble_i #2{%
764      #2\xintloop_again\fi\xint_gobble_i {#2}}%

```

3.23 \xintilooop, \xintilooopindex, \xintbracedilooopindex, \xintouterilooopindex, \xintbracedouterilooopindex, \xintbreakilooop, \xintbreakilooopanddo, \xintilooopskiptonext, \xintilooopskipandredo

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

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

```

765 \def\xintilooop [#1+#2]{%
766      \expandafter\xintilooop_a\the\numexpr #1\expandafter.\the\numexpr #2.%
767 \long\def\xintilooop_a #1.#2.#3#4\repeat{%
768      #3#4\xintilooop_again\fi\xint_gobble_iii {#1}{#2}{#3#4}}%
769 \def\xintilooop_again\fi\xint_gobble_iii #1#2{%
770      \fi\expandafter\xintilooop_again_b\the\numexpr#1+#2.#2.%
771 \long\def\xintilooop_again_b #1.#2.#3{%
772      #3\xintilooop_again\fi\xint_gobble_iii {#1}{#2}{#3}}%
773 \long\def\xintbreakilooop #1\xintilooop_again\fi\xint_gobble_iii #2#3#4{%
774 \long\def\xintbreakilooopanddo
775      #1.#2\xintilooop_again\fi\xint_gobble_iii #3#4#5{#1}%
776 \long\def\xintilooopindex #1\xintilooop_again\fi\xint_gobble_iii #2%
777      {#2#1\xintilooop_again\fi\xint_gobble_iii {#2}}%
778 \long\def\xintbracedilooopindex #1\xintilooop_again\fi\xint_gobble_iii #2%
779      {{#2}#1\xintilooop_again\fi\xint_gobble_iii {#2}}%
780 \long\def\xintouterilooopindex #1\xintilooop_again
781      #2\xintilooop_again\fi\xint_gobble_iii #3%
782      {#3#1\xintilooop_again #2\xintilooop_again\fi\xint_gobble_iii {#3}}%
783 \long\def\xintbracedouterilooopindex #1\xintilooop_again
784      #2\xintilooop_again\fi\xint_gobble_iii #3%
785      {{#3}#1\xintilooop_again #2\xintilooop_again\fi\xint_gobble_iii {#3}}%
786 \long\def\xintilooopskiptonext #1\xintilooop_again\fi\xint_gobble_iii #2#3{%
787      \expandafter\xintilooop_again_b \the\numexpr#2+#3.#3.%
788 \long\def\xintilooopskipandredo #1\xintilooop_again\fi\xint_gobble_iii #2#3#4{%
789      #4\xintilooop_again\fi\xint_gobble_iii {#2}{#3}{#4}}%

```

3.24 \XINT_xflet

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

```

790 \def\XINT_xflet #1%
791 {%
792   \def\XINT_xflet_macro {#1}\XINT_xflet_zapsp
793 }%
794 \def\XINT_xflet_zapsp
795 {%
796   \expandafter\futurelet\expandafter\XINT_token
797   \expandafter\XINT_xflet_sp?\romannumeral`&&@%
798 }%
799 \def\XINT_xflet_sp?
800 {%
801   \ifx\XINT_token\XINT_sptoken
802     \expandafter\XINT_xflet_zapsp
803   \else\expandafter\XINT_xflet_zapspB
804   \fi
805 }%
806 \def\XINT_xflet_zapspB
807 {%
808   \expandafter\futurelet\expandafter\XINT_tokenB
809   \expandafter\XINT_xflet_spB?\romannumeral`&&@%
810 }%
811 \def\XINT_xflet_spB?
812 {%
813   \ifx\XINT_tokenB\XINT_sptoken
814     \expandafter\XINT_xflet_zapspB
815   \else\expandafter\XINT_xflet_eq?
816   \fi
817 }%
818 \def\XINT_xflet_eq?
819 {%
820   \ifx\XINT_token\XINT_tokenB
821     \expandafter\XINT_xflet_macro
822   \else\expandafter\XINT_xflet_zapsp
823   \fi
824 }%

```

3.25 \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.

```

825 \catcode`Z 3
826 \long\def\xintApplyInline #1#2%
827 {%
828   \long\expandafter\def\expandafter\XINT_inline_macro

```



```

829 \expandafter ##\expandafter 1\expandafter {#1{##1}}%
830 \XINT_xflet\XINT_inline_b #2Z% this Z has catcode 3
831 }%
832 \def\XINT_inline_b
833 {%
834     \ifx\XINT_token Z\expandafter\xint_gobble_i
835     \else\expandafter\XINT_inline_d\fi
836 }%
837 \long\def\XINT_inline_d #1%
838 {%
839     \long\def\XINT_item{#1}\XINT_xflet\XINT_inline_e
840 }%
841 \def\XINT_inline_e
842 {%
843     \ifx\XINT_token Z\expandafter\XINT_inline_w
844     \else\expandafter\XINT_inline_f\fi
845 }%
846 \def\XINT_inline_f
847 {%
848     \expandafter\XINT_inline_g\expandafter{\XINT_inline_macro {##1}}%
849 }%
850 \long\def\XINT_inline_g #1%
851 {%
852     \expandafter\XINT_inline_macro\XINT_item
853     \long\def\XINT_inline_macro ##1{#1}\XINT_inline_d
854 }%
855 \def\XINT_inline_w #1%
856 {%
857     \expandafter\XINT_inline_macro\XINT_item
858 }%

```

3.26 \xintFor, \xintFor*, \xintBreakFor, \xintBreakForAndDo

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

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

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

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

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

```

859 \def\XINT_tmpa #1#2{\ifnum #2<#1 \xint_afterfi {{#####2}}\fi}%
860 \def\XINT_tmppb #1#2{\ifnum #1<#2 \xint_afterfi {{#####2}}\fi}%
861 \def\XINT_tmppc #1%

```

```

862 {%
863   \expandafter\edef \csname XINT_for_left#1\endcsname
864     {\xintApplyUnbraced {\XINT_tmpa #1}{123456789}}%
865   \expandafter\edef \csname XINT_for_right#1\endcsname
866     {\xintApplyUnbraced {\XINT_tmpb #1}{123456789}}%
867 }%
868 \xintApplyInline \XINT_tmpc {123456789}%
869 \long\def\xintBreakFor #1Z{%
870 \long\def\xintBreakForAndDo #1#2Z{#1}%
871 \def\xintFor {\let\xintifForFirst\xint_firstoftwo
872   \let\xintifForLast\xint_secondoftwo
873   \futurelet\XINT_token\XINT_for_ifstar }%
874 \def\XINT_for_ifstar {\ifx\XINT_token*\expandafter\XINT_forx
875   \else\expandafter\XINT_for \fi }%
876 \catcode`U 3 % with numexpr
877 \catcode`V 3 % with xintfrac.sty (xint.sty not enough)
878 \catcode`D 3 % with dimexpr
879 \def\XINT_flet_zapsp
880 {%
881   \futurelet\XINT_token\XINT_flet_sp?
882 }%
883 \def\XINT_flet_sp?
884 {%
885   \ifx\XINT_token\XINT_sptoken
886     \xint_afterfi{\expandafter\XINT_flet_zapsp\romannumeral0}%
887   \else\expandafter\XINT_flet_macro
888   \fi
889 }%
890 \long\def\XINT_for #1#2in#3#4#5%
891 {%
892   \expandafter\XINT_toks\expandafter
893     {\expandafter\XINT_for_d\the\numexpr #2\relax {#5}}%
894   \def\XINT_flet_macro {\expandafter\XINT_for_forever?\space}%
895   \expandafter\XINT_flet_zapsp #3Z%
896 }%
897 \def\XINT_for_forever? #1Z%
898 {%
899   \ifx\XINT_token U\XINT_to_forever\fi
900   \ifx\XINT_token V\XINT_to_forever\fi
901   \ifx\XINT_token D\XINT_to_forever\fi
902   \expandafter\the\expandafter\XINT_toks\romannumeral0\xintcsvtolist {#1}Z%
903 }%
904 \def\XINT_to_forever\fi #1\xintcsvtolist #2{\fi \XINT_forever #2}%
905 \long\def\XINT_forx *#1#2in#3#4#5%
906 {%
907   \expandafter\XINT_toks\expandafter
908     {\expandafter\XINT_forx_d\the\numexpr #2\relax {#5}}%
909   \XINT_xflet\XINT_forx_forever? #3Z%
910 }%
911 \def\XINT_forx_forever?
912 {%
913   \ifx\XINT_token U\XINT_to_forxever\fi

```

```

914 \ifx\XINT_token V\XINT_to_forever\fi
915 \ifx\XINT_token D\XINT_to_forever\fi
916 \XINT_forx_empty?
917 }%
918 \def\XINT_to_forever\fi #1\XINT_forx_empty? {\fi \XINT_forever }%
919 \catcode`U 11
920 \catcode`D 11
921 \catcode`V 11
922 \def\XINT_forx_empty?
923 {%
924 \ifx\XINT_token Z\expandafter\xintBreakFor\fi
925 \the\XINT_toks
926 }%
927 \long\def\XINT_for_d #1#2#3%
928 {%
929 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
930 \XINT_toks {{#3}}%
931 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
932 \the\XINT_toks \csname XINT_for_right#1\endcsname }%
933 \XINT_toks {\XINT_x\let\xintifForFirst\xint_seconddoftwo
934 \let\xintifForLast\xint_seconddoftwo\XINT_for_d #1{#2}}%
935 \futurelet\XINT_token\XINT_for_last?
936 }%
937 \long\def\XINT_forx_d #1#2#3%
938 {%
939 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
940 \XINT_toks {{#3}}%
941 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
942 \the\XINT_toks \csname XINT_for_right#1\endcsname }%
943 \XINT_toks {\XINT_x\let\xintifForFirst\xint_seconddoftwo
944 \let\xintifForLast\xint_seconddoftwo\XINT_forx_d #1{#2}}%
945 \XINT_x\let\XINT_for_last?
946 }%
947 \def\XINT_for_last?
948 {%
949 \ifx\XINT_token Z\expandafter\XINT_for_last?yes\fi
950 \the\XINT_toks
951 }%
952 \def\XINT_for_last?yes
953 {%
954 \let\xintifForLast\xint_firstoftwo
955 \xintBreakForAndDo{\XINT_x\xint_gobble_i Z}%
956 }%

```

3.27 \XINT_forever, \xintintegers, \xintdimensions, \xintrationals

New with 1.09e. But this used inadvertently \xintiadd/\xintimul which have the unnecessary \xintnum overhead. Changed in 1.09f to use \xintiadd/\xintiimul which do not have this overhead. Also 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).

```

957 \catcode`U 3
958 \catcode`D 3
959 \catcode`V 3
960 \let\xintegers      U%
961 \let\xintintegers   U%
962 \let\xintdimensions D%
963 \let\xintrationals  V%
964 \def\XINT_forever #1%
965 {%
966   \expandafter\XINT_forever_a
967   \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi a\expandafter\endcsname
968   \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi i\expandafter\endcsname
969   \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi \endcsname
970 }%
971 \catcode`U 11
972 \catcode`D 11
973 \catcode`V 11
974 \def\XINT_?expr_Ua #1#2%
975   {\expandafter{\expandafter\numexpr\the\numexpr #1\expandafter\relax
976     \expandafter\relax\expandafter}%
977   \expandafter{\the\numexpr #2}}%
978 \def\XINT_?expr_Da #1#2%
979   {\expandafter{\expandafter\dimexpr\the\dimexpr #1\expandafter\relax
980     \expandafter s\expandafter p\expandafter\relax\expandafter}%
981   \expandafter{\the\dimexpr #2}}%
982 \catcode`Z 11
983 \def\XINT_?expr_Va #1#2%
984 {%
985   \expandafter\XINT_?expr_Vb\expandafter
986   {\romannumeral`&&\xintrowwithzeros{\xintZapSpacesB{#2}}}%
987   {\romannumeral`&&\xintrowwithzeros{\xintZapSpacesB{#1}}}%
988 }%
989 \catcode`Z 3
990 \def\XINT_?expr_Vb #1#2{\expandafter\XINT_?expr_Vc #2.#1}%
991 \def\XINT_?expr_Vc #1/#2.#3/#4.%
992 {%
993   \xintifEq {#2}{#4}%
994     {\XINT_?expr_Vf {#3}{#1}{#2}}%
995     {\expandafter\XINT_?expr_Vd\expandafter
996       {\romannumeral0\xintiimul {#2}{#4}}%
997       {\romannumeral0\xintiimul {#1}{#4}}%
998       {\romannumeral0\xintiimul {#2}{#3}}%
999     }%
1000 }%
1001 \def\XINT_?expr_Vd #1#2#3{\expandafter\XINT_?expr_Ve\expandafter {#2}{#3}{#1}}%
1002 \def\XINT_?expr_Ve #1#2{\expandafter\XINT_?expr_Vf\expandafter {#2}{#1}}%
1003 \def\XINT_?expr_Vf #1#2#3{{#2/#3}{0}{#1}{#2}{#3}}%
1004 \def\XINT_?expr_Ui {{\numexpr 1\relax}{1}}%
1005 \def\XINT_?expr_Di {{\dimexpr 0pt\relax}{65536}}%
1006 \def\XINT_?expr_Vi {{1/1}{0111}}%
1007 \def\XINT_?expr_U #1#2%
1008   {\expandafter{\expandafter\numexpr\the\numexpr #1+#2\relax\relax}{#2}}%

```

```

1009 \def\XINT_?expr_D #1#2%
1010   {\expandafter{\expandafter\dimexpr\the\numexpr #1+#2\relax sp\relax}{#2}}%
1011 \def\XINT_?expr_V #1#2{\XINT_?expr_Vx #2}%
1012 \def\XINT_?expr_Vx #1#2%
1013 {%
1014   \expandafter\XINT_?expr_Vy\expandafter
1015     {\romannumeral0\xinttiadd {#1}{#2}}{#2}%
1016 }%
1017 \def\XINT_?expr_Vy #1#2#3#4%
1018 {%
1019   \expandafter{\romannumeral0\xinttiadd {#3}{#1}/#4}{#1}{#2}{#3}{#4}%
1020 }%
1021 \def\XINT_forever_a #1#2#3#4%
1022 {%
1023   \ifx #4[\expandafter\XINT_forever_opt_a
1024     \else\expandafter\XINT_forever_b
1025   \fi #1#2#3#4%
1026 }%
1027 \def\XINT_forever_b #1#2#3Z{\expandafter\XINT_forever_c\the\XINT_toks #2#3}%
1028 \long\def\XINT_forever_c #1#2#3#4#5%
1029   {\expandafter\XINT_forever_d\expandafter #2#4#5{#3}Z}%
1030 \def\XINT_forever_opt_a #1#2#3[#4+#5]#6Z%
1031 {%
1032   \expandafter\expandafter\expandafter
1033     \XINT_forever_opt_c\expandafter\the\expandafter\XINT_toks
1034     \romannumeral`&&@#1{#4}{#5}#3%
1035 }%
1036 \long\def\XINT_forever_opt_c #1#2#3#4#5#6{\XINT_forever_d #2{#4}{#5}#6{#3}Z}%
1037 \long\def\XINT_forever_d #1#2#3#4#5%
1038 {%
1039   \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#5}%
1040   \XINT_toks {{#2}}%
1041   \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
1042     \the\XINT_toks \csname XINT_for_right#1\endcsname }%
1043   \XINT_x
1044   \let\xintifForFirst\xint_secondoftwo
1045   \let\xintifForLast\xint_secondoftwo
1046   \expandafter\XINT_forever_d\expandafter #1\romannumeral`&&@#4{#2}{#3}#4{#5}%
1047 }%

```

3.28 \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.

```

1048 \catcode`j 3
1049 \long\def\xintForpair #1#2#3in#4#5#6%
1050 {%
1051   \let\xintifForFirst\xint_firstoftwo

```

```

1052 \let\xintifForLast\xint_secondoftwo
1053 \XINT_toks {\XINT_forpair_d #2{#6}}%
1054 \expandafter\the\expandafter\XINT_toks #4jZ%
1055 }%
1056 \long\def\XINT_forpair_d #1#2#3(#4)#5%
1057 {%
1058 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
1059 \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
1060 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
1061 \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_i\endcsname}%
1062 \ifx #5j\expandafter\XINT_for_last?yes\fi
1063 \XINT_x
1064 \let\xintifForFirst\xint_secondoftwo
1065 \let\xintifForLast\xint_secondoftwo
1066 \XINT_forpair_d #1{#2}%
1067 }%
1068 \long\def\xintForthree #1#2#3in#4#5#6%
1069 {%
1070 \let\xintifForFirst\xint_firstoftwo
1071 \let\xintifForLast\xint_secondoftwo
1072 \XINT_toks {\XINT_forthree_d #2{#6}}%
1073 \expandafter\the\expandafter\XINT_toks #4jZ%
1074 }%
1075 \long\def\XINT_forthree_d #1#2#3(#4)#5%
1076 {%
1077 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
1078 \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
1079 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
1080 \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_ii\endcsname}%
1081 \ifx #5j\expandafter\XINT_for_last?yes\fi
1082 \XINT_x
1083 \let\xintifForFirst\xint_secondoftwo
1084 \let\xintifForLast\xint_secondoftwo
1085 \XINT_forthree_d #1{#2}%
1086 }%
1087 \long\def\xintForthree #1#2#3in#4#5#6%
1088 {%
1089 \let\xintifForFirst\xint_firstoftwo
1090 \let\xintifForLast\xint_secondoftwo
1091 \XINT_toks {\XINT_forthree_d #2{#6}}%
1092 \expandafter\the\expandafter\XINT_toks #4jZ%
1093 }%
1094 \long\def\XINT_forfour_d #1#2#3(#4)#5%
1095 {%
1096 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
1097 \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
1098 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
1099 \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_iii\endcsname}%
1100 \ifx #5j\expandafter\XINT_for_last?yes\fi
1101 \XINT_x
1102 \let\xintifForFirst\xint_secondoftwo
1103 \let\xintifForLast\xint_secondoftwo

```

```

1104 \XINT_forfour_d #1{#2}%
1105 }%
1106 \catcode`Z 11
1107 \catcode`j 11

```

3.29 \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.

```

1108 \def\xintAssign{\def\XINT_flet_macro {\XINT_assign_fork}\XINT_flet_zapsp }%
1109 \def\XINT_assign_fork
1110 {%
1111     \let\XINT_assign_def\def
1112     \ifx\XINT_token[\expandafter\XINT_assign_opt
1113         \else\expandafter\XINT_assign_a
1114     \fi
1115 }%
1116 \def\XINT_assign_opt [#1]%
1117 {%
1118     \ifcsname #1def\endcsname
1119         \expandafter\let\expandafter\XINT_assign_def \csname #1def\endcsname
1120     \else
1121         \expandafter\let\expandafter\XINT_assign_def \csname xint#1def\endcsname
1122     \fi
1123     \XINT_assign_a
1124 }%
1125 \long\def\XINT_assign_a #1\to
1126 {%
1127     \def\XINT_flet_macro{\XINT_assign_b}%
1128     \expandafter\XINT_flet_zapsp\romannumeral`&&@#1\xint:\to
1129 }%
1130 \long\def\XINT_assign_b
1131 {%
1132     \ifx\XINT_token\bgroup
1133         \expandafter\XINT_assign_c
1134     \else\expandafter\XINT_assign_f
1135     \fi
1136 }%
1137 \long\def\XINT_assign_f #1\xint:\to #2%
1138 {%
1139     \XINT_assign_def #2{#1}%
1140 }%
1141 \long\def\XINT_assign_c #1%
1142 {%
1143     \def\xint_temp {#1}%
1144     \ifx\xint_temp\xint_bracedstopper
1145         \expandafter\XINT_assign_e
1146     \else
1147         \expandafter\XINT_assign_d

```

```

1148 \fi
1149 }%
1150 \long\def\XINT_assign_d #1\to #2%
1151 {%
1152 \expandafter\XINT_assign_def\expandafter #2\expandafter{\xint_temp}%
1153 \XINT_assign_c #1\to
1154 }%
1155 \def\XINT_assign_e #1\to {%
1156 \def\xintRelaxArray #1%
1157 {%
1158 \edef\XINT_restoreescapechar {\escapechar\the\escapechar\relax}%
1159 \escapechar -1
1160 \expandafter\def\expandafter\xint_arrayname\expandafter {\string #1}%
1161 \XINT_restoreescapechar
1162 \xintilooop [\csname\xint_arrayname 0\endcsname+-1]
1163 \global
1164 \expandafter\let\csname\xint_arrayname\xintilooopindex\endcsname\relax
1165 \ifnum \xintilooopindex > \xint_c_
1166 \repeat
1167 \global\expandafter\let\csname\xint_arrayname 00\endcsname\relax
1168 \global\let #1\relax
1169 }%
1170 \def\xintAssignArray{\def\XINT_flet_macro {\XINT_assignarray_fork}%
1171 \XINT_flet_zapsp }%
1172 \def\XINT_assignarray_fork
1173 {%
1174 \let\XINT_assignarray_def\def
1175 \ifx\XINT_token[\expandafter\XINT_assignarray_opt
1176 \else\expandafter\XINT_assignarray
1177 \fi
1178 }%
1179 \def\XINT_assignarray_opt [#1]%
1180 {%
1181 \ifcsname #1def\endcsname
1182 \expandafter\let\expandafter\XINT_assignarray_def \csname #1def\endcsname
1183 \else
1184 \expandafter\let\expandafter\XINT_assignarray_def
1185 \csname xint#1def\endcsname
1186 \fi
1187 \XINT_assignarray
1188 }%
1189 \long\def\XINT_assignarray #1\to #2%
1190 {%
1191 \edef\XINT_restoreescapechar {\escapechar\the\escapechar\relax }%
1192 \escapechar -1
1193 \expandafter\def\expandafter\xint_arrayname\expandafter {\string #2}%
1194 \XINT_restoreescapechar
1195 \def\xint_itemcount {0}%
1196 \expandafter\XINT_assignarray_loop \romannumeral`&&@#1\xint:
1197 \csname\xint_arrayname 00\expandafter\endcsname
1198 \csname\xint_arrayname 0\expandafter\endcsname
1199 \expandafter {\xint_arrayname}#2%

```



```

1200 }%
1201 \long\def\XINT_assignarray_loop #1%
1202 {%
1203   \def\xint_temp {#1}%
1204   \ifx\xint_temp\xint_bracedstopper
1205     \expandafter\def\csname\xint_arrayname 0\expandafter\endcsname
1206       \expandafter{\the\numexpr\xint_itemcount}%
1207     \expandafter\expandafter\expandafter\XINT_assignarray_end
1208   \else
1209     \expandafter\def\expandafter\xint_itemcount\expandafter
1210       {\the\numexpr\xint_itemcount+\xint_c_i}%
1211     \expandafter\XINT_assignarray_def
1212       \csname\xint_arrayname\xint_itemcount\expandafter\endcsname
1213       \expandafter{\xint_temp }%
1214     \expandafter\XINT_assignarray_loop
1215   \fi
1216 }%
1217 \def\XINT_assignarray_end #1#2#3#4%
1218 {%
1219   \def #4##1%
1220   {%
1221     \romannumeral0\expandafter #1\expandafter{\the\numexpr ##1}%
1222   }%
1223   \def #1##1%
1224   {%
1225     \ifnum ##1<\xint_c_
1226       \xint_afterfi{\XINT_expandableerror{Array index negative: 0 > ##1} }%
1227     \else
1228       \xint_afterfi {%
1229         \ifnum ##1>#2
1230           \xint_afterfi
1231             {\XINT_expandableerror{Array index beyond range: ##1 > #2} }%
1232           \else\xint_afterfi
1233             {\expandafter\expandafter\expandafter\space\csname #3##1\endcsname}%
1234           \fi}%
1235       \fi
1236   }%
1237 }%
1238 \let\xintDigitsOf\xintAssignArray

```

3.30 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 [\xintNthEltp:f:csv](#), with the exception of the first item, as the list is f-expanded. Perhaps [\xintNthEltpy:f:csv](#) should remove a leading space if present in the picked item; anyway, there are no spaces for the lists handled internally by the Python slicer of [xintexpr](#), except the «nil» object currently represented by exactly one space.

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

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

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

3.30.1 [\xintLength:f:csv](#)

1.2g. Redone for 1.2j. Contrarily to [\xintLength](#) from [xintkernel.sty](#), this one expands its argument.

```

1239 \def\xintLength:f:csv {\romannumeral0\xintlenth:f:csv}%
1240 \def\xintlenth:f:csv #1%
1241 {\long\def\xintlenth:f:csv ##1{%
1242   \expandafter#1\the\numexpr\expandafter\XINT_length:f:csv_a
1243   \romannumeral`&&@##1\xint:,\xint:,\xint:,\xint:,%
1244   \xint:,\xint:,\xint:,\xint:,\xint:,%
1245   \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
1246   \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
1247   \relax
1248 }}\xintlenth:f:csv { }%
```

Must first check if empty list.

```

1249 \long\def\XINT_length:f:csv_a #1%
1250 {%
1251     \xint_gob_til_xint: #1\xint_c_\xint_bye\xint:%
1252     \XINT_length:f:csv_loop #1%
1253 }%
1254 \long\def\XINT_length:f:csv_loop #1,#2,#3,#4,#5,#6,#7,#8,#9,%
1255 {%
1256     \xint_gob_til_xint: #9\XINT_length:f:csv_finish\xint:%
1257     \xint_c_ix+\XINT_length:f:csv_loop
1258 }%
1259 \def\XINT_length:f:csv_finish\xint:\xint_c_ix+\XINT_length:f:csv_loop
1260     #1,#2,#3,#4,#5,#6,#7,#8,#9,{#9\xint_bye}%

```

3.30.2 \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.

```

1261 \def\xintLengthUpTo:f:csv {\romannumeral0\xintlenghupto:f:csv}%
1262 \long\def\xintlenghupto:f:csv #1#2%
1263 {%
1264     \expandafter\XINT_lenghupto:f:csv_a
1265     \the\numexpr#1\expandafter.%
1266     \romannumeral`&&@#2\xint:,\xint:,\xint:,\xint:,%
1267     \xint:,\xint:,\xint:,\xint:,%
1268     \xint_c_viii,\xint_c_vii,\xint_c_vi,\xint_c_v,%
1269     \xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye.%
1270 }%

```

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

```

1271 \long\def\XINT_lenghupto:f:csv_a #1.#2%
1272 {%
1273     \xint_gob_til_xint: #2\XINT_lenghupto:f:csv_empty\xint:%
1274     \XINT_lenghupto:f:csv_loop_b #1.#2%
1275 }%
1276 \def\XINT_lenghupto:f:csv_empty\xint:%
1277     \XINT_lenghupto:f:csv_loop_b #1.#2\xint_bye.{ #1}%
1278 \def\XINT_lenghupto:f:csv_loop_a #1%
1279 {%
1280     \xint_UDsignfork
1281     #1\XINT_lenghupto:f:csv_gt
1282     -\XINT_lenghupto:f:csv_loop_b
1283     \krof #1%
1284 }%
1285 \long\def\XINT_lenghupto:f:csv_gt #1\xint_bye.{-0}%
1286 \long\def\XINT_lenghupto:f:csv_loop_b #1.#2,#3,#4,#5,#6,#7,#8,#9,%
1287 {%
1288     \xint_gob_til_xint: #9\XINT_lenghupto:f:csv_finish_a\xint:%
1289     \expandafter\XINT_lenghupto:f:csv_loop_a\the\numexpr #1-\xint_c_viii.%

```

```

1290 }%
1291 \def\XINT_lengthupto:f:csv_finish_a\xint:
1292   \expandafter\XINT_lengthupto:f:csv_loop_a
1293   \the\numexpr #1-\xint_c_viii.#2,#3,#4,#5,#6,#7,#8,#9,%
1294 {%
1295   \expandafter\XINT_lengthupto:f:csv_finish_b\the\numexpr #1-#9\xint_bye
1296 }%
1297 \def\XINT_lengthupto:f:csv_finish_b #1#2.%
1298 {%
1299   \xint_UDsignfork
1300   #1{-0}%
1301   -{ #1#2}%
1302   \krof
1303 }%

```

3.30.3 \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.

```

1304 \def\xintKeep:f:csv {\romannumeral0\xintkeep:f:csv }%
1305 \long\def\xintkeep:f:csv #1#2%
1306 {%
1307   \expandafter\xint_stop_aftergobble
1308   \romannumeral0\expandafter\XINT_keep:f:csv_a
1309   \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
1310 }%
1311 \def\XINT_keep:f:csv_a #1%
1312 {%
1313   \xint_UDzerominusfork
1314   #1-\XINT_keep:f:csv_keeptime
1315   0#1\XINT_keep:f:csv_neg
1316   0-{\XINT_keep:f:csv_pos #1}%
1317   \krof
1318 }%
1319 \long\def\XINT_keep:f:csv_keeptime .#1{,%
1320 \long\def\XINT_keep:f:csv_neg #1.#2%
1321 {%
1322   \expandafter\XINT_keep:f:csv_neg_done\expandafter,%
1323   \romannumeral0%
1324   \expandafter\XINT_keep:f:csv_neg_a\the\numexpr
1325   #1-\numexpr\XINT_length:f:csv_a
1326   #2\xint:,\xint:,\xint:,\xint:,%
1327   \xint:,\xint:,\xint:,\xint:,\xint:,%
1328   \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
1329   \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
1330   .#2\xint_bye
1331 }%
1332 \def\XINT_keep:f:csv_neg_a #1%
1333 {%

```

```

1334 \xint_UDsignfork
1335 #1{\expandafter\XINT_keep:f:csv_trimloop\the\numexpr-\xint_c_ix+}%
1336 -\XINT_keep:f:csv_keepall
1337 \krof
1338 }%
1339 \def\XINT_keep:f:csv_keepall #1.{ }%
1340 \long\def\XINT_keep:f:csv_neg_done #1\xint_bye{#1}%
1341 \def\XINT_keep:f:csv_trimloop #1#2.%
1342 {%
1343 \xint_gob_til_minus#1\XINT_keep:f:csv_trimloop_finish-%
1344 \expandafter\XINT_keep:f:csv_trimloop
1345 \the\numexpr#1#2-\xint_c_ix\expandafter.\XINT_keep:f:csv_trimloop_trimnine
1346 }%
1347 \long\def\XINT_keep:f:csv_trimloop_trimnine #1,#2,#3,#4,#5,#6,#7,#8,#9,{}%
1348 \def\XINT_keep:f:csv_trimloop_finish-%
1349 \expandafter\XINT_keep:f:csv_trimloop
1350 \the\numexpr-#1-\xint_c_ix\expandafter.\XINT_keep:f:csv_trimloop_trimnine
1351 {\csname XINT_trim:f:csv_finish#1\endcsname}%
1352 \long\def\XINT_keep:f:csv_pos #1.#2%
1353 {%
1354 \expandafter\XINT_keep:f:csv_pos_fork
1355 \romannumeral0\XINT_lengthupto:f:csv_a
1356 #1.#2\xint:,\xint:,\xint:,\xint:,%
1357 \xint:,\xint:,\xint:,\xint:,%
1358 \xint_c_viii,\xint_c_vii,\xint_c_vi,\xint_c_v,%
1359 \xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye.%
1360 .#1.{ }#2\xint_bye%
1361 }%
1362 \def\XINT_keep:f:csv_pos_fork #1#2.%
1363 {%
1364 \xint_UDsignfork
1365 #1{\expandafter\XINT_keep:f:csv_loop\the\numexpr-\xint_c_viii+}%
1366 -\XINT_keep:f:csv_pos_keepall
1367 \krof
1368 }%
1369 \long\def\XINT_keep:f:csv_pos_keepall #1.#2#3\xint_bye{,#3}%
1370 \def\XINT_keep:f:csv_loop #1#2.%
1371 {%
1372 \xint_gob_til_minus#1\XINT_keep:f:csv_loop_end-%
1373 \expandafter\XINT_keep:f:csv_loop
1374 \the\numexpr#1#2-\xint_c_viii\expandafter.\XINT_keep:f:csv_loop_pickeight
1375 }%
1376 \long\def\XINT_keep:f:csv_loop_pickeight
1377 #1#2,#3,#4,#5,#6,#7,#8,#9,{#{#1,#2,#3,#4,#5,#6,#7,#8,#9}}%
1378 \def\XINT_keep:f:csv_loop_end-\expandafter\XINT_keep:f:csv_loop
1379 \the\numexpr-#1-\xint_c_viii\expandafter.\XINT_keep:f:csv_loop_pickeight
1380 {\csname XINT_keep:f:csv_end#1\endcsname}%
1381 \long\expandafter\def\csname XINT_keep:f:csv_end1\endcsname
1382 #1#2,#3,#4,#5,#6,#7,#8,#9\xint_bye {#1,#2,#3,#4,#5,#6,#7,#8}%
1383 \long\expandafter\def\csname XINT_keep:f:csv_end2\endcsname
1384 #1#2,#3,#4,#5,#6,#7,#8\xint_bye {#1,#2,#3,#4,#5,#6,#7}%
1385 \long\expandafter\def\csname XINT_keep:f:csv_end3\endcsname

```

```

1386 #1#2,#3,#4,#5,#6,#7\xint_bye {#1,#2,#3,#4,#5,#6}%
1387 \long\expandafter\def\csname XINT_keep:f:csv_end4\endcsname
1388 #1#2,#3,#4,#5,#6\xint_bye {#1,#2,#3,#4,#5}%
1389 \long\expandafter\def\csname XINT_keep:f:csv_end5\endcsname
1390 #1#2,#3,#4,#5\xint_bye {#1,#2,#3,#4}%
1391 \long\expandafter\def\csname XINT_keep:f:csv_end6\endcsname
1392 #1#2,#3,#4\xint_bye {#1,#2,#3}%
1393 \long\expandafter\def\csname XINT_keep:f:csv_end7\endcsname
1394 #1#2,#3\xint_bye {#1,#2}%
1395 \long\expandafter\def\csname XINT_keep:f:csv_end8\endcsname
1396 #1#2\xint_bye {#1}%

```

3.30.4 \xintTrim:f:csv

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

```

1397 \def\xintTrim:f:csv {\romannumeral0\xinttrim:f:csv}%
1398 \long\def\xinttrim:f:csv #1#2%
1399 {%
1400   \expandafter\xint_stop_aftergobble
1401   \romannumeral0\expandafter\XINT_trim:f:csv_a
1402   \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
1403 }%
1404 \def\XINT_trim:f:csv_a #1%
1405 {%
1406   \xint_UDzerominusfork
1407   #1-\XINT_trim:f:csv_trimnone
1408   0#1\XINT_trim:f:csv_neg
1409   0-{\XINT_trim:f:csv_pos #1}%
1410   \krof
1411 }%
1412 \long\def\XINT_trim:f:csv_trimnone .#1{,#1}%
1413 \long\def\XINT_trim:f:csv_neg #1.#2%
1414 {%
1415   \expandafter\XINT_trim:f:csv_neg_a\the\numexpr
1416   #1-\numexpr\XINT_length:f:csv_a
1417   #2\xint:,\xint:,\xint:,\xint:,%
1418   \xint:,\xint:,\xint:,\xint:,\xint:,%
1419   \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
1420   \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
1421   .{ }#2\xint_bye
1422 }%
1423 \def\XINT_trim:f:csv_neg_a #1%
1424 {%
1425   \xint_UDsignfork
1426   #1f\expandafter\XINT_keep:f:csv_loop\the\numexpr-\xint_c_viii+}%
1427   -\XINT_trim:f:csv_trimall
1428   \krof
1429 }%
1430 \def\XINT_trim:f:csv_trimall {\expandafter,\xint_bye}%
1431 \long\def\XINT_trim:f:csv_pos #1.#2%
1432 {%
1433   \expandafter\XINT_trim:f:csv_pos_done\expandafter,%

```

```

1434 \romannumeral0%
1435 \expandafter\XINT_trim:f:csv_loop\the\numexpr#1-\xint_c_ix.%
1436 #2\xint:,\xint:,\xint:,\xint:,\xint:,%
1437 \xint:,\xint:,\xint:,\xint:,\xint:\xint_bye
1438 }%
1439 \def\XINT_trim:f:csv_loop #1#2.%
1440 {%
1441 \xint_gob_til_minus#1\XINT_trim:f:csv_finish-%
1442 \expandafter\XINT_trim:f:csv_loop\the\numexpr#1#2\XINT_trim:f:csv_loop_trimnine
1443 }%
1444 \long\def\XINT_trim:f:csv_loop_trimnine #1,#2,#3,#4,#5,#6,#7,#8,#9,%
1445 {%
1446 \xint_gob_til_xint: #9\XINT_trim:f:csv_toofew\xint:-\xint_c_ix.%
1447 }%
1448 \def\XINT_trim:f:csv_toofew\xint:{*\xint_c_}%
1449 \def\XINT_trim:f:csv_finish-%
1450 \expandafter\XINT_trim:f:csv_loop\the\numexpr-#1\XINT_trim:f:csv_loop_trimnine
1451 {%
1452 \csname XINT_trim:f:csv_finish#1\endcsname
1453 }%
1454 \long\expandafter\def\csname XINT_trim:f:csv_finish1\endcsname
1455 #1,#2,#3,#4,#5,#6,#7,#8,{ }%
1456 \long\expandafter\def\csname XINT_trim:f:csv_finish2\endcsname
1457 #1,#2,#3,#4,#5,#6,#7,{ }%
1458 \long\expandafter\def\csname XINT_trim:f:csv_finish3\endcsname
1459 #1,#2,#3,#4,#5,#6,{ }%
1460 \long\expandafter\def\csname XINT_trim:f:csv_finish4\endcsname
1461 #1,#2,#3,#4,#5,{ }%
1462 \long\expandafter\def\csname XINT_trim:f:csv_finish5\endcsname
1463 #1,#2,#3,#4,{ }%
1464 \long\expandafter\def\csname XINT_trim:f:csv_finish6\endcsname
1465 #1,#2,#3,{ }%
1466 \long\expandafter\def\csname XINT_trim:f:csv_finish7\endcsname
1467 #1,#2,{ }%
1468 \long\expandafter\def\csname XINT_trim:f:csv_finish8\endcsname
1469 #1,{ }%
1470 \expandafter\let\csname XINT_trim:f:csv_finish9\endcsname\space
1471 \long\def\XINT_trim:f:csv_pos_done #1\xint:#2\xint_bye{#1}%

```

3.30.5 \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.

```

1472 \def\xintNthEltPy:f:csv {\romannumeral0\xintntheltpy:f:csv }%
1473 \long\def\xintntheltpy:f:csv #1#2%
1474 {%
1475 \expandafter\XINT_nthelt:f:csv_a
1476 \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
1477 }%
1478 \def\XINT_nthelt:f:csv_a #1%
1479 {%
1480 \xint_UDsignfork

```

```

1481      #1\XINT_nthelt:f:csv_neg
1482      -\XINT_nthelt:f:csv_pos
1483      \krof #1%
1484 }%
1485 \long\def\XINT_nthelt:f:csv_neg -#1.#2%
1486 {%
1487     \expandafter\XINT_nthelt:f:csv_neg_fork
1488     \the\numexpr\XINT_length:f:csv_a
1489     #2\xint:,\xint:,\xint:,\xint:,%
1490     \xint:,\xint:,\xint:,\xint:,\xint:,%
1491     \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
1492     \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
1493     -#1.#2,\xint_bye
1494 }%
1495 \def\XINT_nthelt:f:csv_neg_fork #1%
1496 {%
1497     \if#1-\expandafter\xint_stop_afterbye\fi
1498     \expandafter\XINT_nthelt:f:csv_neg_done
1499     \romannumeral0%
1500     \expandafter\XINT_keep:f:csv_trimloop\the\numexpr-\xint_c_ix+#1%
1501 }%
1502 \long\def\XINT_nthelt:f:csv_neg_done#1,#2\xint_bye{ #1}%
1503 \long\def\XINT_nthelt:f:csv_pos #1.#2%
1504 {%
1505     \expandafter\XINT_nthelt:f:csv_pos_done
1506     \romannumeral0%
1507     \expandafter\XINT_trim:f:csv_loop\the\numexpr#1-\xint_c_ix.%
1508     #2\xint:,\xint:,\xint:,\xint:,\xint:,%
1509     \xint:,\xint:,\xint:,\xint:,\xint:,\xint_bye
1510 }%
1511 \def\XINT_nthelt:f:csv_pos_done #1{%
1512 \long\def\XINT_nthelt:f:csv_pos_done ##1,##2\xint_bye{%
1513     \xint_gob_til_xint:##1\XINT_nthelt:f:csv_pos_cleanup\xint:#1##1}%
1514 }\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...

```

1515 \xint_firstofone{\long\def\XINT_nthelt:f:csv_pos_cleanup\xint:} %
1516     #1\xint:{ #1}%

```

3.30.6 `\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.

```

1517 \def\xintReverse:f:csv {\romannumeral0\xintreverse:f:csv}%
1518 \long\def\xintreverse:f:csv #1%
1519 {%
1520     \expandafter\XINT_reverse:f:csv_loop

```



```

1521 \expandafter{\expandafter}\romannumeral`&&@#1,%
1522 \xint:,%
1523 \xint_bye,\xint_bye,\xint_bye,\xint_bye,%
1524 \xint_bye,\xint_bye,\xint_bye,\xint_bye,%
1525 \xint:
1526 }%
1527 \long\def\XINT_reverse:f:csv_loop #1#2,#3,#4,#5,#6,#7,#8,#9,%
1528 {%
1529 \xint_bye #9\XINT_reverse:f:csv_cleanup\xint_bye
1530 \XINT_reverse:f:csv_loop {,#9,#8,#7,#6,#5,#4,#3,#2#1}%
1531 }%
1532 \long\def\XINT_reverse:f:csv_cleanup\xint_bye\XINT_reverse:f:csv_loop #1#2\xint:
1533 {%
1534 \XINT_reverse:f:csv_finish #1%
1535 }%
1536 \long\def\XINT_reverse:f:csv_finish #1\xint:,{ }%

```

3.30.7 \xintFirstItem:f:csv

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

```

1537 \def\xintFirstItem:f:csv {\romannumeral0\xintfirstitem:f:csv}%
1538 \long\def\xintfirstitem:f:csv #1%
1539 {%
1540 \expandafter\XINT_first:f:csv_a\romannumeral`&&@#1,\xint_bye
1541 }%
1542 \long\def\XINT_first:f:csv_a #1,#2\xint_bye{ #1}%

```

3.30.8 \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.

```

1543 \def\xintLastItem:f:csv {\romannumeral0\xintlastitem:f:csv}%
1544 \long\def\xintlastitem:f:csv #1%
1545 {%
1546 \expandafter\XINT_last:f:csv_loop\expandafter{\expandafter}\expandafter.%
1547 \romannumeral`&&@#1,%
1548 \xint:\XINT_last_loop_enda,\xint:\XINT_last_loop_endb,%
1549 \xint:\XINT_last_loop_endc,\xint:\XINT_last_loop_endd,%
1550 \xint:\XINT_last_loop_ende,\xint:\XINT_last_loop_endf,%
1551 \xint:\XINT_last_loop_endg,\xint:\XINT_last_loop_endh,\xint_bye
1552 }%
1553 \long\def\XINT_last:f:csv_loop #1.#2,#3,#4,#5,#6,#7,#8,#9,%
1554 {%
1555 \xint_gob_til_xint: #9%
1556 {#8}{#7}{#6}{#5}{#4}{#3}{#2}{#1}\xint:
1557 \XINT_last:f:csv_loop {#9}.%
1558 }%

```

3.30.9 \xintKeep:x:csv

Added to xintexpr at 1.2j.

But data model changed at 1.4, this macro moved to xinttools, not part of publicly supported macros, may be removed at any time.

This macro is used only with positive first argument.

```

1559 \def\xintKeep:x:csv #1#2%
1560 {%
1561     \expandafter\xint_gobble_i
1562     \romannumeral0\expandafter\XINT_keep:x:csv_pos
1563     \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
1564 }%
1565 \def\XINT_keep:x:csv_pos #1.#2%
1566 {%
1567     \expandafter\XINT_keep:x:csv_loop\the\numexpr#1-\xint_c_viii.%
1568     #2\xint_Bye,\xint_Bye,\xint_Bye,\xint_Bye,%
1569     \xint_Bye,\xint_Bye,\xint_Bye,\xint_Bye,\xint_bye
1570 }%
1571 \def\XINT_keep:x:csv_loop #1%
1572 {%
1573     \xint_gob_til_minus#1\XINT_keep:x:csv_finish-%
1574     \XINT_keep:x:csv_loop_pickeight #1%
1575 }%
1576 \def\XINT_keep:x:csv_loop_pickeight #1.#2,#3,#4,#5,#6,#7,#8,#9,%
1577 {%
1578     ,#2,#3,#4,#5,#6,#7,#8,#9%
1579     \expandafter\XINT_keep:x:csv_loop\the\numexpr#1-\xint_c_viii.%
1580 }%
1581 \def\XINT_keep:x:csv_finish-\XINT_keep:x:csv_loop_pickeight -#1.%
1582 {%
1583     \csname XINT_keep:x:csv_finish#1\endcsname
1584 }%
1585 \expandafter\def\csname XINT_keep:x:csv_finish1\endcsname
1586     #1,#2,#3,#4,#5,#6,#7,{,#1,#2,#3,#4,#5,#6,#7\xint_Bye}%
1587 \expandafter\def\csname XINT_keep:x:csv_finish2\endcsname
1588     #1,#2,#3,#4,#5,#6,{,#1,#2,#3,#4,#5,#6\xint_Bye}%
1589 \expandafter\def\csname XINT_keep:x:csv_finish3\endcsname
1590     #1,#2,#3,#4,#5,{,#1,#2,#3,#4,#5\xint_Bye}%
1591 \expandafter\def\csname XINT_keep:x:csv_finish4\endcsname
1592     #1,#2,#3,#4,{,#1,#2,#3,#4\xint_Bye}%
1593 \expandafter\def\csname XINT_keep:x:csv_finish5\endcsname
1594     #1,#2,#3,{,#1,#2,#3\xint_Bye}%
1595 \expandafter\def\csname XINT_keep:x:csv_finish6\endcsname
1596     #1,#2,{,#1,#2\xint_Bye}%
1597 \expandafter\def\csname XINT_keep:x:csv_finish7\endcsname
1598     #1,{,#1\xint_Bye}%
1599 \expandafter\let\csname XINT_keep:x:csv_finish8\endcsname\xint_Bye

```

3.30.10 Public names for the undocumented csv macros: \xintCSVLength, \xintCSVKeep, \xintCSVKeepx, \xintCSVTrim, \xintCSVNthEltPy, \xintCSVReverse, \xintCSVFirstItem, \xintCSVLastItem

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

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.

```
1600 \let\xintCSVLength \xintLength:f:csv
1601 \let\xintCSVKeep \xintKeep:f:csv
1602 \let\xintCSVKeepx \xintKeep:x:csv
1603 \let\xintCSVTrim \xintTrim:f:csv
1604 \let\xintCSVNthEltPy \xintNthEltPy:f:csv
1605 \let\xintCSVReverse \xintReverse:f:csv
1606 \let\xintCSVFirstItem\xintFirstItem:f:csv
1607 \let\xintCSVLastItem \xintLastItem:f:csv
1608 \let\XINT_tmpa\relax \let\XINT_tmpb\relax \let\XINT_tmppc\relax
1609 \XINT_restorecatcodes_endinput%
```

4 Package [xintcore](#) implementation

.1	Catcodes, ε -T _E X and reload detection . . .	60	.25	\XINT_zeroes_forviii	73
.2	Package identification	61	.26	\XINT_sepbyviii_Z	73
.3	(WIP!) Error conditions and exceptions . . .	61	.27	\XINT_sepbyviii_andcount	74
.4	Counts for holding needed constants . . .	63	.28	\XINT_rsepbyviii	74
Routines handling integers as lists of token digits 64			.29	\XINT_sepandrev	75
.5	\XINT_cuz_small	64	.30	\XINT_sepandrev_andcount	75
.6	\xintNum, \xintiNum	64	.31	\XINT_rev_nounsep	76
.7	\xintiiSgn	65	.32	\XINT_unrevbyviii	76
.8	\xintiiOpp	66	Core arithmetic		76
.9	\xintiiAbs	66	.33	\xintiiAdd	77
.10	\xintFDg	66	.34	\xintiiCmp	80
.11	\xintLDg	67	.35	\xintiiSub	82
.12	\xintDouble	67	.36	\xintiiMul	87
.13	\xintHalf	68	.37	\xintiiDivision	91
.14	\xintInc	68	Derived arithmetic		107
.15	\xintDec	69	.38	\xintiiQuo, \xintiiRem	107
.16	\xintDSL	69	.39	\xintiiDivRound	107
.17	\xintDSR	69	.40	\xintiiDivTrunc	108
.18	\xintDSRr	70	.41	\xintiiModTrunc	108
Blocks of eight digits42	\xintiiDivMod	109
.19	\XINT_cuz	70	.43	\xintiiDivFloor	110
.20	\XINT_cuz_byviii	71	.44	\xintiiMod	110
.21	\XINT_unsep_loop	71	.45	\xintiiSqr	110
.22	\XINT_unsep_cuzsmall	72	.46	\xintiiPow	111
.23	\XINT_div_unsepQ	72	.47	\xintiiFac	114
.24	\XINT_div_unsepR	73	.48	\XINT_useiimessage	117

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

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

The commenting continues (2020/02/19) to be very sparse: actually it got worse than ever with release 1.2. I will possibly add comments at a later date, but for the time being the new routines are not commented at all.

1.3 removes all macros which were deprecated at 1.2o.

4.1 Catcodes, ε -T_EX and reload detection

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

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

```

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

```

12 \let\z\endgroup
13 \expandafter\let\expandafter\x\csname ver@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 [2020/02/19 v1.4a Expandable arithmetic on big integers (JFB)]%

```

4.3 (WIP!) Error conditions and exceptions

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

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

and the Python3 implementation in its Decimal module.

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

X3.274 rajoute LostDigits

Python rajoute FloatOperation (et n'inclut pas InsufficientStorage)

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

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

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

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

Various conditions can raise the same signal.

Only signals, not conditions, raise Flags.

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

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

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

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

Only four conditions are currently possibly encountered:

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

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

```

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

```

```

74 \fi
75 \xint_orthat{%
76 % the flag raised is named after the signal #1, but we show condition #2
77 \XINT_expandableerror{#2 (hit <RET> thrice)}%
78 \XINT_expandableerror{#3}%
79 \XINT_expandableerror{next: #5}%
80 % not for X3.274
81 %\XINT_expandableerror{<RET>, or I\xintUse{...}<RET>, or I\xintCTRLC<RET>}%
82 \xint_stop_atfirstofone{#5}%
83 }%
84 }%
85 %% \let\xintUse\xint_stop_atfirstofthree % defined in xint.sty
86 \def\XINT_ifFlagRaised #1{%
87 \ifcsname XINT_#1Flag_ON\endcsname
88 \expandafter\xint_firstoftwo
89 \else
90 \expandafter\xint_secondoftwo
91 \fi}%
92 \def\XINT_resetFlag #1%
93 {\expandafter\let\csname XINT_#1Flag_ON\endcsname\XINT_undefined}%
94 \def\XINT_resetFlags {% WIP
95 \XINT_resetFlag{InvalidOperation}% also from DivisionUndefined
96 \XINT_resetFlag{DivisionByZero}%
97 \XINT_resetFlag{Underflow}% (\xintiiPow with negative exponent)
98 \XINT_resetFlag{Overflow}% not encountered so far in xint code 1.21
99 % .. others ..
100 }%
101 \def\XINT_RaiseFlag #1{\expandafter\xint_gobble_i\csname XINT_#1Flag_ON\endcsname}%
102 \catcode`. 11
103 \let\XINT_Clamped.handler\xint_firstofone % WIP
104 \def\XINT_InvalidOperation.handler#1{NaN}% WIP
105 \def\XINT_ConversionSyntax.handler#1{NaN}% WIP
106 \def\XINT_DivisionByZero.handler#1{SignedInfinity(#1)}% WIP
107 \def\XINT_DivisionImpossible.handler#1{NaN}% WIP
108 \def\XINT_DivisionUndefined.handler#1{NaN}% WIP
109 \let\XINT_Inexact.handler\xint_firstofone % WIP
110 \def\XINT_InvalidContext.handler#1{NaN}% WIP
111 \let\XINT_Rounded.handler\xint_firstofone % WIP
112 \let\XINT_Subnormal.handler\xint_firstofone% WIP
113 \def\XINT_Overflow.handler#1{NaN}% WIP
114 \def\XINT_Underflow.handler#1{NaN}% WIP
115 \catcode`. 12

```

4.4 Counts for holding needed constants

```

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

```

```

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

```

Routines handling integers as lists of token digits

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

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

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

4.5 \XINT_cuz_small

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

```

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

```

4.6 \xintNum, \xintiNum

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

Very old routine got completely rewritten at 1.2l.

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%
146 {%

```



```

147 \xint_gob_til_xint: #9\XINT_num_end\xint:
148 #1#2#3#4#5#6#7#8#9%
149 \ifnum \numexpr #1#2#3#4#5#6#7#8#9+\xint_c_ = \xint_c_

```

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

```

150 \expandafter\XINT_num_loop
151 \else

```

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

```

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

```

4.7 \xintiiSgn

1.2l made \xintiiSgn robust against non terminated input.

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

```

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

```

4.8 \xintiiOpp

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

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

4.9 \xintiiAbs

Attention \xintiiAbs non robust against non terminated input.

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

4.10 \xintFDg

FIRST DIGIT.

1.21: \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
```

```

222     \krof
223 }%
```

4.11 \xintLDg

LAST DIGIT.

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

1.2o deprecates \xintiiLDq, gives to \xintLDq former meaning of \xintiiLDq.

Attention \xintLDq non robust against non terminated input.

```

224 \def\xintLDg {\romannumeral0\xintldg }%
225 \def\xintldg #1{\expandafter\XINT_ldg_fork\romannumeral`&&@#1%
226   \XINT_ldg_c{}{}{}{}{}{}{}{}\xint_bye\relax}%
227 \def\XINT_ldg_fork #1%
228 {%
229   \xint_UDsignfork
230     #1\XINT_ldg
231       -{\XINT_ldg#1}%
232   \krof
233 }%
234 \def\XINT_ldg #1{%
235 \def\XINT_ldg ##1##2##3##4##5##6##7##8##9%
236   {\expandafter#1%
237     \the\numexpr##9##8##7##6##5##4##3##2##1*\xint_c_+\XINT_ldg_a##9}%
238 }\XINT_ldg{ }%
239 \def\XINT_ldg_a#1#2{\XINT_ldg_cbye#2\XINT_ldg_d#1\XINT_ldg_c\XINT_ldg_b#2}%
240 \def\XINT_ldg_b#1#2#3#4#5#6#7#8#9#{#9#8#7#6#5#4#3#2#1*\xint_c_+\XINT_ldg_a#9}%
241 \def\XINT_ldg_c    #1#2\xint_bye{#1}%
242 \def\XINT_ldg_cbye #1\XINT_ldg_c{}%
243 \def\XINT_ldg_d#1#2\xint_bye{#1}%

```

4.12 \xintDouble

Attention \xintDouble non robust against non terminated input.

```

244 \def\xintDouble {\romannumeral0\xintdouble}%
245 \def\xintdouble #1{\expandafter\XINT_dbl_fork\romannumeral`&&@#1%
246         \xint_bye2345678\xint_bye*\xint_c_ii\relax}%
247 \def\XINT_dbl_fork #1%
248 {%
249     \xint_UDsignfork
250     #1\XINT_dbl_neg
251     -\XINT_dbl
252     \krof #1%
253 }%
254 \def\XINT_dbl_neg-{\expandafter-\romannumeral0\XINT_dbl}%
255 \def\XINT_dbl #1{%
256 \def\XINT_dbl ##1##2##3##4##5##6##7##8%
257     {\expandafter#1\the\numexpr##1##2##3##4##5##6##7##8\XINT_dbl_a}%
258 }\XINT_dbl{ }%
259 \def\XINT_dbl_a #1#2#3#4#5#6#7#8%
260     {\expandafter\XINT_dbl_e\the\numexpr 1#1#2#3#4#5#6#7#8\XINT_dbl_a}%

```

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

4.13 \xintHalf

Attention \xintHalf non robust against non terminated input.

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

4.14 \xintInc

1.2i much delayed complete rewrite in 1.2 style.

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

Attention \xintInc non robust against non terminated input.

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

4.15 \xintDec

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

Attention \xintDec non robust against non terminated input.

```

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

```

4.16 \xintDSL

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

Attention \xintDSL non robust against non terminated input.

```

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

```

4.17 \xintDSR

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

Attention \xintDSR non robust against non terminated input.

```

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

```

```

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

```

4.18 \xintDSRr

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

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

Attention \xintDSRr non robust against non terminated input.

```

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

```

Blocks of eight digits

The lingua of release 1.2.

4.19 \XINT_cuz

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

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

Note 2015/11/28: with only four digits the gob_til_fourzeroes had proved in some old testing faster than \ifnum test. But with eight digits, the execution times are much closer, as I tested back then.

```

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

```

4.20 \XINT_cuz_byviii

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

```

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

```

4.21 \XINT_unsep_loop

This is used as

```

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

```

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

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

Done at 1.21 for usage by other macros. The similar code in earlier releases was strangely in $O(N^2)$ style, apparently to avoid some memory constraints. But these memory constraints related to \numexpr chaining seems to be in many places in xint code base. The 1.21 version is written in the 1.2i style of \xintInc etc... and is compatible with some 1! block without digits among the treated blocks, they will disappear.

```

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

```

```

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

```

4.22 \XINT_unsep_cuzsmall

This is used as

```

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

```

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

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

```

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

```

4.23 \XINT_div_unsepQ

This is used by division to remove separators from the produced quotient. The quotient is produced in the correct order. The routine will also remove leading zeroes. An extra initial block of 8 zeroes is possible and thus if present must be removed. Then the next eight digits must be cleaned of leading zeroes. Attention that there might be a single block of 8 zeroes. Expansion launched by \romannumeral0.

Rewritten for 1.2l in 1.2i style.

```

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

```



```

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

```

4.24 \XINT_div_unsepR

This is used by division to remove separators from the produced remainder. The remainder is here in correct order. It must be cleaned of leading zeroes, possibly all the way.

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

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

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

```

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

```

4.25 \XINT_zeroes_forviii

\romannumeral0\XINT_zeroes_forviii #1\R\R\R\R\R\R\R\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

```

```

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 }%

```

```

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.)

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_add_fork #1#2\xint:#3\xint:{\XINT_add_nfork #1#3\xint:#2\xint:}%
538 \def\XINT_add_nfork #1#2%
539 {%
540     \xint_UDzerofork
541     #1\XINT_add_firstiszero
542     #2\XINT_add_secondiszero
543     0{}}%
544     \krof
545     \xint_UDsignsfork
546     #1#2\XINT_add_minusminus
547     #1-\XINT_add_minusplus
548     #2-\XINT_add_plusminus
549     --\XINT_add_plusplus
550     \krof #1#2%
551 }%
552 \def\XINT_add_firstiszero #1\krof 0#2#3\xint:#4\xint:{ #2#3}%
553 \def\XINT_add_secondiszero #1\krof #20#3\xint:#4\xint:{ #2#4}%
554 \def\XINT_add_minusminus #1#2%
555     {\expandafter-\romannumeral0\XINT_add_pp_a {}}}%
556 \def\XINT_add_minusplus #1#2{\XINT_sub_mm_a {}}#2}%
557 \def\XINT_add_plusminus #1#2%
558     {\expandafter\XINT_opp\romannumeral0\XINT_sub_mm_a #1{}}}%
559 \def\XINT_add_pp_a #1#2#3\xint:
560 {%
561     \expandafter\XINT_add_pp_b
562     \romannumeral0\expandafter\XINT_sepandrev_andcount
563     \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
564     #2#3\XINT_rsepybviii_end_A 2345678%
565     \XINT_rsepybviii_end_B 2345678\relax\xint_c_ii\xint_c_i
566     \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
567     \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_W
568     \X #1%
569 }%
570 \let\XINT_add_plusplus \XINT_add_pp_a
571 \def\XINT_add_pp_b #1\xint:#2\X #3\xint:

```

```

572 {%
573   \expandafter\XINT_add_checklengths
574   \the\numexpr #1\expandafter\xint:%
575   \romannumeral0\expandafter\XINT_sepandrev_andcount
576   \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\{10}0000001\W
577   #3\XINT_rsepbyviii_end_A 2345678%
578   \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
579   \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
580   \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
581   1;!1;!1;!1;!1\W #21;!1;!1;!1;!1\W
582   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\W
583 }%

```

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

```

584 \def\XINT_add_checklengths #1\xint:#2\xint:%
585 {%
586   \ifnum #2>#1
587     \expandafter\XINT_add_exchange
588   \else
589     \expandafter\XINT_add_A
590   \fi
591   #1\xint:#2\xint:%
592 }%
593 \def\XINT_add_exchange #1\xint:#2\xint:#3\W #4\W
594 {%
595   \XINT_add_A #2\xint:#1\xint:#4\W #3\W
596 }%
597 \def\XINT_add_A #1\xint:#2\xint:%
598 {%
599   \ifnum #1>\xint_c_vi
600     \expandafter\XINT_add_aa
601   \else \expandafter\XINT_add_aa_small
602   \fi
603 }%
604 \def\XINT_add_aa {\expandafter\XINT_add_out\the\numexpr\XINT_add_a \xint_c_ii}%
605 \def\XINT_add_out{\expandafter\XINT_cuz_small\romannumeral0\XINT_unrevbyviii {}}%
606 \def\XINT_add_aa_small
607   {\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).

```

608 \def\XINT_add_a #1!#2!#3!#4!#5\W
609   #6!#7!#8!#9!%
610 {%
611   \XINT_add_b

```

```

612      #1!#6!#2!#7!#3!#8!#4!#9!%
613      #5\W
614 }%
615 \def\XINT_add_b #1#2#3!#4!%
616 {%
617     \xint_gob_til_sc #2\XINT_add_bi ;%
618     \expandafter\XINT_add_c\the\numexpr#1+1#2#3+#4-\xint_c_ii\xint:%
619 }%
620 \def\XINT_add_bi;\expandafter\XINT_add_c
621     \the\numexpr#1+#2+#3-\xint_c_ii\xint:#4!#5!#6!#7!#8!#9!\W
622 {%
623     \XINT_add_k #1#3!#5!#7!#9!%
624 }%
625 \def\XINT_add_c #1#2\xint:%
626 {%
627     1#2\expandafter!\the\numexpr\XINT_add_d #1%
628 }%
629 \def\XINT_add_d #1#2#3!#4!%
630 {%
631     \xint_gob_til_sc #2\XINT_add_di ;%
632     \expandafter\XINT_add_e\the\numexpr#1+1#2#3+#4-\xint_c_ii\xint:%
633 }%
634 \def\XINT_add_di;\expandafter\XINT_add_e
635     \the\numexpr#1+#2+#3-\xint_c_ii\xint:#4!#5!#6!#7!#8\W
636 {%
637     \XINT_add_k #1#3!#5!#7!%
638 }%
639 \def\XINT_add_e #1#2\xint:%
640 {%
641     1#2\expandafter!\the\numexpr\XINT_add_f #1%
642 }%
643 \def\XINT_add_f #1#2#3!#4!%
644 {%
645     \xint_gob_til_sc #2\XINT_add_fi ;%
646     \expandafter\XINT_add_g\the\numexpr#1+1#2#3+#4-\xint_c_ii\xint:%
647 }%
648 \def\XINT_add_fi;\expandafter\XINT_add_g
649     \the\numexpr#1+#2+#3-\xint_c_ii\xint:#4!#5!#6\W
650 {%
651     \XINT_add_k #1#3!#5!%
652 }%
653 \def\XINT_add_g #1#2\xint:%
654 {%
655     1#2\expandafter!\the\numexpr\XINT_add_h #1%
656 }%
657 \def\XINT_add_h #1#2#3!#4!%
658 {%
659     \xint_gob_til_sc #2\XINT_add_hi ;%
660     \expandafter\XINT_add_i\the\numexpr#1+1#2#3+#4-\xint_c_ii\xint:%
661 }%
662 \def\XINT_add_hi;%
663     \expandafter\XINT_add_i\the\numexpr#1+#2+#3-\xint_c_ii\xint:#4\W

```

```

664 {%
665   \XINT_add_k #1#3!%
666 }%
667 \def\XINT_add_i #1#2\xint:%
668 {%
669   1#2\expandafter!\the\numexpr\XINT_add_a #1%
670 }%

671 \def\XINT_add_k #1{\if #12\expandafter\XINT_add_ke\else\expandafter\XINT_add_l \fi}%
672 \def\XINT_add_ke #11;#2\W {\XINT_add_kf #11;!}%
673 \def\XINT_add_kf 1{1\relax }%
674 \def\XINT_add_l 1#1#2{\xint_gob_til_sc #1\XINT_add_lf ;\XINT_add_m 1#1#2}%
675 \def\XINT_add_lf #1\W {1\relax 00000001!1;!}%
676 \def\XINT_add_m #1!\{\expandafter\XINT_add_n\the\numexpr\xint_c_i+#1\xint:}%
677 \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>.)

```

678 \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.

```

679 \def\xintiiCmp {\romannumeral0\xintiicmp }%
680 \def\xintiicmp #1{\expandafter\XINT_iicmp\romannumeral`&&@#1\xint:}%
681 \def\XINT_iicmp #1#2\xint:#3%
682 {%
683   \expandafter\XINT_cmp_nfork\expandafter #1\romannumeral`&&@#3\xint:#2\xint:
684 }%
685 \def\XINT_cmp_nfork #1#2%
686 {%
687   \xint_UDzerofork
688   #1\XINT_cmp_firstiszero
689   #2\XINT_cmp_secondiszero
690   0{}}%
691   \krof
692   \xint_UDsignsfork
693   #1#2\XINT_cmp_minusminus
694   #1-\XINT_cmp_minusplus
695   #2-\XINT_cmp_plusminus
696   --\XINT_cmp_plusplus
697   \krof #1#2%
698 }%
699 \def\XINT_cmp_firstiszero #1\krof 0#2#3\xint:#4\xint:
700 {%
701   \xint_UDzerominusfork
702   #2-{ 0}%
703   0#2{ 1}%
704   0-{ -1}%
705   \krof
706 }%

```



```

707 \def\XINT_cmp_secondiszero #1\krof #2#3\xint:#4\xint:
708 {%
709     \xint_UDzerominusfork
710     #2-{ 0}%
711     0#2{ -1}%
712     0-{ 1}%
713     \krof
714 }%
715 \def\XINT_cmp_plusminus #1\xint:#2\xint:{ 1}%
716 \def\XINT_cmp_minusplus #1\xint:#2\xint:{ -1}%
717 \def\XINT_cmp_minusminus
718     --{\expandafter\XINT_opp\romannumeral0\XINT_cmp_plusplus {}{}}%
719 \def\XINT_cmp_plusplus #1#2#3\xint:
720 {%
721     \expandafter\XINT_cmp_pp
722     \the\numexpr\expandafter\XINT_sepbyviii_andcount
723     \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
724     #2#3\XINT_sepbyviii_end 2345678\relax
725     \xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!%
726     \xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_\W
727     #1%
728 }%
729 \def\XINT_cmp_pp #1\xint:#2\xint:#3\xint:
730 {%
731     \expandafter\XINT_cmp_checklengths
732     \the\numexpr #2\expandafter\xint:%
733     \the\numexpr\expandafter\XINT_sepbyviii_andcount
734     \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
735     #3\XINT_sepbyviii_end 2345678\relax
736     \xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!%
737     \xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_\W
738     #1;!1;!1;!1;!1;\W
739 }%
740 \def\XINT_cmp_checklengths #1\xint:#2\xint:#3\xint:
741 {%
742     \ifnum #1=#3
743         \expandafter\xint_firstoftwo
744     \else
745         \expandafter\xint_secondoftwo
746     \fi
747     \XINT_cmp_a {\XINT_cmp_distinctlengths {#1}{#3}}#2;!1;!1;!1;\W
748 }%
749 \def\XINT_cmp_distinctlengths #1#2#3\W #4\W
750 {%
751     \ifnum #1>#2
752         \expandafter\xint_firstoftwo
753     \else
754         \expandafter\xint_secondoftwo
755     \fi
756     { -1}{ 1}%
757 }%
758 \def\XINT_cmp_a 1#1!1#2!1#3!1#4!1#5\W 1#6!1#7!1#8!1#9!%

```

```

759 {%
760   \xint_gob_til_sc #1\XINT_cmp_equal ;%
761   \ifnum #1>#6 \XINT_cmp_gt\fi
762   \ifnum #1<#6 \XINT_cmp_lt\fi
763   \xint_gob_til_sc #2\XINT_cmp_equal ;%
764   \ifnum #2>#7 \XINT_cmp_gt\fi
765   \ifnum #2<#7 \XINT_cmp_lt\fi
766   \xint_gob_til_sc #3\XINT_cmp_equal ;%
767   \ifnum #3>#8 \XINT_cmp_gt\fi
768   \ifnum #3<#8 \XINT_cmp_lt\fi
769   \xint_gob_til_sc #4\XINT_cmp_equal ;%
770   \ifnum #4>#9 \XINT_cmp_gt\fi
771   \ifnum #4<#9 \XINT_cmp_lt\fi
772   \XINT_cmp_a #5\W
773 }%
774 \def\XINT_cmp_lt#1{\def\XINT_cmp_lt\fi ##1\W ##2\W {\fi#1-1}}\XINT_cmp_lt{ }%
775 \def\XINT_cmp_gt#1{\def\XINT_cmp_gt\fi ##1\W ##2\W {\fi#1+1}}\XINT_cmp_gt{ }%
776 \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 deciding upon some internal format for `xintfrac` macros), thus I wanted to uniformize delimiters in particular and have some core macros inputting and outputting such formats. But the way division is implemented makes it currently very hard to obtain a satisfactory solution. For subtraction I got there almost, but there was added overhead and, as the core sub-routine still assumed the shorter number will be positioned first, one would need to record the length also in the basic internal format, or add the overhead to not make assumption on which one is shorter. I thus but back-tracked my steps but in passing I improved the efficiency (probably) in the worst case branch.

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

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

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

```

777 \def\xintiisub {\romannumeral0\xintiisub }%
778 \def\xintiisub #1{\expandafter\XINT_iisub\romannumeral`&&@#1\xint:}%
779 \def\XINT_iisub #1#2\xint:#3%
780 {%
781   \expandafter\XINT_sub_nfork\expandafter
782   #1\romannumeral`&&@#3\xint:#2\xint:
783 }%
784 \def\XINT_sub_nfork #1#2%
785 {%
786   \xint_UDzerofork
787   #1\XINT_sub_firstiszero
788   #2\XINT_sub_secondiszero
789   0{ }%

```

```

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

```

```

842 \expandafter\XINT_opp\romannumeral0\XINT_sub_aa #2\W #1\W
843 }%
844 \def\XINT_sub_aa
845 {%
846 \expandafter\XINT_sub_out\the\numexpr\XINT_sub_a\xint_c_i
847 }%

```

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.

```

848 \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.

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.

```

849 \def\XINT_sub_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
850 {%
851 \XINT_sub_b
852 #1!#6!#2!#7!#3!#8!#4!#9!%
853 #5\W
854 }%

```

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.

```

855 \def\XINT_sub_b #1#2#3#4!#5!%
856 {%
857     \xint_gob_til_sc #3\XINT_sub_bi ;%
858     \expandafter\XINT_sub_c\the\numexpr#1+1#5-#3#4-\xint_c_i\xint:%
859 }%
860 \def\XINT_sub_c 1#1#2\xint:%
861 {%
862     1#2\expandafter!\the\numexpr\XINT_sub_d #1%
863 }%
864 \def\XINT_sub_d #1#2#3#4!#5!%
865 {%
866     \xint_gob_til_sc #3\XINT_sub_di ;%
867     \expandafter\XINT_sub_e\the\numexpr#1+1#5-#3#4-\xint_c_i\xint:%
868 }%
869 \def\XINT_sub_e 1#1#2\xint:%
870 {%
871     1#2\expandafter!\the\numexpr\XINT_sub_f #1%
872 }%
873 \def\XINT_sub_f #1#2#3#4!#5!%
874 {%
875     \xint_gob_til_sc #3\XINT_sub_fi ;%
876     \expandafter\XINT_sub_g\the\numexpr#1+1#5-#3#4-\xint_c_i\xint:%
877 }%
878 \def\XINT_sub_g 1#1#2\xint:%
879 {%
880     1#2\expandafter!\the\numexpr\XINT_sub_h #1%
881 }%
882 \def\XINT_sub_h #1#2#3#4!#5!%
883 {%
884     \xint_gob_til_sc #3\XINT_sub_hi ;%
885     \expandafter\XINT_sub_i\the\numexpr#1+1#5-#3#4-\xint_c_i\xint:%
886 }%
887 \def\XINT_sub_i 1#1#2\xint:%
888 {%
889     1#2\expandafter!\the\numexpr\XINT_sub_a #1%
890 }%
891 \def\XINT_sub_bi;%
892     \expandafter\XINT_sub_c\the\numexpr#1+1#2-#3\xint:
893     #4!#5!#6!#7!#8!#9!\W
894 {%
895     \XINT_sub_k #1#2!#5!#7!#9!%
896 }%
897 \def\XINT_sub_di;%
898     \expandafter\XINT_sub_e\the\numexpr#1+1#2-#3\xint:
899     #4!#5!#6!#7!#8\W
900 {%
901     \XINT_sub_k #1#2!#5!#7!%
902 }%
903 \def\XINT_sub_fi;%
904     \expandafter\XINT_sub_g\the\numexpr#1+1#2-#3\xint:
905     #4!#5!#6\W
906 {%

```

```

907 \XINT_sub_k #1#2!#5!%
908 }%
909 \def\XINT_sub_hi;%
910 \expandafter\XINT_sub_i\the\numexpr#1+#2-#3\xint:
911 #4\W
912 {%
913 \XINT_sub_k #1#2!%
914 }%

```

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

If not, then we are certain that even if there is carry it will not propagate beyond the end of A. But it may propagate far transforming chains of 00000000 into 99999999, and if it does go to the final block which possibly is just 1<00000001>!, we will have those eight zeros to clean up.

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

```

915 \def\XINT_sub_k #1#2#3%
916 {%
917     \xint_gob_til_sc #3\XINT_sub_p;\XINT_sub_l #1#2#3%
918 }%
919 \def\XINT_sub_l #1%
920     {\xint_UDzerofork #1\XINT_sub_l_carry 0\XINT_sub_l_Ia\krof}%
921 \def\XINT_sub_l_Ia 1#1;!#2\W{1\relax#1;!1\XINT_sub_fix_none!}%

922 \def\XINT_sub_l_carry 1#1!{\ifcase #1
923     \expandafter \XINT_sub_l_Id
924     \or \expandafter \XINT_sub_l_Ic
925     \else\expandafter \XINT_sub_l_Ib\fi 1#1!}%
926 \def\XINT_sub_l_Ib #1;!#2\W {-\xint_c_i+#1;!1\XINT_sub_fix_none!}%
927 \def\XINT_sub_l_Ic 1#1!1#2#3!#4;!#5\W
928 {%
929     \xint_gob_til_sc #2\XINT_sub_l_Ica;%
930     1\relax 00000000!1#2#3!#4;!1\XINT_sub_fix_none!%
931 }%

```

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`

```
932 \def\XINT_sub_l_Ica#1\W
933 {%
934     1;!1\XINT_sub_fix_cuz!%
935     1;!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
936     \xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z
937 }%
938 \def\XINT_sub_l_Id #1!%!
939     {199999999\expandafter!\the\numexpr \XINT_sub_l_Id_a}%
940 \def\XINT_sub_l_Id_a #1!{\ifcase #1
941     \expandafter \XINT_sub_l_Id
942     \or \expandafter \XINT_sub_l_Id_b
943     \else\expandafter \XINT_sub_l_Ib\fi #1!}%
944 \def\XINT_sub_l_Id_b #1!1#2#3!#4;#5\W
945 }
```

```

946     \xint_gob_til_sc #2\XINT_sub_1_Ida;%
947     1\relax 00000000!1#2#3!#4;!1\XINT_sub_fix_none!%
948 }%
949 \def\XINT_sub_1_Ida#1\XINT_sub_fix_none{1;!1\XINT_sub_fix_none}%

```

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

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

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

```
950 \def\XINT_sub_p;\XINT_sub_l #1#2\W #3\W  
951 {%  
952     \xint_UDzerofork  
953     #1{1;!1\XINT_sub_fix_neg!  
954         1;!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W  
955         \xint_bye2345678\xint_bye1099999988\relax}% A - B, B > A  
956         0{1;!1\XINT_sub_fix_cuz!  
957             1;!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W}%  
958     \krof  
959     \xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:  
960 }%
```

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.

```

961 \def\XINT_sub_fix_none;{\XINT_cuz_small}%
962 \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<=A branch.

```

963 \def\XINT_sub_fix_neg;%
964 {%
965     \expandafter-\romannumeral0\expandafter
966     \XINT_sub_comp_finish\the\numexpr\XINT_sub_comp_loop
967 }%
968 \def\XINT_sub_comp_finish 0{\XINT_sub_fix_cuz;}%
969 \def\XINT_sub_comp_loop #1#2#3#4#5#6#7#8%
970 {%
971     \expandafter\XINT_sub_comp_clean
972     \the\numexpr \xint_c_xi_e_viii_mone-#1#2#3#4#5#6#7#8\XINT_sub_comp_loop
973 }%

```

#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.

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

4.36 \xintiiMul

Completely rewritten for 1.2.

1.21: \xintiiMul made robust against non terminated input.

```

975 \def\xintiiMul {\romannumeral0\xintiimul }%
976 \def\xintiimul #1%
977 {%
978   \expandafter\XINT_iimul\romannumeral`&&#1\xint:
979 }%
980 \def\XINT_iimul #1#2\xint:#3%
981 {%
982   \expandafter\XINT_mul_nfork\expandafter #1\romannumeral`&&#3\xint:#2\xint:
983 }%

```

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

ATTENTION for example that 1.4 \xintiiPrd uses \XINT_mul_nfork now.

```

984 \def\XINT_mul_fork #1#2\xint:#3\xint:{\XINT_mul_nfork #1#3\xint:#2\xint:}%
985 \def\XINT_mul_nfork #1#2%
986 {%
987   \xint_UDzerofork
988   #1\XINT_mul_zero
989   #2\XINT_mul_zero
990   0{}}%
991   \krof
992   \xint_UDsignsfork
993   #1#2\XINT_mul_minusminus
994   #1-\XINT_mul_minusplus
995   #2-\XINT_mul_plusminus
996   --\XINT_mul_plusplus
997   \krof #1#2%
998 }%
999 \def\XINT_mul_zero #1\krof #2#3\xint:#4\xint:{ 0}%
1000 \def\XINT_mul_minusminus #1#2{\XINT_mul_plusplus {}{}}%
1001 \def\XINT_mul_minusplus #1#2%
1002   {\expandafter-\romannumeral0\XINT_mul_plusplus {}#2}%
1003 \def\XINT_mul_plusminus #1#2%
1004   {\expandafter-\romannumeral0\XINT_mul_plusplus #1{}}%
1005 \def\XINT_mul_plusplus #1#2#3\xint:
1006 {%
1007   \expandafter\XINT_mul_pre_b
1008   \romannumeral0\expandafter\XINT_sepandrev_andcount
1009   \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
1010   #2#3\XINT_rsepybviii_end_A 2345678%
1011   \XINT_rsepybviii_end_B 2345678\relax\xint_c_ii\xint_c_i
1012   \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
1013   \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
1014   \W #1%
1015 }%
1016 \def\XINT_mul_pre_b #1\xint:#2\W #3\xint:
1017 {%
1018   \expandafter\XINT_mul_checklengths
1019   \the\numexpr #1\expandafter\xint:%
1020   \romannumeral0\expandafter\XINT_sepandrev_andcount
1021   \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
1022   #3\XINT_rsepybviii_end_A 2345678%
1023   \XINT_rsepybviii_end_B 2345678\relax\xint_c_ii\xint_c_i

```



```

1024          \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
1025          \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_W
1026      1;!W #21;!%
1027      1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1028 }%

```

Cooking recipe, 2015/10/05.

```

1029 \def\XINT_mul_checklengths #1\xint:#2\xint:%
1030 {%
1031     \ifnum #2=\xint_c_i\expandafter\XINT_mul_smallbyfirst\fi
1032     \ifnum #1=\xint_c_i\expandafter\XINT_mul_smallbysecond\fi
1033     \ifnum #2<#1
1034         \ifnum \numexpr (#2-\xint_c_i)*(#1-#2)<383
1035             \XINT_mul_exchange
1036         \fi
1037     \else
1038         \ifnum \numexpr (#1-\xint_c_i)*(#2-#1)>383
1039             \XINT_mul_exchange
1040         \fi
1041     \fi
1042     \XINT_mul_start
1043 }%
1044 \def\XINT_mul_smallbyfirst #1\XINT_mul_start 1#2!1;!W
1045 {%
1046     \ifnum#2=\xint_c_i\expandafter\XINT_mul_oneisone\fi
1047     \ifnum#2<\xint_c_xxii\expandafter\XINT_mul_verysmall\fi
1048     \expandafter\XINT_mul_out\the\numexpr\XINT_smallmul 1#2!%
1049 }%
1050 \def\XINT_mul_smallbysecond #1\XINT_mul_start #2W 1#3!1;!%
1051 {%
1052     \ifnum#3=\xint_c_i\expandafter\XINT_mul_oneisone\fi
1053     \ifnum#3<\xint_c_xxii\expandafter\XINT_mul_verysmall\fi
1054     \expandafter\XINT_mul_out\the\numexpr\XINT_smallmul 1#3!#2%
1055 }%
1056 \def\XINT_mul_oneisone #1!\{\XINT_mul_out }%
1057 \def\XINT_mul_verysmall\expandafter\XINT_mul_out
1058         \the\numexpr\XINT_smallmul 1#1!%
1059     {\expandafter\XINT_mul_out\the\numexpr\XINT_verysmallmul 0\xint:#1!}%
1060 \def\XINT_mul_exchange #1\XINT_mul_start #2W #31;!%
1061     {\fi\fi\XINT_mul_start #31;!W #2}%

1062 \def\XINT_mul_start
1063     {\expandafter\XINT_mul_out\the\numexpr\XINT_mul_loop 100000000!1;!W}%
1064 \def\XINT_mul_out
1065     {\expandafter\XINT_cuz_small\romannumeral0\XINT_unrevbyviii {}}%

```

Call:

\the\numexpr \XINT_mul_loop 100000000!1;!W #11;!W #21;!

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.

```
1066 \def\XINT_mul_loop #1\W #2\W 1#3!%
1067 {%
1068   \xint_gob_til_sc #3\XINT_mul_e ;%
1069   \expandafter\XINT_mul_a\the\numexpr \XINT_smallmul 1#3!#2\W
1070   #1\W #2\W
1071 }%
```

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

```
1072 \def\XINT_mul_a #1\W #2\W
1073 {%
1074   \expandafter\XINT_mul_b\the\numexpr
1075   \XINT_add_a \xint_c_ii #21;!1;!1;!1\W #11;!1;!1;!1\W\W
1076 }%
1077 \def\XINT_mul_b 1#1!{1#1\expandafter!\the\numexpr\XINT_mul_loop }%
1078 \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.

```
1079 \def\XINT_minimulwc_a 1#1\xint:#2\xint:#3!#4#5#6#7#8\xint:%
1080 {%
1081   \expandafter\XINT_minimulwc_b
1082   \the\numexpr \xint_c_x^ix+#1+#3*#8\xint:
1083   #3*#4#5#6#7+#2*#8\xint:
1084   #2*#4#5#6#7\xint:%
1085 }%
1086 \def\XINT_minimulwc_b 1#1#2#3#4#5#6\xint:#7\xint:%
1087 {%
1088   \expandafter\XINT_minimulwc_c
1089   \the\numexpr \xint_c_x^ix+#1#2#3#4#5+#7\xint:#6\xint:%
1090 }%
1091 \def\XINT_minimulwc_c 1#1#2#3#4#5#6\xint:#7\xint:#8\xint:%
1092 {%
1093   1#6#7\expandafter!%
1094   \the\numexpr\expandafter\XINT_smallmul_a
1095   \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8\xint:%
1096 }%
1097 \def\XINT_smallmul 1#1#2#3#4#5!{\XINT_smallmul_a 1000000000\xint:#1#2#3#4\xint:#5!}%
1098 \def\XINT_smallmul_a #1\xint:#2\xint:#3!1#4!%
1099 {%
1100   \xint_gob_til_sc #4\XINT_smallmul_e;%
```

```

1101 \XINT_minimulwc_a #1\xint:#2\xint:#3!#4\xint:#2\xint:#3!%
1102 }%
1103 \def\XINT_smallmul_e;\XINT_minimulwc_a 1#1\xint:#2;#3!%
1104 {\xint_gob_til_eightzeroes #1\XINT_smallmul_f 000000001\relax #1!1;!}%
1105 \def\XINT_smallmul_f 000000001\relax 00000000!1{1\relax}%

1106 \def\XINT_verysmallmul #1\xint:#2!1#3!%
1107 {%
1108 \xint_gob_til_sc #3\XINT_verysmallmul_e;%
1109 \expandafter\XINT_verysmallmul_a
1110 \the\numexpr #2*#3+#1\xint:#2!%
1111 }%
1112 \def\XINT_verysmallmul_e;\expandafter\XINT_verysmallmul_a\the\numexpr
1113 #1+#2#3\xint:#4!%
1114 {\xint_gob_til_zero #2\XINT_verysmallmul_f 0\xint_c_x^viii+#2#3!1;!}%
1115 \def\XINT_verysmallmul_f #1!1{1\relax}%
1116 \def\XINT_verysmallmul_a #1#2\xint:%
1117 {%
1118 \unless\ifnum #1#2<\xint_c_x^ix
1119 \expandafter\XINT_verysmallmul_bi\else
1120 \expandafter\XINT_verysmallmul_bj\fi
1121 \the\numexpr \xint_c_x^ix+#1#2\xint:%
1122 }%
1123 \def\XINT_verysmallmul_bj{\expandafter\XINT_verysmallmul_cj }%
1124 \def\XINT_verysmallmul_cj 1#1#2\xint:%
1125 {1#2\expandafter!\the\numexpr\XINT_verysmallmul #1\xint:%
1126 \def\XINT_verysmallmul_bi\the\numexpr\xint_c_x^ix+#1#2#3\xint:%
1127 {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.

```

1128 \def\XINT_minimul_a #1\xint:#2!#3#4#5#6#7!%
1129 {%
1130 \expandafter\XINT_minimul_b
1131 \the\numexpr \xint_c_x^viii+#2*#7\xint:#2*#3#4#5#6+#1*#7\xint:#1*#3#4#5#6\xint:%
1132 }%
1133 \def\XINT_minimul_b 1#1#2#3#4#5\xint:#6\xint:%
1134 {%
1135 \expandafter\XINT_minimul_c
1136 \the\numexpr \xint_c_x^ix+#1#2#3#4+#6\xint:#5\xint:%
1137 }%
1138 \def\XINT_minimul_c 1#1#2#3#4#5#6\xint:#7\xint:#8\xint:%
1139 {%
1140 1#6#7\expandafter!\the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8!%
1141 }%

```

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 10⁸,

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

```
1142 \def\xintiDivision    {\romannumeral0\xintiDivision}%
1143 \def\xintiDivision    #1{\expandafter\XINT_iidivision \romannumeral`&&@#1\xint:}%
1144 \def\XINT_iidivision #1#2\xint:#3{\expandafter\XINT_iidivision_a\expandafter #1%
1145                                \romannumeral`&&@#3\xint:#2\xint:}%
```

On regarde les signes de A et de B.

```
1146 \def\XINT_iidivision_a #1#2% #1 de A, #2 de B.
1147 {%
1148     \if0#2\xint_dothis{\XINT_iidivision_divbyzero #1#2}\fi
1149     \if0#1\xint_dothis\XINT_iidivision_aiszero\fi
1150     \if-#2\xint_dothis{\expandafter\XINT_iidivision_bneg
1151                         \romannumeral0\XINT_iidivision_bpos #1}\fi
1152     \xint_orthat{\XINT_iidivision_bpos #1#2}%
1153}%
1154 \def\XINT_iidivision_divbyzero#1#2#3\xint:#4\xint:
1155     {\if0#1\xint_dothis{\XINT_signalcondition{DivisionUndefined}}\fi
1156     \xint_orthat{\XINT_signalcondition{DivisionByZero}}%
1157     {Division of #1#4 by #2#3}{\{0\}{0}}}%
1158 \def\XINT_iidivision_aiszero #1\xint:#2\xint:{\{0\}{0}}%
1159 \def\XINT_iidivision_bneg #1% q->-q, r unchanged
1160     {\expandafter{\romannumeral0\XINT_opp #1}}%
1161 \def\XINT_iidivision_bpos #1%
1162 {%
1163     \xint_UDsignfork
1164     #1\XINT_iidivision_aneg
1165     -{\XINT_iidivision_apos #1}%
1166     \krof
1167}%
```

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.

```
1168 \def\XINT_iidivision_apos #1#2\xint:#3\xint:{\XINT_div_prepare {#2}{#1#3}}%
1169 \def\XINT_iidivision_aneg #1\xint:#2\xint:
1170     {\expandafter
1171     \XINT_iidivision_aneg_b\romannumeral0\XINT_div_prepare {#1}{#2}{#1}}%
```

```

1172 \def\XINT_iidivision_aneg_b #1#2{\if0\XINT_Sgn #2\xint:
1173         \expandafter\XINT_iidivision_aneg_rzero
1174     \else
1175         \expandafter\XINT_iidivision_aneg_rpos
1176     \fi {#1}{#2}}%
1177 \def\XINT_iidivision_aneg_rzero #1#2#3{{-#1}{0}}% necessarily q was >0
1178 \def\XINT_iidivision_aneg_rpos #1%
1179 {%
1180     \expandafter\XINT_iidivision_aneg_end\expandafter
1181     {\expandafter\romannumeral0\xintinc {#1}}% q-> -(1+q)
1182 }%
1183 \def\XINT_iidivision_aneg_end #1#2#3%
1184 {%
1185     \expandafter\xint_exchangetwo_keepbraces
1186     \expandafter{\romannumeral0\XINT_sub_mm_a {}{}#3\xint:#2\xint:}{#1}% r-> b-r
1187 }%

```

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.

```

1188 \def\XINT_div_prepare #1%
1189 {%
1190     \XINT_div_prepare_a #1\R\R\R\R\R\R\R {10}0000001\W !{#1}%
1191 }%
1192 \def\XINT_div_prepare_a #1#2#3#4#5#6#7#8#9%
1193 {%
1194     \xint_gob_til_R #9\XINT_div_prepare_small\R
1195     \XINT_div_prepare_b #9%
1196 }%

```

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).

```

1197 \def\XINT_div_prepare_small\R #1!#2%
1198 {%
1199     \ifcase #2
1200     \or\expandafter\XINT_div_BisOne
1201     \or\expandafter\XINT_div_BisTwo
1202     \else\expandafter\XINT_div_small_a
1203     \fi {#2}%
1204 }%
1205 \def\XINT_div_BisOne #1#2{{#2}{0}}%
1206 \def\XINT_div_BisTwo #1#2%
1207 {%
1208     \expandafter\expandafter\expandafter\XINT_div_BisTwo_a
1209     \ifodd\xintLDg{#2} \expandafter1\else \expandafter0\fi {#2}%
1210 }%
1211 \def\XINT_div_BisTwo_a #1#2%
1212 {%
1213     \expandafter{\romannumeral0\XINT_half
1214     #2\xint_bye\xint_Bye345678\xint_bye
1215     *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax}{#1}%
1216 }%

```

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

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.

```
1217 \def\XINT_div_small_a #1#2%
1218 {%
1219   \expandafter\XINT_div_small_b
1220   \the\numexpr #1/\xint_c_ii\expandafter
1221   \xint:\the\numexpr \xint_c_x^viii+#1\expandafter!%
1222   \romannumeral0%
1223   \XINT_div_small_ba #2\R\R\R\R\R\R\R\R{10}0000001\W
1224   #2\XINT_sepybyviii_Z_end 2345678\relax
1225 }%
```

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

```
1226 \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 $A < 10^8$.

```
1227 \def\XINT_div_small_ba #1#2#3#4#5#6#7#8#9%
1228 {%
1229   \xint_gob_til_R #9\XINT_div_smallsmall\R
1230   \expandafter\XINT_div_dosmallldiv
1231   \the\numexpr\expandafter\XINT_sepybyviii_Z
1232   \romannumeral0\XINT_zeroes_forviii
1233   #1#2#3#4#5#6#7#8#9%
1234 }%
```

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}.

```
1235 \def\XINT_div_smallsmall\R
1236   \expandafter\XINT_div_dosmallldiv
1237   \the\numexpr\expandafter\XINT_sepybyviii_Z
1238   \romannumeral0\XINT_zeroes_forviii #1\R #2\relax
1239   {{\XINT_div_dosmallsmall}}{#1}}%
1240 \def\XINT_div_dosmallsmall #1\xint:1#2!#3%
1241 {%
1242   \expandafter\XINT_div_smallsmallend
1243   \the\numexpr (#3+#1)/#2-\xint_c_i\xint:#2\xint:#3\xint:%
1244 }%
1245 \def\XINT_div_smallsmallend #1\xint:#2\xint:#3\xint:{\expandafter
1246   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #3-#1*#2}}%
```

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

```
1247 \def\XINT_div_dosmallldiv
1248   {{\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.

```

1249 \def\XINT_div_prepare_b
1250   {\expandafter\XINT_div_prepare_c\romannumeral0\XINT_zeroes_forviii }%
1251 \def\XINT_div_prepare_c #1!%
1252 {%
1253   \XINT_div_prepare_d #1.00000000!{#1}%
1254 }%
1255 \def\XINT_div_prepare_d #1#2#3#4#5#6#7#8#9%
1256 {%
1257   \expandafter\XINT_div_prepare_e\xint_gob_til_dot #1#2#3#4#5#6#7#8#9!%
1258 }%
1259 \def\XINT_div_prepare_e #1!#2!#3#4%
1260 {%
1261   \XINT_div_prepare_f #4#3\X {#1}{#3}%
1262 }%

```

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.

```

1263 \def\XINT_div_prepare_f #1#2#3#4#5#6#7#8#9\X
1264 {%
1265   \expandafter\XINT_div_prepare_g
1266   \the\numexpr #1#2#3#4#5#6#7#8+\xint_c_i\expandafter
1267   \xint:\the\numexpr (#1#2#3#4#5#6#7#8+\xint_c_i)/\xint_c_ii\expandafter
1268   \xint:\the\numexpr #1#2#3#4#5#6#7#8\expandafter
1269   \xint:\romannumeral0\XINT_sepandrev_andcount
1270   #1#2#3#4#5#6#7#8#9\XINT_rsepbyviii_end_A 2345678%
1271   \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
1272   \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
1273   \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_W
1274   \X
1275 }%
1276 \def\XINT_div_prepare_g #1\xint:#2\xint:#3\xint:#4\xint:#5\X #6#7#8%
1277 {%
1278   \expandafter\XINT_div_prepare_h
1279   \the\numexpr\expandafter\XINT_sepbyviii_andcount
1280   \romannumeral0\XINT_zeroes_forviii #8#7\R\R\R\R\R\R\R\{10}0000001\W
1281   #8#7\XINT_sepbyviii_end 2345678\relax
1282   \xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!%
1283   \xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_W
1284   {#1}{#2}{#3}{#4}{#5}{#6}%
1285 }%
1286 \def\XINT_div_prepare_h #11\xint:#2\xint:#3#4#5#6#7#8%
1287 {%
1288   \XINT_div_start_a {#2}{#6}{#1}{#3}{#4}{#5}{#7}{#8}%
1289 }%

```

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.

```

1290 \def\XINT_div_start_a #1#2%
1291 {%
1292     \ifnum #1 < #2
1293         \expandafter\XINT_div_zeroQ
1294     \else
1295         \expandafter\XINT_div_start_b
1296     \fi
1297     {#1}{#2}%
1298 }%
1299 \def\XINT_div_zeroQ #1#2#3#4#5#6#7%
1300 {%
1301     \expandafter\XINT_div_zeroQ_end
1302     \romannumeral0\XINT_unsep_cuzsmall
1303     #3\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax\xint:
1304 }%
1305 \def\XINT_div_zeroQ_end #1\xint:#2%
1306     {\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»

```

1307 \def\XINT_div_start_b #1#2#3#4#5#6%
1308 {%
1309     \expandafter\XINT_div_finish\the\numexpr
1310     \XINT_div_start_c {#2}\xint:#3\xint:{#6}{#{1}{#2}{#{4}{#5}}{#6}}%
1311 }%
1312 \def\XINT_div_finish
1313 {%
1314     \expandafter\XINT_div_finish_a \romannumeral`&&\XINT_div_unsepQ
1315 }%
1316 \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».

```

1317 \def\XINT_div_finish_b #1%
1318 {%
1319     \if0#1%
1320         \expandafter\XINT_div_finish_bRzero
1321     \else
1322         \expandafter\XINT_div_finish_bRpos
1323     \fi
1324     #1%
1325 }%
1326 \def\XINT_div_finish_bRzero 0\xint:#1#2{#{1}{0}}%
1327 \def\XINT_div_finish_bRpos #1\xint:#2#3%
1328 {%
1329     \expandafter\xint_exchangetwo_keepbraces\XINT_div_cleanR #1#3\xint:{#2}%
1330 }%
1331 \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.


```

1332 \def\XINT_div_start_c #1%
1333 {%
1334     \ifnum #1>\xint_c_vi
1335         \expandafter\XINT_div_start_ca
1336     \else
1337         \expandafter\XINT_div_start_cb
1338     \fi {#1}%
1339 }%
1340 \def\XINT_div_start_ca #1#2\xint:#3!#4!#5!#6!#7!#8!#9!%
1341 {%
1342     \expandafter\XINT_div_start_c\expandafter
1343     {\the\numexpr #1-\xint_c_vii}{#2#3!#4!#5!#6!#7!#8!#9!\xint:%
1344 }%
1345 \def\XINT_div_start_cb #1%
1346     {\csname XINT_div_start_c_\romannumeral\xintexpr#1\endcsname}%
1347 \def\XINT_div_start_c_i #1\xint:#2!%
1348     {\XINT_div_start_c_ #1#2!\xint:}%
1349 \def\XINT_div_start_c_ii #1\xint:#2!#3!%
1350     {\XINT_div_start_c_ #1#2!#3!\xint:}%
1351 \def\XINT_div_start_c_iii #1\xint:#2!#3!#4!%
1352     {\XINT_div_start_c_ #1#2!#3!#4!\xint:}%
1353 \def\XINT_div_start_c_iv #1\xint:#2!#3!#4!#5!%
1354     {\XINT_div_start_c_ #1#2!#3!#4!#5!\xint:}%
1355 \def\XINT_div_start_c_v #1\xint:#2!#3!#4!#5!#6!%
1356     {\XINT_div_start_c_ #1#2!#3!#4!#5!#6!\xint:}%
1357 \def\XINT_div_start_c_vi #1\xint:#2!#3!#4!#5!#6!#7!%
1358     {\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».

```

1359 \def\XINT_div_start_c_ 1#1!#2\xint:#3\xint:#4#5#6%
1360 {%
1361     \XINT_div_I_a {#1}{#4}{1#1!#2}{#6}{00000000}#5{#3}{#6}%
1362 }%

```

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

```

1363 \def\XINT_div_I_a #1#2%
1364 {%
1365     \expandafter\XINT_div_I_b\the\numexpr #1/#2\xint:{#1}{#2}%
1366 }%
1367 \def\XINT_div_I_b #1%
1368 {%
1369     \xint_gob_til_zero #1\XINT_div_I_czero 0\XINT_div_I_c #1%
1370 }%

```

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».

```

1371 \def\XINT_div_I_czero 0\XINT_div_I_c 0\xint:#1#2#3#4#5{1#5\XINT_div_I_g {#3}}%
1372 \def\XINT_div_I_c #1\xint:#2#3%
1373 {%

```

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

```
1374 \expandafter\XINT_div_I_da\the\numexpr #2-#1*#3\xint:#1\xint:{#2}{#3}%
1375 }%
```

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

```
1376 \def\XINT_div_I_da #1\xint:%
1377 {%
1378   \ifnum #1>\xint_c_ix
1379     \expandafter\XINT_div_I_dP
1380   \else
1381     \ifnum #1<\xint_c_
1382       \expandafter\expandafter\expandafter\XINT_div_I_dN
1383     \else
1384       \expandafter\expandafter\expandafter\XINT_div_I_db
1385     \fi
1386   \fi
1387 }%
```

attention très mauvaises notations avec _b et _db.

```
1388 \def\XINT_div_I_dN #1\xint:%
1389 {%
1390   \expandafter\XINT_div_I_b\the\numexpr #1-\xint_c_i\xint:%
1391 }%
1392 \def\XINT_div_I_db #1\xint:#2#3#4#5%
1393 {%
1394   \expandafter\XINT_div_I_dc\expandafter #1%
1395   \romannumeral0\expandafter\XINT_div_sub\expandafter
1396     {\romannumeral0\XINT_rev_nounsep }#4\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1397     {\the\numexpr\XINT_div_verysmallmul #1!#51;!}%
1398   \Z {#4}{#5}%
1399 }%
```

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

```
1400 \def\XINT_div_I_dc #1#2%
1401 {%
1402   \if-#2\expandafter\XINT_div_I_dd\else\expandafter\XINT_div_I_de\fi
1403   #1#2%
1404 }%
1405 \def\XINT_div_I_dd #1-\Z
1406 {%
1407   \if #11\expandafter\XINT_div_I_dz\fi
1408   \expandafter\XINT_div_I_dP\the\numexpr #1-\xint_c_i\xint: XX%
1409 }%
1410 \def\XINT_div_I_dz #1XX#2#3#4%
1411 {%
1412   1#4\XINT_div_I_g {#2}%
1413 }%
1414 \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»

```

1415 \def\XINT_div_I_dP #1\xint:#2#3#4#5#6%
1416 {%
1417     1#6+#1\expandafter\XINT_div_I_g\expandafter
1418     {\romannumeral0\expandafter\XINT_div_sub\expandafter
1419     {\romannumeral0\XINT_rev_nounsep {}}#4\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1420     {\the\numexpr\XINT_div_verysmallmul #1!#51;!}%
1421 }%
1422 }%

    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»

1423 \def\XINT_div_I_g #1#2#3#4#5#6#7%
1424 {%
1425     \expandafter !\the\numexpr
1426     \ifnum#2=#3
1427         \expandafter\XINT_div_exittofinish
1428     \else
1429         \expandafter\XINT_div_I_h
1430     \fi
1431     {#4}#1\xint:#6\xint:{{#4}{#5}{#3}{#2}}{#7}%
1432 }%

    {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»

1433 \def\XINT_div_exittofinish #1#2\xint:#3\xint:#4#5%
1434 {%
1435     1\expandafter\expandafter\expandafter!\expandafter\XINT_div_unsepQ_delim
1436     \romannumeral0\XINT_div_unsepR #2#3%
1437     \xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax\R\xint:
1438 }%

    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»

1439 \def\XINT_div_I_h #1\xint:#2!#3\xint:#4#5%
1440 {%
1441     \XINT_div_II_b #1#2!\xint:{#5}{#4}{#3}{#5}%
1442 }%

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

1443 \def\XINT_div_II_b #1#2!#3!%
1444 {%
1445     \xint_gob_til_eightzeroes #2\XINT_div_II_skipc 00000000%
1446     \XINT_div_II_c #1{1#2}{#3}%
1447 }%

```

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

```
1448 \def\XINT_div_II_skipc 00000000\XINT_div_II_c #1#2#3#4#5\xint:#6#7%
1449 {%
1450     \XINT_div_II_k #7{#4!#5}{#6}{00000000}%
1451 }%
```

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

```
1452 \def\XINT_div_II_c #1#2#3#4%
1453 {%
1454     \expandafter\XINT_div_II_d\the\numexpr\XINT_div_xmini
1455     #1\xint:#2!#3!#4!{#1}{#2}#3!#4!%
1456 }%
1457 \def\XINT_div_xmini #1%
1458 {%
1459     \xint_gob_til_one #1\XINT_div_xmini_a 1\XINT_div_mini #1%
1460 }%
1461 \def\XINT_div_xmini_a 1\XINT_div_mini 1#1%
1462 {%
1463     \xint_gob_til_zero #1\XINT_div_xmini_b 0\XINT_div_mini 1#1%
1464 }%
1465 \def\XINT_div_xmini_b 0\XINT_div_mini 10#1#2#3#4#5#6#7%
1466 {%
1467     \xint_gob_til_zero #7\XINT_div_xmini_c 0\XINT_div_mini 10#1#2#3#4#5#6#7%
1468 }%
```

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

```
1469 \def\XINT_div_xmini_c 0\XINT_div_mini 100000000\xint:50000000!#1!#2!{#1!}%

```

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

```
1470 \def\XINT_div_II_d 1#1#2#3#4#5!#6#7#8\xint:#9%
1471 {%
1472     \expandafter\XINT_div_II_e
1473     \romannumeral0\expandafter\XINT_div_sub\expandafter
1474     {\romannumeral0\XINT_rev_nounsep {#8\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1475     {\the\numexpr\XINT_div_smallmul_a 100000000\xint:#1#2#3#4\xint:#5!#91;!}%
1476     \xint:{#6}{#7}{#9}{#1#2#3#4#5}%
1477 }%
```

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

```
1478 \def\XINT_div_II_e 1#1!%
1479 {%
1480     \xint_gob_til_eightzeroes #1\XINT_div_II_skipf 00000000%
1481     \XINT_div_II_f 1#1!%
1482 }%
```

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»

```
1483 \def\XINT_div_II_skipf 00000000\XINT_div_II_f 100000000!#1\xint:#2#3#4#5#6%
1484 {%
1485     \XINT_div_II_k #6{#1}{#4}{#5}%
1486 }%
```

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.

```
1487 \def\XINT_div_II_f #1!#2!#3\xint:%
1488 {%
1489     \XINT_div_II_fa {#1!#2!}{#1!#2!#3}%
1490 }%
1491 \def\XINT_div_II_fa #1#2#3#4%
1492 {%
1493     \expandafter\XINT_div_II_g \the\numexpr\XINT_div_xmini #3\xint:#4!#1{#2}%
1494 }%
```

#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»

```
1495 \def\XINT_div_II_g 1#1#2#3#4#5!#6#7#8%
1496 {%
1497     \expandafter \XINT_div_II_h
1498     \the\numexpr 1#1#2#3#4#5+#8\expandafter\expandafter\expandafter
1499     \xint:\expandafter\expandafter\expandafter
1500     {\expandafter\xint_gob_til_exclam
1501     \romannumeral0\expandafter\XINT_div_sub\expandafter
1502     {\romannumeral0\XINT_rev_nounsep }#6\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1503     {\the\numexpr\XINT_div_smallmul_a 100000000\xint:#1#2#3#4\xint:#5!#71;!}%
1504     {#7}%
1505 }%
```

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»

```
1506 \def\XINT_div_II_h 1#1\xint:#2#3#4%
1507 {%
1508     \XINT_div_II_k #4{#2}{#3}{#1}%
1509 }%
```

{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»

```
1510 \def\XINT_div_II_k #1#2#3#4#5%
1511 {%
1512     \expandafter\XINT_div_II_l \the\numexpr #4-\xint_c_i\xint:{#3}#1{#2}#5\xint:%
1513 }%
1514 \def\XINT_div_II_l #1\xint:#2#3#4#5!#6!%
1515 {%
1516     \XINT_div_II_m {{#1}{#2}}{{#3}{#4}}{{#5}}{{#5}{#6}}1#6!%
1517 }%
```

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

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

```
1518 \def\XINT_div_II_m #1#2#3#4\xint:#5#6%
1519 {%
1520     \XINT_div_I_a {#3}{#2}{#4}{#5}{#6}#1%
1521 }%
```

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

```
1522 \def\XINT_div_minimulwc_a 1#1\xint:#2\xint:#3!#4#5#6#7#8\xint:%
1523 {%
1524     \expandafter\XINT_div_minimulwc_b
1525     \the\numexpr \xint_c_x^ix+#1+#3*#8\xint:#3*#4#5#6#7+#2*#8\xint:#2*#4#5#6#7\xint:%
1526 }%
1527 \def\XINT_div_minimulwc_b 1#1#2#3#4#5#6\xint:#7\xint:%
1528 {%
1529     \expandafter\XINT_div_minimulwc_c
1530     \the\numexpr \xint_c_x^ix+#1#2#3#4#5+#7\xint:#6\xint:%
1531 }%
1532 \def\XINT_div_minimulwc_c 1#1#2#3#4#5#6\xint:#7\xint:#8\xint:%
1533 {%
1534     1#6#7\expandafter!%
1535     \the\numexpr\expandafter\XINT_div_smallmul_a
1536     \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8\xint:%
1537 }%
1538 \def\XINT_div_smallmul_a #1\xint:#2\xint:#3!1#4!%
1539 {%
1540     \xint_gob_til_sc #4\XINT_div_smallmul_e;%
1541     \XINT_div_minimulwc_a #1\xint:#2\xint:#3!#4\xint:#2\xint:#3!%
1542 }%
1543 \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.

```
1544 \def\XINT_div_verysmallmul #1%
1545     {\xint_gob_til_one #1\XINT_div_verysmallisone 1\XINT_div_verysmallmul_a 0\xint:#1}%
1546 \def\XINT_div_verysmallisone 1\XINT_div_verysmallmul_a 0\xint:1!1#11;!%
1547     {1\relax #1100000000!}%
1548 \def\XINT_div_verysmallmul_a #1\xint:#2!1#3!%
1549 {%
1550     \xint_gob_til_sc #3\XINT_div_verysmallmul_e;%
1551     \expandafter\XINT_div_verysmallmul_b
1552     \the\numexpr \xint_c_x^ix+#2*#3+#1\xint:#2!%
1553 }%
1554 \def\XINT_div_verysmallmul_b 1#1#2\xint:%
1555     {1#2\expandafter!\the\numexpr\XINT_div_verysmallmul_a #1\xint:}%
1556 \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.

```

1557 \def\XINT_div_sub #1#2%
1558 {%
1559     \expandafter\XINT_div_sub_clean
1560     \the\numexpr\expandafter\XINT_div_sub_a\expandafter
1561     1#2;!!;!!;!!\W #1;!!;!!;!!\W
1562 }%
1563 \def\XINT_div_sub_clean #1-#2#3\W
1564 {%
1565     \if1#2\expandafter\XINT_rev_nounsep\else\expandafter\XINT_div_sub_neg\fi
1566     {}#1\R!\R!\R!\R!\R!\R!\R!\R!\W
1567 }%
1568 \def\XINT_div_sub_neg #1\W { -}%
1569 \def\XINT_div_sub_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%
1570 {%
1571     \XINT_div_sub_b #1!#6!#2!#7!#3!#8!#4!#9!#5\W
1572 }%
1573 \def\XINT_div_sub_b #1#2#3!#4!%
1574 {%
1575     \xint_gob_til_sc #4\XINT_div_sub_bi ;%
1576     \expandafter\XINT_div_sub_c\the\numexpr#1-#3+1#4-\xint_c_i\xint:%
1577 }%
1578 \def\XINT_div_sub_c 1#1#2\xint:%
1579 {%
1580     1#2\expandafter!\the\numexpr\XINT_div_sub_d #1%
1581 }%
1582 \def\XINT_div_sub_d #1#2#3!#4!%
1583 {%
1584     \xint_gob_til_sc #4\XINT_div_sub_di ;%
1585     \expandafter\XINT_div_sub_e\the\numexpr#1-#3+1#4-\xint_c_i\xint:%
1586 }%
1587 \def\XINT_div_sub_e 1#1#2\xint:%
1588 {%
1589     1#2\expandafter!\the\numexpr\XINT_div_sub_f #1%
1590 }%
1591 \def\XINT_div_sub_f #1#2#3!#4!%
1592 {%
1593     \xint_gob_til_sc #4\XINT_div_sub_fi ;%
1594     \expandafter\XINT_div_sub_g\the\numexpr#1-#3+1#4-\xint_c_i\xint:%
1595 }%
1596 \def\XINT_div_sub_g 1#1#2\xint:%
1597 {%
1598     1#2\expandafter!\the\numexpr\XINT_div_sub_h #1%
1599 }%
1600 \def\XINT_div_sub_h #1#2#3!#4!%
1601 {%
1602     \xint_gob_til_sc #4\XINT_div_sub_hi ;%
1603     \expandafter\XINT_div_sub_i\the\numexpr#1-#3+1#4-\xint_c_i\xint:%
1604 }%
1605 \def\XINT_div_sub_i 1#1#2\xint:%
1606 {%
1607     1#2\expandafter!\the\numexpr\XINT_div_sub_a #1%
1608 }%

```

```

1609 \def\XINT_div_sub_bi;%
1610   \expandafter\XINT_div_sub_c\the\numexpr#1-#2+#3\xint:#4!#5!#6!#7!#8!#9!;! \W
1611 {%
1612   \XINT_div_sub_l #1#2!#5!#7!#9!%
1613 }%
1614 \def\XINT_div_sub_di;%
1615   \expandafter\XINT_div_sub_e\the\numexpr#1-#2+#3\xint:#4!#5!#6!#7!#8\W
1616 {%
1617   \XINT_div_sub_l #1#2!#5!#7!%
1618 }%
1619 \def\XINT_div_sub_fi;%
1620   \expandafter\XINT_div_sub_g\the\numexpr#1-#2+#3\xint:#4!#5!#6\W
1621 {%
1622   \XINT_div_sub_l #1#2!#5!%
1623 }%
1624 \def\XINT_div_sub_hi;%
1625   \expandafter\XINT_div_sub_i\the\numexpr#1-#2+#3\xint:#4\W
1626 {%
1627   \XINT_div_sub_l #1#2!%
1628 }%
1629 \def\XINT_div_sub_l #1%
1630 {%
1631   \xint_UDzerofork
1632   #1{-2\relax}%
1633   0\XINT_div_sub_r
1634   \krof
1635 }%
1636 \def\XINT_div_sub_r #1!%
1637 {%
1638   -\ifnum 0#1=\xint_c_ 1\else2\fi\relax
1639 }%

```

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.

```

1640 \def\XINT_sdiv_out #1;!#2!%
1641   {\expandafter
1642   {\romannumeral0\XINT_unsep_cuzsmall
1643   #1\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax}%
1644   {#2}}%

```

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

```

1645 \def\XINT_smalldivx_a #1\xint:1#2!1#3!%
1646 {%
1647   \expandafter\XINT_smalldivx_b
1648   \the\numexpr (#3+#1)/#2-\xint_c_i!#1\xint:#2!#3!%
1649 }%
1650 \def\XINT_smalldivx_b #1#2!%

```



```

1651 {%
1652     \if0#1\else
1653         \xint_c_x^viii+#1#2\xint_afterfi{\expandafter!\the\numexpr}\fi
1654     \XINT_smallldiv_c #1#2!%
1655 }%
1656 \def\XINT_smallldiv_c #1!#2\xint:#3!#4!%
1657 {%
1658     \expandafter\XINT_smallldiv_d\the\numexpr #4-#1*#3!#2\xint:#3!%
1659 }%

```

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.

```

1660 \def\XINT_smallldiv_d #1!#2!1#3#4!%
1661 {%
1662     \xint_gob_til_sc #3\XINT_smallldiv_end ;%
1663     \XINT_smallldiv_e #1!#2!1#3#4!%
1664 }%
1665 \def\XINT_smallldiv_end;\XINT_smallldiv_e #1!#2!1;!{1!;!#1!}%

```

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

```

1666 \def\XINT_smallldiv_e #1!#2\xint:#3!%
1667 {%
1668     \expandafter\XINT_smallldiv_f\the\numexpr
1669     \xint_c_xi_e_viii_mone+#1*\xint_c_x^viii/#3!#2\xint:#3!#1!%
1670 }%
1671 \def\XINT_smallldiv_f 1#1#2#3#4#5#6!#7\xint:#8!%
1672 {%
1673     \xint_gob_til_zero #1\XINT_smallldiv_fz 0%
1674     \expandafter\XINT_smallldiv_g
1675     \the\numexpr\XINT_minimul_a #2#3#4#5\xint:#6!#8!#2#3#4#5#6!#7\xint:#8!%
1676 }%
1677 \def\XINT_smallldiv_fz 0%
1678     \expandafter\XINT_smallldiv_g\the\numexpr\XINT_minimul_a
1679     9999\xint:9999!#1!99999999!#2!0!1#3!%
1680 {%
1681     \XINT_smallldiv_i \xint:#3!\xint_c_!#2!%
1682 }%
1683 \def\XINT_smallldiv_g 1#1!1#2!#3!#4!#5!#6!%
1684 {%
1685     \expandafter\XINT_smallldiv_h\the\numexpr 1#6-#1\xint:#2!#5!#3!#4!%
1686 }%
1687 \def\XINT_smallldiv_h 1#1#2\xint:#3!#4!%
1688 {%
1689     \expandafter\XINT_smallldiv_i\the\numexpr #4-#3+#1-\xint_c_i\xint:#2!%
1690 }%
1691 \def\XINT_smallldiv_i #1\xint:#2!#3!#4\xint:#5!%
1692 {%
1693     \expandafter\XINT_smallldiv_j\the\numexpr (#1#2+#4)/#5-\xint_c_i!#3!#1#2!#4\xint:#5!%
1694 }%

```

```

1695 \def\XINT_smallldiv_j #1!#2!%
1696 {%
1697     \xint_c_x^viii+#1+#2\expandafter!\the\numexpr\XINT_smallldiv_k
1698     #1!%
1699 }%

```

On boucle vers \XINT_smallldiv_d.

```

1700 \def\XINT_smallldiv_k #1!#2!#3\xint:#4!%
1701 {%
1702     \expandafter\XINT_smallldiv_d\the\numexpr #2-#1*#4!#3\xint:#4!%
1703 }%

```

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>.

```

1704 \def\XINT_div_mini #1\xint:#2!#3!%
1705 {%
1706     \expandafter\XINT_div_mini_a\the\numexpr
1707     \xint_c_xi_e_viii_mone+#3*\xint_c_x^viii/#1!#1\xint:#2!#3!%
1708 }%

```

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.

```

1709 \def\XINT_div_mini_a 1#1#2#3#4#5#6!#7\xint:#8!%
1710 {%
1711     \xint_gob_til_zero #1\XINT_div_mini_w 0%
1712     \expandafter\XINT_div_mini_b
1713     \the\numexpr\XINT_minimul_a #2#3#4#5\xint:#6!#7!#2#3#4#5#6!#7\xint:#8!%
1714 }%
1715 \def\XINT_div_mini_w 0%
1716     \expandafter\XINT_div_mini_b\the\numexpr\XINT_minimul_a
1717     9999\xint:9999!#1!99999999!#2\xint:#3!00000000!#4!%
1718 {%
1719     \xint_c_x^viii_mone+(#4+#3)/#2!%
1720 }%
1721 \def\XINT_div_mini_b 1#1!1#2!#3!#4!#5!#6!%
1722 {%
1723     \expandafter\XINT_div_mini_c
1724     \the\numexpr 1#6-#1\xint:#2!#5!#3!#4!%
1725 }%
1726 \def\XINT_div_mini_c 1#1#2\xint:#3!#4!%
1727 {%
1728     \expandafter\XINT_div_mini_d
1729     \the\numexpr #4-#3+#1-\xint_c_i\xint:#2!%

```

```

1730 }%
1731 \def\XINT_div_mini_d #1\xint:#2!#3!#4\xint:#5!%
1732 {%
1733     \xint_c_x^viii_mone+#3+(#1#2+#5)/#4!%
1734 }%

```

Derived arithmetic

4.38 \xintiQuo, \xintiRem

```

1735 \def\xintiQuo {\romannumeral0\xintiQuo }%
1736 \def\xintiRem {\romannumeral0\xintiRem }%
1737 \def\xintiQuo
1738     {\expandafter\xint_stop_atfirstoftwo\romannumeral0\xintiDivision }%
1739 \def\xintiRem
1740     {\expandafter\xint_stop_atsecondoftwo\romannumeral0\xintiDivision }%

```

4.39 \xintiDivRound

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

1.2l: \xintiDivRound made robust against non terminated input.

```

1741 \def\xintiDivRound {\romannumeral0\xintiDivRound }%
1742 \def\xintiDivRound #1{\expandafter\XINT_iidivround\romannumeral`&&@#1\xint:}%
1743 \def\XINT_iidivround #1#2\xint:#3%
1744     {\expandafter\XINT_iidivround_a\expandafter #1\romannumeral`&&@#3\xint:#2\xint:}%
1745 \def\XINT_iidivround_a #1#2% #1 de A, #2 de B.
1746 {%
1747     \if0#2\xint_dothis{\XINT_iidivround_divbyzero#1#2}\fi
1748     \if0#1\xint_dothis\XINT_iidivround_aiszero\fi
1749     \if-#2\xint_dothis{\XINT_iidivround_bneg #1}\fi
1750     \xint_orthat{\XINT_iidivround_bpos #1#2}%
1751 }%
1752 \def\XINT_iidivround_divbyzero #1#2#3\xint:#4\xint:
1753     {\XINT_signalcondition{DivisionByZero}{Division of #1#4 by #2#3}{0}}%
1754 \def\XINT_iidivround_aiszero #1\xint:#2\xint:{ 0}%
1755 \def\XINT_iidivround_bpos #1%
1756 {%
1757     \xint_UDsignfork
1758         #1{\xintiopp\XINT_iidivround_pos {}}%
1759         -{\XINT_iidivround_pos #1}%
1760     \krof
1761 }%
1762 \def\XINT_iidivround_bneg #1%
1763 {%
1764     \xint_UDsignfork
1765         #1{\XINT_iidivround_pos {}}%
1766         -{\xintiopp\XINT_iidivround_pos #1}%
1767     \krof
1768 }%
1769 \def\XINT_iidivround_pos #1#2\xint:#3\xint:
1770 {%
1771     \expandafter\expandafter\expandafter\XINT_dsrr

```

```

1772 \expandafter\xint_firstoftwo
1773 \romannumeral0\XINT_div_prepare {#2}{#1#30}%
1774 \xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax
1775 }%

```

4.40 \xintiDivTrunc

1.21: \xintiDivTrunc made robust against non terminated input.

```

1776 \def\xintiDivTrunc {\romannumeral0\xintiDivTrunc }%
1777 \def\xintiDivTrunc #1{\expandafter\XINT_iDivTrunc\romannumeral`&&#1\xint:}%
1778 \def\XINT_iDivTrunc #1#2\xint:#3{\expandafter\XINT_iDivTrunc_a\expandafter #1%
1779 \romannumeral`&&#3\xint:#2\xint:}%
1780 \def\XINT_iDivTrunc_a #1#2# #1 de A, #2 de B.
1781 {%
1782 \if0#2\xint_dothis{\XINT_iDivTrunc_divbyzero#1#2}\fi
1783 \if0#1\xint_dothis\XINT_iDivTrunc_aiszero\fi
1784 \if-#2\xint_dothis{\XINT_iDivTrunc_bneg #1}\fi
1785 \xint_orthat{\XINT_iDivTrunc_bpos #1#2}%
1786 }%

```

Attention to not move DivRound code beyond that point.

```

1787 \let\XINT_iDivTrunc_divbyzero\XINT_iDivRound_divbyzero
1788 \let\XINT_iDivTrunc_aiszero \XINT_iDivRound_aiszero
1789 \def\XINT_iDivTrunc_bpos #1%
1790 {%
1791 \xint_UDsignfork
1792 #1{\xintiOpp\XINT_iDivTrunc_pos {}}%
1793 -{\XINT_iDivTrunc_pos #1}%
1794 \krof
1795 }%
1796 \def\XINT_iDivTrunc_bneg #1%
1797 {%
1798 \xint_UDsignfork
1799 #1{\XINT_iDivTrunc_pos {}}%
1800 -{\xintiOpp\XINT_iDivTrunc_pos #1}%
1801 \krof
1802 }%
1803 \def\XINT_iDivTrunc_pos #1#2\xint:#3\xint:
1804 {\expandafter\xint_stop_atfirstoftwo
1805 \romannumeral0\XINT_div_prepare {#2}{#1#3}}%

```

4.41 \xintiModTrunc

Renamed from \xintiMod to \xintiModTrunc at 1.2p.

```

1806 \def\xintiModTrunc {\romannumeral0\xintiModTrunc }%
1807 \def\xintiModTrunc #1{\expandafter\XINT_iModTrunc\romannumeral`&&#1\xint:}%
1808 \def\XINT_iModTrunc #1#2\xint:#3{\expandafter\XINT_iModTrunc_a\expandafter #1%
1809 \romannumeral`&&#3\xint:#2\xint:}%
1810 \def\XINT_iModTrunc_a #1#2# #1 de A, #2 de B.
1811 {%

```

```

1812 \if0#2\xint_dothis{\XINT_iimodtrunc_divbyzero#1#2}\fi
1813 \if0#1\xint_dothis\XINT_iimodtrunc_aiszero\fi
1814 \if-#2\xint_dothis{\XINT_iimodtrunc_bneg #1}\fi
1815 \xint_orthat{\XINT_iimodtrunc_bpos #1#2}%
1816 }%

```

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

```

1817 \let\XINT_iimodtrunc_divbyzero\XINT_iidivround_divbyzero
1818 \let\XINT_iimodtrunc_aiszero \XINT_iidivround_aiszero
1819 \def\XINT_iimodtrunc_bpos #1%
1820 {%
1821 \xint_UDsignfork
1822 #1{\xintiiopt\XINT_iimodtrunc_pos {}}%
1823 -{\XINT_iimodtrunc_pos #1}%
1824 \krof
1825 }%
1826 \def\XINT_iimodtrunc_bneg #1%
1827 {%
1828 \xint_UDsignfork
1829 #1{\xintiiopt\XINT_iimodtrunc_pos {}}%
1830 -{\XINT_iimodtrunc_pos #1}%
1831 \krof
1832 }%
1833 \def\XINT_iimodtrunc_pos #1#2\xint:#3\xint:
1834 {\expandafter\xint_stop_atsecondoftwo\romannumeral0\XINT_div_prepare
1835 {#2}{#1#3}}%

```

4.42 \xintiDivMod

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

```

1836 \def\xintiDivMod {\romannumeral0\xintiDivMod }%
1837 \def\xintiDivMod #1{\expandafter\XINT_iidivmod\romannumeral`&&#1\xint:}%
1838 \def\XINT_iidivmod #1#2\xint:#3{\expandafter\XINT_iidivmod_a\expandafter #1%
1839 \romannumeral`&&#3\xint:#2\xint:}%
1840 \def\XINT_iidivmod_a #1#2% #1 de A, #2 de B.
1841 {%
1842 \if0#2\xint_dothis{\XINT_iidivmod_divbyzero#1#2}\fi
1843 \if0#1\xint_dothis\XINT_iidivmod_aiszero\fi
1844 \if-#2\xint_dothis{\XINT_iidivmod_bneg #1}\fi
1845 \xint_orthat{\XINT_iidivmod_bpos #1#2}%
1846 }%
1847 \def\XINT_iidivmod_divbyzero #1#2\xint:#3\xint:
1848 {%
1849 \XINT_signalcondition{DivisionByZero}{Division by #2 of #1#3}{}%
1850 {{0}{0}}% à revoir...
1851 }%
1852 \def\XINT_iidivmod_aiszero #1\xint:#2\xint:{{0}{0}}%
1853 \def\XINT_iidivmod_bneg #1%
1854 {%
1855 \expandafter\XINT_iidivmod_bneg_finish

```

```

1856 \romannumeral0\xint_UDsignfork
1857 #1{\XINT_iidivmod_bpos {}}%
1858 -{\XINT_iidivmod_bpos {-#1}}%
1859 \krof
1860 }%
1861 \def\XINT_iidivmod_bneg_finish#1#2%
1862 {%
1863 \expandafter\xint_exchangetwo_keepbraces\expandafter
1864 {\romannumeral0\xintiiopt#2}{#1}%
1865 }%
1866 \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.

```

1867 \def\xintiiDivFloor {\romannumeral0\xintiividivfloor}%
1868 \def\xintiividivfloor {\expandafter\xint_stop_atfirstoftwo
1869 \romannumeral0\xintiividivmod}%

```

4.44 \xintiiMod

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

```

1870 \def\xintiiMod {\romannumeral0\xintiimod}%
1871 \def\xintiimod {\expandafter\xint_stop_atsecondoftwo
1872 \romannumeral0\xintiividivmod}%

```

4.45 \xintiiSqr

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

```

1873 \def\xintiiSqr {\romannumeral0\xintiisqr }%
1874 \def\xintiisqr #1%
1875 {%
1876 \expandafter\XINT_sqr\romannumeral0\xintiiaabs{#1}\xint:
1877 }%
1878 \def\XINT_sqr #1\xint:
1879 {%
1880 \expandafter\XINT_sqr_a
1881 \romannumeral0\expandafter\XINT_sepandrev_andcount
1882 \romannumeral0\XINT_zeroes_forviii #1\R\R\R\R\R\R\R\R{10}0000001\W
1883 #1\XINT_rsepybviii_end_A 2345678%
1884 \XINT_rsepybviii_end_B 2345678\relax\xint_c_ii\xint_c_i
1885 \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
1886 \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
1887 \xint:
1888 }%

```

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


```

1927      0-\XINT_pow_Apos #2}%
1928      \krof {#1}%
1929 }%
1930 \def\XINT_pow_AisZero #1#2\xint:
1931 {%
1932      \ifcase\XINT_cntSgn #1\xint:
1933          \xint_afterfi { 1}%
1934      \or
1935          \xint_afterfi { 0}%
1936      \else
1937          \xint_afterfi
1938          {\XINT_signalcondition{DivisionByZero}{Zero to power #1}{0}}%
1939      \fi
1940 }%
1941 \def\XINT_pow_Aneg #1%
1942 {%
1943      \ifodd #1
1944          \expandafter\XINT_opp\romannumeral0%
1945      \fi
1946      \XINT_pow_Apos {}{#1}%
1947 }%
1948 \def\XINT_pow_Apos #1#2{\XINT_pow_Apos_a {#2}#1}%
1949 \def\XINT_pow_Apos_a #1#2#3%
1950 {%
1951      \xint_gob_til_xint: #3\XINT_pow_Apos_short\xint:
1952      \XINT_pow_AatleastTwo {#1}#2#3%
1953 }%
1954 \def\XINT_pow_Apos_short\xint:\XINT_pow_AatleastTwo #1#2\xint:
1955 {%
1956      \ifcase #2
1957          \xintError:thiscannothappen
1958      \or \expandafter\XINT_pow_AisOne
1959      \else\expandafter\XINT_pow_AatleastTwo
1960      \fi {#1}#2\xint:
1961 }%
1962 \def\XINT_pow_AisOne #1\xint:{ 1}%
1963 \def\XINT_pow_AatleastTwo #1%
1964 {%
1965      \ifcase\XINT_cntSgn #1\xint:
1966          \expandafter\XINT_pow_BisZero
1967      \or
1968          \expandafter\XINT_pow_I_in
1969      \else
1970          \expandafter\XINT_pow_BisNegative
1971      \fi
1972      {#1}%
1973 }%
1974 \def\XINT_pow_BisNegative #1\xint:{\XINT_signalcondition{Underflow}{Inverse power
1975      can not be represented by an integer}{0}}%
1976 \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).


```

1977 \def\XINT_pow_I_in #1#2\xint:
1978 {%
1979     \expandafter\XINT_pow_I_loop
1980     \the\numexpr #1\expandafter\xint:%
1981     \romannumeral0\expandafter\XINT_sepandrev
1982     \romannumeral0\XINT_zeroes_forviii #2\R\R\R\R\R\R\R\{10}0000001\W
1983     #2\XINT_rsepybyviii_end_A 2345678%
1984     \XINT_rsepybyviii_end_B 2345678\relax XX%
1985     \R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\W
1986     1;!\W
1987     1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1988 }%
1989 \def\XINT_pow_I_loop #1\xint:%
1990 {%
1991     \ifnum #1 = \xint_c_i\expandafter\XINT_pow_I_exit\fi
1992     \ifodd #1
1993         \expandafter\XINT_pow_II_in
1994     \else
1995         \expandafter\XINT_pow_I_squareit
1996     \fi #1\xint:%
1997 }%
1998 \def\XINT_pow_I_exit \ifodd #1\fi #2\xint:#3\W {\XINT_mul_out #3}%

```

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

```

1999 \def\XINT_pow_I_squareit #1\xint:#2\W%
2000 {%
2001     \expandafter\XINT_pow_I_loop
2002     \the\numexpr #1/\xint_c_ii\expandafter\xint:%
2003     \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W
2004 }%
2005 \def\XINT_pow_mulbutcheckifsmall #1!1#2%
2006 {%
2007     \xint_gob_til_sc #2\XINT_pow_mul_small;%
2008     \XINT_mul_loop 100000000!1;!\W #1!1#2%
2009 }%
2010 \def\XINT_pow_mul_small;\XINT_mul_loop
2011     100000000!1;!\W 1#1!1;!\W
2012 {%
2013     \XINT_smallmul 1#1!%
2014 }%
2015 \def\XINT_pow_II_in #1\xint:#2\W
2016 {%
2017     \expandafter\XINT_pow_II_loop
2018     \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter\xint:%
2019     \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W #2\W
2020 }%
2021 \def\XINT_pow_II_loop #1\xint:%
2022 {%
2023     \ifnum #1 = \xint_c_i\expandafter\XINT_pow_II_exit\fi
2024     \ifodd #1

```

```

2025     \expandafter\XINT_pow_II_odda
2026   \else
2027     \expandafter\XINT_pow_II_even
2028   \fi #1\xint:%
2029}%
2030\def\XINT_pow_II_exit\ifodd #1\fi #2\xint:#3\W #4\W
2031{%
2032  \expandafter\XINT_mul_out
2033  \the\numexpr\XINT_pow_mulbutcheckifsmall #4\W #3%
2034}%
2035\def\XINT_pow_II_even #1\xint:#2\W
2036{%
2037  \expandafter\XINT_pow_II_loop
2038  \the\numexpr #1/\xint_c_ii\expandafter\xint:%
2039  \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W
2040}%
2041\def\XINT_pow_II_odda #1\xint:#2\W #3\W
2042{%
2043  \expandafter\XINT_pow_II_oddb
2044  \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter\xint:%
2045  \the\numexpr\XINT_pow_mulbutcheckifsmall #3\W #2\W #2\W
2046}%
2047\def\XINT_pow_II_oddb #1\xint:#2\W #3\W
2048{%
2049  \expandafter\XINT_pow_II_loop
2050  \the\numexpr #1\expandafter\xint:%
2051  \the\numexpr\XINT_pow_mulbutcheckifsmall #3\W #3\W #2\W
2052}%

```

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}%
\def\tfac #1{\expandafter\XINT_mul_out
  \romannumeral-0\ufac {1}{#1}1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W}%
\def\ufac #1#2{\ifcase\numexpr#2-#1\relax
  \expandafter\xint_firstofthree

```

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

```

\or
\expandafter\xint_secondofthree
\else
\expandafter\xint_thirdofthree
\fi
{\the\numexpr\xint_c_x^viii+#1!1;!}%
{\the\numexpr\xint_c_x^viii+#1*#2!1;!}%
{\expandafter\vfac\the\numexpr (#1+#2)/\xint_c_ii.#1.#2.}%
}%
\def\vfac #1.#2.#3.%
{%
\expandafter
\vfac\expandafter
{\romannumeral-`0\expandafter
\ufac\expandafter{\the\numexpr #1+\xint_c_i}{#3}}%
{\ufac {#2}{#1}}%
}%
\def\wfac #1#2{\expandafter\zfac\romannumeral-`0#2\W #1}%
\def\zfac {\the\numexpr\XINT_mul_loop 100000000!1;! \W }% core multiplication...
\catcode`_ 8
\catcode`^ 7

```

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

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

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

```

2053 \def\xintiifac {\romannumeral0\xintiifac }%
2054 \def\xintiifac #1{\expandafter\XINT_fac_fork\the\numexpr#1.%}
2055 \def\XINT_fac_fork #1#2.%
2056 {%
2057     \xint_UDzerominusfork
2058     #1-\XINT_fac_zero
2059     0#1\XINT_fac_neg
2060     0-\XINT_fac_checksize
2061     \krof #1#2.%
2062 }%
2063 \def\XINT_fac_zero #1.{ 1}%
2064 \def\XINT_fac_neg #1.{\XINT_signalcondition{InvalidOperation}{Factorial of
2065     negative: (#1)!}}{0}}%
2066 \def\XINT_fac_checksize #1.%
2067 {%
2068     \ifnum #1>\xint_c_x^iv \xint_dothis{\XINT_fac_toobig #1.}\fi
2069     \ifnum #1>465 \xint_dothis{\XINT_fac_bigloop_a #1.}\fi
2070     \ifnum #1>101 \xint_dothis{\XINT_fac_medloop_a #1.\XINT_mul_out}\fi
2071         \xint_orthat{\XINT_fac_smallloop_a #1.\XINT_mul_out}%
2072     1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
2073 }%
2074 \def\XINT_fac_toobig #1.#2\W{\XINT_signalcondition{InvalidOperation}{Factorial
2075     of too big argument: #1 > 10000}}{0}}%
2076 \def\XINT_fac_bigloop_a #1.%

```

```

2077 {%
2078   \expandafter\XINT_fac_bigloop_b \the\numexpr
2079   #1+\xint_c_i-\xint_c_ii*((#1-464)/\xint_c_ii).#1.%
2080 }%
2081 \def\XINT_fac_bigloop_b #1.#2.%
2082 {%
2083   \expandafter\XINT_fac_medloop_a
2084   \the\numexpr #1-\xint_c_i.{\XINT_fac_bigloop_loop #1.#2.}%
2085 }%
2086 \def\XINT_fac_bigloop_loop #1.#2.%
2087 {%
2088   \ifnum #1>#2 \expandafter\XINT_fac_bigloop_exit\fi
2089   \expandafter\XINT_fac_bigloop_loop
2090   \the\numexpr #1+\xint_c_ii\expandafter.%
2091   \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_bigloop_mul #1!%
2092 }%
2093 \def\XINT_fac_bigloop_exit #1!{\XINT_mul_out}%
2094 \def\XINT_fac_bigloop_mul #1!%
2095 {%
2096   \expandafter\XINT_smallmul
2097   \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2098 }%
2099 \def\XINT_fac_medloop_a #1.%
2100 {%
2101   \expandafter\XINT_fac_medloop_b
2102   \the\numexpr #1+\xint_c_i-\xint_c_iii*((#1-100)/\xint_c_iii).#1.%
2103 }%
2104 \def\XINT_fac_medloop_b #1.#2.%
2105 {%
2106   \expandafter\XINT_fac_smallloop_a
2107   \the\numexpr #1-\xint_c_i.{\XINT_fac_medloop_loop #1.#2.}%
2108 }%
2109 \def\XINT_fac_medloop_loop #1.#2.%
2110 {%
2111   \ifnum #1>#2 \expandafter\XINT_fac_loop_exit\fi
2112   \expandafter\XINT_fac_medloop_loop
2113   \the\numexpr #1+\xint_c_iii\expandafter.%
2114   \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_medloop_mul #1!%
2115 }%
2116 \def\XINT_fac_medloop_mul #1!%
2117 {%
2118   \expandafter\XINT_smallmul
2119   \the\numexpr
2120   \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2121 }%
2122 \def\XINT_fac_smallloop_a #1.%
2123 {%
2124   \csname
2125     XINT_fac_smallloop_\the\numexpr #1-\xint_c_iv*(#1/\xint_c_iv)\relax
2126   \endcsname #1.%
2127 }%
2128 \expandafter\def\csname XINT_fac_smallloop_1\endcsname #1.%

```

```

2129 {%
2130     \XINT_fac_smallloop_loop 2.#1.1000000001!1;!%
2131 }%
2132 \expandafter\def\csname XINT_fac_smallloop_-2\endcsname #1.%
2133 {%
2134     \XINT_fac_smallloop_loop 3.#1.1000000002!1;!%
2135 }%
2136 \expandafter\def\csname XINT_fac_smallloop_-1\endcsname #1.%
2137 {%
2138     \XINT_fac_smallloop_loop 4.#1.1000000006!1;!%
2139 }%
2140 \expandafter\def\csname XINT_fac_smallloop_0\endcsname #1.%
2141 {%
2142     \XINT_fac_smallloop_loop 5.#1.10000000024!1;!%
2143 }%
2144 \def\XINT_fac_smallloop_loop #1.#2.%
2145 {%
2146     \ifnum #1>#2 \expandafter\XINT_fac_loop_exit\fi
2147     \expandafter\XINT_fac_smallloop_loop
2148     \the\numexpr #1+\xint_c_iv\expandafter.%
2149     \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_smallloop_mul #1!%
2150 }%
2151 \def\XINT_fac_smallloop_mul #1!%
2152 {%
2153     \expandafter\XINT_smallmul
2154     \the\numexpr
2155         \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2156 }%
2157 \def\XINT_fac_loop_exit #1!#2;!#3{#3#2;!}%

```

4.48 \XINT_useiimessage

1.2o

```

2158 \def\XINT_useiimessage #1% used in LaTeX only
2159 {%
2160     \XINT_ifFlagRaised {#1}%
2161     {\@backslashchar#1
2162         (load xintfrac or use \@backslashchar xintii\xint_gobble_iv#1!)\MessageBreak}%
2163     }%
2164 }%
2165 \XINT_restorecatcodes_endinput%

```

5 Package *xint* implementation

.1	Package identification	119	.35	<code>\xintiiifNotZero</code>	132
.2	More token management	119	.36	<code>\xintiiifOne</code>	132
.3	(WIP) A constant needed by <code>\xintRandomDigits</code> et al.	119	.37	<code>\xintiiifOdd</code>	132
.4	<code>\xintLen</code> , <code>\xintiLen</code>	120	.38	<code>\xintifTrueAelseB</code> , <code>\xintifFalseAelseB</code>	132
.5	<code>\xintiiLogTen</code>	120	.39	<code>\xintIsTrue</code> , <code>\xintIsFalse</code>	133
.6	<code>\xintReverseDigits</code>	120	.40	<code>\xintNOT</code>	133
.7	<code>\xintiiE</code>	121	.41	<code>\xintAND</code> , <code>\xintOR</code> , <code>\xintXOR</code>	133
.8	<code>\xintDecSplit</code>	122	.42	<code>\xintANDof</code>	133
.9	<code>\xintDecSplitL</code>	123	.43	<code>\xintORof</code>	134
.10	<code>\xintDecSplitR</code>	124	.44	<code>\xintXORof</code>	134
.11	<code>\xintDSHr</code>	124	.45	<code>\xintiiMax</code>	134
.12	<code>\xintDSH</code>	125	.46	<code>\xintiiMin</code>	135
.13	<code>\xintDSx</code>	125	.47	<code>\xintiiMaxof</code>	136
.14	<code>\xintiiEq</code>	127	.48	<code>\xintiiMinof</code>	137
.15	<code>\xintiiNotEq</code>	127	.49	<code>\xintiiSum</code>	137
.16	<code>\xintiiGeq</code>	127	.50	<code>\xintiiPrd</code>	138
.17	<code>\xintiiGt</code>	128	.51	<code>\xintiiSquareRoot</code>	139
.18	<code>\xintiiLt</code>	128	.52	<code>\xintiiSqrt</code> , <code>\xintiiSqrtR</code>	146
.19	<code>\xintiiGtorEq</code>	128	.53	<code>\xintiiBinomial</code>	146
.20	<code>\xintiiLtorEq</code>	128	.54	<code>\xintiiPFactorial</code>	152
.21	<code>\xintiiIsZero</code>	128	.55	<code>\xintBool</code> , <code>\xintToggle</code>	155
.22	<code>\xintiiIsNotZero</code>	128	.56	<code>\xintiiGCD</code>	155
.23	<code>\xintiiIsOne</code>	128	.57	<code>\xintiiLCM</code>	156
.24	<code>\xintiiOdd</code>	129	.58	<code>\xintiiGCDof</code>	156
.25	<code>\xintiiEven</code>	129	.59	<code>\xintiiLCMof</code>	157
.26	<code>\xintiiMON</code>	129	.60	(WIP) <code>\xintRandomDigits</code>	157
.27	<code>\xintiiMMON</code>	129	.61	(WIP) <code>\XINT_eightrandomdigits</code> , <code>\xintEightRandomDigits</code>	157
.28	<code>\xintSgnFork</code>	130	.62	(WIP) <code>\xintRandBit</code>	158
.29	<code>\xintiiifSgn</code>	130	.63	(WIP) <code>\xintXRandomDigits</code>	158
.30	<code>\xintiiifCmp</code>	130	.64	(WIP) <code>\xintiiRandRangeAtoB</code>	158
.31	<code>\xintiiifEq</code>	131	.65	(WIP) <code>\xintiiRandRange</code>	159
.32	<code>\xintiiifGt</code>	131	.66	(WIP) Adjustments for engines without <code>uniformdeviate</code> primitive	160
.33	<code>\xintiiifLt</code>	131			
.34	<code>\xintiiifZero</code>	131			

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 \beginingroup\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 [2020/02/19 v1.4a 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:\xint:
65   \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
66   \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye\relax
67 }}\xintilen{ }%
68 \def\xintLen {\romannumeral0\xintlen }%
69 \let\xintlen\xintilen

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

5.5 \xintiLogTen

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

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

```

5.6 \xintReverseDigits

1.2.

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

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

This macro is currently not used elsewhere in xint code.


```

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

```

5.7 \xintiiE

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

```

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

```

```
130 \krof #1%
131 }%
```

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

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

5.8 \xintDecSplit

DECIMAL SPLIT

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

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

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

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

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

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

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

```

164 \expandafter\XINT_split_fromleft_clean
165 \the\numexpr1#2#3#4#5#6#7#8#9\expandafter
166 \XINT_split_fromleft_a\the\numexpr\xint_c_viii-#1.%
167 }%
168 \def\XINT_split_fromleft_end_a #1.%
169 {%
170 \expandafter\XINT_split_fromleft_clean
171 \the\numexpr1\csname XINT_split_fromleft_end#1\endcsname
172 }%
173 \def\XINT_split_fromleft_clean 1{ }%
174 \expandafter\def\csname XINT_split_fromleft_end7\endcsname #1%
175 {#1\XINT_split_fromleft_end_b}%
176 \expandafter\def\csname XINT_split_fromleft_end6\endcsname #1#2%
177 {#1#2\XINT_split_fromleft_end_b}%
178 \expandafter\def\csname XINT_split_fromleft_end5\endcsname #1#2#3%
179 {#1#2#3\XINT_split_fromleft_end_b}%
180 \expandafter\def\csname XINT_split_fromleft_end4\endcsname #1#2#3#4%
181 {#1#2#3#4\XINT_split_fromleft_end_b}%
182 \expandafter\def\csname XINT_split_fromleft_end3\endcsname #1#2#3#4#5%
183 {#1#2#3#4#5\XINT_split_fromleft_end_b}%
184 \expandafter\def\csname XINT_split_fromleft_end2\endcsname #1#2#3#4#5#6%
185 {#1#2#3#4#5#6\XINT_split_fromleft_end_b}%
186 \expandafter\def\csname XINT_split_fromleft_end1\endcsname #1#2#3#4#5#6#7%
187 {#1#2#3#4#5#6#7\XINT_split_fromleft_end_b}%
188 \expandafter\def\csname XINT_split_fromleft_end0\endcsname #1#2#3#4#5#6#7#8%
189 {#1#2#3#4#5#6#7#8\XINT_split_fromleft_end_b}%

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

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

```

5.9 \xintDecSplitL

```

208 \def\xintDecSplitL {\romannumeral0\xintdecsplitl }%
209 \def\xintdecsplitl #1#2%
210 {%

```

```

211 \expandafter\XINT_splitl_finish
212 \romannumeral0\expandafter\XINT_split_xfork
213 \the\numexpr #1\expandafter.\romannumeral`&&@#2%
214 \xint_bye2345678\xint_bye..%
215 }%
216 \def\XINT_splitl_finish #1.#2.{ #1}%

```

5.10 \xintDecSplitR

```

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

```

5.11 \xintDSHr

DECIMAL SHIFTS \xintDSH {x}{A}
 si $x \leq 0$, fait $A \rightarrow A \cdot 10^{|x|}$. si $x > 0$, et $A \geq 0$, fait $A \rightarrow \text{quo}(A, 10^x)$
 si $x > 0$, et $A < 0$, fait $A \rightarrow -\text{quo}(-A, 10^x)$
 (donc pour $x > 0$ c'est comme DSR itéré x fois)
 \xintDSHr donne le 'reste' (si $x \leq 0$ donne zéro).
 Badly named macros.
 Rewritten for 1.2i, this was old code and \xintDSx has changed interface.

```

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

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

229 \expandafter\XINT_dshr_fork\the\numexpr#1\expandafter.\romannumeral`&&@#2;%
230 }%
231 \def\XINT_dshr_fork #1%
232 {%
233 \xint_UDzerominusfork
234 0#1\XINT_dshr_xzeroorneg
235 #1-\XINT_dshr_xzeroorneg
236 0-\XINT_dshr_xpositive
237 \krof #1%
238 }%
239 \def\XINT_dshr_xzeroorneg #1;{ 0}%
240 \def\XINT_dshr_xpositive
241 {%

242 \expandafter\xint_stop_atsecondoftwo\romannumeral0\XINT_dsx_xisPos
243 }%

```

5.12 \xintDSH

```

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

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

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

    \expandafter\xint_stop_atfirstoftwo\romannumeral0\XINT_dsx_xisPos
260 }%

```

5.13 \xintDSx

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

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

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

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

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

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

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

Rewritten for 1.2i, this was old code.

```

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

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

```

```

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

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

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

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

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

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

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

```

```

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

```

5.14 \xintiiEq

no \xintiieq.

```

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

```

5.15 \xintiiNotEq

Pour xintexpr. Pas de version en lowercase.

```

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

```

5.16 \xintiiGeq

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

1.2l made \xintiiGeq robust against non terminated items.

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

This macro should have been called \xintGEq for example.

```

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

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

```

```

355 \def\XINT_geq_secondiszero #1\krof #2#3\xint:#4\xint:{ 1}%
356 \def\XINT_geq_plusminus #1-{\XINT_geq_plusplus #1{}}%
357 \def\XINT_geq_minusplus -#1{\XINT_geq_plusplus {}#1}%
358 \def\XINT_geq_minusminus --{\XINT_geq_plusplus {}{}}%
359 \def\XINT_geq_plusplus
360   {\expandafter\XINT_geq_finish\romannumeral0\XINT_cmp_plusplus}%
361 \def\XINT_geq_finish #1{\if-#1\expandafter\XINT_geq_no
362   \else\expandafter\XINT_geq_yes\fi}%
363 \def\XINT_geq_no 1{ 0}%
364 \def\XINT_geq_yes { 1}%

```

5.17 \xintiiGt

```

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

```

5.18 \xintiiLt

```

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

```

5.19 \xintiiGtorEq

```

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

```

5.20 \xintiiLtorEq

```

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

```

5.21 \xintiiIsZero

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

```

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

```

5.22 \xintiiIsNotZero

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

```

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

```

5.23 \xintiiIsOne

Added in 1.03. 1.09a defines \xintIsOne. 1.1a adds \xintiiIsOne.

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

```

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

```



```

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

```

5.24 \xintiiOdd

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

```

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

```

5.25 \xintiiEven

```

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

```

5.26 \xintiiMON

MINUS ONE TO THE POWER N

```

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

```

5.27 \xintiiMMON

MINUS ONE TO THE POWER N-1

```

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

```

```

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

```

5.28 \xintSgnFork

Expandable three-way fork added in 1.07. The argument #1 must expand to non-self-ending -1,0 or 1. 1.09i with `_thenstop` (now `_stop_at...`).

```

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

```

5.29 \xintiiifSgn

Expandable three-way fork added in 1.09a. Branches expandably depending on whether <0, =0, >0. Choice of branch guaranteed in two steps.

1.09i has `\xint_firstofthreeafterstop` (now `\xint_stop_atfirstofthree`) etc for faster expansion.

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

```

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

```

5.30 \xintiiifCmp

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

```

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

```

5.31 `\xintiiifEq`

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

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

5.32 `\xintiiifGt`

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

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

5.33 `\xintiiifLt`

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

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

5.34 `\xintiiifZero`

Expandable two-way fork added in 1.09a. Branches expandably depending on whether the argument is zero (branch A) or not (branch B). 1.09i restyling. By the way it appears (not thoroughly tested, though) that `\if` tests are faster than `\ifnum` tests. 1.1 adds ii versions.

1.2o deprecates `\xintifZero`.

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

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

5.35 `\xintiiifNotZero`

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

5.36 `\xintiiifOne`

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

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

5.37 `\xintiiifOdd`

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

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

5.38 `\xintifTrueAelseB`, `\xintifFalseAelseB`

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

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

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

```

5.39 \xintIsTrue, \xintIsFalse

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

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

5.40 \xintNOT

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

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

5.41 \xintAND, \xintOR, \xintXOR

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

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

5.42 \xintANDof

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

1.2l made \xintANDof robust against non terminated items.

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

1.4. This macro as well as ORof and XORof were formally not used by xintexpr, which uses comma separated items, but at 1.4 xintexpr uses braced items. And the macros here got slightly refactored and \XINT_ANDof added for usage by xintexpr and the NewExpr hook. For some random reason I decided to use ^ as delimiter this has to do that other macros in xintfrac in same family (such as \xintGCDoF, \xintSum) also use \xint: internally and although not strictly needed having two separate ones clarifies.

```
527 \def\xintANDof {\romannumeral0\xintandof }%
528 \def\xintandof #1{\expandafter\XINT_andof\romannumeral`&&@#1^}%
529 \def\XINT_ANDof {\romannumeral0\XINT_andof}%
530 \def\XINT_andof #1%
531 {%
532     \xint_gob_til_^ #1\XINT_andof_yes ^%
533     \xintiiifNotZero{#1}\XINT_andof\XINT_andof_no
```

```
534 }%
535 \def\XINT_andof_no #1^{ 0}%
536 \def\XINT_andof_yes ^#1\XINT_andof_no{ 1}%
```

5.43 \xintORof

New with 1.09a. Works also with an empty list. Empty items however are not accepted.

1.2l made \xintORof robust against non terminated items.

Refactored at 1.4.

```
537 \def\xintORof {\romannumeral0\xintorof }%
538 \def\xintorof #1{\expandafter\XINT_orof\romannumeral`&&@#1^}%
539 \def\XINT_ORof {\romannumeral0\XINT_orof}%
540 \def\XINT_orof #1%
541 {%
542   \xint_gob_til_ ^ #1\XINT_orof_no ^%
543   \xintiiifNotZero{#1}\XINT_orof_yes\XINT_orof
544 }%
545 \def\XINT_orof_yes#1^{ 1}%
546 \def\XINT_orof_no ^#1\XINT_orof{ 0}%
```

5.44 \xintXORof

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

1.2l made \xintXORof robust against non terminated items.

Refactored at 1.4 to use \numexpr (or an \ifnum). I have not tested if more efficient or not or if one can do better without \the. \XINT_XORof for xintexpr matters.

```
547 \def\xintXORof {\romannumeral0\xintxorof }%
548 \def\xintxorof #1{\expandafter\XINT_xorof\romannumeral`&&@#1^}%
549 \def\XINT_XORof {\romannumeral0\XINT_xorof}%
550 \def\XINT_xorof {\if1\the\numexpr\XINT_xorof_a}%
551 \def\XINT_xorof_a #1%
552 {%
553   \xint_gob_til_ ^ #1\XINT_xorof_e ^%
554   \xintiiifNotZero{#1}{-}{ }\XINT_xorof_a
555 }%
556 \def\XINT_xorof_e ^#1\XINT_xorof_a
557 {1\relax\xint_afterfi{ 0}\else\xint_afterfi{ 1}\fi}%
```

5.45 \xintiiMax

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

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

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

```
564 {%
565   \expandafter\XINT_max_fork\romannumeral`&&@#2\xint:#1\xint:
566 }%
```

#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.

```
567 \def\XINT_max_fork #1#2\xint:#3#4\xint:
568 {%
569   \xint_UDsignsfork
570     #1#3\XINT_max_minusminus % A < 0, B < 0
571     #1-\XINT_max_plusminus % B < 0, A >= 0
572     #3-\XINT_max_minusplus % A < 0, B >= 0
573     --{\xint_UDzerosfork
574       #1#3\XINT_max_zerozero % A = B = 0
575       #10\XINT_max_pluszero % B = 0, A > 0
576       #30\XINT_max_zeropplus % A = 0, B > 0
577       00\XINT_max_plusplus % A, B > 0
578     \krof }%
579   \krof
580   #3#1#2\xint:#4\xint:
581   \expandafter\xint_stop_atfirstoftwo
582   \else
583   \expandafter\xint_stop_atsecondoftwo
584   \fi
585   {#3#4}{#1#2}%
586 }%
```

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

```
587 \def\XINT_max_zerozero #1\fi{\xint_stop_atfirstoftwo }%
588 \def\XINT_max_zeropplus #1\fi{\xint_stop_atsecondoftwo }%
589 \def\XINT_max_pluszero #1\fi{\xint_stop_atfirstoftwo }%
590 \def\XINT_max_minusplus #1\fi{\xint_stop_atsecondoftwo }%
591 \def\XINT_max_plusminus #1\fi{\xint_stop_atfirstoftwo }%
592 \def\XINT_max_plusplus
593 {%
594   \if1\romannumeral0\XINT_geq_plusplus
595 }%
```

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.

```
596 \def\XINT_max_minusminus --%
597 {%
598   \unless\if1\romannumeral0\XINT_geq_plusplus{}}%
599 }%
```

5.46 \xintiiMin

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

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

```

600 \def\xintiiMin {\romannumeral0\xintiimin }%
601 \def\xintiimin #1%
602 {%
603   \expandafter\xint_iimin \romannumeral`&&@#1\xint:
604 }%
605 \def\xint_iimin #1\xint:#2%
606 {%
607   \expandafter\XINT_min_fork\romannumeral`&&@#2\xint:#1\xint:
608 }%
609 \def\XINT_min_fork #1#2\xint:#3#4\xint:
610 {%
611   \xint_UDsignsfork
612     #1#3\XINT_min_minusminus % A < 0, B < 0
613     #1-\XINT_min_plusminus % B < 0, A >= 0
614     #3-\XINT_min_minusplus % A < 0, B >= 0
615     --{\xint_UDzerosfork
616       #1#3\XINT_min_zerozero % A = B = 0
617       #10\XINT_min_pluszero % B = 0, A > 0
618       #30\XINT_min_zeropplus % A = 0, B > 0
619       00\XINT_min_plusplus % A, B > 0
620     \krof }%
621   \krof
622   #3#1#2\xint:#4\xint:
623   \expandafter\xint_stop_atsecondoftwo
624   \else
625   \expandafter\xint_stop_atfirstoftwo
626   \fi
627   {#3#4}{#1#2}%
628 }%
629 \def\XINT_min_zerozero #1\fi{\xint_stop_atfirstoftwo }%
630 \def\XINT_min_zeropplus #1\fi{\xint_stop_atfirstoftwo }%
631 \def\XINT_min_pluszero #1\fi{\xint_stop_atsecondoftwo }%
632 \def\XINT_min_minusplus #1\fi{\xint_stop_atfirstoftwo }%
633 \def\XINT_min_plusminus #1\fi{\xint_stop_atsecondoftwo }%
634 \def\XINT_min_plusplus
635 {%
636   \if1\romannumeral0\XINT_geq_plusplus
637 }%
638 \def\XINT_min_minusminus --%
639 {%
640   \unless\if1\romannumeral0\XINT_geq_plusplus{}}%
641 }%

```

5.47 `\xintiiMaxof`

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

NOT compatible with empty list.

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

1.4 refactors code to allow empty argument. For usage by \xintiexpr. Slight deterioration, will come back.

```

642 \def\xintiiMaxof {\romannumeral0\xintiimaxof }%
643 \def\xintiimaxof #1{\expandafter\XINT_iimaxof\romannumeral`&&@#1^}%
644 \def\XINT_iiMaxof{\romannumeral0\XINT_iimaxof}%
645 \def\XINT_iimaxof#1%
646 {%
647   \xint_gob_til_ ^ #1\XINT_iimaxof_empty ^%
648   \expandafter\XINT_iimaxof_loop\romannumeral`&&@#1\xint:
649 }%
650 \def\XINT_iimaxof_empty ^#1\xint:{ 0}%
651 \def\XINT_iimaxof_loop #1\xint:#2%
652 {%
653   \xint_gob_til_ ^ #2\XINT_iimaxof_e ^%
654   \expandafter\XINT_iimaxof_loop\romannumeral0\xintiimax{#1}{#2}\xint:
655 }%
656 \def\XINT_iimaxof_e ^#1\xintiimax #2#3\xint:{ #2}%

```

5.48 \xintiiMinof

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

1.4 refactoring for \xintiexpr matters.

```

657 \def\xintiiMinof {\romannumeral0\xintiiminof }%
658 \def\xintiiminof #1{\expandafter\XINT_iiminof\romannumeral`&&@#1^}%
659 \def\XINT_iiMinof{\romannumeral0\XINT_iiminof}%
660 \def\XINT_iiminof#1%
661 {%
662   \xint_gob_til_ ^ #1\XINT_iiminof_empty ^%
663   \expandafter\XINT_iiminof_loop\romannumeral`&&@#1\xint:
664 }%
665 \def\XINT_iiminof_empty ^#1\xint:{ 0}%
666 \def\XINT_iiminof_loop #1\xint:#2%
667 {%
668   \xint_gob_til_ ^ #2\XINT_iiminof_e ^%
669   \expandafter\XINT_iiminof_loop\romannumeral0\xintiimin{#1}{#2}\xint:
670 }%
671 \def\XINT_iiminof_e ^#1\xintiimin #2#3\xint:{ #2}%

```

5.49 \xintiiSum

\xintiiSum {{a}{b}...{z}} Refactored at 1.4 for matters initially related to xintexpr delimiter choice.

```

672 \def\xintiiSum {\romannumeral0\xintiisum }%
673 \def\xintiisum #1{\expandafter\XINT_iisum\romannumeral`&&@#1^}%
674 \def\XINT_iiSum{\romannumeral0\XINT_iisum}%
675 \def\XINT_iisum #1%
676 {%
677   \expandafter\XINT_iisum_a\romannumeral`&&@#1\xint:
678 }%
679 \def\XINT_iisum_a #1%

```

```

680 {%
681   \xint_gob_til_^ #1\XINT_iisum_empty ^%
682   \XINT_iisum_loop #1%
683 }%
684 \def\XINT_iisum_empty ^#1\xint:{ 0}%

    bad coding as it depends on internal conventions of \XINT_add_nfork

685 \def\XINT_iisum_loop #1#2\xint:#3%
686 {%
687   \expandafter\XINT_iisum_loop_a
688   \expandafter#1\romannumeral`&&@#3\xint:#2\xint:\xint:
689 }%
690 \def\XINT_iisum_loop_a #1#2%
691 {%
692   \xint_gob_til_^ #2\XINT_iisum_loop_end ^%
693   \expandafter\XINT_iisum_loop\romannumeral0\XINT_add_nfork #1#2%
694 }%

    see previous comment!

695 \def\XINT_iisum_loop_end ^#1\XINT_add_nfork #2#3\xint:#4\xint:\xint:{ #2#4}%

```

5.50 \xintiiPrd

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

Macros renamed and refactored (slightly more macros here to supposedly bring micro-gain) at 1.4 to match changes in xintfrac of delimiter, in sync with some usage in xintexpr.

Contrarily to the xintfrac version \xintPrd, this one aborts as soon as it hits a zero value.

```

696 \def\xintiiPrd {\romannumeral0\xintiiprd }%
697 \def\xintiiprd #1{\expandafter\XINT_iiprd\romannumeral`&&@#1^}%
698 \def\XINT_iiprd{\romannumeral0\XINT_iiprd}%

```

The above romannumeral caused f-expansion of the list argument. We f-expand below the first item and each successive items because we do not use \xintiiMul but jump directly into \XINT_mul_nfork.

```

699 \def\XINT_iiprd #1%
700 {%
701   \expandafter\XINT_iiprd_a\romannumeral`&&@#1\xint:
702 }%
703 \def\XINT_iiprd_a #1%
704 {%
705   \xint_gob_til_^ #1\XINT_iiprd_empty ^%
706   \xint_gob_til_zero #1\XINT_iiprd_zero 0%
707   \XINT_iiprd_loop #1%
708 }%
709 \def\XINT_iiprd_empty ^#1\xint:{ 1}%
710 \def\XINT_iiprd_zero 0#1^{ 0}%

```

bad coding as it depends on internal conventions of \XINT_mul_nfork

```

711 \def\XINT_iiprd_loop #1#2\xint:#3%
712 {%
713   \expandafter\XINT_iiprd_loop_a

```

```

714 \expandafter#1\romannumeral`&&@#3\xint:#2\xint:\xint:
715 }%
716 \def\XINT_iiprd_loop_a #1#2%
717 {%
718 \xint_gob_til_ ^ #2\XINT_iiprd_loop_end ^%
719 \xint_gob_til_zero #2\XINT_iiprd_zero 0%
720 \expandafter\XINT_iiprd_loop\romannumeral0\XINT_mul_nfork #1#2%
721 }%

    see previous comment!

722 \def\XINT_iiprd_loop_end ^#1\XINT_mul_nfork #2#3\xint:#4\xint:\xint:{ #2#4}%

```

5.51 \xintiiSquareRoot

First done with 1.08.

1.1 added \xintiiSquareRoot.

1.1a added \xintiiSqrtR.

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

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

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

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

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

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

```

723 \def\xintiiSquareRoot {\romannumeral0\xintiisquareroot }%
724 \def\xintiisquareroot #1{\expandafter\XINT_sqrt_checkin\romannumeral`&&@#1\xint:}%
725 \def\XINT_sqrt_checkin #1%
726 {%
727 \xint_UDzerominusfork
728 #1-\XINT_sqrt_iszero
729 0#1\XINT_sqrt_isneg
730 0-\XINT_sqrt
731 \krof #1%
732 }%
733 \def\XINT_sqrt_iszero #1\xint:{{1}\{1\}}%
734 \def\XINT_sqrt_isneg #1\xint:{\XINT_signalcondition{InvalidOperation}{square
735 root of negative: #1}\{0}\{0}\{0}\}%
736 \def\XINT_sqrt #1\xint:
737 {%

```

```

738 \expandafter\XINT_sqrt_start\romannumeral0\xintlength {#1}.#1.%
739 }%
740 \def\XINT_sqrt_start #1.%
741 {%
742 \ifnum #1<\xint_c_x\xint_dothis\XINT_sqrt_small_a\fi
743 \xint_orthat\XINT_sqrt_big_a #1.%
744 }%
745 \def\XINT_sqrt_small_a #1.{\XINT_sqrt_a #1.\XINT_sqrt_small_d }%
746 \def\XINT_sqrt_big_a #1.{\XINT_sqrt_a #1.\XINT_sqrt_big_d }%
747 \def\XINT_sqrt_a #1.%
748 {%
749 \ifodd #1
750 \expandafter\XINT_sqrt_b0
751 \else
752 \expandafter\XINT_sqrt_bE
753 \fi
754 #1.%
755 }%

756 \def\XINT_sqrt_bE #1.#2#3#4%
757 {%
758 \XINT_sqrt_c {#3#4}#2{#1}#3#4%
759 }%

760 \def\XINT_sqrt_b0 #1.#2#3%
761 {%
762 \XINT_sqrt_c #3#2{#1}#3%
763 }%

764 \def\XINT_sqrt_c #1#2%
765 {%
766 \expandafter #2%
767 \the\numexpr \ifnum #1>\xint_c_ii
768 \ifnum #1>\xint_c_vi
769 \ifnum #1>12 \ifnum #1>20 \ifnum #1>30
770 \ifnum #1>42 \ifnum #1>56 \ifnum #1>72
771 \ifnum #1>90
772 10\else 9\fi \else 8\fi \else 7\fi \else 6\fi \else 5\fi
773 \else 4\fi \else 3\fi \else 2\fi \else 1\fi .%
774 }%

775 \def\XINT_sqrt_small_d #1.#2%
776 {%
777 \expandafter\XINT_sqrt_small_e
778 \the\numexpr #1\ifcase \numexpr #2/\xint_c_ii-\xint_c_i\relax
779 \or 0\or 00\or 000\or 0000\fi .%
780 }%

781 \def\XINT_sqrt_small_e #1.#2.%
782 {%
783 \expandafter\XINT_sqrt_small_ea\the\numexpr #1*#1-#2.#1.%
784 }%

```

```

785 \def\XINT_sqrt_small_ea #1%
786 {%
787   \if0#1\xint_dothis\XINT_sqrt_small_ez\fi
788   \if-#1\xint_dothis\XINT_sqrt_small_eb\fi
789   \xint_orthat\XINT_sqrt_small_f #1%
790 }%
791 \def\XINT_sqrt_small_ez 0.#1.{\expandafter{\the\numexpr#1+\xint_c_i
792   \expandafter}\expandafter{\the\numexpr #1*\xint_c_ii+\xint_c_i}}%

793 \def\XINT_sqrt_small_eb -#1.#2.%
794 {%
795   \expandafter\XINT_sqrt_small_ec \the\numexpr
796   (#1-\xint_c_i+#2)/(\xint_c_ii*#2).#1.#2.%
797 }%

798 \def\XINT_sqrt_small_ec #1.#2.#3.%
799 {%
800   \expandafter\XINT_sqrt_small_f \the\numexpr
801   -#2+\xint_c_ii*#3*#1+#1*#1\expandafter.\the\numexpr #3+#1.%
802 }%

803 \def\XINT_sqrt_small_f #1.#2.%
804 {%
805   \expandafter\XINT_sqrt_small_g
806   \the\numexpr (#1+#2)/(\xint_c_ii*#2)-\xint_c_i.#1.#2.%
807 }%

808 \def\XINT_sqrt_small_g #1#2.%
809 {%
810   \if 0#1%
811     \expandafter\XINT_sqrt_small_end
812   \else
813     \expandafter\XINT_sqrt_small_h
814   \fi
815   #1#2.%
816 }%

817 \def\XINT_sqrt_small_h #1.#2.#3.%
818 {%
819   \expandafter\XINT_sqrt_small_f
820   \the\numexpr #2-\xint_c_ii*#1*#3+#1*#1\expandafter.%
821   \the\numexpr #3-#1.%
822 }%
823 \def\XINT_sqrt_small_end #1.#2.#3.{{#3}{#2}}%

824 \def\XINT_sqrt_big_d #1.#2%
825 {%
826   \ifodd #2 \xint_dothis{\expandafter\XINT_sqrt_big_e0}\fi
827   \xint_orthat{\expandafter\XINT_sqrt_big_eE}%

```

```

828 \the\numexpr (#2-\xint_c_i)/\xint_c_ii.#1;%
829 }%

830 \def\xint_sqrt_big_eE #1;#2#3#4#5#6#7#8#9%
831 {%
832 \xint_sqrt_big_eE_a #1;{#2#3#4#5#6#7#8#9}%
833 }%

834 \def\xint_sqrt_big_eE_a #1.#2;#3%
835 {%
836 \expandafter\xint_sqrt_bigormed_f
837 \romannumeral0\xint_sqrt_small_e #2000.#3.#1;%
838 }%

839 \def\xint_sqrt_big_e0 #1;#2#3#4#5#6#7#8#9%
840 {%
841 \xint_sqrt_big_e0_a #1;{#2#3#4#5#6#7#8#9}%
842 }%
843 \def\xint_sqrt_big_e0_a #1.#2;#3#4%
844 {%
845 \expandafter\xint_sqrt_bigormed_f
846 \romannumeral0\xint_sqrt_small_e #20000.#3#4.#1;%
847 }%

848 \def\xint_sqrt_bigormed_f #1#2#3;%
849 {%
850 \ifnum#3<\xint_c_ix
851 \xint_dothis {\csname XINT_sqrt_med_f\romannumeral#3\endcsname}%
852 \fi
853 \xint_orthat\xint_sqrt_big_f #1.#2.#3;%
854 }%
855 \def\xint_sqrt_med_fv {\xint_sqrt_med_fa .}%
856 \def\xint_sqrt_med_fvi {\xint_sqrt_med_fa 0.}%
857 \def\xint_sqrt_med_fvii {\xint_sqrt_med_fa 00.}%
858 \def\xint_sqrt_med_fviii{\xint_sqrt_med_fa 000.}%

859 \def\xint_sqrt_med_fa #1.#2.#3.#4;%
860 {%
861 \expandafter\xint_sqrt_med_fb
862 \the\numexpr (#30#1-5#1)/(\xint_c_ii*#2).#1.#2.#3.%
863 }%

864 \def\xint_sqrt_med_fb #1.#2.#3.#4.#5.%
865 {%
866 \expandafter\xint_sqrt_small_ea
867 \the\numexpr (#40#2-\xint_c_ii*#3*#1)*10#2+(#1*#1-#5)\expandafter.%
868 \the\numexpr #30#2-#1.%
869 }%

```

```

870 \def\XINT_sqrt_big_f #1;#2#3#4#5#6#7#8#9%
871 {%
872   \XINT_sqrt_big_fa #1;{#2#3#4#5#6#7#8#9}%
873 }%

874 \def\XINT_sqrt_big_fa #1.#2.#3;#4%
875 {%
876   \expandafter\XINT_sqrt_big_ga
877   \the\numexpr #3-\xint_c_viii\expandafter.%
878   \romannumeral0\XINT_sqrt_med_fa 000.#1.#2.;#4.%
879 }%

880 \def\XINT_sqrt_big_ga #1.#2#3%
881 {%
882   \ifnum #1>\xint_c_viii
883     \expandafter\XINT_sqrt_big_gb\else
884     \expandafter\XINT_sqrt_big_ka
885   \fi #1.#3.#2.%
886 }%

887 \def\XINT_sqrt_big_gb #1.#2.#3.%
888 {%
889   \expandafter\XINT_sqrt_big_gc
890   \the\numexpr (\xint_c_ii*#2-\xint_c_i)*\xint_c_x^viii/(\xint_c_iv*#3).%
891   #3.#2.#1;%
892 }%

893 \def\XINT_sqrt_big_gc #1.#2.#3.%
894 {%
895   \expandafter\XINT_sqrt_big_gd
896   \romannumeral0\xintiiaadd
897     {\xintiiSub {#3000000000}\xintDouble{\xintiiMul{#2}{#1}}000000000}%
898     {\xintiiSqr {#1}}.%
899   \romannumeral0\xintiisub{#2000000000}{#1}.%
900 }%

901 \def\XINT_sqrt_big_gd #1.#2.%
902 {%
903   \expandafter\XINT_sqrt_big_ge #2.#1.%
904 }%

905 \def\XINT_sqrt_big_ge #1;#2#3#4#5#6#7#8#9%
906   {\XINT_sqrt_big_gf #1.#2#3#4#5#6#7#8#9;}%
907 \def\XINT_sqrt_big_gf #1;#2#3#4#5#6#7#8#9%
908   {\XINT_sqrt_big_gg #1#2#3#4#5#6#7#8#9.}%

909 \def\XINT_sqrt_big_gg #1.#2.#3.#4.%
910 {%
911   \expandafter\XINT_sqrt_big_gloop

```

```

912 \expandafter\xint_c_xvi\expandafter.%
913 \the\numexpr #3-\xint_c_viii\expandafter.%
914 \romannumeral0\xintiisub {#2}{\xintiNum{#4}}.#1.%
915 }%

916 \def\XINT_sqrt_big_gloop #1.#2.%
917 {%
918 \unless\ifnum #1<#2 \xint_dothis\XINT_sqrt_big_ka \fi
919 \xint_orthat{\XINT_sqrt_big_gi #1.}#2.%
920 }%

921 \def\XINT_sqrt_big_gi #1.%
922 {%
923 \expandafter\XINT_sqrt_big_gj\romannumeral\xintreplicate{#1}0.#1.%
924 }%

925 \def\XINT_sqrt_big_gj #1.#2.#3.#4.#5.%
926 {%
927 \expandafter\XINT_sqrt_big_gk
928 \romannumeral0\xintiidivision {#4#1}%
929 {\XINT_dbl #5\xint_bye2345678\xint_bye*\xint_c_ii\relax}.%
930 #1.#5.#2.#3.%
931 }%

932 \def\XINT_sqrt_big_gk #1#2.#3.#4.%
933 {%
934 \expandafter\XINT_sqrt_big_gl
935 \romannumeral0\xintiiadd {#2#3}{\xintiiSqr{#1}}.%
936 \romannumeral0\xintiisub {#4#3}{#1}.%
937 }%

938 \def\XINT_sqrt_big_gl #1.#2.%
939 {%
940 \expandafter\XINT_sqrt_big_gm #2.#1.%
941 }%

942 \def\XINT_sqrt_big_gm #1.#2.#3.#4.#5.%
943 {%
944 \expandafter\XINT_sqrt_big_gn

945 \romannumeral0\XINT_split_fromleft\xint_c_ii*#3.#5\xint_bye2345678\xint_bye.%
946 #1.#2.#3.#4.%
947 }%

948 \def\XINT_sqrt_big_gn #1.#2.#3.#4.#5.#6.%
949 {%
950 \expandafter\XINT_sqrt_big_gloop
951 \the\numexpr \xint_c_ii*#5\expandafter.%
952 \the\numexpr #6-#5\expandafter.%
953 \romannumeral0\xintiisub{#4}{\xintiNum{#1}}.#3.#2.%
954 }%

```



```

955 \def\XINT_sqrt_big_ka #1.#2.#3.#4.%
956 {%
957     \expandafter\XINT_sqrt_big_kb

958     \romannumeral0\XINT_dsx_addzeros {#1}#3;.%
959     \romannumeral0\xintiisub
960     {\XINT_dsx_addzerosnofuss {\xint_c_ii*#1}#2;}%
961     {\xintiNum{#4}}}%
962 }%
963 \def\XINT_sqrt_big_kb #1.#2.%
964 {%
965     \expandafter\XINT_sqrt_big_kc #2.#1.%
966 }%

967 \def\XINT_sqrt_big_kc #1%
968 {%
969     \if0#1\xint_dothis\XINT_sqrt_big_kz\fi
970     \xint_orthat\XINT_sqrt_big_kloop #1%
971 }%
972 \def\XINT_sqrt_big_kz 0.#1.%
973 {%
974     \expandafter\XINT_sqrt_big_kend
975     \romannumeral0%
976     \xintinc{\XINT_dbl#1\xint_bye2345678\xint_bye*\xint_c_ii\relax}.#1.%
977 }%
978 \def\XINT_sqrt_big_kend #1.#2.%
979 {%
980     \expandafter{\romannumeral0\xintinc{#2}}{#1}%
981 }%

982 \def\XINT_sqrt_big_kloop #1.#2.%
983 {%
984     \expandafter\XINT_sqrt_big_ke
985     \romannumeral0\xintiidivision{#1}%
986     {\romannumeral0\XINT_dbl #2\xint_bye2345678\xint_bye*\xint_c_ii\relax}{#2}%
987 }%

988 \def\XINT_sqrt_big_ke #1%
989 {%
990     \if0\XINT_Sgn #1\xint:
991         \expandafter \XINT_sqrt_big_end
992     \else \expandafter \XINT_sqrt_big_kf
993     \fi {#1}%
994 }%

995 \def\XINT_sqrt_big_kf #1#2#3%
996 {%
997     \expandafter\XINT_sqrt_big_kg
998     \romannumeral0\xintiisub {#3}{#1}%.%
999     \romannumeral0\xintiiadd {#2}{\xintiiSqr {#1}}%.%

```

```
1000 }%
1001 \def\XINT_sqrt_big_kg #1.#2.%
1002 {%
1003     \expandafter\XINT_sqrt_big_kloop #2.#1.%
1004 }%
```

```
1005 \def\XINT_sqrt_big_end #1#2#3{{{#3}{#2}}}%
```

5.52 \xintiisqrt, \xintiisqrtR

```
1006 \def\xintiisqrt {\romannumeral0\xintiisqrt }%
1007 \def\xintiisqrt {\expandafter\XINT_sqrt_post\romannumeral0\xintiisquareroot }%
1008 \def\XINT_sqrt_post #1#2{\XINT_dec #1\XINT_dec_bye234567890\xint_bye}%
1009 \def\xintiisqrtR {\romannumeral0\xintiisqrtr }%
1010 \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 $0 < \#2 < \#1$, $<= N - 0.75 < N$, donc rounded- $\rightarrow \#1$ si $\#2 \geq \#1$, $(\#1 - .5)^2 \geq N + .25 > N$, donc rounded- $\rightarrow \#1 - 1$.

```
1011 \def\XINT_sqrtr_post #1#2%
1012     {\xintiiflt {#2}{#1}{ #1}{\XINT_dec #1\XINT_dec_bye234567890\xint_bye}}%
```

5.53 \xintiibinomial

2015/11/28-29 for 1.2f.

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

I really lack some kind of infinity or NaN value.

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

```
1013 \def\xintiibinomial {\romannumeral0\xintiibinomial }%
1014 \def\xintiibinomial #1#2%
1015 {%
1016     \expandafter\XINT_binom_pre\the\numexpr #1\expandafter.\the\numexpr #2.%
1017 }%
1018 \def\XINT_binom_pre #1.#2.%
1019 {%
1020     \expandafter\XINT_binom_fork \the\numexpr#1-#2.#2.#1.%
1021 }%
```

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.

```
1022 \def\XINT_binom_fork #1#2.#3#4.#5#6.%
1023 {%
1024     \if-#5\xint_dothis{\XINT_signalcondition{InvalidOperation}{Binomial with
1025         negative first arg: #5#6}{\{0\}}\fi
1026     \if-#1\xint_dothis{ 0}\fi
1027     \if-#3\xint_dothis{ 0}\fi
1028     \if0#1\xint_dothis{ 1}\fi
1029     \if0#3\xint_dothis{ 1}\fi
```

```

1030 \ifnum #5#6>\xint_c_x^viii_mone\xint_dothis
1031 {\XINT_signalcondition{InvalidOperation}{Binomial with too
1032   large argument: 99999999 < #5#6}{\{0}\}\fi
1033 \ifnum #1#2>#3#4 \xint_dothis{\XINT_binom_a #1#2.#3#4.}\fi
1034 \xint_orthat{\XINT_binom_a #3#4.#1#2.}%
1035 }%

```

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

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

```

1036 \def\XINT_binom_a #1.#2.%
1037 {%
1038   \expandafter\XINT_binom_b\the\numexpr \xint_c_i+#1.1.#2.100000001!1!;!0!%
1039 }%

```

$y = x - k + 1$. $j = 1$. k . On va évaluer par $y/1 * (y+1)/2 * (y+2)/3$ etc... On essaie de regrouper de manière à utiliser au mieux $\backslash numexpr$. 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 $\backslash xintiiFac$. 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 $vsmall$).

```

1040 \def\XINT_binom_b #1.%
1041 {%
1042   \ifnum #1>9999 \xint_dothis\XINT_binom_vbigloop \fi
1043   \ifnum #1>463 \xint_dothis\XINT_binom_bigloop \fi
1044   \ifnum #1>98 \xint_dothis\XINT_binom_medloop \fi
1045   \ifnum #1>29 \xint_dothis\XINT_binom_smallloop \fi
1046   \xint_orthat\XINT_binom_vsmallloop #1.%
1047 }%

```

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.

```

1048 \def\XINT_binom_smallloop #1.#2.#3.%
1049 {%
1050   \ifcase\numexpr #3-#2\relax
1051     \expandafter\XINT_binom_end_
1052   \or \expandafter\XINT_binom_end_i
1053   \or \expandafter\XINT_binom_end_ii
1054   \or \expandafter\XINT_binom_end_iii
1055   \else\expandafter\XINT_binom_smallloop_a
1056   \fi #1.#2.#3.%
1057 }%

```

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

```

1058 \def\XINT_binom_smallloop_a #1.#2.#3.%
1059 {%
1060   \expandafter\XINT_binom_smallloop_b

```

```

1061 \the\numexpr #1+\xint_c_iv\expandafter.%
1062 \the\numexpr #2+\xint_c_iv\expandafter.%
1063 \the\numexpr #3\expandafter.%
1064 \the\numexpr\expandafter\XINT_binom_div
1065 \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1066 !\romannumeral0\expandafter\XINT_binom_mul
1067 \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1068 }%
1069 \def\XINT_binom_smallloop_b #1.%
1070 {%
1071 \ifnum #1>98 \expandafter\XINT_binom_medloop \else
1072 \expandafter\XINT_binom_smallloop \fi #1.%
1073 }%

```

Ici on prend trois par trois.

```

1074 \def\XINT_binom_medloop #1.#2.#3.%
1075 {%
1076 \ifcase\numexpr #3-#2\relax
1077 \expandafter\XINT_binom_end_
1078 \or \expandafter\XINT_binom_end_i
1079 \or \expandafter\XINT_binom_end_ii
1080 \else\expandafter\XINT_binom_medloop_a
1081 \fi #1.#2.#3.%
1082 }%
1083 \def\XINT_binom_medloop_a #1.#2.#3.%
1084 {%
1085 \expandafter\XINT_binom_medloop_b
1086 \the\numexpr #1+\xint_c_iii\expandafter.%
1087 \the\numexpr #2+\xint_c_iii\expandafter.%
1088 \the\numexpr #3\expandafter.%
1089 \the\numexpr\expandafter\XINT_binom_div
1090 \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
1091 !\romannumeral0\expandafter\XINT_binom_mul
1092 \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1093 }%
1094 \def\XINT_binom_medloop_b #1.%
1095 {%
1096 \ifnum #1>463 \expandafter\XINT_binom_bigloop \else
1097 \expandafter\XINT_binom_medloop \fi #1.%
1098 }%

```

Ici on prend deux par deux.

```

1099 \def\XINT_binom_bigloop #1.#2.#3.%
1100 {%
1101 \ifcase\numexpr #3-#2\relax
1102 \expandafter\XINT_binom_end_
1103 \or \expandafter\XINT_binom_end_i
1104 \else\expandafter\XINT_binom_bigloop_a
1105 \fi #1.#2.#3.%
1106 }%
1107 \def\XINT_binom_bigloop_a #1.#2.#3.%
1108 {%

```

```

1109 \expandafter\XINT_binom_bigloop_b
1110 \the\numexpr #1+\xint_c_ii\expandafter.%
1111 \the\numexpr #2+\xint_c_ii\expandafter.%
1112 \the\numexpr #3\expandafter.%
1113 \the\numexpr\expandafter\XINT_binom_div
1114 \the\numexpr #2*(#2+\xint_c_i)\expandafter
1115 !\romannumeral0\expandafter\XINT_binom_mul
1116 \the\numexpr #1*(#1+\xint_c_i)!%
1117 }%
1118 \def\XINT_binom_bigloop_b #1.%
1119 {%
1120 \ifnum #1>9999 \expandafter\XINT_binom_vbigloop \else
1121 \expandafter\XINT_binom_bigloop \fi #1.%
1122 }%

```

Et finalement un par un.

```

1123 \def\XINT_binom_vbigloop #1.#2.#3.%
1124 {%
1125 \ifnum #3=#2
1126 \expandafter\XINT_binom_end_
1127 \else\expandafter\XINT_binom_vbigloop_a
1128 \fi #1.#2.#3.%
1129 }%
1130 \def\XINT_binom_vbigloop_a #1.#2.#3.%
1131 {%
1132 \expandafter\XINT_binom_vbigloop
1133 \the\numexpr #1+\xint_c_i\expandafter.%
1134 \the\numexpr #2+\xint_c_i\expandafter.%
1135 \the\numexpr #3\expandafter.%
1136 \the\numexpr\expandafter\XINT_binom_div\the\numexpr #2\expandafter
1137 !\romannumeral0\XINT_binom_mul #1!%
1138 }%

```

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.

```

1139 \def\XINT_binom_vsmallloop #1.#2.#3.%
1140 {%
1141 \ifcase\numexpr #3-#2\relax
1142 \expandafter\XINT_binom_vsmallend_
1143 \or \expandafter\XINT_binom_vsmallend_i
1144 \or \expandafter\XINT_binom_vsmallend_ii
1145 \or \expandafter\XINT_binom_vsmallend_iii
1146 \else\expandafter\XINT_binom_vsmallloop_a
1147 \fi #1.#2.#3.%
1148 }%
1149 \def\XINT_binom_vsmallloop_a #1.%
1150 {%
1151 \ifnum #1>26 \expandafter\XINT_binom_smallloop_a \else
1152 \expandafter\XINT_binom_vsmallloop_b \fi #1.%

```

```

1153 }%
1154 \def\XINT_binom_vsmallloop_b #1.#2.#3.%
1155 {%
1156     \expandafter\XINT_binom_vsmallloop
1157     \the\numexpr #1+\xint_c_iv\expandafter.%
1158     \the\numexpr #2+\xint_c_iv\expandafter.%
1159     \the\numexpr #3\expandafter.%
1160     \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1161     \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1162     !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1163 }%
1164 \def\XINT_binom_mul #1!#21!;!0!%
1165 {%
1166     \expandafter\XINT_rev_nounsep\expandafter{\expandafter}%
1167     \the\numexpr\expandafter\XINT_smallmul
1168     \the\numexpr\xint_c_x^viii+#1\expandafter
1169     !\romannumeral0\XINT_rev_nounsep {}1;!#2%
1170     \R!\R!\R!\R!\R!\R!\R!\R!\R!\W
1171     \R!\R!\R!\R!\R!\R!\R!\R!\R!\W
1172     1;!%
1173 }%
1174 \def\XINT_binom_div #1!1;!%
1175 {%
1176     \expandafter\XINT_smalldivx_a
1177     \the\numexpr #1/\xint_c_ii\expandafter\xint:
1178     \the\numexpr \xint_c_x^viii+#1!%
1179 }%

```

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

```

1180 \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.

```

1181 \def\XINT_binom_end_iii #1.#2.#3.%
1182 {%
1183     \expandafter\XINT_binom_finish
1184     \the\numexpr\expandafter\XINT_binom_div
1185         \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1186         !\romannumeral0\expandafter\XINT_binom_mul
1187         \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1188 }%
1189 \def\XINT_binom_end_ii #1.#2.#3.%
1190 {%
1191     \expandafter\XINT_binom_finish
1192     \the\numexpr\expandafter\XINT_binom_div
1193         \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
1194         !\romannumeral0\expandafter\XINT_binom_mul
1195         \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1196 }%
1197 \def\XINT_binom_end_i #1.#2.#3.%

```

```

1198 {%
1199   \expandafter\XINT_binom_finish
1200   \the\numexpr\expandafter\XINT_binom_div
1201     \the\numexpr #2*(#2+\xint_c_i)\expandafter
1202     !\romannumeral0\expandafter\XINT_binom_mul
1203     \the\numexpr #1*(#1+\xint_c_i)!%
1204 }%
1205 \def\XINT_binom_end_ #1.#2.#3.%
1206 {%
1207   \expandafter\XINT_binom_finish
1208   \the\numexpr\expandafter\XINT_binom_div\the\numexpr #2\expandafter
1209   !\romannumeral0\XINT_binom_mul #1!%
1210 }%
1211 \def\XINT_binom_finish #1;!0!%
1212   {\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).

```

1213 \def\XINT_binom_vsmallend_iii #1.%
1214 {%
1215   \ifnum #1>26 \expandafter\XINT_binom_end_iii \else
1216     \expandafter\XINT_binom_vsmallend_iiib \fi #1.%
1217 }%
1218 \def\XINT_binom_vsmallend_iiib #1.#2.#3.%
1219 {%
1220   \expandafter\XINT_binom_vsmallfinish
1221   \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1222     \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1223     !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1224 }%
1225 \def\XINT_binom_vsmallend_ii #1.%
1226 {%
1227   \ifnum #1>27 \expandafter\XINT_binom_end_ii \else
1228     \expandafter\XINT_binom_vsmallend_iib \fi #1.%
1229 }%
1230 \def\XINT_binom_vsmallend_iib #1.#2.#3.%
1231 {%
1232   \expandafter\XINT_binom_vsmallfinish
1233   \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1234     \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
1235     !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1236 }%
1237 \def\XINT_binom_vsmallend_i #1.%
1238 {%
1239   \ifnum #1>28 \expandafter\XINT_binom_end_i \else
1240     \expandafter\XINT_binom_vsmallend_ib \fi #1.%
1241 }%
1242 \def\XINT_binom_vsmallend_ib #1.#2.#3.%
1243 {%
1244   \expandafter\XINT_binom_vsmallfinish
1245   \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1246   \the\numexpr #2*(#2+\xint_c_i)\expandafter

```

```

1247      !\the\numexpr #1*(#1+\xint_c_i)!%
1248 }%
1249 \def\xINT_binom_vsmallend_ #1.%
1250 {%
1251     \ifnum #1>29 \expandafter\xINT_binom_end_ \else
1252         \expandafter\xINT_binom_vsmallend_b \fi #1.%
1253 }%
1254 \def\xINT_binom_vsmallend_b #1.#2.#3.%
1255 {%
1256     \expandafter\xINT_binom_vsmallfinish
1257     \the\numexpr\xINT_binom_vsmallmuldiv #2!#1!%
1258 }%
1259 \def\xINT_binom_vsmallfinish#1{%
1260 \def\xINT_binom_vsmallfinish1##1!1!;!0!{\expandafter#1\the\numexpr##1\relax}%
1261 }\xINT_binom_vsmallfinish{ }%

```

5.54 \xintiipFactorial

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.

```

1262 \def\xintiipFactorial {\romannumeral0\xintiipfactorial }%
1263 \def\xintiipfactorial #1#2%
1264 {%
1265     \expandafter\xINT_pfac_fork\the\numexpr#1\expandafter.\the\numexpr #2.%
1266 }%
1267 \def\xintPFactorial{\romannumeral0\xintpfactorial}%
1268 \let\xintpfactorial\xintiipfactorial

```

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

```

1269 \def\xINT_pfac_fork #1#2.#3#4.%
1270 {%
1271     \unless\ifnum #1#2<#3#4 \xint_dothis\xINT_pfac_one\fi
1272     \if-#3\xint_dothis\xINT_pfac_neg\fi
1273     \if-#1\xint_dothis\xINT_pfac_zero\fi
1274     \ifnum #3#4>\xint_c_x^viii_mone\xint_dothis\xINT_pfac_outofrange\fi
1275     \xint_orthat \xINT_pfac_a #1#2.#3#4.%
1276 }%
1277 \def\xINT_pfac_outofrange #1.#2.%
1278     {\xINT_signalcondition{InvalidOperation}{PFactorial with
1279     too big second arg: 99999999 < #2}}{0}}%
1280 \def\xINT_pfac_one #1.#2.{ 1}%
1281 \def\xINT_pfac_zero #1.#2.{ 0}%
1282 \def\xINT_pfac_neg -#1.-#2.%
1283 {%
1284     \ifnum #1>\xint_c_x^viii\xint_dothis\xINT_pfac_outofrange\fi

```



```

1285 \xint_orthat
1286 {\ifodd\numexpr#2-#1\relax\xint_afterfi{\expandafter-\romannumeral`&&@\}\fi
1287 \expandafter\XINT_pfac_a }%
1288 \the\numexpr #2-\xint_c_i\expandafter.\the\numexpr#1-\xint_c_i.%
1289 }%
1290 \def\XINT_pfac_a #1.#2.%
1291 {%
1292 \expandafter\XINT_pfac_b\the\numexpr \xint_c_i+#1.#2.10000000011;!;%
1293 1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1294 }%
1295 \def\XINT_pfac_b #1.%
1296 {%
1297 \ifnum #1>9999 \xint_dothis\XINT_pfac_vbigloop \fi
1298 \ifnum #1>463 \xint_dothis\XINT_pfac_bigloop \fi
1299 \ifnum #1>98 \xint_dothis\XINT_pfac_medloop \fi
1300 \xint_orthat\XINT_pfac_smallloop #1.%
1301 }%
1302 \def\XINT_pfac_smallloop #1.#2.%
1303 {%
1304 \ifcase\numexpr #2-#1\relax
1305 \expandafter\XINT_pfac_end_
1306 \or \expandafter\XINT_pfac_end_i
1307 \or \expandafter\XINT_pfac_end_ii
1308 \or \expandafter\XINT_pfac_end_iii
1309 \else\expandafter\XINT_pfac_smallloop_a
1310 \fi #1.#2.%
1311 }%
1312 \def\XINT_pfac_smallloop_a #1.#2.%
1313 {%
1314 \expandafter\XINT_pfac_smallloop_b
1315 \the\numexpr #1+\xint_c_iv\expandafter.%
1316 \the\numexpr #2\expandafter.%
1317 \the\numexpr\expandafter\XINT_smallmul
1318 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1319 }%
1320 \def\XINT_pfac_smallloop_b #1.%
1321 {%
1322 \ifnum #1>98 \expandafter\XINT_pfac_medloop \else
1323 \expandafter\XINT_pfac_smallloop \fi #1.%
1324 }%
1325 \def\XINT_pfac_medloop #1.#2.%
1326 {%
1327 \ifcase\numexpr #2-#1\relax
1328 \expandafter\XINT_pfac_end_
1329 \or \expandafter\XINT_pfac_end_i
1330 \or \expandafter\XINT_pfac_end_ii
1331 \else\expandafter\XINT_pfac_medloop_a
1332 \fi #1.#2.%
1333 }%
1334 \def\XINT_pfac_medloop_a #1.#2.%
1335 {%
1336 \expandafter\XINT_pfac_medloop_b

```

```

1337 \the\numexpr #1+\xint_c_iii\expandafter.%
1338 \the\numexpr #2\expandafter.%
1339 \the\numexpr\expandafter\XINT_smallmul
1340 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1341 }%
1342 \def\XINT_pfac_medloop_b #1.%
1343 {%
1344 \ifnum #1>463 \expandafter\XINT_pfac_bigloop \else
1345 \expandafter\XINT_pfac_medloop \fi #1.%
1346 }%
1347 \def\XINT_pfac_bigloop #1.#2.%
1348 {%
1349 \ifcase\numexpr #2-#1\relax
1350 \expandafter\XINT_pfac_end_
1351 \or \expandafter\XINT_pfac_end_i
1352 \else\expandafter\XINT_pfac_bigloop_a
1353 \fi #1.#2.%
1354 }%
1355 \def\XINT_pfac_bigloop_a #1.#2.%
1356 {%
1357 \expandafter\XINT_pfac_bigloop_b
1358 \the\numexpr #1+\xint_c_ii\expandafter.%
1359 \the\numexpr #2\expandafter.%
1360 \the\numexpr\expandafter
1361 \XINT_smallmul\the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
1362 }%
1363 \def\XINT_pfac_bigloop_b #1.%
1364 {%
1365 \ifnum #1>9999 \expandafter\XINT_pfac_vbigloop \else
1366 \expandafter\XINT_pfac_bigloop \fi #1.%
1367 }%
1368 \def\XINT_pfac_vbigloop #1.#2.%
1369 {%
1370 \ifnum #2=#1
1371 \expandafter\XINT_pfac_end_
1372 \else\expandafter\XINT_pfac_vbigloop_a
1373 \fi #1.#2.%
1374 }%
1375 \def\XINT_pfac_vbigloop_a #1.#2.%
1376 {%
1377 \expandafter\XINT_pfac_vbigloop
1378 \the\numexpr #1+\xint_c_i\expandafter.%
1379 \the\numexpr #2\expandafter.%
1380 \the\numexpr\expandafter\XINT_smallmul\the\numexpr\xint_c_x^viii+#1!%
1381 }%
1382 \def\XINT_pfac_end_iii #1.#2.%
1383 {%
1384 \expandafter\XINT_mul_out
1385 \the\numexpr\expandafter\XINT_smallmul
1386 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1387 }%
1388 \def\XINT_pfac_end_ii #1.#2.%

```

```

1389 {%
1390     \expandafter\XINT_mul_out
1391     \the\numexpr\expandafter\XINT_smallmul
1392     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1393 }%
1394 \def\XINT_pfac_end_i #1.#2.%
1395 {%
1396     \expandafter\XINT_mul_out
1397     \the\numexpr\expandafter\XINT_smallmul
1398     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
1399 }%
1400 \def\XINT_pfac_end_ #1.#2.%
1401 {%
1402     \expandafter\XINT_mul_out
1403     \the\numexpr\expandafter\XINT_smallmul\the\numexpr \xint_c_x^viii+#1!%
1404 }%

```

5.55 \xintBool, \xintToggle

1.09c

```

1405 \def\xintBool #1{\romannumeral`&&@%
1406         \csname if#1\endcsname\expandafter1\else\expandafter0\fi }%
1407 \def\xintToggle #1{\romannumeral`&&@\iftoggle{#1}{1}{0}}%

```

5.56 \xintiiGCD

Copied over \xintiiGCD code from xintgcd at 1.3d in order to support gcd() function in \xintiiexpr.

At 1.4 original code removed from xintgcd as the latter now requires xint.

```

1408 \def\xintiiGCD {\romannumeral0\xintiigcd }%
1409 \def\xintiigcd #1{\expandafter\XINT_iigcd\romannumeral0\xintiiabs#1\xint:}%
1410 \def\XINT_iigcd #1#2\xint:#3%
1411 {%
1412     \expandafter\XINT_gcd_fork\expandafter#1%
1413     \romannumeral0\xintiiabs#3\xint:#1#2\xint:
1414 }%
1415 \def\XINT_gcd_fork #1#2%
1416 {%
1417     \xint_UDzerofork
1418     #1\XINT_gcd_Aiszero
1419     #2\XINT_gcd_Biszero
1420     0\XINT_gcd_loop
1421     \krof
1422     #2%
1423 }%
1424 \def\XINT_gcd_AisZero #1\xint:#2\xint:{ #1}%
1425 \def\XINT_gcd_BisZero #1\xint:#2\xint:{ #2}%
1426 \def\XINT_gcd_loop #1\xint:#2\xint:
1427 {%
1428     \expandafter\expandafter\expandafter\XINT_gcd_CheckRem
1429     \expandafter\xint_secondoftwo
1430     \romannumeral0\XINT_div_prepare {#1}{#2}\xint:#1\xint:
1431 }%

```

```

1432 \def\XINT_gcd_CheckRem #1%
1433 {%
1434     \xint_gob_til_zero #1\XINT_gcd_end0\XINT_gcd_loop #1%
1435 }%
1436 \def\XINT_gcd_end0\XINT_gcd_loop #1\xint:#2\xint:{ #2}%

```

5.57 \xintiilCM

Copied over \xintiilCM code from xintgcd at 1.3d in order to support lcm() function in \xintiexpr.

At 1.4 original code removed from xintgcd as the latter now requires xint.

```

1437 \def\xintiilCM {\romannumeral0\xintiilcm}%
1438 \def\xintiilcm #1{\expandafter\XINT_iilcm\romannumeral0\xintiabs#1\xint:}%
1439 \def\XINT_iilcm #1#2\xint:#3%
1440 {%
1441     \expandafter\XINT_lcm_fork\expandafter#1%
1442         \romannumeral0\xintiabs#3\xint:#1#2\xint:
1443 }%
1444 \def\XINT_lcm_fork #1#2%
1445 {%
1446     \xint_UDzerofork
1447     #1\XINT_lcm_iszero
1448     #2\XINT_lcm_iszero
1449     0\XINT_lcm_notzero
1450     \krof
1451     #2%
1452 }%
1453 \def\XINT_lcm_iszero #1\xint:#2\xint:{ 0}%
1454 \def\XINT_lcm_notzero #1\xint:#2\xint:
1455 {%
1456     \expandafter\XINT_lcm_end\romannumeral0%
1457     \expandafter\expandafter\expandafter\XINT_gcd_CheckRem
1458     \expandafter\xint_secondoftwo
1459     \romannumeral0\XINT_div_prepare {#1}{#2}\xint:#1\xint:
1460     \xint:#1\xint:#2\xint:
1461 }%
1462 \def\XINT_lcm_end #1\xint:#2\xint:#3\xint:{\xintiimul {#2}{\xintiiQuo{#3}{#1}}}%

```

5.58 \xintiiGCDof

New with 1.09a (xintgcd.sty).

1.2l adds protection against items being non-terminated \the\numexpr.

1.4 renames the macro into \xintiiGCDof and moves it here. Terminator modified to ^ for direct call by \xintiexpr function. See comments in xintfrac.sty about \xintGCDof macro there.

```

1463 \def\xintiiGCDof {\romannumeral0\xintiigcdof}%
1464 \def\xintiigcdof #1{\expandafter\XINT_iigcdof_a\romannumeral`&&@#1^}%
1465 \def\XINT_iigCDof {\romannumeral0\XINT_iigcdof_a}%
1466 \def\XINT_iigcdof_a #1{\expandafter\XINT_iigcdof_b\romannumeral`&&@#1!}%
1467 \def\XINT_iigcdof_b #1!#2{\expandafter\XINT_iigcdof_c\romannumeral`&&@#2!{#1}!}%
1468 \def\XINT_iigcdof_c #1{\xint_gob_til_ ^ #1\XINT_iigcdof_e ^\XINT_iigcdof_d #1}%
1469 \def\XINT_iigcdof_d #1!{\expandafter\XINT_iigcdof_b\romannumeral0\xintiigcd {#1}}%
1470 \def\XINT_iigcdof_e #1!#2!{ #2}%

```

5.59 \xintiilCMof

See comments of \xintiiGCDoF

```

1471 \def\xintiilCMof      {\romannumeral0\xintiilcmof }%
1472 \def\xintiilcmof      #1{\expandafter\XINT_iilcmof_a\romannumeral`&&@#1^}%
1473 \def\XINT_iilCMof      {\romannumeral0\XINT_iilcmof_a}%
1474 \def\XINT_iilcmof_a    #1{\expandafter\XINT_iilcmof_b\romannumeral`&&@#1!}%
1475 \def\XINT_iilcmof_b    #1!#2{\expandafter\XINT_iilcmof_c\romannumeral`&&@#2!{#1!}%
1476 \def\XINT_iilcmof_c    #1{\xint_gob_til_^ #1\XINT_iilcmof_e ^\XINT_iilcmof_d #1}%
1477 \def\XINT_iilcmof_d    #1!{\expandafter\XINT_iilcmof_b\romannumeral0\xintiilcm {#1}}%
1478 \def\XINT_iilcmof_e    #1!#2!{ #2}%

```

5.60 (WIP) \xintRandomDigits

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

```

1479 \def\xintRandomDigits{\romannumeral0\xintrandomdigits}%
1480 \def\xintrandomdigits#1%
1481 {%
1482   \csname xint_gob_andstop_\expandafter\XINT_randomdigits\the\numexpr#1\xint:
1483 }%
1484 \def\XINT_randomdigits#1\xint:
1485 {%
1486   \expandafter\XINT_randomdigits_a
1487   \the\numexpr(#1+\xint_c_iii)/\xint_c_viii\xint:#1\xint:
1488 }%
1489 \def\XINT_randomdigits_a#1\xint:#2\xint:
1490 {%
1491   \romannumeral\numexpr\xint_c_viii*#1-#2\csname XINT_%
1492   \romannumeral\XINT_replicate #1\endcsname \csname
1493   XINT_rdg\endcsname
1494 }%
1495 \def\XINT_rdg
1496 {%
1497   \expandafter\XINT_rdg_aux\the\numexpr%
1498   \xint_c_nine_x^viii%
1499   -\xint_texuniformdeviate\xint_c_ii^vii%
1500   -\xint_c_ii^vii*\xint_texuniformdeviate\xint_c_ii^vii%
1501   -\xint_c_ii^xiv*\xint_texuniformdeviate\xint_c_ii^vii%
1502   -\xint_c_ii^xxi*\xint_texuniformdeviate\xint_c_ii^vii%
1503   +\xint_texuniformdeviate\xint_c_x^viii%
1504   \relax%
1505 }%
1506 \def\XINT_rdg_aux#1{XINT_rdg\endcsname}%
1507 \let\XINT_XINT_rdg\endcsname

```

5.61 (WIP) \XINT_eightrandomdigits, \xintEightRandomDigits

1.3b. 1.4 adds some public alias...

```

1508 \def\XINT_eightrandomdigits
1509 {%
1510     \expandafter\xint_gobble_i\the\numexpr%
1511         \xint_c_nine_x^viii%
1512         -\xint_texuniformdeviate\xint_c_ii^vii%
1513         -\xint_c_ii^vii*\xint_texuniformdeviate\xint_c_ii^vii%
1514         -\xint_c_ii^xiv*\xint_texuniformdeviate\xint_c_ii^vii%
1515         -\xint_c_ii^xxi*\xint_texuniformdeviate\xint_c_ii^vii%
1516         +\xint_texuniformdeviate\xint_c_x^viii%
1517     \relax%
1518 }%
1519 \let\xintEightRandomDigits\XINT_eightrandomdigits
1520 \def\xintRandBit{\xint_texuniformdeviate\xint_c_ii}%

```

5.62 (WIP) \xintRandBit

1.4 And let's add also \xintRandBit while we are at it.

```

1521 \def\xintRandBit{\xint_texuniformdeviate\xint_c_ii}%

```

5.63 (WIP) \xintXRandomDigits

1.3b.

```

1522 \def\xintXRandomDigits#1%
1523 {%
1524     \csname xint_gobble_\expandafter\XINT_xrandomdigits\the\numexpr#1\xint:
1525 }%
1526 \def\XINT_xrandomdigits#1\xint:
1527 {%
1528     \expandafter\XINT_xrandomdigits_a
1529     \the\numexpr(#1+\xint_c_iii)/\xint_c_viii\xint:#1\xint:
1530 }%
1531 \def\XINT_xrandomdigits_a#1\xint:#2\xint:
1532 {%
1533     \romannumeral\numexpr\xint_c_viii*#1-#2\expandafter\endcsname
1534     \romannumeral`&&\romannumeral
1535     \XINT_replicate #1\endcsname\XINT_eightrandomdigits
1536 }%

```

5.64 (WIP) \xintiiRandRangeAtoB

1.3b. Support for randrange() function.

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

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

It is an error if b<=a, as in Python.

```

1537 \def\xintiiRandRangeAtoB{\romannumeral`&&\xintiiRandRangeAtoB}%
1538 \def\xintiiRandRangeAtoB#1%
1539 {%
1540     \expandafter\XINT_randrangeAtoB_a\romannumeral`&&#1\xint:

```

```

1541 }%
1542 \def\XINT_randrangeAtoB_a#1\xint:#2%
1543 {%
1544     \xintiiaadd{\expandafter\XINT_randrange
1545                 \romannumeral0\xintiisub{#2}{#1}\xint:}%
1546     {#1}%
1547 }%

```

5.65 (WIP) \xintiiRandRange

1.3b. Support for randrange().

```

1548 \def\xintiiRandRange{\romannumeral`&&\xintiirandrange}%
1549 \def\xintiirandrange#1%
1550 {%
1551     \expandafter\XINT_randrange\romannumeral`&&#1\xint:
1552 }%
1553 \def\XINT_randrange #1%
1554 {%
1555     \xint_UDzerominusfork
1556     #1-\XINT_randrange_err:empty
1557     0#1\XINT_randrange_err:empty
1558     0-\XINT_randrange_a
1559     \krof #1%
1560 }%
1561 \def\XINT_randrange_err:empty#1\xint:
1562 {%
1563     \XINT_expandableerror{Empty range for randrange.} 0%
1564 }%
1565 \def\XINT_randrange_a #1\xint:
1566 {%
1567     \expandafter\XINT_randrange_b\romannumeral0\xintlength{#1}.#1\xint:
1568 }%
1569 \def\XINT_randrange_b #1.%
1570 {%
1571     \ifnum#1<\xint_c_x\xint_dothis{\the\numexpr\XINT_uniformdeviate{}}\fi
1572     \xint_orthat{\XINT_randrange_c #1.}%
1573 }%
1574 \def\XINT_randrange_c #1.#2#3#4#5#6#7#8#9%
1575 {%
1576     \expandafter\XINT_randrange_d
1577     \the\numexpr\expandafter\XINT_uniformdeviate\expandafter
1578     {\expandafter}\the\numexpr\xint_c_i+#2#3#4#5#6#7#8#9\xint:\xint:
1579     #2#3#4#5#6#7#8#9\xint:#1\xint:
1580 }%

```

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.

```

1581 \def\XINT_randrange_d #1\xint:#2\xint:

```

```

1582 {%
1583   \ifnum#1=\xint_c_\xint_dothis\XINT_randrange_Z\fi
1584   \ifnum#1=#2 \xint_dothis\XINT_randrange_A\fi
1585   \xint_orthat\XINT_randrange_e #1\xint:
1586 }%
1587 \def\XINT_randrange_e #1\xint:#2\xint:#3\xint:
1588 {%
1589   \the\numexpr#1\expandafter\relax
1590   \romannumeral0\xintrandomdigits{#2-\xint_c_viii}%
1591 }%

```

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...

```

1592 \def\XINT_randrange_Z 0\xint:#1\xint:#2\xint:
1593 {%
1594   \xintinum{\xintRandomDigits{#1-\xint_c_viii}}%
1595 }%

```

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.

```

1596 \def\XINT_randrange_A #1\xint:#2\xint:#3\xint:
1597 {%
1598   \expandafter\XINT_randrange_B
1599   \romannumeral0\xintrandomdigits{#2-\xint_c_viii}\xint:
1600   #3\xint:#2.#1\xint:
1601 }%
1602 \def\XINT_randrange_B #1\xint:#2\xint:#3.#4\xint:
1603 {%
1604   \xintiiiflt{#1}{#2}{\XINT_randrange_E}{\XINT_randrange_again}%
1605   #4#1\xint:#3.#4#2\xint:
1606 }%
1607 \def\XINT_randrange_E #1\xint:#2\xint:{ #1}%
1608 \def\XINT_randrange_again #1\xint:{\XINT_randrange_c}%

```

5.66 (WIP) Adjustments for engines without uniformdeviate primitive

1.3b.

```

1609 \ifdefined\xint_texuniformdeviate
1610 \else
1611   \def\xintrandomdigits#1%
1612   {%
1613     \XINT_expandableerror
1614     {No uniformdeviate at engine level, returning 0.} 0%
1615   }%
1616   \let\xintXRandomDigits\xintRandomDigits
1617   \def\XINT_randrange#1\xint:
1618   {%
1619     \XINT_expandableerror
1620     {No uniformdeviate at engine level, returning 0.} 0%

```


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

```
1621 }%
1622 \fi
1623 \XINT_restorecatcodes_endinput%
```

6 Package xintbinhex implementation

.1	Catcodes, ε -TeX and reload detection . .	162	.6	<code>\xintDecToBin</code>	167
.2	Package identification	163	.7	<code>\xintHexToDec</code>	168
.3	Constants, etc...	163	.8	<code>\xintBinToDec</code>	170
.4	Helper macros	164	.9	<code>\xintBinToHex</code>	171
.4.1	<code>\XINT_zeroes_foriv</code>	164	.10	<code>\xintHexToBin</code>	172
.5	<code>\xintDecToHex</code>	164	.11	<code>\xintCHexToBin</code>	172

The commenting is currently (2020/02/19) very sparse.

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

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

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

6.1 Catcodes, ε -TeX and reload detection

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

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

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2 \catcode13=5      % ^^M
3 \endlinechar=13 %
4 \catcode123=1     % {
5 \catcode125=2     % }
6 \catcode64=11     % @
7 \catcode35=6      % #
8 \catcode44=12     % ,
9 \catcode45=12     % -
10 \catcode46=12     % .
11 \catcode58=12     % :
12 \let\z\endgroup
13 \expandafter\let\expandafter\x\csname ver@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 [2020/02/19 v1.4a Expandable binary and hexadecimal conversions (JFB)]%

```

6.3 Constants, etc...

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

```

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

```

```

77 \expandafter\def\csname XINT_csbth_1110\endcsname {\endcsname E}%
78 \expandafter\def\csname XINT_csbth_1111\endcsname {\endcsname F}%
79 \let\XINT_csbth_none \endcsname
80 \expandafter\def\csname XINT_cshtb_0\endcsname {\endcsname0000}%
81 \expandafter\def\csname XINT_cshtb_1\endcsname {\endcsname0001}%
82 \expandafter\def\csname XINT_cshtb_2\endcsname {\endcsname0010}%
83 \expandafter\def\csname XINT_cshtb_3\endcsname {\endcsname0011}%
84 \expandafter\def\csname XINT_cshtb_4\endcsname {\endcsname0100}%
85 \expandafter\def\csname XINT_cshtb_5\endcsname {\endcsname0101}%
86 \expandafter\def\csname XINT_cshtb_6\endcsname {\endcsname0110}%
87 \expandafter\def\csname XINT_cshtb_7\endcsname {\endcsname0111}%
88 \expandafter\def\csname XINT_cshtb_8\endcsname {\endcsname1000}%
89 \expandafter\def\csname XINT_cshtb_9\endcsname {\endcsname1001}%
90 \def\XINT_cshtb_A {\endcsname1010}%
91 \def\XINT_cshtb_B {\endcsname1011}%
92 \def\XINT_cshtb_C {\endcsname1100}%
93 \def\XINT_cshtb_D {\endcsname1101}%
94 \def\XINT_cshtb_E {\endcsname1110}%
95 \def\XINT_cshtb_F {\endcsname1111}%
96 \let\XINT_cshtb_none \endcsname

```

6.4 Helper macros

6.4.1 \XINT_zeroes_foriv

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

```

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_hb_loop}%
472 \XINT_restorecatcodes_endinput%

```

7 Package xintgcd implementation

.1	Catcodes, ε -TeX and reload detection . .	174	.5	<code>\xintBezoutAlgorithm</code>	180
.2	Package identification	175	.6	<code>\xintTypesetEuclideanAlgorithm</code>	182
.3	<code>\xintBezout</code>	175	.7	<code>\xintTypesetBezoutAlgorithm</code>	183
.4	<code>\xintEuclideanAlgorithm</code>	179			

The commenting is currently (2020/02/19) very sparse.

Release 1.09h has modified a bit the `\xintTypesetEuclideanAlgorithm` and `\xintTypesetBezoutAlgorithm` layout with respect to line indentation in particular. And they use the `xinttools \xintloop` rather than the Plain TeX or \mathbb{X} 's `\loop`.

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 $AU+BV=D$.

From 1.1 to 1.3f the package loaded only `xintcore`. At 1.4 it now automatically loads both of `xint` and `xinttools` (the latter being in fact a requirement of `\xintTypesetEuclideanAlgorithm` and `\xintTypesetBezoutAlgorithm` since 1.09h).

 **Changed at 1.4!**

At 1.4 `\xintGCD`, `\xintLCM`, `\xintGCDof`, and `\xintLCMof` are removed from the package: they are provided only by `xintfrac` and they handle general fractions, not only integers.

The original integer-only macros have been renamed into respectively `\xintiigcd`, `\xintiilcm`, `\xintiigcdof`, and `\xintiilcmof` and got relocated into `xint` package.

7.1 Catcodes, ε -TeX and reload detection

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

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

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2   \catcode13=5      % ^^M
3   \endlinechar=13 %
4   \catcode123=1     % {
5   \catcode125=2     % }
6   \catcode64=11    % @
7   \catcode35=6     % #
8   \catcode44=12    % ,
9   \catcode45=12    % -
10  \catcode46=12    % .
11  \catcode58=12    % :
12  \def\z{\endgroup}%
13  \expandafter\let\expandafter\x\csname ver@xintgcd.sty\endcsname
14  \expandafter\let\expandafter\w\csname ver@xint.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{xintgcd}{numexpr not available, aborting input}%
25  \aftergroup\endinput

```

```

26 \else
27   \ifx\x\relax % plain-TeX, first loading of xintgcd.sty
28     \ifx\w\relax % but xint.sty not yet loaded.
29       \expandafter\def\expandafter\z\expandafter{\z\input xint.sty\relax}%
30     \fi
31     \ifx\t\relax % but xinttools.sty not yet loaded.
32       \expandafter\def\expandafter\z\expandafter{\z\input xinttools.sty\relax}%
33     \fi
34   \else
35     \def\empty {}%
36     \ifx\x\empty % LaTeX, first loading,
37     % variable is initialized, but \ProvidesPackage not yet seen
38       \ifx\w\relax % xint.sty not yet loaded.
39         \expandafter\def\expandafter\z\expandafter{\z\RequirePackage{xint}}%
40       \fi
41       \ifx\t\relax % xinttools.sty not yet loaded.
42         \expandafter\def\expandafter\z\expandafter{\z\RequirePackage{xinttools}}%
43       \fi
44     \else
45       \aftergroup\endinput % xintgcd already loaded.
46     \fi
47   \fi
48 \fi
49 \z%
50 \XINTsetupcatcodes% defined in xintkernel.sty

```

7.2 Package identification

```

51 \XINT_providespackage
52 \ProvidesPackage{xintgcd}%
53 [2020/02/19 v1.4a Euclidean algorithm with xint package (JFB)]%

```

7.3 \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.

```

54 \def\xintBezout {\romannumeral0\xintbezout }%
55 \def\xintbezout #1%
56 {%

```

```

57 \expandafter\XINT_bezout\expandafter {\romannumeral0\xintnum{#1}}%
58 }%
59 \def\XINT_bezout #1#2%
60 {%
61 \expandafter\XINT_bezout_fork \romannumeral0\xintnum{#2}\Z #1\Z
62 }%

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

63 \def\XINT_bezout_fork #1#2\Z #3#4\Z
64 {%
65 \xint_UDzerosfork
66 #1#3\XINT_bezout_botharezero
67 #10\XINT_bezout_secondiszero
68 #30\XINT_bezout_firstiszero
69 00\xint_UDsignsfork
70 \krof
71 #1#3\XINT_bezout_minusminus % A < 0, B < 0
72 #1-\XINT_bezout_minusplus % A > 0, B < 0
73 #3-\XINT_bezout_plusminus % A < 0, B > 0
74 --\XINT_bezout_plusplus % A > 0, B > 0
75 \krof
76 {#2}{#4}#1#3% #1#2=B, #3#4=A
77 }%
78 \def\XINT_bezout_botharezero #1\krof#2#300{{0}{0}{0}}%
79 \def\XINT_bezout_firstiszero #1\krof#2#3#4#5%
80 {%
81 \xint_UDsignfork
82 #4{{0}{-1}}{#2}}%
83 -{{0}{1}}{#4#2}}%
84 \krof
85 }%
86 \def\XINT_bezout_secondiszero #1\krof#2#3#4#5%
87 {%
88 \xint_UDsignfork
89 #5{{-1}{0}}{#3}}%
90 -{{1}{0}}{#5#3}}%
91 \krof
92 }%

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

93 \def\XINT_bezout_minusminus #1#2#3#4%
94 {%
95 \expandafter\XINT_bezout_mm_post
96 \romannumeral0\expandafter\XINT_bezout_preloop_a
97 \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
98 }%
99 \def\XINT_bezout_mm_post #1#2%
100 {%
101 \expandafter\XINT_bezout_mm_postb\expandafter
102 {\romannumeral0\xintiopp{#2}}{\romannumeral0\xintiopp{#1}}%
103 }%
104 \def\XINT_bezout_mm_postb #1#2{\expandafter{#2}{#1}}%

```


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

```

105 \def\XINT_bezout_minusplus #1#2#3#4%
106 {%
107     \expandafter\XINT_bezout_mp_post
108     \romannumeral0\expandafter\XINT_bezout_preloop_a
109     \romannumeral0\XINT_div_prepare {#1}{#4#2}{#1}%
110 }%
111 \def\XINT_bezout_mp_post #1#2%
112 {%
113     \expandafter\xint_exchangetwo_keepbraces\expandafter
114     {\romannumeral0\xintiiopp {#2}}{#1}%
115 }%

```

plusminus A < 0, B > 0

```

116 \def\XINT_bezout_plusminus #1#2#3#4%
117 {%
118     \expandafter\XINT_bezout_pm_post
119     \romannumeral0\expandafter\XINT_bezout_preloop_a
120     \romannumeral0\XINT_div_prepare {#3#1}{#2}{#3#1}%
121 }%
122 \def\XINT_bezout_pm_post #1{\expandafter{\romannumeral0\xintiiopp{#1}}}%

```

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

```

123 \def\XINT_bezout_plusplus #1#2#3#4%
124 {%
125     \expandafter\XINT_bezout_preloop_a
126     \romannumeral0\XINT_div_prepare {#3#1}{#4#2}{#3#1}%
127 }%

```

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.

```

128 \def\XINT_bezout_preloop_a #1#2#3%
129 {%
130     \if0#1\xint_dothis\XINT_bezout_preloop_exchange\fi
131     \if0#2\xint_dothis\XINT_bezout_preloop_exit\fi
132     \xint_orthat{\expandafter\XINT_bezout_loop_B}%
133     \romannumeral0\XINT_div_prepare {#2}{#3}{#2}{#1}110%

```

```

134 }%
135 \def\XINT_bezout_preloop_exit
136   \romannumeral0\XINT_div_prepare #1#2#3#4#5#6#7%
137 {%
138   {0}{1}{2}%
139 }%
140 \def\XINT_bezout_preloop_exchange
141 {%
142   \expandafter\xint_exchangetwo_keepbraces
143   \romannumeral0\expandafter\XINT_bezout_preloop_A
144 }%
145 \def\XINT_bezout_preloop_A #1#2#3#4%
146 {%
147   \if0#2\xint_dothis\XINT_bezout_preloop_exit\fi
148   \xint_orthat{\expandafter\XINT_bezout_loop_B}%
149   \romannumeral0\XINT_div_prepare {#2}{#3}{#2}{#1}%
150 }%
151 \def\XINT_bezout_loop_B #1#2%
152 {%
153   \if0#2\expandafter\XINT_bezout_exitA
154   \else\expandafter\XINT_bezout_loop_C
155   \fi {#1}{#2}%
156 }%

```

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...

```

157 \def\XINT_bezout_loop_C #1#2#3#4#5#6#7%
158 {%
159   \expandafter\XINT_bezout_loop_D\expandafter
160     {\romannumeral0\xintiiadd{\XINT_mul_plusplus{}}{#1\xint:#4\xint:}{#6}}%
161     {\romannumeral0\xintiiadd{\XINT_mul_plusplus{}}{#1\xint:#5\xint:}{#7}}%
162     {#2}{#3}{#4}{#5}%
163 }%
164 \def\XINT_bezout_loop_D #1#2%
165 {%
166   \expandafter\XINT_bezout_loop_E\expandafter{#2}{#1}%
167 }%
168 \def\XINT_bezout_loop_E #1#2#3#4%
169 {%
170   \expandafter\XINT_bezout_loop_b
171   \romannumeral0\XINT_div_prepare {#3}{#4}{#3}{#2}{#1}%
172 }%
173 \def\XINT_bezout_loop_b #1#2%
174 {%
175   \if0#2\expandafter\XINT_bezout_exitA
176   \else\expandafter\XINT_bezout_loop_c
177   \fi {#1}{#2}%
178 }%
179 \def\XINT_bezout_loop_c #1#2#3#4#5#6#7%

```

```

180 {%
181   \expandafter\XINT_bezout_loop_d\expandafter
182     {\romannumeral0\xintiadd{\XINT_mul_plusplus{}}{#1\xint:#4\xint:}{#6}}%
183     {\romannumeral0\xintiadd{\XINT_mul_plusplus{}}{#1\xint:#5\xint:}{#7}}%
184     {#2}{#3}{#4}{#5}%
185 }%
186 \def\XINT_bezout_loop_d #1#2%
187 {%
188   \expandafter\XINT_bezout_loop_e\expandafter{#2}{#1}%
189 }%
190 \def\XINT_bezout_loop_e #1#2#3#4%
191 {%
192   \expandafter\XINT_bezout_loop_B
193   \romannumeral0\XINT_div_prepare {#3}{#4}{#3}{#2}{#1}%
194 }%

```

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.

```

195 \def\XINT_bezout_exita #1#2#3#4#5#6#7{{-#5}{#4}{#3}}%
196 \def\XINT_bezout_exita #1#2#3#4#5#6#7{{#5}{-#4}{#3}}%

```

7.4 \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.

```

197 \def\xintEuclideanAlgorithm {\romannumeral0\xinteucclideanalgorithm }%
198 \def\xinteucclideanalgorithm #1%
199 {%
200   \expandafter\XINT_euc\expandafter{\romannumeral0\xintiabs{\xintNum{#1}}}%
201 }%
202 \def\XINT_euc #1#2%
203 {%
204   \expandafter\XINT_euc_fork\romannumeral0\xintiabs{\xintNum{#2}}\Z #1\Z
205 }%

```

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

```

206 \def\XINT_euc_fork #1#2\Z #3#4\Z
207 {%
208   \xint_UDzerofork
209   #1\XINT_euc_BisZero
210   #3\XINT_euc_AisZero
211   0\XINT_euc_a
212   \krof
213   {0}{#1#2}{#3#4}{#3#4}{#1#2}}\Z
214 }%

```

Le {} pour protéger $\{A\}\{B\}$ si on s'arrête après une étape (B divise A). On va renvoyer: $\{N\}\{A\}\{D=r(n)\}\{B\}\{q_1\}\{r_1\}\{q_2\}\{r_2\}\{q_3\}\{r_3\}\dots\{q_N\}\{r_N=0\}$

```

215 \def\XINT_euc_AisZero #1#2#3#4#5#6{{1}{0}{#2}{#2}{0}{0}}%
216 \def\XINT_euc_BisZero #1#2#3#4#5#6{{1}{0}{#3}{#3}{0}{0}}%

    {n}{rn}{an}{{qn}{rn}}...{{A}{B}}{}{}Z
a(n) = r(n-1). Pour n=0 on a juste {0}{B}{A}{{A}{B}}{}{}Z
\XINT_div_prepare {u}{v} divide v par u

217 \def\XINT_euc_a #1#2#3%
218 {%
219     \expandafter\XINT_euc_b\the\numexpr #1+\xint_c_i\expandafter.%
220     \romannumeral0\XINT_div_prepare {#2}{#3}{#2}%
221 }%

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

222 \def\XINT_euc_b #1.#2#3#4%
223 {%
224     \XINT_euc_c #3Z {#1}{#3}{#4}{{#2}{#3}}%
225 }%

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

226 \def\XINT_euc_c #1#2Z
227 {%
228     \xint_gob_til_zero #1\XINT_euc_end0\XINT_euc_a
229 }%

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

230 \def\XINT_euc_end0\XINT_euc_a #1#2#3#4Z%
231 {%
232     \expandafter\XINT_euc_end_a
233     \romannumeral0%
234     \XINT_rord_main {}#4{{#1}{#3}}%
235     \xint:
236     \xint_bye\xint_bye\xint_bye\xint_bye
237     \xint_bye\xint_bye\xint_bye\xint_bye
238     \xint:
239 }%
240 \def\XINT_euc_end_a #1#2#3{{#1}{#3}{#2}}%

```

7.5 \xintBezoutAlgorithm

Pour Bezout: objectif, renvoyer

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

```

241 \def\xintBezoutAlgorithm {\romannumeral0\xintbezoutalgorithm }%
242 \def\xintbezoutalgorithm #1%
243 {%
244     \expandafter \XINT_bezalg

```

```

245 \expandafter{\romannumeral0\xintiiabs{\xintNum{#1}}}%
246 }%
247 \def\XINT_bezalg #1#2%
248 {%
249 \expandafter\XINT_bezalg_fork\romannumeral0\xintiiabs{\xintNum{#2}}\Z #1\Z
250 }%

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

251 \def\XINT_bezalg_fork #1#2\Z #3#4\Z
252 {%
253 \xint_UDzerofork
254 #1\XINT_bezalg_BisZero
255 #3\XINT_bezalg_AisZero
256 0\XINT_bezalg_a
257 \krof
258 0{#1#2}{#3#4}1001{#3#4}{#1#2}}{\Z
259 }%
260 \def\XINT_bezalg_AisZero #1#2#3\Z{1}{0}{0}{1}{#2}{#2}{1}{0}{0}{0}{1}}%
261 \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)}... division de #3 par #2

262 \def\XINT_bezalg_a #1#2#3%
263 {%
264 \expandafter\XINT_bezalg_b\the\numexpr #1+\xint_c_i\expandafter.%
265 \romannumeral0\XINT_div_prepare {#2}{#3}{#2}%
266 }%

    {n+1}{q(n+1)}{r(n+1)}{r(n)}{alpha(n)}{beta(n)}{alpha(n-1)}{beta(n-1)}...

267 \def\XINT_bezalg_b #1.#2#3#4#5#6#7#8%
268 {%
269 \expandafter\XINT_bezalg_c\expandafter
270 {\romannumeral0\xintiiadd {\xintiiMul {#6}{#2}}{#8}}%
271 {\romannumeral0\xintiiadd {\xintiiMul {#5}{#2}}{#7}}%
272 {#1}{#2}{#3}{#4}{#5}{#6}%
273 }%

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

274 \def\XINT_bezalg_c #1#2#3#4#5#6%
275 {%
276 \expandafter\XINT_bezalg_d\expandafter {#2}{#3}{#4}{#5}{#6}{#1}%
277 }%

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

278 \def\XINT_bezalg_d #1#2#3#4#5#6#7#8%
279 {%
280 \XINT_bezalg_e #4\Z {#2}{#4}{#5}{#1}{#6}{#7}{#8}{#3}{#4}{#1}{#6}}%
281 }%

```

```

    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.

282 \def\XINT_bezalg_e #1#2\Z
283 {%
284     \xint_gob_til_zero #1\XINT_bezalg_end0\XINT_bezalg_a
285 }%

    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}

286 \def\XINT_bezalg_end0\XINT_bezalg_a #1#2#3#4#5#6#7#8\Z
287 {%
288     \expandafter\XINT_bezalg_end_a
289     \romannumeral0%
290     \XINT_rord_main {}#8{{#1}{#3}}%
291     \xint:
292     \xint_bye\xint_bye\xint_bye\xint_bye
293     \xint_bye\xint_bye\xint_bye\xint_bye
294     \xint:
295 }%

    {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}

296 \def\XINT_bezalg_end_a #1#2#3#4{{#1}{#3}}{0}{1}{#2}{#4}{1}{0}}%

```

7.6 \xintTypesetEuclideanAlgorithm

TYPESETTING

Organisation:

```

{N}{A}{D}{B}{q1}{r1}{q2}{r2}{q3}{r3}...{qN}{rN=0}
\U1 = N = nombre d'étapes, \U3 = PGCD, \U2 = A, \U4=B q1 = \U5, q2 = \U7 --> qn = \U<2n+3>, rn =
\U<2n+4> bn = rn. B = r0. A=r(-1)
r(n-2) = q(n)r(n-1)+r(n) (n e étape)
\U{2n} = \U{2n+3} \times \U{2n+2} + \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

```

```

297 \def\xintTypesetEuclideanAlgorithm {%
298     \unless\ifdefined\xintAssignArray
299         \errmessage
300         {xintgcd: package xinttools is required for \string\xintTypesetEuclideanAlgorithm}%
301         \expandafter\xint_gobble_iii
302     \fi
303     \XINT_TypesetEuclideanAlgorithm
304 }%

```

```

305 \def\XINT_TypesetEuclideanAlgorithm #1#2%
306 {% l'algo remplace #1 et #2 par |#1| et |#2|
307   \par
308   \beginngroup
309     \xintAssignArray\xintEuclideanAlgorithm {#1}{#2}\to\U
310     \edef\A{\U2}\edef\B{\U4}\edef\N{\U1}%
311     \setbox 0 \vbox{\halign {$##$\cr \A\cr \B \cr}}%
312     \count 255 1
313     \xintloop
314       \indent\hbox to \wd 0 {\hfil$\U{\numexpr 2*\count255\relax}$}%
315       ${} = \U{\numexpr 2*\count255 + 3\relax}
316       \times \U{\numexpr 2*\count255 + 2\relax}
317       + \U{\numexpr 2*\count255 + 4\relax}$%
318     \ifnum \count255 < \N
319       \par
320       \advance \count255 1
321     \repeat
322   \endgroup
323 }%

```

7.7 \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

```

324 \def\xintTypesetBezoutAlgorithm {%
325   \unless\ifdefined\xintAssignArray
326     \errmessage
327       {xintgcd: package xinttools is required for \string\xintTypesetBezoutAlgorithm}%
328     \expandafter\xint_gobble_iii
329   \fi
330   \XINT_TypesetBezoutAlgorithm
331 }%
332 \def\XINT_TypesetBezoutAlgorithm #1#2%
333 {%
334   \par
335   \beginngroup
336     \xintAssignArray\xintBezoutAlgorithm {#1}{#2}\to\BEZ
337     \edef\A{\BEZ2}\edef\B{\BEZ6}\edef\N{\BEZ1}% A = |#1|, B = |#2|
338     \setbox 0 \vbox{\halign {$##$\cr \A\cr \B \cr}}%
339     \count255 1
340     \xintloop
341       \indent\hbox to \wd 0 {\hfil$\BEZ{4*\count255 - 2}$}%
342       ${} = \BEZ{4*\count255 + 5}
343       \times \BEZ{4*\count255 + 2}
344       + \BEZ{4*\count255 + 6}$\hfill\break
345     \hbox to \wd 0 {\hfil$\BEZ{4*\count255 + 7}$}%
346     ${} = \BEZ{4*\count255 + 5}

```

```

347     \times \BEZ{4*\count255 + 3}
348     + \BEZ{4*\count255 - 1}$\hfill\break
349     \hbox to \wd 0 {\hfil$\BEZ{4*\count255 +8}$}%
350     ${} = \BEZ{4*\count255 + 5}
351     \times \BEZ{4*\count255 + 4}
352     + \BEZ{4*\count255 }$
353     \par
354     \ifnum \count255 < \N
355     \advance \count255 1
356 \repeat
357 \edef\U{\BEZ{4*\N + 4}}%
358 \edef\V{\BEZ{4*\N + 3}}%
359 \edef\D{\BEZ5}%
360 \ifodd\N
361     $\U\times\A - \V\times \B = -\D$%
362 \else
363     $\U\times\A - \V\times\B = \D$%
364 \fi
365 \par
366 \endgroup
367 }%
368 \XINT_restorecatcodes_endinput%

```


8 Package **xintfrac** implementation

.1	Catcodes, ε -TeX and reload detection . . .	186	.46	<code>\xintDiv</code>	221
.2	Package identification	187	.47	<code>\xintDivFloor</code>	222
.3	<code>\XINT_cntSgnFork</code>	187	.48	<code>\xintDivTrunc</code>	222
.4	<code>\xintLen</code>	187	.49	<code>\xintDivRound</code>	222
.5	<code>\XINT_outfrac</code>	187	.50	<code>\xintModTrunc</code>	222
.6	<code>\XINT_inFrac</code>	188	.51	<code>\xintDivMod</code>	223
.7	<code>\XINT_frac_gen</code>	190	.52	<code>\xintMod</code>	224
.8	<code>\XINT_factortens</code>	192	.53	<code>\xintIsOne</code>	225
.9	<code>\xintEq</code> , <code>\xintNotEq</code> , <code>\xintGt</code> , <code>\xintLt</code> , <code>\xintGtorEq</code> , <code>\xintLtorEq</code> , <code>\xintIsZero</code> , <code>\xintIsNotZero</code> , <code>\xintOdd</code> , <code>\xintEven</code> , <code>\xintifSgn</code> , <code>\xintifCmp</code> , <code>\xintifEq</code> , <code>\xintifGt</code> , <code>\xintifLt</code> , <code>\xintifZero</code> , <code>\xintifNotZero</code> , <code>\xintifOne</code> , <code>\xintifOdd</code>	193	.54	<code>\xintGeq</code>	226
.10	<code>\xintRaw</code>	195	.55	<code>\xintMax</code>	227
.11	<code>\xintiLogTen</code>	195	.56	<code>\xintMaxof</code>	227
.12	<code>\xintPRaw</code>	196	.57	<code>\xintMin</code>	228
.13	<code>\xintSPRaw</code> , <code>\xintFracToSci</code>	196	.58	<code>\xintMinof</code>	228
.14	<code>\xintRawWithZeros</code>	197	.59	<code>\xintCmp</code>	229
.15	<code>\xintDecToString</code>	198	.60	<code>\xintAbs</code>	230
.16	<code>\xintFloor</code> , <code>\xintiFloor</code>	198	.61	<code>\xintOpp</code>	230
.17	<code>\xintCeil</code> , <code>\xintiCeil</code>	199	.62	<code>\xintInv</code>	231
.18	<code>\xintNumerator</code>	199	.63	<code>\xintSgn</code>	231
.19	<code>\xintDenominator</code>	199	.64	<code>\xintGCD</code> , <code>\xintLCM</code>	231
.20	<code>\xintFrac</code>	200	.65	<code>\xintGCDof</code>	231
.21	<code>\xintSignedFrac</code>	200	.66	<code>\xintLCMof</code>	232
.22	<code>\xintFwOver</code>	201	.67	Floating point macros	233
.23	<code>\xintSignedFwOver</code>	201	.68	<code>\xintDigits</code> , <code>\xintSetDigits</code>	234
.24	<code>\xintREZ</code>	202	.69	<code>\xintFloat</code>	234
.25	<code>\xintE</code>	202	.70	<code>\XINTinFloat</code> , <code>\XINTinFloatS</code> , <code>\XINTiLogTen</code>	235
.26	<code>\xintIrr</code> , <code>\xintPIrr</code>	203	.71	<code>\xintPFloat</code>	242
.27	<code>\xintifInt</code>	204	.72	<code>\XINTinFloatFracdigits</code>	244
.28	<code>\xintIsInt</code>	205	.73	<code>\xintFloatAdd</code> , <code>\XINTinFloatAdd</code>	244
.29	<code>\xintJrr</code>	205	.74	<code>\xintFloatSub</code> , <code>\XINTinFloatSub</code>	245
.30	<code>\xintTFrac</code>	206	.75	<code>\xintFloatMul</code> , <code>\XINTinFloatMul</code>	246
.31	<code>\xintTrunc</code> , <code>\xintiTrunc</code>	206	.76	<code>\XINTinFloatInv</code>	246
.32	<code>\xintTTrunc</code>	209	.77	<code>\xintFloatDiv</code> , <code>\XINTinFloatDiv</code>	247
.33	<code>\xintNum</code>	209	.78	<code>\xintFloatPow</code> , <code>\XINTinFloatPow</code>	247
.34	<code>\xintRound</code> , <code>\xintiRound</code>	209	.79	<code>\xintFloatPower</code> , <code>\XINTinFloatPower</code>	251
.35	<code>\xintXTrunc</code>	210	.80	<code>\xintFloatFac</code> , <code>\XINTFloatFac</code>	255
.36	<code>\xintAdd</code>	216	.81	<code>\xintFloatPFactorial</code> , <code>\XINTinFloatPFactorial</code>	260
.37	<code>\xintSub</code>	218	.82	<code>\xintFloatBinomial</code> , <code>\XINTinFloatBinomial</code>	264
.38	<code>\xintSum</code>	218	.83	<code>\xintFloatSqrt</code> , <code>\XINTinFloatSqrt</code>	265
.39	<code>\xintMul</code>	219	.84	<code>\xintFloatE</code> , <code>\XINTinFloatE</code>	268
.40	<code>\xintSqr</code>	219	.85	<code>\XINTinFloatMod</code>	269
.41	<code>\xintPow</code>	219	.86	<code>\XINTinFloatDivFloor</code>	269
.42	<code>\xintFac</code>	220	.87	<code>\XINTinFloatDivMod</code>	269
.43	<code>\xintBinomial</code>	220	.88	<code>\xintifFloatInt</code>	270
.44	<code>\xintPFactorial</code>	221	.89	<code>\xintFloatIsInt</code>	270
.45	<code>\xintPrd</code>	221	.90	<code>\XINTinFloatdigits</code> , <code>\XINTinFloatSqrtdigits</code> , <code>\XINTinFloatFacdigits</code> , <code>\XINTiLogTendigits</code>	270

.91	(WIP) \XINTinRandomFloatS, \XINTinRandomFloatSdigits	270	.94	\PoorManPowerOfTen	272
.92	(WIP) \XINTinRandomFloatSixteen	271	.95	\PoorManPower	272
.93	\PoorManLogBaseTen	271	.96	Support macros for natural logarithm and exponential xintexpr functions	273

The commenting is currently (2020/02/19) very sparse.

8.1 Catcodes, ε -TeX and reload detection

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

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2  \catcode13=5      % ^^M
3  \endlinechar=13 %
4  \catcode123=1     % {
5  \catcode125=2     % }
6  \catcode64=11     % @
7  \catcode35=6      % #
8  \catcode44=12     % ,
9  \catcode45=12     % -
10 \catcode46=12     % .
11 \catcode58=12     % :
12 \let\z\endgroup
13 \expandafter\let\expandafter\x\csname ver@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 [2020/02/19 v1.4a Expandable operations on fractions (JFB)]%
```

8.3 \XINT_cntSgnFork

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

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

8.4 \xintLen

The used formula is disputable, the idea is that A/1 and A should have same length. Venerable code rewritten for 1.2i, following updates to \xintLength in xintkernel.sty. And sadly, I forgot on this occasion that this macro is not supposed to count the sign... Fixed in 1.2k.

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

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

8.5 \XINT_outfrac

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

```
68 \def\XINT_outfrac #1#2#3%
69 {%
```

```

70 \ifcase\XINT_cntSgn #3\xint:
71   \expandafter \XINT_outfrac_divisionbyzero
72 \or
73   \expandafter \XINT_outfrac_P
74 \else
75   \expandafter \XINT_outfrac_N
76 \fi
77 {#2}{#3}[#1]%
78 }%
79 \def\XINT_outfrac_divisionbyzero #1#2%
80 {%
81   \XINT_signalcondition{DivisionByZero}{Division of #1 by #2}{0/1[0]}%
82 }%
83 \def\XINT_outfrac_P#1{%
84 \def\XINT_outfrac_P ##1##2%
85   {\if0\XINT_Sgn ##1\xint:\expandafter\XINT_outfrac_Zero\fi##1/##2}%
86 }\XINT_outfrac_P{ }%
87 \def\XINT_outfrac_Zero #1[#2]{ 0/1[0]}%
88 \def\XINT_outfrac_N #1#2%
89 {%
90   \expandafter\XINT_outfrac_N_a\expandafter
91   {\romannumeral0\XINT_opp #2}{\romannumeral0\XINT_opp #1}%
92 }%
93 \def\XINT_outfrac_N_a #1#2%
94 {%
95   \expandafter\XINT_outfrac_P\expandafter {#2}{#1}%
96 }%

```

8.6 \XINT_inFrac

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

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

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

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

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

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

While I was at it 1.2 added `\numexpr` parsing of the N, which earlier was restricted to be only explicit digits. I allowed `[]` with empty N, but the way I did it in 1.2 with `\the\numexpr 0#1` was buggy, as it did not allow `#1` to be a `\count` for example or itself a `\numexpr` (although such inputs

were not previously allowed, I later turned out to use them in the code itself, e.g. the float factorial of version 1.2f). The better way would be `\the\numexpr#1+\xint_c_` but 1.2f finally does only `\the\numexpr #1` and `#1` is not allowed to be empty.

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

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

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

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

1.2l fixes frailty of `\XINT_infrac` (hence basically of all `xintfrac` macros) respective to non terminated `\numexpr` input: `\xintRaw{\the\numexpr1}` for example. The issue was that `\numexpr` sees the `/` and expands what's next. But even `\numexpr 1//` for example creates an error, and to my mind this is a defect of `\numexpr`. It should be able to trace back and see that `/` was used as delimiter not as operator. Anyway, I thus fixed this problem belatedly here regarding `\XINT_infrac`.

```

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

```

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

```

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

```

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

```

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

```

8.7 \XINT_frac_gen

Extended in 1.07 to recognize and accept scientific notation both at the numerator and (possible) denominator. Only a lowercase e will do here, but uppercase E is possible within 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:\xint:
162 \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
163 \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
164 ~#2#1!%
165 }%
166 \def\XINT_frac_gen_C #1!#2.#3%
167 {%
168 \xint_UDXINTWfork
169 #3\XINT_frac_gen_Ca
170 \XINT_W\XINT_frac_gen_Cb
171 \krof
172 #1!#2.#3%
173 }%
174 \def\XINT_frac_gen_Ca #1~#2!#3e#4e#5\XINT_T
175 {%
176 \expandafter\XINT_frac_gen_F\the\numexpr #4-#1\expandafter
177 ~\romannumeral0\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
178 #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z~#3~%
179 }%
180 \def\XINT_frac_gen_Cb #1.#2e%
181 {%
182 \expandafter\XINT_frac_gen_Cc\romannumeral`&&@#2.#1e%
183 }%
184 \def\XINT_frac_gen_Cc #1.#2~#3!#4e#5e#6\XINT_T
185 {%
186 \expandafter\XINT_frac_gen_F\the\numexpr #5-#2-%
187
188 \numexpr\XINT_length_loop
189 #1\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
190 \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
191 \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
192
193 \relax\expandafter~%
194 \romannumeral0\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
195 #3\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z
196 ~#4#1~%
197 }%
198 \def\XINT_frac_gen_F #1~#2%
199 {%
200 \xint_UDzerominusfork
201 #2-\XINT_frac_gen_Gdivbyzero
202 0#2{\XINT_frac_gen_G -{}}%
203 0-{\XINT_frac_gen_G }{#2}%
204 \krof #1~%
205 }%
206 \def\XINT_frac_gen_Gdivbyzero #1~~#2~%
207 {%
208 \expandafter\XINT_frac_gen_Gdivbyzero_a
209 \romannumeral0\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
210 #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z~#1~%

```

```

209 }%
210 \def\XINT_frac_gen_Gdivbyzero_a #1~#2~%
211 {%
212     \XINT_signalcondition{DivisionByZero}{Division of #1 by zero}{\{#2\}{#1\}{0}}%
213 }%
214 \def\XINT_frac_gen_G #1#2#3~#4~#5~%
215 {%
216     \expandafter\XINT_frac_gen_Ga
217     \romannumeral0\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
218     #1#5\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z~#3~{#2#4}%
219 }%
220 \def\XINT_frac_gen_Ga #1#2~#3~%
221 {%
222     \xint_gob_til_zero #1\XINT_frac_gen_zero 0%
223     {#3}{#1#2}%
224 }%
225 \def\XINT_frac_gen_zero 0#1#2#3{\{0\}{0\}{1}}%

```

8.8 \XINT_factortens

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

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

Testing shows significant gain at 100 digits or more.

```

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

```



```

251 \def\XINT_factortens_f #1#2\xint_c_i#3.#4.#5.%
252   {\expandafter\XINT_factortens_g\the\numexpr#1+#5.#3.}%
253 \def\XINT_factortens_g #1.#2.{.\XINT_factortens_y#2.#1.}%
254 \def\XINT_factortens_e #1..#2.%
255   {\expandafter.\expandafter\XINT_factortens_c
256     \the\numexpr\xint_c_ix+#2.}%

```

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

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

```

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

```

```

295         \else\expandafter\xint_stop_atfirstofthree
296     \fi
297 }%
298 \def\xintifCmp{\romannumeral0\xintifcmp }%
299 \def\xintifcmp #1#2%
300 {%
301     \ifcase\xintCmp {#1}{#2}
302         \expandafter\xint_stop_atsecondofthree
303     \or\expandafter\xint_stop_atthirdofthree
304     \else\expandafter\xint_stop_atfirstofthree
305     \fi
306 }%
307 \def\xintifEq {\romannumeral0\xintifeq }%
308 \def\xintifeq #1#2%
309 {%
310     \if0\xintCmp{#1}{#2}%
311         \expandafter\xint_stop_atfirstoftwo
312     \else\expandafter\xint_stop_atsecondoftwo
313     \fi
314 }%
315 \def\xintifGt {\romannumeral0\xintifgt }%
316 \def\xintifgt #1#2%
317 {%
318     \if1\xintCmp{#1}{#2}%
319         \expandafter\xint_stop_atfirstoftwo
320     \else\expandafter\xint_stop_atsecondoftwo
321     \fi
322 }%
323 \def\xintifLt {\romannumeral0\xintiflt }%
324 \def\xintiflt #1#2%
325 {%
326     \ifnum\xintCmp{#1}{#2}<\xint_c_
327         \expandafter\xint_stop_atfirstoftwo
328     \else \expandafter\xint_stop_atsecondoftwo
329     \fi
330 }%
331 \def\xintifZero {\romannumeral0\xintifzero }%
332 \def\xintifzero #1%
333 {%
334     \if0\xintSgn{#1}%
335         \expandafter\xint_stop_atfirstoftwo
336     \else
337         \expandafter\xint_stop_atsecondoftwo
338     \fi
339 }%
340 \def\xintifNotZero{\romannumeral0\xintifnotzero }%
341 \def\xintifnotzero #1%
342 {%
343     \if0\xintSgn{#1}%
344         \expandafter\xint_stop_atsecondoftwo
345     \else
346         \expandafter\xint_stop_atfirstoftwo

```

```

347     \fi
348 }%
349 \def\xintifOne {\romannumeral0\xintifone }%
350 \def\xintifone #1%
351 {%
352     \if1\xintIsOne{#1}%
353     \expandafter\xint_stop_atfirstoftwo
354     \else
355     \expandafter\xint_stop_atsecondoftwo
356     \fi
357 }%
358 \def\xintifOdd {\romannumeral0\xintifodd }%
359 \def\xintifodd #1%
360 {%
361     \if\xintOdd{#1}1%
362     \expandafter\xint_stop_atfirstoftwo
363     \else
364     \expandafter\xint_stop_atsecondoftwo
365     \fi
366 }%

```

8.10 \xintRaw

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

```

367 \def\xintRaw {\romannumeral0\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 \xintiLogTen

New at 1.3e

```

373 \def\xintiLogTen {\the\numexpr\xintilogten}%
374 \def\xintilogten
375 {%
376     \expandafter\XINT_ilogten\romannumeral0\xintraw
377 }%
378 \def\XINT_ilogten #1%
379 {%
380     \xint_UDzerominusfork
381     0#1\XINT_ilogten_p
382     #1-\XINT_ilogten_z
383     0-\XINT_ilogten_p#1}%
384     \krof
385 }%
386 \def\XINT_ilogten_z #1[#2]{-7FFF8000\relax}%
387 \def\XINT_ilogten_p #1/#2[#3]%
388 {%

```

```

389      #3+\expandafter\XINT_ilogten_a
390      \the\numexpr\xintLength{#1}\expandafter.\the\numexpr\xintLength{#2}.#1.#2.%
391 }%
392 \def\XINT_ilogten_a #1.#2.%
393 {%
394     #1-#2\ifnum#1>#2
395         \expandafter\XINT_ilogten_aa
396     \else
397         \expandafter\XINT_ilogten_ab
398     \fi #1.#2.%
399 }%
400 \def\XINT_ilogten_aa #1.#2.#3.#4.%
401 {%
402     \xintiiflt{#3}{\XINT_dsx_addzerosnofuss{#1-#2}#4;}{-1}{}\relax
403 }%
404 \def\XINT_ilogten_ab #1.#2.#3.#4.%
405 {%
406     \xintiiflt{\XINT_dsx_addzerosnofuss{#2-#1}#3;}{#4}{-1}{}\relax
407 }%

```

8.12 \xintPRaw

1.09b

```

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

```

8.13 \xintSPRaw, \xintFracToSci

This private macro was for usage by \xinttheexpr. It got moved here at 1.4.

Attention that \xintSPRaw assumes that if the number has no [N] part it does not have a fraction part /B either. Indeed this was the case always with 1.3f (parsing of an integer by \xintexpr does not add the [0] because the code is shared with \xintiexpr and when there is /B, \xintexpr always

adds [0]; even qfrac() parses via \xintRaw; and reduce() internally uses \xintIrr whose outputs is A/B but it add [0]).

\xintFracToSci is now used in its place. As reduce() does not anymore append the [0] at 1.4, \xintFracToSci has to recognize A, A[N], A/B and A/B[N] but does not have to parse multiple plus or minus signs or scientific part etc like \xintRaw knows. It has to identify say 0/5 (although I don't think that can arise) and -0 is never occurring.

The difference with former case is that it outputs AeN/B hence does not anymore use the xintfrac.sty raw format. It will not printe the /B if B=1 and not print the «eN» if N is zero.

If input is empty \xintFracToSci output is also empty, whereas \xintRaw produces 0/1[0] out of empty. But \XINTexprprint anyhow has it own special routine for empty input.

```

429 \def\xintSPraw    {\romannumeral0\xintspraw}%
430 \def\xintspraw   #1{\expandafter\XINT_spraw\romannumeral`&&@#1[\W]}%
431 \def\XINT_spraw  #1[#2#3]{\xint_gob_til_W #2\XINT_spraw_a\W\XINT_spraw_p #1[#2#3]}%
432 \def\XINT_spraw_a\W\XINT_spraw_p #1[\W]{ #1}%
433 \def\XINT_spraw_p #1[\W]{\xintpraw {#1}}%
434 \def\xintFracToSci #1%
435   {\expandafter\XINT_FracToSci\romannumeral`&&@#1/\W[\R]}%
436 \def\XINT_FracToSci #1/#2#3[#4%
437 {%
438   \xint_gob_til_W #2\XINT_FracToSci_no\W
439   \xint_gob_til_R #4\XINT_FracToSci_yesno\R
440   \XINT_FracToSci_yesyes #1/#2#3[#4%
441}%
442 \def\XINT_FracToSci_no #1\XINT_FracToSci_yesyes #2[#3%
443 {%
444   \xint_gob_til_R #3\XINT_FracToSci_nono\R
445   \XINT_FracToSci_noyes #2[#3%
446}%
447 \def\XINT_FracToSci_nono\R\XINT_FracToSci_noyes #1/\W[\R]{#1}%
448 \def\XINT_FracToSci_noyes #1#2[#3]/\W[\R]%
449 {%
450   #1\xint_gob_til_zero#1\expandafter\iffalse\xint_gobble_ii0\iftrue
451   #2\ifnum #3=\xint_c_\else\xintFracToSciE#3\fi\fi
452}%
453 \def\XINT_FracToSci_yesno\R\XINT_FracToSci_yesyes #1#2/#3/\W[\R]%
454 {%
455   #1\xint_gob_til_zero#1\expandafter\iffalse\xint_gobble_ii0\iftrue
456   #2\if\XINT_isOne{#3}1\else/#3\fi\fi
457}%
458 \def\XINT_FracToSci_yesyes #1#2/#3[#4]/\W[\R]%
459 {%
460   #1\xint_gob_til_zero#1\expandafter\iffalse\xint_gobble_ii0\iftrue
461   #2\ifnum #4=\xint_c_\else\xintFracToSciE#4\fi
462   \if\XINT_isOne{#3}1\else/#3\fi\fi
463}%
464 \def\xintFracToSciE{e}%

```

8.14 \xintRawWithZeros

This was called \xintRaw in versions earlier than 1.07

```

465 \def\xintRawWithZeros {\romannumeral0\xinrawwithzeros}%

```

```

466 \def\xintraewithzeros
467 {%
468     \expandafter\XINT_rawz_fork\romannumeral0\XINT_infrac
469 }%

470 \def\XINT_rawz_fork #1%
471 {%
472     \ifnum#1<\xint_c_
473         \expandafter\XINT_rawz_Ba
474     \else
475         \expandafter\XINT_rawz_A
476     \fi
477     #1.%
478 }%
479 \def\XINT_rawz_A #1.#2#3{\XINT_dsx_addzeros{#1}#2;/#3}%
480 \def\XINT_rawz_Ba -#1.#2#3{\expandafter\XINT_rawz_Bb
481     \expandafter{\romannumeral0\XINT_dsx_addzeros{#1}#3;}{#2}}%
482 \def\XINT_rawz_Bb #1#2{ #2/#1}%

```

8.15 \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.

```

483 \def\xintDecToString{\romannumeral0\xintdectostr}%
484 \def\xintdectostr#1{\expandafter\XINT_dectostr\romannumeral0\xintra{#1}}%
485 \def\XINT_dectostr #1/#2[#3]{\xintiifZero {#1}%
486     \XINT_dectostr_z
487     {\if1\XINT_isOne{#2}\expandafter\XINT_dectostr_a
488         \else\expandafter\XINT_dectostr_b
489         \fi}%
490     #1/#2[#3]}%
491 }%
492 \def\XINT_dectostr_z#1[#2]{ 0}%
493 \def\XINT_dectostr_a#1/#2[#3]{%
494     \ifnum#3<\xint_c_\xint_dothis{\xinttrunc{-#3}{#1[#3]}}\fi
495     \xint_orthat{\xintiie{#1}{#3}}%
496 }%
497 \def\XINT_dectostr_b#1/#2[#3]{% just to handle this somehow
498     \ifnum#3<\xint_c_\xint_dothis{\xinttrunc{-#3}{#1[#3]}/#2}\fi
499     \xint_orthat{\xintiie{#1}{#3}/#2}%
500 }%

```

8.16 \xintFloor, \xintiFloor

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

```

501 \def\xintFloor {\romannumeral0\xintfloor }%
502 \def\xintfloor #1% devrais-je faire \xintREZ?
503     {\expandafter\XINT_ifloor \romannumeral0\xintraewithzeros {#1}./1[0]}%
504 \def\xintiFloor {\romannumeral0\xintifloor }%

```

```

505 \def\xintifloor #1%
506     {\expandafter\XINT_ifloor \romannumeral0\xintraffracwithzeros {#1}.}%
507 \def\XINT_ifloor #1/#2.{\xintiifrac {#1}{#2}}%

```

8.17 \xintCeil, \xintiCeil

1.09a

```

508 \def\xintCeil {\romannumeral0\xintceil }%
509 \def\xintceil #1{\xintiifrac {\xintFloor {\xintOpp{#1}}}%
510 \def\xintiCeil {\romannumeral0\xinticeil }%
511 \def\xinticeil #1{\xintiifrac {\xintiFloor {\xintOpp{#1}}}%

```

8.18 \xintNumerator

```

512 \def\xintNumerator {\romannumeral0\xintnumerator }%
513 \def\xintnumerator
514 {%
515     \expandafter\XINT_numer\romannumeral0\XINT_infrac
516 }%
517 \def\XINT_numer #1%
518 {%
519     \ifcase\XINT_cntSgn #1\xint:
520     \expandafter\XINT_numer_B
521     \or
522     \expandafter\XINT_numer_A
523     \else
524     \expandafter\XINT_numer_B
525     \fi
526     {#1}%
527 }%
528 \def\XINT_numer_A #1#2#3{\XINT_dsx_addzeros{#1}#2;}%
529 \def\XINT_numer_B #1#2#3{ #2}%

```

8.19 \xintDenominator

```

530 \def\xintDenominator {\romannumeral0\xintdenominator }%
531 \def\xintdenominator
532 {%
533     \expandafter\XINT_denom_fork\romannumeral0\XINT_infrac
534 }%
535 \def\XINT_denom_fork #1%
536 {%
537     \ifnum#1<\xint_c_
538     \expandafter\XINT_denom_B
539     \else
540     \expandafter\XINT_denom_A
541     \fi
542     #1.%
543 }%
544 \def\XINT_denom_A #1.#2#3{ #3}%
545 \def\XINT_denom_B -#1.#2#3{\XINT_dsx_addzeros{#1}#3;}%

```

8.20 \xintFrac

Useless typesetting macro.

```

546 \def\xintFrac {\romannumeral0\xintfrac }%
547 \def\xintfrac #1%
548 {%
549     \expandafter\XINT_fracfrac_A\romannumeral0\XINT_infrac {#1}%
550 }%
551 \def\XINT_fracfrac_A #1{\XINT_fracfrac_B #1\Z }%
552 \catcode`^=7
553 \def\XINT_fracfrac_B #1#2\Z
554 {%
555     \xint_gob_til_zero #1\XINT_fracfrac_C 0\XINT_fracfrac_D {10^{#1#2}}%
556 }%
557 \def\XINT_fracfrac_C 0\XINT_fracfrac_D #1#2#3%
558 {%
559     \if1\XINT_isOne {#3}%
560         \xint_afterfi {\expandafter\xint_stop_atfirstoftwo\xint_gobble_ii }%
561     \fi
562     \space
563     \frac {#2}{#3}%
564 }%
565 \def\XINT_fracfrac_D #1#2#3%
566 {%
567     \if1\XINT_isOne {#3}\XINT_fracfrac_E\fi
568     \space
569     \frac {#2}{#3}#1%
570 }%
571 \def\XINT_fracfrac_E \fi\space\frac #1#2{\fi \space #1\cdot }%

```

8.21 \xintSignedFrac

```

572 \def\xintSignedFrac {\romannumeral0\xintsignedfrac }%
573 \def\xintsignedfrac #1%
574 {%
575     \expandafter\XINT_sgnfrac_a\romannumeral0\XINT_infrac {#1}%
576 }%
577 \def\XINT_sgnfrac_a #1#2%
578 {%
579     \XINT_sgnfrac_b #2\Z {#1}%
580 }%
581 \def\XINT_sgnfrac_b #1%
582 {%
583     \xint_UDsignfork
584         #1\XINT_sgnfrac_N
585         -{\XINT_sgnfrac_P #1}%
586     \krof
587 }%
588 \def\XINT_sgnfrac_P #1\Z #2%
589 {%
590     \XINT_fracfrac_A {#2}{#1}%
591 }%

```



```

592 \def\XINT_sgnfrac_N
593 {%
594   \expandafter\romannumeral0\XINT_sgnfrac_P
595 }%

```

8.22 \xintFwOver

```

596 \def\xintFwOver {\romannumeral0\xintfwover }%
597 \def\xintfwover #1%
598 {%
599   \expandafter\XINT_fwover_A\romannumeral0\XINT_infrac {#1}%
600 }%
601 \def\XINT_fwover_A #1{\XINT_fwover_B #1\Z }%
602 \def\XINT_fwover_B #1#2\Z
603 {%
604   \xint_gob_til_zero #1\XINT_fwover_C 0\XINT_fwover_D {10^{#1#2}}%
605 }%
606 \catcode\^ =11
607 \def\XINT_fwover_C #1#2#3#4#5%
608 {%
609   \if0\XINT_isOne {#5}\xint_afterfi { {#4\over #5}}%
610   \else\xint_afterfi { #4}%
611   \fi
612 }%
613 \def\XINT_fwover_D #1#2#3%
614 {%
615   \if0\XINT_isOne {#3}\xint_afterfi { {#2\over #3}}%
616   \else\xint_afterfi { #2\cdot }%
617   \fi
618   #1%
619 }%

```

8.23 \xintSignedFwOver

```

620 \def\xintSignedFwOver {\romannumeral0\xintsignedfwover }%
621 \def\xintsignedfwover #1%
622 {%
623   \expandafter\XINT_sgnfwover_a\romannumeral0\XINT_infrac {#1}%
624 }%
625 \def\XINT_sgnfwover_a #1#2%
626 {%
627   \XINT_sgnfwover_b #2\Z {#1}%
628 }%
629 \def\XINT_sgnfwover_b #1%
630 {%
631   \xint_UDsignfork
632   #1\XINT_sgnfwover_N
633   -{\XINT_sgnfwover_P #1}%
634   \krof
635 }%
636 \def\XINT_sgnfwover_P #1\Z #2%
637 {%
638   \XINT_fwover_A {#2}{#1}%
639 }%
640 \def\XINT_sgnfwover_N

```

```
641 {%
642   \expandafter-\romannumeral0\XINT_sgnfwover_P
643 }%
```

8.24 \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.

```
644 \def\xintREZ {\romannumeral0\xintrez }%
645 \def\xintrez
646 {%
647   \expandafter\XINT_rez_A\romannumeral0\XINT_infrac
648 }%
649 \def\XINT_rez_A #1#2%
650 {%
651   \XINT_rez_AB #2\Z {#1}%
652 }%
653 \def\XINT_rez_AB #1%
654 {%
655   \xint_UDzerominusfork
656   #1-\XINT_rez_zero
657   0#1\XINT_rez_neg
658   0-{\XINT_rez_B #1}%
659   \krof
660 }%
661 \def\XINT_rez_zero #1\Z #2#3{ 0/1[0]}%
662 \def\XINT_rez_neg {\expandafter-\romannumeral0\XINT_rez_B }%
663 \def\XINT_rez_B #1\Z
664 {%
665   \expandafter\XINT_rez_C\romannumeral0\XINT_factortens {#1}%
666 }%
667 \def\XINT_rez_C #1.#2.#3#4%
668 {%
669   \expandafter\XINT_rez_D\romannumeral0\XINT_factortens {#4}#3+#2.#1.%
670 }%
671 \def\XINT_rez_D #1.#2.#3.%
672 {%
673   \expandafter\XINT_rez_E\the\numexpr #3-#2.#1.%
674 }%
675 \def\XINT_rez_E #1.#2.#3.{ #3/#2[#1]}%
```

8.25 \xintE

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

1.1 modifies and moves \xintiiE to xint.sty.

```
676 \def\xintE {\romannumeral0\xinte }%
677 \def\xinte #1%
678 {%
679   \expandafter\XINT_e \romannumeral0\XINT_infrac {#1}%
680 }%
```

```

681 \def\XINT_e #1#2#3#4%
682 {%
683     \expandafter\XINT_e_end\the\numexpr #1+#4.{#2}{#3}%
684 }%
685 \def\XINT_e_end #1.#2#3{ #2/#3[#1]}%

```

8.26 \xintIrr, \xintPIrr

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

```

686 \def\xintIrr {\romannumeral0\xintirr }%
687 \def\xintPIrr{\romannumeral0\xintpirr }%
688 \def\xintirr #1%
689 {%
690     \expandafter\XINT_irr_start\romannumeral0\xintraawwithzeros {#1}\Z
691 }%
692 \def\xintpirr #1%
693 {%
694     \expandafter\XINT_pirr_start\romannumeral0\xintraaw{#1}%
695 }%
696 \def\XINT_irr_start #1#2/#3\Z
697 {%
698     \if0\XINT_isOne {#3}%
699     \xint_afterfi
700     {\xint_UDsignfork
701         #1\XINT_irr_negative
702         -{\XINT_irr_nonneg #1}%
703     \krof}%
704 \else
705     \xint_afterfi{\XINT_irr_denomisine #1}%
706 \fi
707 #2\Z {#3}%
708 }%
709 \def\XINT_pirr_start #1#2/#3[%
710 {%
711     \if0\XINT_isOne {#3}%
712     \xint_afterfi
713     {\xint_UDsignfork
714         #1\XINT_irr_negative
715         -{\XINT_irr_nonneg #1}%
716     \krof}%
717 \else
718     \xint_afterfi{\XINT_irr_denomisine #1}%
719 \fi
720 #2\Z {#3}[%
721 }%
722 \def\XINT_irr_denomisine #1\Z #2{ #1/1}% changed in 1.08
723 \def\XINT_irr_negative #1\Z #2{\XINT_irr_D #1\Z #2\Z -}%
724 \def\XINT_irr_nonneg #1\Z #2{\XINT_irr_D #1\Z #2\Z \space}%
725 \def\XINT_irr_D #1#2\Z #3#4\Z
726 {%
727     \xint_UDzerosfork
728     #3#1\XINT_irr_indeterminate

```

```

729      #30\XINT_irr_divisionbyzero
730      #10\XINT_irr_zero
731      00\XINT_irr_loop_a
732      \krof
733      {#3#4}{#1#2}{#3#4}{#1#2}%
734 }%
735 \def\XINT_irr_indeterminate #1#2#3#4#5%
736 {%
737     \XINT_signalcondition{DivisionUndefined}{indeterminate: 0/0}{0/1}%
738 }%
739 \def\XINT_irr_divisionbyzero #1#2#3#4#5%
740 {%
741     \XINT_signalcondition{DivisionByZero}{vanishing denominator: #5#2/0}{0/1}%
742 }%
743 \def\XINT_irr_zero #1#2#3#4#5{ 0/1}% changed in 1.08
744 \def\XINT_irr_loop_a #1#2%
745 {%
746     \expandafter\XINT_irr_loop_d
747     \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
748 }%
749 \def\XINT_irr_loop_d #1#2%
750 {%
751     \XINT_irr_loop_e #2\Z
752 }%
753 \def\XINT_irr_loop_e #1#2\Z
754 {%
755     \xint_gob_til_zero #1\XINT_irr_loop_exit0\XINT_irr_loop_a {#1#2}%
756 }%
757 \def\XINT_irr_loop_exit0\XINT_irr_loop_a #1#2#3#4%
758 {%
759     \expandafter\XINT_irr_loop_exitb\expandafter
760     {\romannumeral0\xintiique {#3}{#2}}%
761     {\romannumeral0\xintiique {#4}{#2}}%
762 }%
763 \def\XINT_irr_loop_exitb #1#2%
764 {%
765     \expandafter\XINT_irr_finish\expandafter {#2}{#1}%
766 }%
767 \def\XINT_irr_finish #1#2#3{#3#1/#2}% changed in 1.08

```

8.27 \xintifInt

```

768 \def\xintifInt {\romannumeral0\xintifint }%
769 \def\xintifint #1{\expandafter\XINT_ifint\romannumeral0\xintraawithzeros {#1}.}%
770 \def\XINT_ifint #1/#2.%
771 {%
772     \if 0\xintiiRem {#1}{#2}%
773     \expandafter\xint_stop_atfirstoftwo
774     \else
775     \expandafter\xint_stop_atsecondoftwo
776     \fi
777 }%

```

8.28 \xintIsInt

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

```
778 \def\xintIsInt    {\romannumeral0\xintisint }%
779 \def\xintisint #1%
780   {\expandafter\XINT_ifint\romannumeral0\xintraawithzeros {#1}.10}%
```

8.29 \xintJrr

```
781 \def\xintJrr {\romannumeral0\xintjrr }%
782 \def\xintjrr #1%
783 {%
784   \expandafter\XINT_jrr_start\romannumeral0\xintraawithzeros {#1}\Z
785 }%
786 \def\XINT_jrr_start #1#2/#3\Z
787 {%
788   \if0\XINT_isOne {#3}\xint_afterfi
789     {\xint_UDsignfork
790       #1\XINT_jrr_negative
791       -{\XINT_jrr_nonneg #1}%
792       \krof}%
793   \else
794     \xint_afterfi{\XINT_jrr_denomisone #1}%
795     \fi
796     #2\Z {#3}%
797 }%
798 \def\XINT_jrr_denomisone #1\Z #2{ #1/1}% changed in 1.08
799 \def\XINT_jrr_negative  #1\Z #2{\XINT_jrr_D #1\Z #2\Z -}%
800 \def\XINT_jrr_nonneg    #1\Z #2{\XINT_jrr_D #1\Z #2\Z \space}%
801 \def\XINT_jrr_D #1#2\Z #3#4\Z
802 {%
803   \xint_UDzerosfork
804     #3#1\XINT_jrr_indeterminate
805     #30\XINT_jrr_divisionbyzero
806     #10\XINT_jrr_zero
807     00\XINT_jrr_loop_a
808   \krof
809   {#3#4}{#1#2}1001%
810 }%
811 \def\XINT_jrr_indeterminate #1#2#3#4#5#6#7%
812 {%
813   \XINT_signalcondition{DivisionUndefined}{indeterminate: 0/0}{0/1}%
814 }%
815 \def\XINT_jrr_divisionbyzero #1#2#3#4#5#6#7%
816 {%
817   \XINT_signalcondition{DivisionByZero}{Vanishing denominator: #7#2/0}{0/1}%
818 }%
819 \def\XINT_jrr_zero #1#2#3#4#5#6#7{ 0/1}% changed in 1.08
820 \def\XINT_jrr_loop_a #1#2%
821 {%
822   \expandafter\XINT_jrr_loop_b
823   \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
```

```

824 }%
825 \def\XINT_jrr_loop_b #1#2#3#4#5#6#7%
826 {%
827     \expandafter \XINT_jrr_loop_c \expandafter
828         {\romannumeral0\xintiadd{\XINT_mul_fork #4\xint:#1\xint:}{#6}}%
829         {\romannumeral0\xintiadd{\XINT_mul_fork #5\xint:#1\xint:}{#7}}%
830     {#2}{#3}{#4}{#5}%
831 }%
832 \def\XINT_jrr_loop_c #1#2%
833 {%
834     \expandafter \XINT_jrr_loop_d \expandafter{#2}{#1}%
835 }%
836 \def\XINT_jrr_loop_d #1#2#3#4%
837 {%
838     \XINT_jrr_loop_e #3\Z {#4}{#2}{#1}%
839 }%
840 \def\XINT_jrr_loop_e #1#2\Z
841 {%
842     \xint_gob_til_zero #1\XINT_jrr_loop_exit0\XINT_jrr_loop_a {#1#2}%
843 }%
844 \def\XINT_jrr_loop_exit0\XINT_jrr_loop_a #1#2#3#4#5#6%
845 {%
846     \XINT_irr_finish {#3}{#4}%
847 }%

```

8.30 \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 - 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.

```

848 \def\xintTFrac {\romannumeral0\xinttfrac }%
849 \def\xinttfrac #1{\expandafter\XINT_tfrac_fork\romannumeral0\xintrawwithzeros {#1}\Z }%
850 \def\XINT_tfrac_fork #1%
851 {%
852     \xint_UDzerominusfork
853     #1-\XINT_tfrac_zero
854     0#1{\xintioopp\XINT_tfrac_P }%
855     0-{\XINT_tfrac_P #1}%
856     \krof
857 }%
858 \def\XINT_tfrac_zero #1\Z { 0/1[0]}%
859 \def\XINT_tfrac_P #1/#2\Z {\expandafter\XINT_rez_AB
860     \romannumeral0\xintiirem{#1}{#2}\Z {0}{#2}}%

```

8.31 \xintTrunc, \xintiTrunc

This of course has a long history. Only showing here some comments.

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.

1.4a (2020/02/18) adds handling of a negative first argument.

Zero input still gives single digit 0 output as I did not want to complicate the code. But if quantization gives 0, the exponent $[D]$ will be there. Well actually eD because of problem that sign of original is preserved in output so we can have -0 and I can not use $-0[D]$ notation as it is not legal for strict format. So I will use $-0eD$ hence eD generally even though this means so slight suboptimality for `trunc()` function in `\xintexpr`.

The idea to give a meaning to negative D (in the context of optional argument to `\xintiexpr`) was suggested a long time ago by Kpym (October 20, 2015). His suggestion was then to treat it as positive D but trim trailing zeroes. But since then, there is `\xintDecToString` which can be combined with `\xintREZ`, and I feel matters of formatting output require a whole module (or rather use existing third-party tools), and I decided to opt rather for an operation similar as the `quantize()` of Python Decimal module. I.e. we truncate (or round) to an integer multiple of a given power of 10.

Other reason to decide to do this is that it looks as if I don't even need to understand the original code to hack into its ending via `\XINT_trunc_G` or `\XINT_itrunc_G`. For the latter it looks as if logically I simply have to do nothing. For the former I simply have to add some eD postfix.

```

861 \def\xintTrunc {\romannumeral0\xinttrunc}%
862 \def\xintiTrunc {\romannumeral0\xintitrunc}%
863 \def\xinttrunc #1{\expandafter\XINT_trunc\the\numexpr#1.\XINT_trunc_G}%
864 \def\xintitrunc #1{\expandafter\XINT_trunc\the\numexpr#1.\XINT_itrunc_G}%
865 \def\XINT_trunc #1.#2#3%
866 {%
867   \expandafter\XINT_trunc_a\romannumeral0\XINT_infrac{#3}#1.#2%
868}%
869 \def\XINT_trunc_a #1#2#3#4.#5%
870 {%
871   \if0\XINT_Sgn#2\xint:\xint_dothis\XINT_trunc_zero\fi
872   \if1\XINT_is_One#3XY\xint_dothis\XINT_trunc_sp_b\fi
873   \xint_orthat\XINT_trunc_b #1+#4.{#2}{#3}#5#4.%
874}%
875 \def\XINT_trunc_zero #1.#2.{ 0}%
876 \def\XINT_trunc_b {\expandafter\XINT_trunc_B\the\numexpr}%
877 \def\XINT_trunc_sp_b {\expandafter\XINT_trunc_sp_B\the\numexpr}%
878 \def\XINT_trunc_B #1%
879 {%
880   \xint_UDsignfork
881     #1\XINT_trunc_C
882     -\XINT_trunc_D
883   \krof #1%
884}%
885 \def\XINT_trunc_sp_B #1%
886 {%
887   \xint_UDsignfork
888     #1\XINT_trunc_sp_C
889     -\XINT_trunc_sp_D
890   \krof #1%
891}%

```

```

892 \def\XINT_trunc_C -#1.#2#3%
893 {%
894     \expandafter\XINT_trunc_CE
895     \romannumeral0\XINT_dsx_addzeros{#1}#3;.{#2}%
896 }%
897 \def\XINT_trunc_CE #1.#2{\XINT_trunc_E #2.{#1}}%

898 \def\XINT_trunc_sp_C -#1.#2#3{\XINT_trunc_sp_Ca #2.#1}%
899 \def\XINT_trunc_sp_Ca #1%
900 {%
901     \xint_UDsignfork
902     #1{\XINT_trunc_sp_Cb -}%
903     -{\XINT_trunc_sp_Cb \space#1}%
904     \krof
905 }%
906 \def\XINT_trunc_sp_Cb #1#2.#3.%
907 {%
908     \expandafter\XINT_trunc_sp_Cc
909     \romannumeral0\expandafter\XINT_split_fromright_a
910     \the\numexpr#3-\numexpr\XINT_length_loop
911     #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
912     \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
913     \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
914     .#2\xint_bye2345678\xint_bye..#1%
915 }%
916 \def\XINT_trunc_sp_Cc #1%
917 {%
918     \if.#1\xint_dothis{\XINT_trunc_sp_Cd 0.}\fi
919     \xint_orthat {\XINT_trunc_sp_Cd #1}%
920 }%
921 \def\XINT_trunc_sp_Cd #1.#2.#3%
922 {%
923     \XINT_trunc_sp_F #3#1.%
924 }%
925 \def\XINT_trunc_D #1.#2%
926 {%
927     \expandafter\XINT_trunc_E
928     \romannumeral0\XINT_dsx_addzeros {#1}#2;.%
929 }%
930 \def\XINT_trunc_sp_D #1.#2#3%
931 {%
932     \expandafter\XINT_trunc_sp_E
933     \romannumeral0\XINT_dsx_addzeros {#1}#2;.%
934 }%
935 \def\XINT_trunc_E #1%
936 {%
937     \xint_UDsignfork
938     #1{\XINT_trunc_F -}%
939     -{\XINT_trunc_F \space#1}%
940     \krof
941 }%
942 \def\XINT_trunc_sp_E #1%

```



```

943 {%
944   \xint_UDsignfork
945     #1{\XINT_trunc_sp_F -}%
946     -{\XINT_trunc_sp_F\space#1}%
947   \krof
948 }%
949 \def\XINT_trunc_F #1#2.#3#4%
950   {\expandafter#4\romannumeral`&&\expandafter\xint_firstoftwo
951     \romannumeral0\XINT_div_prepare {#3}{#2}.#1}%
952 \def\XINT_trunc_sp_F #1#2.#3{#3#2.#1}%
953 \def\XINT_itrunc_G #1#2.#3#4.%
954 {%
955   \if#10\xint_dothis{ 0}\fi
956   \xint_orthat{#3#1}#2%
957 }%
958 \def\XINT_trunc_G #1.#2#3#4.%
959 {%
960   \xint_gob_til_minus#3\XINT_trunc_Hc-%
961   \expandafter\XINT_trunc_H
962   \the\numexpr\romannumeral0\xintlength {#1}-#3#4.#3#4.{#1}#2%
963 }%
964 \def\XINT_trunc_Hc-\expandafter\XINT_trunc_H
965   \the\numexpr\romannumeral0\xintlength #1.-#2.#3#4{#4#3e#2}%
966 \def\XINT_trunc_H #1.#2.%
967 {%
968   \ifnum #1 > \xint_c_ \xint_dothis{\XINT_trunc_Ha {#2}}\fi
969   \xint_orthat {\XINT_trunc_Hb {-#1}}% -0,--1,--2, ...
970 }%
971 \def\XINT_trunc_Ha%
972 {%
973   \expandafter\XINT_trunc_Haa\romannumeral0\xintdecsplit
974 }%
975 \def\XINT_trunc_Haa #1#2#3{#3#1.#2}%
976 \def\XINT_trunc_Hb #1#2#3%
977 {%
978   \expandafter #3\expandafter0\expandafter.%
979   \romannumeral\xintreplicate{#1}0#2%
980 }%

```

8.32 \xintTTrunc

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

```

981 \def\xintTTrunc {\romannumeral0\xintttrunc }%
982 \def\xintttrunc {\xintitrunc\xint_c_}%

```

8.33 \xintNum

```

983 \let\xintnum \xintttrunc

```

8.34 \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`.

At 1.4a, first argument can be negative. This is handled at `\XINT_trunc_G`.

```

984 \def\xintRound {\romannumeral0\xintround}%
985 \def\xintiRound {\romannumeral0\xintiround}%
986 \def\xintround #1{\expandafter\XINT_round\the\numexpr #1.\XINT_round_A}%
987 \def\xintiround #1{\expandafter\XINT_round\the\numexpr #1.\XINT_iround_A}%
988 \def\XINT_round #1.{\expandafter\XINT_round_aa\the\numexpr #1+\xint_c_i.#1.}%
989 \def\XINT_round_aa #1.#2.#3#4%
990 {%
991   \expandafter\XINT_round_a\romannumeral0\XINT_infrac{#4}#1.#3#2.%
992 }%
993 \def\XINT_round_a #1#2#3#4.%
994 {%
995   \if0\XINT_Sgn#2\xint:\xint_dothis\XINT_trunc_zero\fi
996   \if1\XINT_is_One#3XY\xint_dothis\XINT_trunc_sp_b\fi
997   \xint_orthat\XINT_trunc_b #1+#4.{#2}{#3}%
998 }%
999 \def\XINT_round_A{\expandafter\XINT_trunc_G\romannumeral0\XINT_round_B}%
1000 \def\XINT_iround_A{\expandafter\XINT_itrunc_G\romannumeral0\XINT_round_B}%
1001 \def\XINT_round_B #1.%
1002   {\XINT_dsrr #1\xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax.}%

```

8.35 `\xintXTrunc`

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

- no more use of `\xintiloop` from `xinttools.sty` (replaced by `\xintreplicate...` from `xintkernel.sty`),
- no more use in `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.

```

1003 \def\xintXTrunc #1%#2%
1004 {%
1005   \expandafter\XINT_xtrunc_a
1006   \the\numexpr #1\expandafter.\romannumeral0\xintraw
1007 }%
1008 \def\XINT_xtrunc_a #1.% ?? faire autre chose
1009 {%
1010   \expandafter\XINT_xtrunc_b\the\numexpr\ifnum#1<\xint_c_i \xint_c_i-\fi #1.%
1011 }%

1012 \def\XINT_xtrunc_b #1.#2{\XINT_xtrunc_c #2{#1}}%

1013 \def\XINT_xtrunc_c #1%
1014 {%
1015   \xint_UDzerominusfork

```

```

1016      #1-\XINT_xtrunc_zero
1017      0#1{-\XINT_xtrunc_d {}}%
1018      0-{\XINT_xtrunc_d #1}%
1019      \krof
1020 }%[
1021 \def\XINT_xtrunc_zero #1#2]{0.\romannumeral\xintreplicate{#1}0}%

1022 \def\XINT_xtrunc_d #1#2#3/#4[#5]%
1023 {%
1024     \XINT_xtrunc_prepare_a#4\R\R\R\R\R\R\R {10}0000001\W
1025     !{#4};{#5}{#2}{#1#3}%
1026 }%
1027 \def\XINT_xtrunc_prepare_a #1#2#3#4#5#6#7#8#9%
1028 {%
1029     \xint_gob_til_R #9\XINT_xtrunc_prepare_small\R
1030     \XINT_xtrunc_prepare_b #9%
1031 }%
1032 \def\XINT_xtrunc_prepare_small\R #1!#2;%
1033 {%
1034     \ifcase #2
1035     \or\expandafter\XINT_xtrunc_BisOne
1036     \or\expandafter\XINT_xtrunc_BisTwo
1037     \or
1038     \or\expandafter\XINT_xtrunc_BisFour
1039     \or\expandafter\XINT_xtrunc_BisFive
1040     \or
1041     \or
1042     \or\expandafter\XINT_xtrunc_BisEight
1043     \fi\XINT_xtrunc_BisSmall {#2}%
1044 }%

1045 \def\XINT_xtrunc_BisOne\XINT_xtrunc_BisSmall #1#2#3#4%
1046     {\XINT_xtrunc_sp_e {#2}{#4}{#3}}%
1047 \def\XINT_xtrunc_BisTwo\XINT_xtrunc_BisSmall #1#2#3#4%
1048 {%
1049     \expandafter\XINT_xtrunc_sp_e\expandafter
1050     {\the\numexpr #2-\xint_c_i\expandafter}\expandafter
1051     {\romannumeral0\xintiimul 5{#4}}{#3}%
1052 }%
1053 \def\XINT_xtrunc_BisFour\XINT_xtrunc_BisSmall #1#2#3#4%
1054 {%
1055     \expandafter\XINT_xtrunc_sp_e\expandafter
1056     {\the\numexpr #2-\xint_c_ii\expandafter}\expandafter
1057     {\romannumeral0\xintiimul {25}{#4}}{#3}%
1058 }%
1059 \def\XINT_xtrunc_BisFive\XINT_xtrunc_BisSmall #1#2#3#4%
1060 {%
1061     \expandafter\XINT_xtrunc_sp_e\expandafter
1062     {\the\numexpr #2-\xint_c_i\expandafter}\expandafter
1063     {\romannumeral0\xintdouble {#4}}{#3}%
1064 }%
1065 \def\XINT_xtrunc_BisEight\XINT_xtrunc_BisSmall #1#2#3#4%

```

```

1066 {%
1067   \expandafter\XINT_xtrunc_sp_e\expandafter
1068   {\the\numexpr #2-\xint_c_iii\expandafter}\expandafter
1069   {\romannumeral0\xintiimul {125}{#4}}{#3}%
1070 }%
1071 \def\XINT_xtrunc_BisSmall #1%
1072 {%
1073   \expandafter\XINT_xtrunc_e\expandafter
1074   {\expandafter\XINT_xtrunc_small_a
1075   \the\numexpr #1/\xint_c_ii\expandafter
1076   .\the\numexpr \xint_c_x^viii+#1!}%
1077 }%

1078 \def\XINT_xtrunc_small_a #1.#2!#3%
1079 {%
1080   \expandafter\XINT_div_small_b\the\numexpr #1\expandafter
1081   \xint:\the\numexpr #2\expandafter!%
1082   \romannumeral0\XINT_div_small_ba #3\R\R\R\R\R\R\R\R{10}0000001\W
1083   #3\XINT_sepbyviii_Z_end 2345678\relax
1084 }%

1085 \def\XINT_xtrunc_prepare_b
1086   {\expandafter\XINT_xtrunc_prepare_c\romannumeral0\XINT_zeroes_forviii }%
1087 \def\XINT_xtrunc_prepare_c #1!%
1088 {%
1089   \XINT_xtrunc_prepare_d #1.00000000!{#1}%
1090 }%
1091 \def\XINT_xtrunc_prepare_d #1#2#3#4#5#6#7#8#9%
1092 {%
1093   \expandafter\XINT_xtrunc_prepare_e
1094   \xint_gob_til_dot #1#2#3#4#5#6#7#8#9!%
1095 }%
1096 \def\XINT_xtrunc_prepare_e #1!#2!#3#4%
1097 {%
1098   \XINT_xtrunc_prepare_f #4#3\X {#1}{#3}%
1099 }%
1100 \def\XINT_xtrunc_prepare_f #1#2#3#4#5#6#7#8#9\X
1101 {%
1102   \expandafter\XINT_xtrunc_prepare_g\expandafter
1103   \XINT_div_prepare_g
1104   \the\numexpr #1#2#3#4#5#6#7#8+\xint_c_i\expandafter
1105   \xint:\the\numexpr (#1#2#3#4#5#6#7#8+\xint_c_i)/\xint_c_ii\expandafter
1106   \xint:\the\numexpr #1#2#3#4#5#6#7#8\expandafter
1107   \xint:\romannumeral0\XINT_sepandrev_andcount
1108   #1#2#3#4#5#6#7#8#9\XINT_rsepbyviii_end_A 2345678%
1109   \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
1110   \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
1111   \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_W
1112   \X
1113 }%

1114 \def\XINT_xtrunc_prepare_g #1;{\XINT_xtrunc_e {#1}}%

```

```

1115 \def\XINT_xtrunc_e #1#2%
1116 {%
1117     \ifnum #2<\xint_c_
1118         \expandafter\XINT_xtrunc_I
1119     \else
1120         \expandafter\XINT_xtrunc_II
1121     \fi #2\xint:{#1}%
1122 }%

1123 \def\XINT_xtrunc_I -#1\xint:#2#3#4%
1124 {%
1125     \expandafter\XINT_xtrunc_I_a\romannumeral0#2{#4}{#2}{#1}{#3}%
1126 }%

1127 \def\XINT_xtrunc_I_a #1#2#3#4#5%
1128 {%
1129     \expandafter\XINT_xtrunc_I_b\the\numexpr #4-#5\xint:#4\xint:{#5}{#2}{#3}{#1}%
1130 }%

1131 \def\XINT_xtrunc_I_b #1%
1132 {%
1133     \xint_UDsignfork
1134     #1\XINT_xtrunc_IA_c
1135     -\XINT_xtrunc_IB_c
1136     \krof #1%
1137 }%

1138 \def\XINT_xtrunc_IA_c -#1\xint:#2\xint:#3#4#5#6%
1139 {%
1140     \expandafter\XINT_xtrunc_IA_d
1141     \the\numexpr#2-\xintlength{#6}\xint:{#6}%
1142     \expandafter\XINT_xtrunc_IA_xd
1143     \the\numexpr (#1+\xint_c_ii^v)/\xint_c_ii^vi-\xint_c_i\xint:#1\xint:{#5}{#4}%
1144 }%

1145 \def\XINT_xtrunc_IA_d #1%
1146 {%
1147     \xint_UDsignfork
1148     #1\XINT_xtrunc_IAA_e
1149     -\XINT_xtrunc_IAB_e
1150     \krof #1%
1151 }%

1152 \def\XINT_xtrunc_IAA_e -#1\xint:#2%
1153 {%
1154     \romannumeral0\XINT_split_fromleft
1155     #1.#2\xint_gobble_i\xint_bye2345678\xint_bye..%
1156 }%

```

```

1157 \def\XINT_xtrunc_IAB_e #1\xint:#2%
1158 {%
1159     0.\romannumeral\XINT_rep#1\endcsname0#2%
1160 }%

1161 \def\XINT_xtrunc_IA_xd #1\xint:#2\xint:%
1162 {%
1163     \expandafter\XINT_xtrunc_IA_xe\the\numexpr #2-\xint_c_ii^vi*#1\xint:#1\xint:%
1164 }%

1165 \def\XINT_xtrunc_IA_xe #1\xint:#2\xint:#3#4%
1166 {%
1167     \XINT_xtrunc_loop {#2}{#4}{#3}{#1}%
1168 }%

1169 \def\XINT_xtrunc_IB_c #1\xint:#2\xint:#3#4#5#6%
1170 {%
1171     \expandafter\XINT_xtrunc_IB_d
1172     \romannumeral0\XINT_split_xfork #1.#6\xint_bye2345678\xint_bye..{#3}%
1173 }%

1174 \def\XINT_xtrunc_IB_d #1.#2.#3%
1175 {%
1176     \expandafter\XINT_xtrunc_IA_d\the\numexpr#3-\xintLength {#1}\xint:{#1}%
1177 }%

1178 \def\XINT_xtrunc_II #1\xint:%
1179 {%
1180     \expandafter\XINT_xtrunc_II_a\romannumeral\xintreplicate{#1}0\xint:%
1181 }%
1182 \def\XINT_xtrunc_II_a #1\xint:#2#3#4%
1183 {%
1184     \expandafter\XINT_xtrunc_II_b
1185     \the\numexpr (#3+\xint_c_ii^v)/\xint_c_ii^vi-\xint_c_i\expandafter\xint:%
1186     \the\numexpr #3\expandafter\xint:\romannumeral0#2{#4#1}{#2}%
1187 }%
1188 \def\XINT_xtrunc_II_b #1\xint:#2\xint:%
1189 {%
1190     \expandafter\XINT_xtrunc_II_c\the\numexpr #2-\xint_c_ii^vi*#1\xint:#1\xint:%
1191 }%

1192 \def\XINT_xtrunc_II_c #1\xint:#2\xint:#3#4#5%
1193 {%
1194     #3.\XINT_xtrunc_loop {#2}{#4}{#5}{#1}%
1195 }%

1196 \def\XINT_xtrunc_loop #1%
1197 {%
1198     \ifnum #1=\xint_c_ \expandafter\XINT_xtrunc_transition\fi

```

[illegible]

```

1246 \def\XINT_xtrunc_sp_IA_b -#1\xint:#2\xint:#3#4%
1247 {%
1248   \expandafter\XINT_xtrunc_sp_IA_c
1249   \the\numexpr#2-\xintlength{#4}\xint:{#4}\romannumeral\XINT_rep#1\endcsname0%
1250 }%

1251 \def\XINT_xtrunc_sp_IA_c #1%
1252 {%
1253   \xint_UDsignfork
1254   #1\XINT_xtrunc_sp_IAA
1255   -\XINT_xtrunc_sp_IAB
1256   \krof #1%
1257 }%

1258 \def\XINT_xtrunc_sp_IAA -#1\xint:#2%
1259 {%
1260   \romannumeral0\XINT_split_fromleft
1261   #1.#2\xint_gobble_i\xint_bye2345678\xint_bye..%
1262 }%

1263 \def\XINT_xtrunc_sp_IAB #1\xint:#2%
1264 {%
1265   0.\romannumeral\XINT_rep#1\endcsname0#2%
1266 }%

1267 \def\XINT_xtrunc_sp_IB_b #1\xint:#2\xint:#3#4%
1268 {%
1269   \expandafter\XINT_xtrunc_sp_IB_c
1270   \romannumeral0\XINT_split_xfork #1.#4\xint_bye2345678\xint_bye..{#3}%
1271 }%

1272 \def\XINT_xtrunc_sp_IB_c #1.#2.#3%
1273 {%
1274   \expandafter\XINT_xtrunc_sp_IA_c\the\numexpr#3-\xintlength {#1}\xint:{#1}%
1275 }%

1276 \def\XINT_xtrunc_sp_II #1\xint:#2#3%
1277 {%
1278   #2\romannumeral\XINT_rep#1\endcsname0.\romannumeral\XINT_rep#3\endcsname0%
1279 }%

```

8.36 \xintAdd

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

```

1280 \def\xintAdd {\romannumeral0\xintadd }%
1281 \def\xintadd #1{\expandafter\XINT_fadd\romannumeral0\xintra #1}%
1282 \def\XINT_fadd #1{\xint_gob_til_zero #1\XINT_fadd_Azero 0\XINT_fadd_a #1}%
1283 \def\XINT_fadd_Azero #1{\xintra #1}%
1284 \def\XINT_fadd_a #1/#2[#3]#4%

```



```

1285   {\expandafter\XINT_fadd_b\romannumeral0\xintra {#4}{#3}{#1}{#2}}%
1286 \def\XINT_fadd_b #1{\xint_gob_til_zero #1\XINT_fadd_Bzero 0\XINT_fadd_c #1}%
1287 \def\XINT_fadd_Bzero #1]#2#3#4{ #3/#4[#2]}%
1288 \def\XINT_fadd_c #1/#2[#3]#4%
1289 {%
1290   \expandafter\XINT_fadd_Aa\the\numexpr #4-#3.{#3}{#4}{#1}{#2}%
1291 }%
1292 \def\XINT_fadd_Aa #1%
1293 {%
1294   \xint_UDzerominusfork
1295     #1-\XINT_fadd_B
1296     0#1\XINT_fadd_Bb
1297     0-\XINT_fadd_Ba
1298   \krof #1%
1299 }%
1300 \def\XINT_fadd_B #1.#2#3#4#5#6#7{\XINT_fadd_C {#4}{#5}{#7}{#6}[#3]}%
1301 \def\XINT_fadd_Ba #1.#2#3#4#5#6#7%
1302 {%
1303   \expandafter\XINT_fadd_C\expandafter
1304     {\romannumeral0\XINT_dsx_addzeros {#1}#6;}%
1305     {#7}{#5}{#4}[#2]}%
1306 }%
1307 \def\XINT_fadd_Bb -#1.#2#3#4#5#6#7%
1308 {%
1309   \expandafter\XINT_fadd_C\expandafter
1310     {\romannumeral0\XINT_dsx_addzeros {#1}#4;}%
1311     {#5}{#7}{#6}[#3]}%
1312 }%
1313 \def\XINT_fadd_iszero #1[#2]{ 0/1[0]}% ou [#2] origine?
1314 \def\XINT_fadd_C #1#2#3%
1315 {%
1316   \expandafter\XINT_fadd_D_b
1317   \romannumeral0\XINT_div_prepare{#2}{#3}{#2}{#2}{#3}{#1}%
1318 }%

```

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.

```

1319 \def\XINT_fadd_D_a #1#2%
1320 {%
1321   \expandafter\XINT_fadd_D_b
1322   \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
1323 }%
1324 \def\XINT_fadd_D_b #1#2{\XINT_fadd_D_c #2\Z}%
1325 \def\XINT_fadd_D_c #1#2\Z
1326 {%
1327   \xint_gob_til_zero #1\XINT_fadd_D_exit0\XINT_fadd_D_a {#1#2}%
1328 }%
1329 \def\XINT_fadd_D_exit0\XINT_fadd_D_a #1#2#3%
1330 {%
1331   \expandafter\XINT_fadd_E
1332   \romannumeral0\xintiiquo {#3}{#2}.{#2}%
1333 }%

```

```

1334 \def\XINT_fadd_E #1.#2#3%
1335 {%
1336     \expandafter\XINT_fadd_F
1337     \romannumeral0\xintiimul{#1}{#3}.\{\xintiiQuo{#3}{#2}\}{#1}%
1338 }%
1339 \def\XINT_fadd_F #1.#2#3#4#5%
1340 {%
1341     \expandafter\XINT_fadd_G
1342     \romannumeral0\xintiiadd{\xintiiMul{#2}{#4}}{\xintiiMul{#3}{#5}}/#1%
1343 }%
1344 \def\XINT_fadd_G #1{%
1345 \def\XINT_fadd_G ##1{\if0##1\expandafter\XINT_fadd_iszero\fi#1#1}%
1346 }\XINT_fadd_G{ }%

```

8.37 \xintSub

Since 1.3 will use least common multiple of denominators.

```

1347 \def\xintSub {\romannumeral0\xintsub }%
1348 \def\xintsub #1{\expandafter\XINT_fsub\romannumeral0\xintra{#1}}%
1349 \def\XINT_fsub #1{\xint_gob_til_zero #1\XINT_fsub_Azero 0\XINT_fsub_a #1}%
1350 \def\XINT_fsub_Azero #1{\xintopp }%
1351 \def\XINT_fsub_a #1/#2[#3]#4%
1352     {\expandafter\XINT_fsub_b\romannumeral0\xintra{#4}{#3}{#1}{#2}}%
1353 \def\XINT_fsub_b #1{\xint_UDzerominusfork
1354     #1-\XINT_fadd_Bzero
1355     0#1\XINT_fadd_c
1356     0-\XINT_fadd_c -#1}%
1357 \krof }%

```

8.38 \xintSum

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

Empty items in the input are not accepted by this macro, but the input may be empty.
 Refactored slightly at 1.4. \XINT_Sum used in xintexpr code.

```

1358 \def\xintSum {\romannumeral0\xintsum }%
1359 \def\xintsum #1{\expandafter\XINT_sum\romannumeral`&&@#1^}%
1360 \def\XINT_Sum{\romannumeral0\XINT_sum}%
1361 \def\XINT_sum#1%
1362 {%
1363     \xint_gob_til_ ^ #1\XINT_sum_empty ^%
1364     \expandafter\XINT_sum_loop\romannumeral0\xintra{#1}\xint:
1365 }%
1366 \def\XINT_sum_empty ^#1\xint:{ 0/1[0]}%
1367 \def\XINT_sum_loop #1\xint:#2%
1368 {%
1369     \xint_gob_til_ ^ #2\XINT_sum_end ^%
1370     \expandafter\XINT_sum_loop
1371     \romannumeral0\xintadd{#1}{\romannumeral0\xintra{#2}}\xint:
1372 }%
1373 \def\XINT_sum_end ^#1\xintadd #2#3\xint:{ #2}%

```

8.39 \xintMul

```

1374 \def\xintMul {\romannumeral0\xintmul }%
1375 \def\xintmul #1{\expandafter\XINT_fmulo\romannumeral0\xintra {#1}.}%
1376 \def\XINT_fmulo #1{\xint_gob_til_zero #1\XINT_fmulo_zero 0\XINT_fmulo_a #1}%
1377 \def\XINT_fmulo_a #1[#2].#3%
1378   {\expandafter\XINT_fmulo_b\romannumeral0\xintra {#3}#1[#2.]}%
1379 \def\XINT_fmulo_b #1{\xint_gob_til_zero #1\XINT_fmulo_zero 0\XINT_fmulo_c #1}%
1380 \def\XINT_fmulo_c #1/#2[#3]#4/#5[#6.]%
1381 {%
1382   \expandafter\XINT_fmulo_d
1383   \expandafter{\the\numexpr #3+#6\expandafter}%
1384   \expandafter{\romannumeral0\xintiimul {#5}{#2}}%
1385   {\romannumeral0\xintiimul {#4}{#1}}%
1386 }%
1387 \def\XINT_fmulo_d #1#2#3%
1388 {%
1389   \expandafter \XINT_fmulo_e \expandafter{#3}{#1}{#2}%
1390 }%
1391 \def\XINT_fmulo_e #1#2{\XINT_outfrac {#2}{#1}}%
1392 \def\XINT_fmulo_zero #1.#2{ 0/1[0]}%

```

8.40 \xintSqr

1.1 modifs comme xintMul.

```

1393 \def\xintSqr {\romannumeral0\xintsqr }%
1394 \def\xintsqr #1{\expandafter\XINT_fsqr\romannumeral0\xintra {#1}}%
1395 \def\XINT_fsqr #1{\xint_gob_til_zero #1\XINT_fsqr_zero 0\XINT_fsqr_a #1}%
1396 \def\XINT_fsqr_a #1/#2[#3]%
1397 {%
1398   \expandafter\XINT_fsqr_b
1399   \expandafter{\the\numexpr #3+#3\expandafter}%
1400   \expandafter{\romannumeral0\xintiisqr {#2}}%
1401   {\romannumeral0\xintiisqr {#1}}%
1402 }%
1403 \def\XINT_fsqr_b #1#2#3{\expandafter \XINT_fmulo_e \expandafter{#3}{#1}{#2}}%
1404 \def\XINT_fsqr_zero #1{ 0/1[0]}%

```

8.41 \xintPow

1.2f: to be coherent with the "i" convention \xintiPow should parse also its exponent via \xintNum when xintfrac.sty is loaded. This was not the case so far. Cependant le problème est que le fait d'appliquer \xintNum rend impossible certains inputs qui auraient pu être gérés par \numexpr. Le \numexpr externe est ici pour intercepter trop grand input.

```

1405 \def\xintipow #1#2%
1406 {%
1407   \expandafter\xint_pow\the\numexpr \xintNum{#2}\expandafter
1408   .\romannumeral0\xintnum{#1}\xint:
1409 }%
1410 \def\xintPow {\romannumeral0\xintpow }%
1411 \def\xintpow #1%
1412 {%

```

```

1413 \expandafter\XINT_fpow\expandafter {\romannumeral0\XINT_infrac {#1}}%
1414 }%
1415 \def\XINT_fpow #1#2%
1416 {%
1417 \expandafter\XINT_fpow_fork\the\numexpr \xintNum{#2}\relax\Z #1%
1418 }%
1419 \def\XINT_fpow_fork #1#2\Z
1420 {%
1421 \xint_UDzerominusfork
1422 #1-\XINT_fpow_zero
1423 0#1\XINT_fpow_neg
1424 0-\XINT_fpow_pos #1}%
1425 \krof
1426 {#2}%
1427 }%
1428 \def\XINT_fpow_zero #1#2#3#4{ 1/1[0]}%
1429 \def\XINT_fpow_pos #1#2#3#4#5%
1430 {%
1431 \expandafter\XINT_fpow_pos_A\expandafter
1432 {\the\numexpr #1#2*#3\expandafter}\expandafter
1433 {\romannumeral0\xintiipow {#5}{#1#2}}%
1434 {\romannumeral0\xintiipow {#4}{#1#2}}%
1435 }%
1436 \def\XINT_fpow_neg #1#2#3#4%
1437 {%
1438 \expandafter\XINT_fpow_pos_A\expandafter
1439 {\the\numexpr -#1*#2\expandafter}\expandafter
1440 {\romannumeral0\xintiipow {#3}{#1}}%
1441 {\romannumeral0\xintiipow {#4}{#1}}%
1442 }%
1443 \def\XINT_fpow_pos_A #1#2#3%
1444 {%
1445 \expandafter\XINT_fpow_pos_B\expandafter {#3}{#1}{#2}%
1446 }%
1447 \def\XINT_fpow_pos_B #1#2{\XINT_outfrac {#2}{#1}}%

```

8.42 \xintFac

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

```

1448 \def\xintFac {\romannumeral0\xintfac}%
1449 \def\xintfac #1{\expandafter\XINT_fac_fork\the\numexpr\xintNum{#1}.[0]}%

```

8.43 \xintBinomial

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

```

1450 \def\xintBinomial {\romannumeral0\xintbinomial}%
1451 \def\xintbinomial #1#2%
1452 {%
1453 \expandafter\XINT_binom_pre

```

```

1454 \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.[0]%
1455 }%

```

8.44 \xintPFactorial

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

```

1456 \def\xintipfactorial #1#2%
1457 {%
1458 \expandafter\XINT_pfac_fork
1459 \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.%
1460 }%
1461 \def\xintPFactorial {\romannumeral0\xintpfactorial}%
1462 \def\xintpfactorial #1#2%
1463 {%
1464 \expandafter\XINT_pfac_fork
1465 \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.[0]%
1466 }%

```

8.45 \xintPrd

Refactored at 1.4. After some hesitation the routine still does not try to detect on the fly a zero item, to abort the loop. Indeed this would add some overhead generally (as we need normalizing the item before checking if it vanishes hence we must then grab things once more).

```

1467 \def\xintPrd {\romannumeral0\xintprd}%
1468 \def\xintprd #1{\expandafter\XINT_prd\romannumeral`&&@#1^}%
1469 \def\XINT_Prd{\romannumeral0\XINT_prd}%
1470 \def\XINT_prd#1%
1471 {%
1472 \xint_gob_til_ ^ #1\XINT_prd_empty ^%
1473 \expandafter\XINT_prd_loop\romannumeral0\xintra{#1}\xint:
1474 }%
1475 \def\XINT_prd_empty ^#1\xint:{ 1/1[0]}%
1476 \def\XINT_prd_loop #1\xint:#2%
1477 {%
1478 \xint_gob_til_ ^ #2\XINT_prd_end ^%
1479 \expandafter\XINT_prd_loop
1480 \romannumeral0\xintmul{#1}{\romannumeral0\xintra{#2}}\xint:
1481 }%
1482 \def\XINT_prd_end ^#1\xintmul #2#3\xint:{ #2}%

```

8.46 \xintDiv

```

1483 \def\xintDiv {\romannumeral0\xintdiv}%
1484 \def\xintdiv #1%
1485 {%
1486 \expandafter\XINT_fdiv\expandafter {\romannumeral0\XINT_infrac {#1}}%
1487 }%
1488 \def\XINT_fdiv #1#2%
1489 {\expandafter\XINT_fdiv_A\romannumeral0\XINT_infrac {#2}#1}%
1490 \def\XINT_fdiv_A #1#2#3#4#5#6%
1491 {%

```

```

1492 \expandafter\XINT_fdiv_B
1493 \expandafter{\the\numexpr #4-#1\expandafter}%
1494 \expandafter{\romannumeral0\xintiimul {#2}{#6}}%
1495 {\romannumeral0\xintiimul {#3}{#5}}%
1496 }%
1497 \def\XINT_fdiv_B #1#2#3%
1498 {%
1499 \expandafter\XINT_fdiv_C
1500 \expandafter{#3}{#1}{#2}%
1501 }%
1502 \def\XINT_fdiv_C #1#2{\XINT_outfrac {#2}{#1}}%

```

8.47 \xintDivFloor

1.1. Changed at 1.2p to not append /1[0] ending but rather output a big integer in strict format, like \xintDivTrunc and \xintDivRound.

```

1503 \def\xintDivFloor {\romannumeral0\xintdivfloor }%
1504 \def\xintdivfloor #1#2{\xintifloor{\xintDiv {#1}{#2}}}%

```

8.48 \xintDivTrunc

1.1. \xintttrunc rather than \xintitrunc0 in 1.1a

```

1505 \def\xintDivTrunc {\romannumeral0\xintdivtrunc }%
1506 \def\xintdivtrunc #1#2{\xintttrunc {\xintDiv {#1}{#2}}}%

```

8.49 \xintDivRound

1.1

```

1507 \def\xintDivRound {\romannumeral0\xintdivround }%
1508 \def\xintdivround #1#2{\xintiround 0{\xintDiv {#1}{#2}}}%

```

8.50 \xintModTrunc

1.1. \xintModTrunc {q1}{q2} computes $q1 - q2 * t(q1/q2)$ with $t(q1/q2)$ equal to the truncated division of two fractions $q1$ and $q2$.

Its former name, prior to 1.2p, was \xintMod.

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

```

1509 \def\xintModTrunc {\romannumeral0\xintmodtrunc }%
1510 \def\xintmodtrunc #1{\expandafter\XINT_modtrunc_a\romannumeral0\xintra{#1}.}%
1511 \def\XINT_modtrunc_a #1#2.#3%
1512 {\expandafter\XINT_modtrunc_b\expandafter #1\romannumeral0\xintra{#3}#2.}%
1513 \def\XINT_modtrunc_b #1#2% #1 de A, #2 de B.
1514 {%
1515 \if0#2\xint_dothis{\XINT_modtrunc_divbyzero #1#2}\fi
1516 \if0#1\xint_dothis\XINT_modtrunc_aiszero\fi
1517 \if-#2\xint_dothis{\XINT_modtrunc_bneg #1}\fi
1518 \xint_orthat{\XINT_modtrunc_bpos #1#2}%
1519 }%
1520 \def\XINT_modtrunc_divbyzero #1#2[#3]#4.%

```

```

1521 {%
1522   \XINT_signalcondition{DivisionByZero}{Division by #2[#3] of #1#4}{0/1[0]}%
1523 }%
1524 \def\XINT_modtrunc_aiszero #1.{ 0/1[0]}%
1525 \def\XINT_modtrunc_bneg #1%
1526 {%
1527   \xint_UDsignfork
1528     #1{\xintiopp\XINT_modtrunc_pos {}}%
1529     -{\XINT_modtrunc_pos #1}%
1530   \krof
1531 }%
1532 \def\XINT_modtrunc_bpos #1%
1533 {%
1534   \xint_UDsignfork
1535     #1{\xintiopp\XINT_modtrunc_pos {}}%
1536     -{\XINT_modtrunc_pos #1}%
1537   \krof
1538 }%

```

Attention. This crucially uses that xint's `\xintiiE{x}{e}` is defined to return x unchanged if e is negative (and x extended by e zeroes if $e \geq 0$).

```

1539 \def\XINT_modtrunc_pos #1#2/#3[#4]#5/#6[#7].%
1540 {%
1541   \expandafter\XINT_modtrunc_pos_a
1542   \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.%
1543   \romannumeral0\expandafter\XINT_mod_D_b
1544   \romannumeral0\XINT_div_prepare{#3}{#6}{#3}{#3}{#6}%
1545   {#1#5}{#7-#4}{#2}{#4-#7}%
1546 }%
1547 \def\XINT_modtrunc_pos_a #1.#2#3#4{\xintiirem {#3}{#4}/#2[#1]}%

```

8.51 \xintDivMod

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

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

```

1548 \def\xintDivMod {\romannumeral0\xintdivmod }%
1549 \def\xintdivmod #1{\expandafter\XINT_divmod_a\romannumeral0\xintra{#1}.}%
1550 \def\XINT_divmod_a #1#2.#3%
1551   {\expandafter\XINT_divmod_b\expandafter #1\romannumeral0\xintra{#3}#2.}%
1552 \def\XINT_divmod_b #1#2% #1 de A, #2 de B.
1553 {%
1554   \if0#2\xint_dothis{\XINT_divmod_divbyzero #1#2}\fi
1555   \if0#1\xint_dothis\XINT_divmod_aiszero\fi
1556   \if-#2\xint_dothis{\XINT_divmod_bneg #1}\fi
1557   \xint_orthat{\XINT_divmod_bpos #1#2}%
1558 }%
1559 \def\XINT_divmod_divbyzero #1#2[#3]#4.%
1560 {%
1561   \XINT_signalcondition{DivisionByZero}{Division by #2[#3] of #1#4}{}%

```

```

1562     {{0}{0/1[0]}}% à revoir...
1563 }%
1564 \def\XINT_divmod_aiszero #1.{{0}{0/1[0]}}%
1565 \def\XINT_divmod_bneg #1f // -g = (-f) // g, f % -g = -((-f) % g)
1566 {%
1567     \expandafter\XINT_divmod_bneg_finish
1568     \romannumeral0\xint_UDsignfork
1569     #1{\XINT_divmod_bpos {}}%
1570     -{\XINT_divmod_bpos {-#1}}%
1571     \krof
1572 }%
1573 \def\XINT_divmod_bneg_finish#1#2%
1574 {%
1575     \expandafter\xint_exchangetwo_keepbraces\expandafter
1576     {\romannumeral0\xintiioopp#2}{#1}%
1577 }%
1578 \def\XINT_divmod_bpos #1#2/#3[#4]#5/#6[#7].%
1579 {%
1580     \expandafter\XINT_divmod_bpos_a
1581     \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.%
1582     \romannumeral0\expandafter\XINT_mod_D_b
1583     \romannumeral0\XINT_div_prepare{#3}{#6}{#3}{#3}{#6}%
1584     {#1#5}{#7-#4}{#2}{#4-#7}%
1585 }%
1586 \def\XINT_divmod_bpos_a #1.#2#3#4%
1587 {%
1588     \expandafter\XINT_divmod_bpos_finish
1589     \romannumeral0\xintiidivision{#3}{#4}{/#2[#1]}%
1590 }%
1591 \def\XINT_divmod_bpos_finish #1#2#3{{#1}{#2#3}}%

```

8.52 \xintMod

1.2p. `\xintMod{q1}{q2}` computes $q1 - q2 \cdot \text{floor}(q1/q2)$. Attention that it relies on `\xintiiE{x}{e}` returning x if $e < 0$.

Prior to 1.2p, that macro had the meaning now attributed to `\xintModTrunc`.

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

```

1592 \def\xintMod {\romannumeral0\xintmod }%
1593 \def\xintmod #1{\expandafter\XINT_mod_a\romannumeral0\xintraw{#1}.}%
1594 \def\XINT_mod_a #1#2.#3%
1595     {\expandafter\XINT_mod_b\expandafter #1\romannumeral0\xintraw{#3}#2.}%
1596 \def\XINT_mod_b #1#2% #1 de A, #2 de B.
1597 {%
1598     \if0#2\xint_dothis{\XINT_mod_divbyzero #1#2}\fi
1599     \if0#1\xint_dothis\XINT_mod_aiszero\fi
1600     \if-#2\xint_dothis{\XINT_mod_bneg #1}\fi
1601     \xint_orthat{\XINT_mod_bpos #1#2}%
1602 }%

```

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

```

1603 \let\XINT_mod_divbyzero\XINT_modtrunc_divbyzero

```



```

1604 \let\XINT_mod_aiszero \XINT_modtrunc_aiszero
1605 \def\XINT_mod_bneg #1# f % -g = - ((-f) % g), for g > 0
1606 {%
1607     \xintiiopt\XINT_UDsignfork
1608     #1{\XINT_mod_bpos {}}%
1609     -{\XINT_mod_bpos {-#1}}%
1610     \krof
1611 }%
1612 \def\XINT_mod_bpos #1#2/#3[#4]#5/#6[#7].%
1613 {%
1614     \expandafter\XINT_mod_bpos_a
1615     \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.%
1616     \romannumeral0\expandafter\XINT_mod_D_b
1617     \romannumeral0\XINT_div_prepare{#3}{#6}{#3}{#3}{#6}%
1618     {#1#5}{#7-#4}{#2}{#4-#7}%
1619 }%
1620 \def\XINT_mod_D_a #1#2%
1621 {%
1622     \expandafter\XINT_mod_D_b
1623     \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
1624 }%
1625 \def\XINT_mod_D_b #1#2{\XINT_mod_D_c #2\Z}%
1626 \def\XINT_mod_D_c #1#2\Z
1627 {%
1628     \xint_gob_til_zero #1\XINT_mod_D_exit0\XINT_mod_D_a {#1#2}%
1629 }%
1630 \def\XINT_mod_D_exit0\XINT_mod_D_a #1#2#3%
1631 {%
1632     \expandafter\XINT_mod_E
1633     \romannumeral0\xintiipro {#3}{#2}.{#2}%
1634 }%
1635 \def\XINT_mod_E #1.#2#3%
1636 {%
1637     \expandafter\XINT_mod_F
1638     \romannumeral0\xintiimul{#1}{#3}.{\xintiipro{#3}{#2}}{#1}%
1639 }%
1640 \def\XINT_mod_F #1.#2#3#4#5#6#7%
1641 {%
1642     {#1}{\xintiipro{\xintiimul{#4}{#3}}{#5}}%
1643     {\xintiipro{\xintiimul{#6}{#2}}{#7}}%
1644 }%
1645 \def\XINT_mod_bpos_a #1.#2#3#4{\xintiirem {#3}{#4}/#2[#1]}%

```

8.53 \xintIsOne

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

```

1646 \def\xintIsOne {\romannumeral0\xintisone }%
1647 \def\xintisone #1{\expandafter\XINT_fracisone
1648     \romannumeral0\xintrawithzeros{#1}\Z }%
1649 \def\XINT_fracisone #1/#2\Z
1650 {\if0\xintiicmp {#1}{#2}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%

```

8.54 \xintGeq

```

1651 \def\xintGeq {\romannumeral0\xintgeq }%
1652 \def\xintgeq #1%
1653 {%
1654   \expandafter\XINT_fgeq\expandafter {\romannumeral0\xintabs {#1}}%
1655 }%
1656 \def\XINT_fgeq #1#2%
1657 {%
1658   \expandafter\XINT_fgeq_A \romannumeral0\xintabs {#2}#1%
1659 }%
1660 \def\XINT_fgeq_A #1%
1661 {%
1662   \xint_gob_til_zero #1\XINT_fgeq_Zii 0%
1663   \XINT_fgeq_B #1%
1664 }%
1665 \def\XINT_fgeq_Zii 0\XINT_fgeq_B #1[#2]#3[#4]{ 1}%
1666 \def\XINT_fgeq_B #1/#2[#3]#4#5/#6[#7]%
1667 {%
1668   \xint_gob_til_zero #4\XINT_fgeq_Zi 0%
1669   \expandafter\XINT_fgeq_C\expandafter
1670   {\the\numexpr #7-#3\expandafter}\expandafter
1671   {\romannumeral0\xintiimul {#4#5}{#2}}%
1672   {\romannumeral0\xintiimul {#6}{#1}}%
1673 }%
1674 \def\XINT_fgeq_Zi 0#1#2#3#4#5#6#7{ 0}%
1675 \def\XINT_fgeq_C #1#2#3%
1676 {%
1677   \expandafter\XINT_fgeq_D\expandafter
1678   {#3}{#1}{#2}%
1679 }%
1680 \def\XINT_fgeq_D #1#2#3%
1681 {%
1682   \expandafter\XINT_cntSgnFork\romannumeral`&&\expandafter\XINT_cntSgn
1683   \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\xint:
1684   { 0}{\XINT_fgeq_E #2\Z {#3}{#1}}{ 1}%
1685 }%
1686 \def\XINT_fgeq_E #1%
1687 {%
1688   \xint_UDsignfork
1689   #1\XINT_fgeq_Fd
1690   -{\XINT_fgeq_Fn #1}%
1691   \krof
1692 }%

1693 \def\XINT_fgeq_Fd #1\Z #2#3%
1694 {%
1695   \expandafter\XINT_fgeq_Fe
1696   \romannumeral0\XINT_dsx_addzeros {#1}#3;\xint:#2\xint:
1697 }%
1698 \def\XINT_fgeq_Fe #1\xint:#2#3\xint:{\XINT_geq_plusplus #2#1\xint:#3\xint:}%
1699 \def\XINT_fgeq_Fn #1\Z #2#3%
1700 {%

```

```

1701 \expandafter\XINT_fgeq_Fo
1702 \romannumeral0\XINT_dsx_addzeros {#1}#2;\xint:#3\xint:
1703 }%
1704 \def\XINT_fgeq_Fo #1#2\xint:#3\xint:{\XINT_geq_plusplus #1#3\xint:#2\xint:}%

```

8.55 \xintMax

```

1705 \def\xintMax {\romannumeral0\xintmax }%
1706 \def\xintmax #1%
1707 {%
1708 \expandafter\XINT_fmax\expandafter {\romannumeral0\xintra #1}%
1709 }%
1710 \def\XINT_fmax #1#2%
1711 {%
1712 \expandafter\XINT_fmax_A\romannumeral0\xintra #2}%1%
1713 }%
1714 \def\XINT_fmax_A #1#2/#3[#4]#5#6/#7[#8]%
1715 {%
1716 \xint_UDsignsfork
1717 #1#5\XINT_fmax_minusminus
1718 -#5\XINT_fmax_firstneg
1719 #1-\XINT_fmax_secondneg
1720 --\XINT_fmax_nonneg_a
1721 \krof
1722 #1#5{#2/#3[#4]}{#6/#7[#8]}%
1723 }%
1724 \def\XINT_fmax_minusminus --%
1725 {\expandafter-\romannumeral0\XINT_fmin_nonneg_b }%
1726 \def\XINT_fmax_firstneg #1-#2#3{ #1#2}%
1727 \def\XINT_fmax_secondneg -#1#2#3{ #1#3}%
1728 \def\XINT_fmax_nonneg_a #1#2#3#4%
1729 {%
1730 \XINT_fmax_nonneg_b {#1#3}{#2#4}%
1731 }%
1732 \def\XINT_fmax_nonneg_b #1#2%
1733 {%
1734 \if0\romannumeral0\XINT_fgeq_A #1#2%
1735 \xint_afterfi{ #1}%
1736 \else \xint_afterfi{ #2}%
1737 \fi
1738 }%

```

8.56 \xintMaxof

1.21 protects \xintMaxof against items with non terminated \the\numexpr expressions.

1.4 renders the macro compatible with an empty argument and it also defines an accessor \XINT_Maxof suitable for xintexpr usage (formerly xintexpr had its own macro handling comma separated values, but it changed internal representation at 1.4).

```

1739 \def\xintMaxof {\romannumeral0\xintmaxof }%
1740 \def\xintmaxof #1{\expandafter\XINT_maxof\romannumeral`&&@#1^}%
1741 \def\XINT_Maxof{\romannumeral0\XINT_maxof}%
1742 \def\XINT_maxof#1%
1743 {%

```

```

1744 \xint_gob_til_ ^ #1\XINT_maxof_empty ^%
1745 \expandafter\XINT_maxof_loop\romannumeral0\xintra{#1}\xint:
1746 }%
1747 \def\XINT_maxof_empty ^#1\xint:{ 0/1[0]}%
1748 \def\XINT_maxof_loop #1\xint:#2%
1749 {%
1750 \xint_gob_til_ ^ #2\XINT_maxof_e ^%
1751 \expandafter\XINT_maxof_loop
1752 \romannumeral0\xintmax{#1}{\romannumeral0\xintra{#2}}\xint:
1753 }%
1754 \def\XINT_maxof_e ^#1\xintmax #2#3\xint:{ #2}%

```

8.57 \xintMin

```

1755 \def\xintMin {\romannumeral0\xintmin }%
1756 \def\xintmin #1%
1757 {%
1758 \expandafter\XINT_fmin\expandafter {\romannumeral0\xintra {#1}}%
1759 }%
1760 \def\XINT_fmin #1#2%
1761 {%
1762 \expandafter\XINT_fmin_A\romannumeral0\xintra {#2}#1%
1763 }%
1764 \def\XINT_fmin_A #1#2/#3[#4]#5#6/#7[#8]%
1765 {%
1766 \xint_UDsignsfork
1767 #1#5\XINT_fmin_minusminus
1768 -#5\XINT_fmin_firstneg
1769 #1-\XINT_fmin_secondneg
1770 --\XINT_fmin_nonneg_a
1771 \krof
1772 #1#5{#2/#3[#4]}{#6/#7[#8]}%
1773 }%
1774 \def\XINT_fmin_minusminus --%
1775 {\expandafter-\romannumeral0\XINT_fmax_nonneg_b }%
1776 \def\XINT_fmin_firstneg #1-#2#3{ -#3}%
1777 \def\XINT_fmin_secondneg -#1#2#3{ -#2}%
1778 \def\XINT_fmin_nonneg_a #1#2#3#4%
1779 {%
1780 \XINT_fmin_nonneg_b {#1#3}{#2#4}%
1781 }%
1782 \def\XINT_fmin_nonneg_b #1#2%
1783 {%
1784 \if0\romannumeral0\XINT_fgeq_A #1#2%
1785 \xint_afterfi{ #2}%
1786 \else \xint_afterfi{ #1}%
1787 \fi
1788 }%

```

8.58 \xintMinof

- 1.21 protects \xintMinof against items with non terminated \the\numexpr expressions.
- 1.4 version is compatible with an empty input (empty items are handled as zero).

```

1789 \def\xintMinof {\romannumeral0\xintminof }%
1790 \def\xintminof #1{\expandafter\XINT_minof\romannumeral`&&@#1^}%
1791 \def\XINT_Minof{\romannumeral0\XINT_minof}%
1792 \def\XINT_minof#1%
1793 {%
1794     \xint_gob_til_ ^ #1\XINT_minof_empty ^%
1795     \expandafter\XINT_minof_loop\romannumeral0\xintraw{#1}\xint:
1796 }%
1797 \def\XINT_minof_empty ^#1\xint:{ 0/1[0]}%
1798 \def\XINT_minof_loop #1\xint:#2%
1799 {%
1800     \xint_gob_til_ ^ #2\XINT_minof_e ^%
1801     \expandafter\XINT_minof_loop\romannumeral0\xintmin{#1}{\romannumeral0\xintraw{#2}}\xint:
1802 }%
1803 \def\XINT_minof_e ^#1\xintmin #2#3\xint:{ #2}%

```

8.59 \xintCmp

```

1804 \def\xintCmp {\romannumeral0\xintcmp }%
1805 \def\xintcmp #1%
1806 {%
1807     \expandafter\XINT_fcmp\expandafter {\romannumeral0\xintraw {#1}}%
1808 }%
1809 \def\XINT_fcmp #1#2%
1810 {%
1811     \expandafter\XINT_fcmp_A\romannumeral0\xintraw {#2}#1%
1812 }%
1813 \def\XINT_fcmp_A #1#2/#3[#4]#5#6/#7[#8]%
1814 {%
1815     \xint_UDsignsfork
1816     #1#5\XINT_fcmp_minusminus
1817     -#5\XINT_fcmp_firstneg
1818     #1-\XINT_fcmp_secondneg
1819     --\XINT_fcmp_nonneg_a
1820     \krof
1821     #1#5{#2/#3[#4]}{#6/#7[#8]}%
1822 }%
1823 \def\XINT_fcmp_minusminus --#1#2{\XINT_fcmp_B #2#1}%
1824 \def\XINT_fcmp_firstneg #1-#2#3{ -1}%
1825 \def\XINT_fcmp_secondneg -#1#2#3{ 1}%
1826 \def\XINT_fcmp_nonneg_a #1#2%
1827 {%
1828     \xint_UDzerosfork
1829     #1#2\XINT_fcmp_zerozero
1830     0#2\XINT_fcmp_firstzero
1831     #10\XINT_fcmp_secondzero
1832     00\XINT_fcmp_pos
1833     \krof
1834     #1#2%
1835 }%
1836 \def\XINT_fcmp_zerozero #1#2#3#4{ 0}%
1837 \def\XINT_fcmp_firstzero #1#2#3#4{ -1}%
1838 \def\XINT_fcmp_secondzero #1#2#3#4{ 1}%

```

```

1839 \def\XINT_fcmp_pos #1#2#3#4%
1840 {%
1841     \XINT_fcmp_B #1#3#2#4%
1842 }%
1843 \def\XINT_fcmp_B #1/#2[#3]#4/#5[#6]%
1844 {%
1845     \expandafter\XINT_fcmp_C\expandafter
1846     {\the\numexpr #6-#3\expandafter}\expandafter
1847     {\romannumeral0\xintiimul {#4}{#2}}%
1848     {\romannumeral0\xintiimul {#5}{#1}}%
1849 }%
1850 \def\XINT_fcmp_C #1#2#3%
1851 {%
1852     \expandafter\XINT_fcmp_D\expandafter
1853     {#3}{#1}{#2}%
1854 }%
1855 \def\XINT_fcmp_D #1#2#3%
1856 {%
1857     \expandafter\XINT_cntSgnFork\romannumeral`&&\expandafter\XINT_cntSgn
1858     \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\xint:
1859     { -1}{\XINT_fcmp_E #2\Z {#3}{#1}}{ 1}%
1860 }%
1861 \def\XINT_fcmp_E #1%
1862 {%
1863     \xint_UDsignfork
1864     #1\XINT_fcmp_Fd
1865     -{\XINT_fcmp_Fn #1}%
1866     \krof
1867 }%

1868 \def\XINT_fcmp_Fd #1\Z #2#3%
1869 {%
1870     \expandafter\XINT_fcmp_Fe
1871     \romannumeral0\XINT_dsx_addzeros {#1}#3;\xint:#2\xint:
1872 }%
1873 \def\XINT_fcmp_Fe #1\xint:#2#3\xint:{\XINT_cmp_plusplus #2#1\xint:#3\xint:}%
1874 \def\XINT_fcmp_Fn #1\Z #2#3%
1875 {%
1876     \expandafter\XINT_fcmp_Fo
1877     \romannumeral0\XINT_dsx_addzeros {#1}#2;\xint:#3\xint:
1878 }%
1879 \def\XINT_fcmp_Fo #1#2\xint:#3\xint:{\XINT_cmp_plusplus #1#3\xint:#2\xint:}%

```

8.60 \xintAbs

```

1880 \def\xintAbs {\romannumeral0\xintabs }%
1881 \def\xintabs #1{\expandafter\XINT_abs\romannumeral0\xintraw {#1}}%

```

8.61 \xintOpp

```

1882 \def\xintOpp {\romannumeral0\xintopp }%
1883 \def\xintopp #1{\expandafter\XINT_opp\romannumeral0\xintraw {#1}}%

```

8.62 \xintInv

1.3d.

```

1884 \def\xintInv    {\romannumeral0\xintinv}%
1885 \def\xintinv #1{\expandafter\XINT_inv\romannumeral0\xintra #1}%
1886 \def\XINT_inv #1%
1887 {%
1888     \xint_UDzerominusfork
1889     #1-\XINT_inv_iszero
1890     0#1\XINT_inv_a
1891     0-\XINT_inv_a {}}%
1892     \krof #1%
1893}%
1894 \def\XINT_inv_iszero #1]%
1895     {\XINT_signalcondition{DivisionByZero}{Division of 1 by zero (#1)}}{\{0/1[0]}}%
1896 \def\XINT_inv_a #1#2/#3[#4#5]%
1897 {%
1898     \xint_UDzerominusfork
1899     #4-\XINT_inv_expiszero
1900     0#4\XINT_inv_b
1901     0-\XINT_inv_b -#4}%
1902     \krof #5.{#1#3/#2}%
1903}%
1904 \def\XINT_inv_expiszero #1.#2{ #2[0]}%
1905 \def\XINT_inv_b #1.#2{ #2[#1]}%

```

8.63 \xintSgn

```

1906 \def\xintSgn    {\romannumeral0\xintsgn}%
1907 \def\xintsgn #1{\expandafter\XINT_sgn\romannumeral0\xintra #1}\xint:}%

```

8.64 \xintGCD, \xintLCM

1.4. They replace the former [xintgcd](#) macros of the same names which truncated to integers their arguments. Fraction-producing [gcd\(\)](#) and [lcm\(\)](#) functions were available since 1.3d [xintexpr](#), with non-public support macros handling comma separated values.

```

1908 \def\xintGCD {\romannumeral0\xintgcd}%
1909 \def\xintgcd #1#2{\XINT_fgcdof{#1}{#2}^}%
1910 \def\xintLCM {\romannumeral0\xintlcm}%
1911 \def\xintlcm #1#2{\XINT_flcmof{#1}{#2}^}%

```

8.65 \xintGCDof

1.4. This inherits from former non public [xintexpr](#) macro called [\xintGCDof:csv](#), handling comma separated items, and former [xintgcd](#) macro called [\xintGCDof](#) which handled braced items to which it applied [\xintNum](#) before handling the computations on integers only. The macro keeps the former name [xintgcd](#), and handles fractions presented as braced items. It is now the support macro for the [gcd\(\)](#) function in [\xintexpr](#) and [\xintfloatexpr](#).

The support macro for the [gcd\(\)](#) function in [\xintiexpr](#) is [\xintiiGCDof](#) which is located in [xint](#).

```

1912 \def\xintGCDof {\romannumeral0\xintgcdof}%
1913 \def\xintgcdof #1{\expandafter\XINT_fgcdof\romannumeral`&&@#1^}%
1914 \def\XINT_GCDof{\romannumeral0\XINT_fgcdof}%

```

This abuses the way \xintiiabs works in order to avoid fetching whole argument again: \xintiiabs
^ raises no error.

```

1915 \def\xINT_fgcdof #1%
1916 {%
1917     \xint_gob_til_ ^ #1\xINT_fgcdof_empty ^%
1918     \expandafter\xINT_fgcdof_loop\romannumeral0\xintiiabs#1\xint:
1919 }%
1920 \def\xINT_fgcdof_empty ^#1\xint:{ 1/1[0]}%
1921 \def\xINT_fgcdof_loop #1\xint:#2%
1922 {%
1923     \expandafter\xINT_fgcdof_loop_a\romannumeral0\xintiiabs#2\xint:#1\xint:
1924 }%
1925 \def\xINT_fgcdof_loop_a#1#2\xint:#3\xint:
1926 {%
1927     \xint_gob_til_ ^ #1\xINT_fgcdof_end ^%
1928     \xint_gob_til_zero #1\xINT_fgcdof_skip 0%
1929     \expandafter\xINT_fgcdof_loop_b\romannumeral0\xintmod{#1#2}{#3}\xint:#3\xint:
1930 }%
1931 \def\xINT_fgcdof_end ^#1\xint:#2\xint:{ #2}%
1932 \def\xINT_fgcdof_skip 0%
1933     \expandafter\xINT_fgcdof_loop_b\romannumeral0\xintmod#1\xint:
1934 {%
1935     \XINT_fgcdof_loop
1936 }%
1937 \def\xINT_fgcdof_loop_b#1#2\xint:#3\xint:
1938 {%
1939     \xint_gob_til_zero #1\xINT_fgcdof_next 0%
1940     \expandafter\xINT_fgcdof_loop_b\romannumeral0\xintmod{#3}{#1#2}\xint:#1#2\xint:
1941 }%
1942 \def\xINT_fgcdof_next 0%
1943     \expandafter\xINT_fgcdof_loop_b\romannumeral0\xintmod#1#2\xint:#3\xint:#4%
1944 {%
1945     \expandafter\xINT_fgcdof_loop_a\romannumeral0\xintiiabs#4\xint:#1\xint:
1946 }%

```

8.66 \xintLCMof

See comments for \xintGCDof. xint provides integer only \xintiilCMof.

```

1947 \def\xintLCMof {\romannumeral0\xintlcmof}%
1948 \def\xintlcmof #1{\expandafter\xINT_flcmof\romannumeral`&&@#1^}%
1949 \def\xINT_LCMof{\romannumeral0\xINT_flcmof}%
1950 \def\xINT_flcmof #1%
1951 {%
1952     \xint_gob_til_ ^ #1\xINT_flcmof_empty ^%
1953     \expandafter\xINT_flcmof_loop\romannumeral0\xintiiabs\xintRaw{#1}\xint:
1954 }%
1955 \def\xINT_flcmof_empty ^#1\xint:{ 0/1[0]}%

```

\XINT_inv expects A/B[N] format which is the case here.

```

1956 \def\xINT_flcmof_loop #1%
1957 {%

```



```

1958 \xint_gob_til_zero #1\XINT_flgcmof_zero 0%
1959 \expandafter\XINT_flgcmof_d\romannumeral0\XINT_inv #1%
1960 }%
1961 \def\XINT_flgcmof_zero #1^{ 0/1[0]}%

```

`\xintRaw{^}` would raise an error thus we delay application of `\xintRaw` to new item. As soon as we hit against a zero item, the l.c.m is known to be zero itself. Else we need to inverse new item, but this requires full A/B[N] raw format, hence the `\xintraw`.

```

1962 \def\XINT_flgcmof_d #1\xint:#2%
1963 {%
1964 \expandafter\XINT_flgcmof_loop_a\romannumeral0\xintiiabs#2\xint:#1\xint:
1965 }%
1966 \def\XINT_flgcmof_loop_a #1#2\xint:%
1967 {%
1968 \xint_gob_til_ ^ #1\XINT_flgcmof_end ^%
1969 \xint_gob_til_zero #1\XINT_flgcmof_zero 0%
1970 \expandafter\XINT_flgcmof_loop_b\romannumeral0\expandafter\XINT_inv
1971 \romannumeral0\xintraw{#1#2}\xint:
1972 }%
1973 \def\XINT_flgcmof_end ^#1\xint:#2\xint:{\XINT_inv #2}%

```

This is Euclide algorithm.

```

1974 \def\XINT_flgcmof_loop_b #1#2\xint:#3\xint:
1975 {%
1976 \xint_gob_til_zero #1\XINT_flgcmof_next 0%
1977 \expandafter\XINT_flgcmof_loop_b\romannumeral0\xintmod{#3}{#1#2}\xint:#1#2\xint:
1978 }%
1979 \def\XINT_flgcmof_next 0%
1980 \expandafter\XINT_flgcmof_loop_b\romannumeral0\xintmod#1#2\xint:#3\xint:#4%
1981 {%
1982 \expandafter\XINT_flgcmof_loop_a\romannumeral0\xintiiabs#4\xint:#1\xint:
1983 }%

```

8.67 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.

8.68 \xintDigits, \xintSetDigits

1.3f modifies the (strange) original signature #1#2 for `\xintDigits` macro into #1=, allowing usage without colon. It also adds `\xintSetDigits`. Starred variants are added by `xintexpr.sty`.

```
1984 \mathchardef\xintdigits 16
1985 \def\xintDigits #1=%
1986   {\afterassignment \xint_gobble_i \mathchardef\xintdigits=}%
1987 \def\xinttheDigits {\number\xintdigits }%
1988 \def\xintSetDigits #1{\mathchardef\xintdigits=\numexpr#1\relax}%
```

8.69 \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.

```

1989 \def\xintFloat    {\romannumeral0\xintfloat }%
1990 \def\xintfloat #1{\XINT_float_chkopt #1\xint:}%
1991 \def\XINT_float_chkopt #1%
1992 {%
1993     \ifx [#1\expandafter\XINT_float_opt
1994         \else\expandafter\XINT_float_noopt
1995     \fi #1%
1996 }%
1997 \def\XINT_float_noopt #1\xint:%
1998 {%
1999     \expandafter\XINT_float_post
2000     \romannumeral0\XINTinfloat[\XINTdigits]{#1}\XINTdigits.%
2001 }%
2002 \def\XINT_float_opt [\xint:#1]%
2003 {%
2004     \expandafter\XINT_float_opt_a\the\numexpr #1.%
2005 }%
2006 \def\XINT_float_opt_a #1.#2%
2007 {%
2008     \expandafter\XINT_float_post
2009     \romannumeral0\XINTinfloat[#1]{#2}#1.%
2010 }%
2011 \def\XINT_float_post #1%
2012 {%
2013     \xint_UDzerominusfork
2014     #1-\XINT_float_zero
2015     0#1\XINT_float_neg
2016     0-\XINT_float_pos
2017     \krof #1%
2018 }%[
2019 \def\XINT_float_zero #1]#2.{ 0.e0}%
2020 \def\XINT_float_neg-\{\expandafter-\romannumeral0\XINT_float_pos}%
2021 \def\XINT_float_pos #1#2[#3]#4.%
2022 {%
2023     \expandafter\XINT_float_pos_done\the\numexpr#3+#4-\xint_c_i.#1.#2;%
2024 }%
2025 \def\XINT_float_pos_done #1.#2;{ #2e#1}%

```

8.70 \XINTinFloat, \XINTinFloatS, \XINTiLogTen

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

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

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

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

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

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

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

1.3e adds `\XINTiLogTen`.

```
2026 \def\xINTinFloat {\romannumeral0\xINTinfloat}%
2027 \def\xINTinfloat
2028   {\expandafter\xINT_infloat_clean\romannumeral0\xINT_infloat}%
```

Attention que ici le fait que l'on grabbe `#1` est important car il pourrait y avoir un zéro (en particulier dans le cas où `input` est nul).

```
2029 \def\xINT_infloat_clean #1%
2030   {\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`

```
2031 \def\xINT_infloat_clean_a !#1.#2[#3]%
2032 {%
2033   \expandafter\xINT_infloat_done
2034   \the\numexpr #3-#1\expandafter.%
2035   \romannumeral0\xINT_dsx_addzeros {#1}#2;;%
2036}%
2037 \def\xINT_infloat_done #1.#2;{ #2[#1]}%
```

variant which allows output with shorter mantissas.

```
2038 \def\xINTinFloatS {\romannumeral0\xINTinfloatS}%
2039 \def\xINTinfloatS
2040   {\expandafter\xINT_infloatS_clean\romannumeral0\xINT_infloat}%
2041 \def\xINT_infloatS_clean #1%
2042   {\if #1!\xint_dothis\xINT_infloatS_clean_a\fi\xint_orthat{ }#1}%
2043 \def\xINT_infloatS_clean_a !#1.{ }%
```

1.3e ajoute `\XINTiLogTen`. Le comportement pour un input nul est non encore finalisé. Il changera lorsque `NaN`, `+Inf`, `-Inf` existeront.

```
2044 \def\xINTFloatiLogTen {\the\numexpr\xINTfloatilogten}%
2045 \def\xINTfloatilogten [#1]#2%
2046   {\expandafter\xINT_floatilogten\romannumeral0\xINT_infloat[#1]{#2}#1.%}
```

```

2047 \def\XINT_floatilogten #1{%
2048   \if #10\xint_dothis\XINT_floatilogten_z\fi
2049   \if #1!\xint_dothis\XINT_floatilogten_a\fi
2050   \xint_orthat\XINT_floatilogten_b #1%
2051 }%
2052 \def\XINT_floatilogten_z 0[0]#1.{-7FFF8000\relax}%
2053 \def\XINT_floatilogten_a !#1.#2[#3]#4.{#3-#1+#4-1\relax}%
2054 \def\XINT_floatilogten_b #1[#2]#3.{#2+#3-1\relax}%

```

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

```

2055 \def\XINT_infloat [#1]#2%
2056 {%
2057   \expandafter\XINT_infloat_a\the\numexpr #1\expandafter.%
2058   \romannumeral0\XINT_infrac {#2}%
2059 }%

```

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

```

2060 \def\XINT_infloat_a #1.#2#3#4%
2061 {%

```

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

```

2062   \if1\XINT_is_One#4XY%
2063     \expandafter\XINT_infloat_sp
2064   \else\expandafter\XINT_infloat_fork
2065   \fi #3.{#1}{#2}{#4}%
2066 }%

```

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.

```

2067 \def\XINT_infloat_sp #1%
2068 {%
2069   \xint_UDzerominusfork
2070   #1-\XINT_infloat_spzero
2071   0#1\XINT_infloat_spneg
2072   0-\XINT_infloat_sppos
2073   \krof #1%
2074 }%

```

Attention surtout pas 0/1[0] ici.

```

2075 \def\XINT_infloat_spzero 0.#1#2#3{ 0[0]}%
2076 \def\XINT_infloat_spneg-%
2077   {\expandafter\XINT_infloat_spnegend\romannumeral0\XINT_infloat_sppos}%
2078 \def\XINT_infloat_spnegend #1%
2079   {\if#1!\expandafter\XINT_infloat_spneg_needzeros\fi -#1}%
2080 \def\XINT_infloat_spneg_needzeros -!#1.{!#1.-}%

```

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

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

```

2081 \def\XINT_infloat_sppos #1.#2#3#4%
2082 {%
2083   \expandafter\XINT_infloat_sp_b\the\numexpr#2-\xintLength{#1}.#1.#2.#3.%
2084 }%

```

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

#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.

```
2085 \def\XINT_infloat_sp_b #1%
2086 {%
2087   \xint_UDzerominusfork
2088   #1-\XINT_infloat_sp_quick
2089   0#1\XINT_infloat_sp_c
2090   0-\XINT_infloat_sp_needzeros
2091   \krof #1%
2092 }%
```

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

```
2093 \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.

```
2094 \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.

```
2095 \def\XINT_infloat_sp_c -#1.#2#3.#4.#5.%
2096 {%
2097   \expandafter\XINT_infloat_Y
2098   \the\numexpr #5+#1\expandafter.%
2099   \romannumeral0\expandafter\XINT_infloat_sp_round
2100   \romannumeral0\XINT_split_fromleft
2101   (\xint_c_i+#4).#2#3\xint_bye2345678\xint_bye..#2%
2102 }%
2103 \def\XINT_infloat_sp_round #1.#2.%
2104 {%
2105   \XINT_dsrr#1\xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax.%
2106 }%
```

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.

```
2107 \def\XINT_infloat_fork #1%
2108 {%
2109   \xint_UDsignfork
2110   #1\XINT_infloat_J
2111   -\XINT_infloat_K
2112   \krof #1%
2113 }%
2114 \def\XINT_infloat_J-{\expandafter-\romannumeral0\XINT_infloat_K }%
```

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

```
2115 \def\XINT_infloat_K #1.#2%
2116 {%
2117     \expandafter\XINT_infloat_L
2118     \the\numexpr\xintLength{#1}\expandafter.\the\numexpr #2+\xint_c_iv.{#1}{#2}%
2119 }%
```

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

```
2120 \def\XINT_infloat_L #1.#2.%
2121 {%
2122     \ifnum #1>#2
2123         \expandafter\XINT_infloat_Ma
2124     \else
2125         \expandafter\XINT_infloat_Mb
2126     \fi #1.#2.%
2127 }%
```

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

```
2128 \def\XINT_infloat_Ma #1.#2.#3%
2129 {%
2130     \expandafter\XINT_infloat_MtoN\expandafter-\expandafter0\expandafter.%
2131     \romannumeral0\XINT_split_fromleft#2.#3\xint_bye2345678\xint_bye..%
2132     #2.#1.{#3}%
2133 }%
```

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

Here A is short. We set $u = P+4 - |A|$, and $A'' = A$ ($A' = 10^u A$)
output: $u.A''.$ $P+4.$ $|A|. \{A\}_{P\{n\{B\}}$

```
2134 \def\XINT_infloat_Mb #1.#2.#3%
2135 {%
2136     \expandafter\XINT_infloat_MtoN\the\numexpr#2-#1.%
2137     #3..#2.#1.{#3}%
2138 }%
```

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

```
2139 \def\XINT_infloat_MtoN #1.#2.#3.#4.#5.#6#7#8#9%
2140 {%
2141     \expandafter\XINT_infloat_N
2142     \the\numexpr\xintLength{#9}.#4.{#9}#1.#2.#7.#5.#8.{#6}{#9}%
2143 }%
2144 \def\XINT_infloat_N #1.#2.%
2145 {%
2146     \ifnum #1>#2
2147         \expandafter\XINT_infloat_Oa
2148     \else
2149         \expandafter\XINT_infloat_Ob
2150     \fi #1.#2.%
2151 }%
```

```
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}
```

```
2152 \def\XINT_infloat_Oa #1.#2.#3%
2153 {%
2154     \expandafter\XINT_infloat_P\expandafter-\expandafter0\expandafter.%
2155     \romannumeral0\XINT_split_fromleft#2.#3\xint_bye2345678\xint_bye..%
2156     #1.%
2157 }%
```

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

```
2158 \def\XINT_infloat_Ob #1.#2.#3%
2159 {%
2160     \expandafter\XINT_infloat_P\the\numexpr#2-#1.#3..#1.%
2161 }%
```

```
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.

```
2162 \def\XINT_infloat_P #1#2.#3.#4.#5.#6#7.#8.#9.%
2163 {%
2164     \csname XINT_infloat_Q\if-#1\else\if-#6\else q\fi\fi\expandafter\endcsname
2165     \romannumeral0\xintiiquo
2166     {\romannumeral0\XINT_dsx_addzerosnofuss
2167         {#6#7-#1#2+#9+\xint_c_iii\if-#1\else\if-#6\else-\xint_c_ii\fi\fi}#8;}%
2168     {#3}.#9.#5.%
2169 }%
```

«quick» branch.

```
2170 \def\XINT_infloat_Qq #1.#2.%
2171 {%
2172     \expandafter\XINT_infloat_Rq
2173     \romannumeral0\XINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..#2.%
2174 }%
2175 \def\XINT_infloat_Rq #1.#2#3.%
2176 {%
2177     \ifnum#2<\xint_c_v
2178         \expandafter\XINT_infloat_SEq
2179     \else\expandafter\XINT_infloat_SUP
2180     \fi
2181     {\if.#3.\xint_c_\else\xint_c_i\fi}#1.%
2182 }%
```

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

```
2183 \def\XINT_infloat_Q #1.#2.%
2184 {%
2185     \expandafter\XINT_infloat_R
2186     \romannumeral0\XINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..#2.%
```



```

2187 }%
2188 \def\XINT_infloat_R #1.#2#3#4#5.%
2189 {%
2190     \if.#5.\expandafter\XINT_infloat_Sa\else\expandafter\XINT_infloat_Sb\fi
2191     #2#3#4#5.#1.%
2192 }%

```

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

```

2193 \def\XINT_infloat_Sa #1.%
2194 {%
2195     \ifnum#1>500 \xint_dothis\XINT_infloat_SUP\fi
2196     \ifnum#1<499 \xint_dothis\XINT_infloat_SEq\fi
2197     \xint_orthat\XINT_infloat_X\xint_c_
2198 }%
2199 \def\XINT_infloat_Sb #1.%
2200 {%
2201     \ifnum#1>5009 \xint_dothis\XINT_infloat_SUP\fi
2202     \ifnum#1<4990 \xint_dothis\XINT_infloat_SEq\fi
2203     \xint_orthat\XINT_infloat_X\xint_c_i
2204 }%

```

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

```

2205 \def\XINT_infloat_SEq #1#2.#3.#4.#5.#6.#7#8%
2206 {%
2207     \expandafter\XINT_infloat_SY
2208     \the\numexpr #6+#5-#4-#3+#1.#2.%
2209 }%
2210 \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.

```

2211 \def\XINT_infloat_SUP #1#2#3.#4.#5.#6.#7.#8#9%
2212 {%
2213     \expandafter\XINT_infloat_Y
2214     \the\numexpr#7+#6-#5-#4+#1\expandafter.%
2215     \romannumeral0\xintinc{#2#3}.#2%
2216 }%

```

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

```

2217 \def\XINT_infloat_X #1#2#3.#4.#5.#6.#7.#8#9%
2218 {%
2219     \expandafter\XINT_infloat_Y
2220     \the\numexpr #7+#6-#5-#4+#1\expandafter.%

```

```

2221 \romannumeral`&&@\romannumeral0\xintiiiflt
2222 {\xintDSH{#6-#5-#4+#1}{\xintDouble{#8}}}%
2223 {\xintiiMul{\xintInc{\xintDouble{#2#3}}}{#9}}%
2224 \xint_firstofone
2225 \xintinc{#2#3}.#2%
2226 }%

```

check for rounding up to next power of ten.

```

2227 \def\xint_infloat_Y #1{%
2228 \def\xint_infloat_Y ##1.##2##3.##4%
2229 {%
2230 \if##49\if##21\expandafter\expandafter\expandafter\xint_infloat_Z\fi\fi
2231 #1##2##3[##1]%
2232 }}\xint_infloat_Y{ }%

```

#1=1, #2=0.

```

2233 \def\xint_infloat_Z #1#2#3[#4]%
2234 {%
2235 \expandafter\xint_infloat_ZZ\the\numexpr#4+\xint_c_i.#3.%
2236 }%
2237 \def\xint_infloat_ZZ #1.#2.{ 1#2[#1]}%

```

8.71 \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).

```

2238 \def\xintPFloat {\romannumeral0\xintpfloat }%
2239 \def\xintpfloat #1{\XINT_pfloat_chkopt #1\xint:}%
2240 \def\xint_pfloat_chkopt #1%
2241 {%
2242 \ifx [#1\expandafter\xint_pfloat_opt
2243 \else\expandafter\xint_pfloat_noopt
2244 \fi #1%
2245 }%
2246 \def\xint_pfloat_noopt #1\xint:%
2247 {%
2248 \expandafter\xint_pfloat_a
2249 \romannumeral0\xintfloat [\XINTdigits]{#1};\XINTdigits.%
2250 }%

2251 \def\xint_pfloat_opt [\xint:#1]%
2252 {%
2253 \expandafter\xint_pfloat_opt_a \the\numexpr #1.%
2254 }%
2255 \def\xint_pfloat_opt_a #1.#2%

```

```

2256 {%
2257     \expandafter\XINT_pfloat_a\romannumeral0\xintfloat [#1]{#2};#1.%
2258 }%
2259 \def\XINT_pfloat_a #1%
2260 {%
2261     \xint_UDzerominusfork
2262     #1-\XINT_pfloat_zero
2263     0#1\XINT_pfloat_neg
2264     0-\XINT_pfloat_pos
2265     \krof #1%
2266 }%

2267 \def\XINT_pfloat_zero #1;#2.{ 0.}%
2268 \def\XINT_pfloat_neg-{\expandafter-\romannumeral0\XINT_pfloat_pos }%

2269 \def\XINT_pfloat_pos #1.#2e#3;#4.%
2270 {%
2271     \ifnum #3>\xint_c_v \xint_dothis\XINT_pfloat_no\fi
2272     \ifnum #3<-\xint_c_v \xint_dothis\XINT_pfloat_no\fi
2273     \ifnum #3<\xint_c_ \xint_dothis\XINT_pfloat_N\fi
2274     \ifnum #3>\numexpr #4-\xint_c_i\relax \xint_dothis\XINT_pfloat_Ps\fi
2275     \xint_orthat\XINT_pfloat_P #1#2e#3;%
2276 }%
2277 \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.

2278 \def\XINT_pfloat_N #1e-#2;%
2279 {%
2280     \csname XINT_pfloat_N_\romannumeral#2\endcsname #1%
2281 }%
2282 \def\XINT_pfloat_N_i { 0.}%
2283 \def\XINT_pfloat_N_ii { 0.0}%
2284 \def\XINT_pfloat_N_iii{ 0.00}%
2285 \def\XINT_pfloat_N_iv { 0.000}%
2286 \def\XINT_pfloat_N_v { 0.0000}%

2287 \def\XINT_pfloat_P #1e#2;%
2288 {%
2289     \csname XINT_pfloat_P_\romannumeral#2\endcsname #1%
2290 }%
2291 \def\XINT_pfloat_P_ #1{ #1.}%
2292 \def\XINT_pfloat_P_i #1#2{ #1#2.}%
2293 \def\XINT_pfloat_P_ii #1#2#3{ #1#2#3.}%
2294 \def\XINT_pfloat_P_iii#1#2#3#4{ #1#2#3#4.}%
2295 \def\XINT_pfloat_P_iv #1#2#3#4#5{ #1#2#3#4#5.}%
2296 \def\XINT_pfloat_P_v #1#2#3#4#5#6{ #1#2#3#4#5#6.}%

2297 \def\XINT_pfloat_Ps #1e#2;%
2298 {%

```

```

2299 \csname XINT_pfloat_Ps\romannumeral#2\endcsname #100000;%
2300 }%
2301 \def\XINT_pfloat_Psi #1#2#3;{ #1#2.%}%
2302 \def\XINT_pfloat_Psii #1#2#3#4;{ #1#2#3.%}%
2303 \def\XINT_pfloat_Psiii#1#2#3#4#5;{ #1#2#3#4.%}%
2304 \def\XINT_pfloat_Psiv #1#2#3#4#5#6;{ #1#2#3#4#5.%}%
2305 \def\XINT_pfloat_Psv #1#2#3#4#5#6#7;{ #1#2#3#4#5#6.%}%

```

8.72 \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 \xinttfrac was just a first implementation.

```

2306 \def\XINTinFloatFracdigits {\romannumeral0\XINTinfloatfracdigits }%
2307 \def\XINTinfloatfracdigits #1%
2308 {%
2309 \expandafter\XINT_infloatfracdg_a\expandafter {\romannumeral0\xinttfrac{#1}}%
2310 }%
2311 \def\XINT_infloatfracdg_a {\XINTinfloat [\XINTdigits]}%

```

8.73 \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.

```

2312 \def\xintFloatAdd {\romannumeral0\xintfloatadd }%
2313 \def\xintfloatadd #1{\XINT_fladd_chkopt \xintfloat #1\xint:}%
2314 \def\XINTinFloatAdd {\romannumeral0\XINTinfloatadd }%
2315 \def\XINTinfloatadd #1{\XINT_fladd_chkopt \XINTinfloatS #1\xint:}%
2316 \def\XINT_fladd_chkopt #1#2%
2317 {%
2318 \ifx [#2\expandafter\XINT_fladd_opt
2319 \else\expandafter\XINT_fladd_noopt
2320 \fi #1#2%
2321 }%
2322 \def\XINT_fladd_noopt #1#2\xint:#3%
2323 {%
2324 #1[\XINTdigits]%
2325 {\expandafter\XINT_FL_add_a
2326 \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{#3}}%
2327 }%
2328 \def\XINT_fladd_opt #1[\xint:#2]#3#4%
2329 {%
2330 \expandafter\XINT_fladd_opt_a\the\numexpr #2.#1%

```

```

2331 }%
2332 \def\XINT_fladd_opt_a #1.#2#3#4%
2333 {%
2334     #2[#1]{\expandafter\XINT_FL_add_a\romannumeral0\XINTinfloat[#1]{#3}#1.{#4}}%
2335 }%
2336 \def\XINT_FL_add_a #1%
2337 {%
2338     \xint_gob_til_zero #1\XINT_FL_add_zero 0\XINT_FL_add_b #1%
2339 }%

2340 \def\XINT_FL_add_zero #1.#2{#2}%[
2341 \def\XINT_FL_add_b #1]#2.#3%
2342 {%
2343     \expandafter\XINT_FL_add_c\romannumeral0\XINTinfloat[#2]{#3}#2.#1)%
2344 }%

2345 \def\XINT_FL_add_c #1%
2346 {%
2347     \xint_gob_til_zero #1\XINT_FL_add_zero 0\XINT_FL_add_d #1%
2348 }%

2349 \def\XINT_FL_add_d #1[#2]#3.#4[#5]%
2350 {%
2351     \ifnum\numexpr #2-#3-#5>\xint_c_\xint_dothis\xint_firstoftwo\fi
2352     \ifnum\numexpr #5-#3-#2>\xint_c_\xint_dothis\xint_secondoftwo\fi
2353     \xint_orthat\xintAdd {#1[#2]}{#4[#5]}%
2354 }%

```

8.74 \xintFloatSub, \XINTinFloatSub

First done 1.07.

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

```

2355 \def\xintFloatSub {\romannumeral0\xintfloatsub }%
2356 \def\xintfloatsub #1{\XINT_fsub_chkopt \xintfloat #1\xint:}%
2357 \def\XINTinFloatSub {\romannumeral0\XINTinfloatsub }%
2358 \def\XINTinfloatsub #1{\XINT_fsub_chkopt \XINTinfloatS #1\xint:}%
2359 \def\XINT_fsub_chkopt #1#2%
2360 {%
2361     \ifx [#2\expandafter\XINT_fsub_opt
2362     \else\expandafter\XINT_fsub_noopt
2363     \fi #1#2%
2364 }%
2365 \def\XINT_fsub_noopt #1#2\xint:#3%
2366 {%
2367     #1[\XINTdigits]%
2368     {\expandafter\XINT_FL_add_a
2369     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{\xintOpp{#3}}}%
2370 }%
2371 \def\XINT_fsub_opt #1[\xint:#2]%#3#4%
2372 {%

```

```

2373 \expandafter\XINT_flsb_opt_a\the\numexpr #2.#1%
2374 }%
2375 \def\XINT_flsb_opt_a #1.#2#3#4%
2376 {%
2377 #2[#1]{\expandafter\XINT_FL_add_a\romannumeral0\XINTinfloat[#1]{#3}#1.{\xintOpp{#4}}}%
2378 }%

```

8.75 \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).

```

2379 \def\xintFloatMul {\romannumeral0\xintfloatmul }%
2380 \def\xintfloatmul #1{\XINT_flmul_chkopt \xintfloat #1\xint:}%
2381 \def\XINTinFloatMul {\romannumeral0\XINTinfloatmul }%
2382 \def\XINTinfloatmul #1{\XINT_flmul_chkopt \XINTinfloatS #1\xint:}%
2383 \def\XINT_flmul_chkopt #1#2%
2384 {%
2385 \ifx [#2\expandafter\XINT_flmul_opt
2386 \else\expandafter\XINT_flmul_noopt
2387 \fi #1#2%
2388 }%
2389 \def\XINT_flmul_noopt #1#2\xint:#3%
2390 {%
2391 #1[\XINTdigits]%
2392 {\expandafter\XINT_FL_mul_a
2393 \romannumeral0\XINTinfloatS[\XINTdigits]{#2}\XINTdigits.{#3}}%
2394 }%
2395 \def\XINT_flmul_opt #1[\xint:#2]#3#4%
2396 {%
2397 \expandafter\XINT_flmul_opt_a\the\numexpr #2.#1%
2398 }%
2399 \def\XINT_flmul_opt_a #1.#2#3#4%
2400 {%
2401 #2[#1]{\expandafter\XINT_FL_mul_a\romannumeral0\XINTinfloatS[#1]{#3}#1.{#4}}%
2402 }%
2403 \def\XINT_FL_mul_a #1[#2]#3.#4%
2404 {%
2405 \expandafter\XINT_FL_mul_b\romannumeral0\XINTinfloatS[#3]{#4}#1[#2]%
2406 }%
2407 \def\XINT_FL_mul_b #1[#2]#3[#4]{\xintiiMul{#3}{#1}/1[#4+#2]}%

```

8.76 \XINTinFloatInv

Added belatedly at 1.3e, to support inv() function. We use Short output, for rare inv(\xintexpr 1/3\relax) case. I need to think the whole thing out at some later date.

```
2408 \def\XINTinFloatInv#1{\XINTinFloatS[\XINTdigits]{\xintInv{#1}}}%
```

8.77 \xintFloatDiv, \XINTinFloatDiv

1.07.

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

1.2g handles the inputs via \XINTinFloatS which will be more efficient when the precision is large and the input is for example a small constant like 2.

The actual rounding of the quotient is handled via \xintfloat (or \XINTinfloatS).

1.2k does the same kind of improvement in \XINT_FL_div_b as for multiplication: earlier code was unnecessarily high level.

```
2409 \def\xintFloatDiv {\romannumeral0\xintfloatdiv }%
2410 \def\xintfloatdiv #1{\XINT_fldiv_chkopt \xintfloat #1\xint:}%
2411 \def\XINTinFloatDiv {\romannumeral0\XINTinfloatdiv }%
2412 \def\XINTinfloatdiv #1{\XINT_fldiv_chkopt \XINTinfloatS #1\xint:}%
2413 \def\XINT_fldiv_chkopt #1#2%
2414 {%
2415     \ifx [#2\expandafter\XINT_fldiv_opt
2416         \else\expandafter\XINT_fldiv_noopt
2417     \fi #1#2%
2418 }%

2419 \def\XINT_fldiv_noopt #1#2\xint:#3%
2420 {%
2421     #1[\XINTdigits]%
2422     {\expandafter\XINT_FL_div_a
2423         \romannumeral0\XINTinfloatS[\XINTdigits]{#3}\XINTdigits.{#2}}%
2424 }%
2425 \def\XINT_fldiv_opt #1[\xint:#2]%#3#4%
2426 {%
2427     \expandafter\XINT_fldiv_opt_a\the\numexpr #2.#1%
2428 }%

2429 \def\XINT_fldiv_opt_a #1.#2#3#4%
2430 {%
2431     #2[#1]{\expandafter\XINT_FL_div_a\romannumeral0\XINTinfloatS[#1]{#4}#1.{#3}}%
2432 }%
2433 \def\XINT_FL_div_a #1[#2]#3.#4%
2434 {%
2435     \expandafter\XINT_FL_div_b\romannumeral0\XINTinfloatS[#3]{#4}/#1e#2%
2436 }%

2437 \def\XINT_FL_div_b #1[#2]{#1e#2}%
```

8.78 \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).

```

2438 \def\xintFloatPow {\romannumeral0\xintfloatpow}%
2439 \def\xintfloatpow #1{\XINT_flpow_chkopt \xintfloat #1\xint:}%
2440 \def\XINTinFloatPow {\romannumeral0\XINTinfloatpow}%
2441 \def\XINTinfloatpow #1{\XINT_flpow_chkopt \XINTinfloatS #1\xint:}%
2442 \def\XINT_flpow_chkopt #1#2%
2443 {%
2444   \ifx [#2\expandafter\XINT_flpow_opt
2445     \else\expandafter\XINT_flpow_noopt
2446   \fi
2447   #1#2%
2448 }%
2449 \def\XINT_flpow_noopt #1#2\xint:#3%
2450 {%
2451   \expandafter\XINT_flpow_checkB_a
2452   \the\numexpr #3.\XINTdigits.{#2}{#1[\XINTdigits]]}%
2453 }%
2454 \def\XINT_flpow_opt #1[\xint:#2]%
2455 {%
2456   \expandafter\XINT_flpow_opt_a\the\numexpr #2.#1%
2457 }%
2458 \def\XINT_flpow_opt_a #1.#2#3#4%
2459 {%
2460   \expandafter\XINT_flpow_checkB_a\the\numexpr #4.#1.{#3}{#2[#1]}%
2461 }%
2462 \def\XINT_flpow_checkB_a #1%
2463 {%
2464   \xint_UDzerominusfork
2465   #1-\XINT_flpow_BisZero
2466   0#1{\XINT_flpow_checkB_b -}%
2467   0-{\XINT_flpow_checkB_b }{#1}%
2468   \krof
2469 }%
2470 \def\XINT_flpow_BisZero .#1.#2#3{#3{1[0]}}%

2471 \def\XINT_flpow_checkB_b #1#2.#3.%
2472 {%
2473   \expandafter\XINT_flpow_checkB_c
2474   \the\numexpr\xintLength{#2}+\xint_c_iii.#3.#2.{#1}%
2475 }%

2476 \def\XINT_flpow_checkB_c #1.#2.%
2477 {%
2478   \expandafter\XINT_flpow_checkB_d\the\numexpr#1+#2.#1.#2.%
2479 }%

```

1.2f rounds input to P digits, first.

```

2480 \def\XINT_flpow_checkB_d #1.#2.#3.#4.#5#6%
2481 {%

```



```

2482 \expandafter \XINT_flpow_aa
2483 \romannumeral0\XINTinfloat [#3]{#6}{#2}{#1}{#4}{#5}%
2484 }%

2485 \def\XINT_flpow_aa #1[#2]#3%
2486 {%
2487 \expandafter\XINT_flpow_ab\the\numexpr #2-#3\expandafter.%
2488 \romannumeral\XINT_rep #3\endcsname0.#1.%
2489 }%

2490 \def\XINT_flpow_ab #1.#2.#3.{\XINT_flpow_a #3#2[#1]}%

2491 \def\XINT_flpow_a #1%
2492 {%
2493 \xint_UDzerominusfork
2494 #1-\XINT_flpow_zero
2495 0#1{\XINT_flpow_b \iftrue}%
2496 0-{\XINT_flpow_b \iffalse#1}%
2497 \krof
2498 }%
2499 \def\XINT_flpow_zero #1[#2]#3#4#5#6%
2500 {%
2501 #6{\if 1#51\xint_dothis {0[0]}\fi
2502 \xint_orthat
2503 {\XINT_signalcondition{DivisionByZero}{0 to the power #4}{0[0]}}%
2504 }%
2505 }%

2506 \def\XINT_flpow_b #1#2[#3]#4#5%
2507 {%
2508 \XINT_flpow_loopI #5.#3.#2.#4.{#1\ifodd #5 \xint_c_i\fi\fi}%
2509 }%

2510 \def\XINT_flpow_truncate #1.#2.#3.%
2511 {%
2512 \expandafter\XINT_flpow_truncate_a
2513 \romannumeral0\XINT_split_fromleft
2514 #3.#2\xint_bye2345678\xint_bye..#1.#3.%
2515 }%

2516 \def\XINT_flpow_truncate_a #1.#2.#3.{#3+\xintLength{#2}.#1.%
2517 \def\XINT_flpow_loopI #1.%
2518 {%
2519 \ifnum #1=\xint_c_i\expandafter\XINT_flpow_ItoIII\fi
2520 \ifodd #1
2521 \expandafter\XINT_flpow_loopI_odd
2522 \else
2523 \expandafter\XINT_flpow_loopI_even
2524 \fi
2525 #1.%
2526 }%

```

```

2527 \def\XINT_flpow_ItoIII\ifodd #1\fi #2.#3.#4.#5.#6%
2528 {%
2529     \expandafter\XINT_flpow_III\the\numexpr #6+\xint_c_.#3.#4.#5.%
2530 }%

2531 \def\XINT_flpow_loopI_even #1.#2.#3.%#4.%
2532 {%
2533     \expandafter\XINT_flpow_loopI
2534     \the\numexpr #1/\xint_c_ii\expandafter.%
2535     \the\numexpr\expandafter\XINT_flpow_truncate
2536     \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.%
2537 }%
2538 \def\XINT_flpow_loopI_odd #1.#2.#3.#4.%
2539 {%
2540     \expandafter\XINT_flpow_loopII
2541     \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter.%
2542     \the\numexpr\expandafter\XINT_flpow_truncate
2543     \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#2.#3.%
2544 }%
2545 \def\XINT_flpow_loopII #1.%
2546 {%
2547     \ifnum #1 = \xint_c_i\expandafter\XINT_flpow_ItoIII\fi
2548     \ifodd #1
2549         \expandafter\XINT_flpow_loopII_odd
2550     \else
2551         \expandafter\XINT_flpow_loopII_even
2552     \fi
2553     #1.%
2554 }%
2555 \def\XINT_flpow_loopII_even #1.#2.#3.%#4.%
2556 {%
2557     \expandafter\XINT_flpow_loopII
2558     \the\numexpr #1/\xint_c_ii\expandafter.%
2559     \the\numexpr\expandafter\XINT_flpow_truncate
2560     \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.%
2561 }%
2562 \def\XINT_flpow_loopII_odd #1.#2.#3.#4.#5.#6.%
2563 {%
2564     \expandafter\XINT_flpow_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_flpow_loopII_odda #1.#2.#3.#4.#5.#6.%
2570 {%
2571     \expandafter\XINT_flpow_loopII
2572     \the\numexpr #4/\xint_c_ii-\xint_c_i\expandafter.%
2573     \the\numexpr\expandafter\XINT_flpow_truncate
2574     \the\numexpr\xint_c_ii*#5\expandafter.\romannumeral0\xintiisqr{#6}.#3.%
2575     #1.#2.%
2576 }%

```

```

2577 \def\XINT_flpow_IItoIII\ifodd #1\fi #2.#3.#4.#5.#6.#7.#8%
2578 {%
2579     \expandafter\XINT_flpow_III\the\numexpr #8+\xint_c_\expandafter.%
2580     \the\numexpr\expandafter\XINT_flpow_truncate
2581     \the\numexpr#3+#6\expandafter.\romannumeral0\xintiimul{#4}{#7}.#5.%
2582 }%

```

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).

```

2583 \def\XINT_flpow_III #1.#2.#3.#4.#5%
2584 {%
2585     \expandafter\XINT_flpow_IIIend
2586     \xint_UDsignfork
2587     #5{{1/#3[-#2]}}}%
2588     -{{#3[#2]}}}%
2589     \krof #1%
2590 }%

2591 \def\XINT_flpow_IIIend #1#2#3%
2592     {#3{\if#21\xint_afterfi{\expandafter-\romannumeral`&&@\fi#1}}}%

```

8.79 `\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.

```

2593 \def\xintFloatPower {\romannumeral0\xintfloatpower}%
2594 \def\xintfloatpower #1{\XINT_flpower_chkopt \xintfloat #1\xint:}%
2595 \def\XINTinFloatPower {\romannumeral0\XINTinfloatpower }%
2596 \def\XINTinfloatpower #1{\XINT_flpower_chkopt \XINTinfloatS #1\xint:}%

```

First the special macro for use by the expression parser which checks if one raises to an half-integer exponent. This is always with `\XINTdigits` precision. Rewritten for 1.2k in order for the final square root to keep three guard digits.

We have to be careful that exponent #2 is not constrained by TeX bound. And we must allow fractions. The 1.2k variant does a rounding to nearest integer of half-integer, 1.2f did a truncation rather (this is done after truncation of #2 to fixed point with one digit after mark.) We try to recognize quickly the case of integer exponent, for speed, but there is overhead of going through `\xintiTrunc1`.

```

2597 \def\xINTinFloatPowerH {\romannumeral0\xINTinfloatpowerh }%

2598 \def\xINTinfloatpowerh #1#2%
2599 {%
2600     \expandafter\xINT_flpowerh_a\romannumeral0\xintitrunc1{#2};%
2601     \XINTdigits.{#1}{\XINTinfloatS[\XINTdigits]}%
2602 }%

2603 \def\xINT_flpowerh_a #1;%
2604 {%
2605     \if0\xintLDg{#1}\expandafter\xINT_flpowerh_int
2606     \else\expandafter\xINT_flpowerh_b
2607     \fi #1.%
2608 }%
2609 \def\xINT_flpowerh_int #1%
2610 {%
2611     \if0#1\expandafter\xINT_flpower_BisZero
2612     \else\expandafter\xINT_flpowerh_i
2613     \fi #1%
2614 }%
2615 \def\xINT_flpowerh_i #10.{\expandafter\xINT_flpower_checkB_a#1.}%
2616 \def\xINT_flpowerh_b #1.%
2617 {%
2618     \expandafter\xINT_flpowerh_c\romannumeral0\xintdsrr{\xintDouble{#1}}.%
2619 }%
2620 \def\xINT_flpowerh_c #1.%
2621 {%
2622     \ifodd\xintLDg{#1} %<- intentional space
2623     \expandafter\xINT_flpowerh_d\else\expandafter\xINT_flpowerh_e
2624     \fi #1.%
2625 }%
2626 \def\xINT_flpowerh_d #1.\XINTdigits.#2#3%
2627 {%
2628     \XINT_flpower_checkB_a #1.\XINTdigits.{#2}\XINT_flpowerh_finish
2629 }%
2630 \def\xINT_flpowerh_finish #1%
2631     {\XINTinfloatS[\XINTdigits]{\XINTinFloatSqrt[\XINTdigits+\xint_c_iii]{#1}}}%
2632 \def\xINT_flpowerh_e #1.%
2633     {\expandafter\xINT_flpower_checkB_a\romannumeral0\xinthalff{#1}.}%

    Start of macro. Check for optional argument.

2634 \def\xINT_flpower_chkopt #1#2%
2635 {%
2636     \ifx [#2\expandafter\xINT_flpower_opt
2637     \else\expandafter\xINT_flpower_noopt

```

```

2638 \fi
2639 #1#2%
2640 }%
2641 \def\XINT_flpower_noopt #1#2\xint:#3%
2642 {%
2643 \expandafter\XINT_flpower_checkB_a
2644 \romannumeral0\xintnum{#3}.\XINTdigits.{#2}{#1[\XINTdigits]]}%
2645 }%
2646 \def\XINT_flpower_opt #1[\xint:#2]%
2647 {%
2648 \expandafter\XINT_flpower_opt_a\the\numexpr #2.#1%
2649 }%
2650 \def\XINT_flpower_opt_a #1.#2#3#4%
2651 {%
2652 \expandafter\XINT_flpower_checkB_a
2653 \romannumeral0\xintnum{#4}.#1.{#3}{#2[#1]]}%
2654 }%
2655 \def\XINT_flpower_checkB_a #1%
2656 {%
2657 \xint_UDzerominusfork
2658 #1-{ \XINT_flpower_BisZero 0}%
2659 0#1{ \XINT_flpower_checkB_b -}%
2660 0-{ \XINT_flpower_checkB_b }{#1}%
2661 \krof
2662 }%
2663 \def\XINT_flpower_BisZero 0.#1.#2#3{#3{1[0]}}%
2664 \def\XINT_flpower_checkB_b #1#2.#3.%
2665 {%
2666 \expandafter\XINT_flpower_checkB_c
2667 \the\numexpr\xintLength{#2}+\xint_c_iii.#3.#2.{#1}%
2668 }%

2669 \def\XINT_flpower_checkB_c #1.#2.%
2670 {%
2671 \expandafter\XINT_flpower_checkB_d\the\numexpr#1+#2.#1.#2.%
2672 }%

2673 \def\XINT_flpower_checkB_d #1.#2.#3.#4.#5#6%
2674 {%
2675 \expandafter \XINT_flpower_aa
2676 \romannumeral0\XINTinfloat [#3]{#6}{#2}{#1}{#4}{#5}%
2677 }%

2678 \def\XINT_flpower_aa #1[#2]#3%
2679 {%
2680 \expandafter\XINT_flpower_ab\the\numexpr #2-#3\expandafter.%
2681 \romannumeral\XINT_rep #3\endcsname0.#1.%
2682 }%
2683 \def\XINT_flpower_ab #1.#2.#3.{\XINT_flpower_a #3#2[#1]]}%
2684 \def\XINT_flpower_a #1%
2685 {%

```

```

2686 \xint_UDzerominusfork
2687 #1-\XINT_flpow_zero
2688 0#1{\XINT_flpower_b \iftrue}%
2689 0-{\XINT_flpower_b \iffalse#1}%
2690 \krof
2691 }%
2692 \def\XINT_flpower_b #1#2[#3]#4#5%
2693 {%
2694 \XINT_flpower_loopI #5.#3.#2.#4.{#1\xintiiOdd{#5}\fi}%
2695 }%
2696 \def\XINT_flpower_loopI #1.%
2697 {%
2698 \if1\XINT_isOne {#1}\xint_dothis\XINT_flpower_ItoIII\fi
2699 \ifodd\xintLDg{#1} %<- intentional space
2700 \xint_dothis{\expandafter\XINT_flpower_loopI_odd}\fi
2701 \xint_orthat{\expandafter\XINT_flpower_loopI_even}%

2702 \romannumeral0\XINT_half
2703 #1\xint_bye\xint_Bye345678\xint_bye
2704 *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax.%
2705 }%
2706 \def\XINT_flpower_ItoIII #1.#2.#3.#4.#5%
2707 {%
2708 \expandafter\XINT_flpow_III\the\numexpr #5+\xint_c_.#2.#3.#4.%
2709 }%
2710 \def\XINT_flpower_loopI_even #1.#2.#3.#4.%
2711 {%
2712 \expandafter\XINT_flpower_toloopI
2713 \the\numexpr\expandafter\XINT_flpow_truncate
2714 \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#1.%
2715 }%
2716 \def\XINT_flpower_toloopI #1.#2.#3.#4.{\XINT_flpower_loopI #4.#1.#2.#3.}%
2717 \def\XINT_flpower_loopI_odd #1.#2.#3.#4.%
2718 {%
2719 \expandafter\XINT_flpower_toloopII
2720 \the\numexpr\expandafter\XINT_flpow_truncate
2721 \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.%
2722 #1.#2.#3.%
2723 }%
2724 \def\XINT_flpower_toloopII #1.#2.#3.#4.{\XINT_flpower_loopII #4.#1.#2.#3.}%
2725 \def\XINT_flpower_loopII #1.%
2726 {%
2727 \if1\XINT_isOne{#1}\xint_dothis\XINT_flpower_ItoIII\fi
2728 \ifodd\xintLDg{#1} %<- intentional space
2729 \xint_dothis{\expandafter\XINT_flpower_loopII_odd}\fi
2730 \xint_orthat{\expandafter\XINT_flpower_loopII_even}%

2731 \romannumeral0\XINT_half#1\xint_bye\xint_Bye345678\xint_bye
2732 *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax.%
2733 }%
2734 \def\XINT_flpower_loopII_even #1.#2.#3.#4.%
2735 {%

```

```

2736 \expandafter\XINT_flpower_toloopII
2737 \the\numexpr\expandafter\XINT_flpow_truncate
2738 \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#1.%
2739 }%
2740 \def\XINT_flpower_loopII_odd #1.#2.#3.#4.#5.#6.%
2741 {%
2742 \expandafter\XINT_flpower_loopII_odda
2743 \the\numexpr\expandafter\XINT_flpow_truncate
2744 \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2745 #1.#2.#3.%
2746 }%
2747 \def\XINT_flpower_loopII_odda #1.#2.#3.#4.#5.#6.%
2748 {%
2749 \expandafter\XINT_flpower_toloopII
2750 \the\numexpr\expandafter\XINT_flpow_truncate
2751 \the\numexpr\xint_c_ii*#5\expandafter.\romannumeral0\xintiisqr{#6}.#3.%
2752 #4.#1.#2.%
2753 }%
2754 \def\XINT_flpower_IItoIII #1.#2.#3.#4.#5.#6.#7%
2755 {%
2756 \expandafter\XINT_flpow_III\the\numexpr #7+\xint_c_\expandafter.%
2757 \the\numexpr\expandafter\XINT_flpow_truncate
2758 \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2759 }%

```

8.80 \xintFloatFac, \XINTFloatFac

Done at 1.2. At 1.3e \XINTinFloatFac outputs using \XINTinFloatS.

```

2760 \def\xintFloatFac {\romannumeral0\xintfloatfac}%
2761 \def\xintfloatfac #1{\XINT_flfac_chkopt \xintfloat #1\xint:}%
2762 \def\XINTinFloatFac {\romannumeral0\XINTinfloatfac }%
2763 \def\XINTinfloatfac #1{\XINT_flfac_chkopt \XINTinfloatS #1\xint:}%
2764 \def\XINT_flfac_chkopt #1#2%
2765 {%
2766 \ifx [#2\expandafter\XINT_flfac_opt
2767 \else\expandafter\XINT_flfac_noopt
2768 \fi
2769 #1#2%
2770 }%
2771 \def\XINT_flfac_noopt #1#2\xint:
2772 {%
2773 \expandafter\XINT_FL_fac_fork_a
2774 \the\numexpr \xintNum{#2}.\xint_c_i \XINTdigits\XINT_FL_fac_out{#1[\XINTdigits]]}%
2775 }%
2776 \def\XINT_flfac_opt #1[\xint:#2]%
2777 {%
2778 \expandafter\XINT_flfac_opt_a\the\numexpr #2.#1%
2779 }%
2780 \def\XINT_flfac_opt_a #1.#2#3%
2781 {%
2782 \expandafter\XINT_FL_fac_fork_a\the\numexpr \xintNum{#3}.\xint_c_i {#1}\XINT_FL_fac_out{#2[#1]]}%
2783 }%

```

```

2784 \def\XINT_FL_fac_fork_a #1%
2785 {%
2786     \xint_UDzerominusfork
2787     #1-\XINT_FL_fac_iszero
2788     0#1\XINT_FL_fac_isneg
2789     0-\XINT_FL_fac_fork_b #1}%
2790     \krof
2791 }%
2792 \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.

```

2793 \def\XINT_FL_fac_isneg #1.#2#3#4#5%
2794 {%
2795     #5{\XINT_signalcondition{InvalidOperation}
2796         {Factorial of negative: (-#1)!}{0[0]}}%
2797 }%
2798 \def\XINT_FL_fac_fork_b #1.%
2799 {%
2800     \ifnum #1>\xint_c_x^viii_mone\xint_dothis\XINT_FL_fac_toobig\fi
2801     \ifnum #1>\xint_c_x^iv\xint_dothis\XINT_FL_fac_vbig \fi
2802     \ifnum #1>465 \xint_dothis\XINT_FL_fac_big\fi
2803     \ifnum #1>101 \xint_dothis\XINT_FL_fac_med\fi
2804     \xint_orthat\XINT_FL_fac_small
2805     #1.%
2806 }%
2807 \def\XINT_FL_fac_toobig #1.#2#3#4#5%
2808 {%
2809     #5{\XINT_signalcondition{InvalidOperation}
2810         {Factorial of too big: (#1)!}{0[0]}}%
2811 }%

```

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

We then want $\text{ceil}((P+k+n)/8)$. Using \numexpr rounding division (ARRRRRGGGHHHH), if m is a positive integer, $\text{ceil}(m/8)$ can be computed as $(m+3)/8$. Thus with $m=P+10+k$, this gives $Q \leftarrow (P+13+k)/8$. The routine actually computes $8(Q-1)$ for use in \XINT_FL_fac_addzeros.

With 1.2f the formula is $m=P+9+k$, $Q \leftarrow (P+12+k)/8$, and we use now $4=12-8$ rather than the earlier $5=13-8$. Whatever happens, the value computed in \XINT_FL_fac_increaseP is at least 8. There will always be an extra block.

Note: with Digits:=32; Maple gives for 200!:

```

> factorial(200.);
375
0.78865786736479050355236321393218 10

```


My 1.2f routine (and also 1.2) outputs:

7.8865786736479050355236321393219e374

and this is the correct rounding because for 40 digits it computes

7.886578673647905035523632139321850622951e374

Maple's result (contrarily to xint) is thus not the correct rounding but still it is less than 0.6 ulp wrong.

```

2812 \def\XINT_FL_fac_vbig
2813   {\expandafter\XINT_FL_fac_vbigloop_a
2814     \the\numexpr \XINT_FL_fac_increaseP \xint_c_i   }%
2815 \def\XINT_FL_fac_big
2816   {\expandafter\XINT_FL_fac_bigloop_a
2817     \the\numexpr \XINT_FL_fac_increaseP \xint_c_ii  }%
2818 \def\XINT_FL_fac_med
2819   {\expandafter\XINT_FL_fac_medloop_a
2820     \the\numexpr \XINT_FL_fac_increaseP \xint_c_iii }%
2821 \def\XINT_FL_fac_small
2822   {\expandafter\XINT_FL_fac_smallloop_a
2823     \the\numexpr \XINT_FL_fac_increaseP \xint_c_iv   }%
2824 \def\XINT_FL_fac_increaseP #1#2.#3#4%
2825 {%
2826   #2\expandafter.\the\numexpr\xint_c_viii*%
2827   ((\xint_c_iv+#4+\expandafter\XINT_FL_fac_countdigits
2828     \the\numexpr #2/((#1*#3)\relax 87654321\Z)/\xint_c_viii).)%
2829 }%
2830 \def\XINT_FL_fac_countdigits #1#2#3#4#5#6#7#8{\XINT_FL_fac_countdone }%
2831 \def\XINT_FL_fac_countdone  #1#2\Z {#1}%
2832 \def\XINT_FL_fac_out #1;![#2]#3%
2833   {#3{\romannumeral0\XINT_mul_out
2834     #1;!1\R!1\R!1\R!1\R!%
2835     1\R!1\R!1\R!1\R!\W [#2]}}%
2836 \def\XINT_FL_fac_vbigloop_a #1.#2.%
2837 {%
2838   \XINT_FL_fac_bigloop_a \xint_c_x^iv.#2.%
2839   {\expandafter\XINT_FL_fac_vbigloop_loop\the\numexpr 100010001\expandafter.%
2840     \the\numexpr \xint_c_x^viii+#1.}%
2841 }%
2842 \def\XINT_FL_fac_vbigloop_loop #1.#2.%
2843 {%
2844   \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2845   \expandafter\XINT_FL_fac_vbigloop_loop
2846   \the\numexpr #1+\xint_c_i\expandafter.%
2847   \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_mul #1!%
2848 }%
2849 \def\XINT_FL_fac_bigloop_a #1.%
2850 {%
2851   \expandafter\XINT_FL_fac_bigloop_b \the\numexpr
2852   #1+\xint_c_i-\xint_c_ii*((#1-464)/\xint_c_ii).#1.%
2853 }%
2854 \def\XINT_FL_fac_bigloop_b #1.#2.#3.%
2855 {%
2856   \expandafter\XINT_FL_fac_medloop_a
2857   \the\numexpr #1-\xint_c_i.#3.{\XINT_FL_fac_bigloop_loop #1.#2.}%

```

```

2858 }%
2859 \def\XINT_FL_fac_bigloop_loop #1.#2.%
2860 {%
2861     \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2862     \expandafter\XINT_FL_fac_bigloop_loop
2863     \the\numexpr #1+\xint_c_ii\expandafter.%
2864     \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_bigloop_mul #1!%
2865 }%
2866 \def\XINT_FL_fac_bigloop_mul #1!%
2867 {%
2868     \expandafter\XINT_FL_fac_mul
2869     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2870 }%
2871 \def\XINT_FL_fac_medloop_a #1.%
2872 {%
2873     \expandafter\XINT_FL_fac_medloop_b
2874     \the\numexpr #1+\xint_c_i-\xint_c_iii*((#1-100)/\xint_c_iii).#1.%
2875 }%
2876 \def\XINT_FL_fac_medloop_b #1.#2.#3.%
2877 {%
2878     \expandafter\XINT_FL_fac_smallloop_a
2879     \the\numexpr #1-\xint_c_i.#3.{\XINT_FL_fac_medloop_loop #1.#2.}%
2880 }%
2881 \def\XINT_FL_fac_medloop_loop #1.#2.%
2882 {%
2883     \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2884     \expandafter\XINT_FL_fac_medloop_loop
2885     \the\numexpr #1+\xint_c_iii\expandafter.%
2886     \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_medloop_mul #1!%
2887 }%
2888 \def\XINT_FL_fac_medloop_mul #1!%
2889 {%
2890     \expandafter\XINT_FL_fac_mul
2891     \the\numexpr
2892         \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2893 }%
2894 \def\XINT_FL_fac_smallloop_a #1.%
2895 {%
2896     \csname
2897         XINT_FL_fac_smallloop_\the\numexpr #1-\xint_c_iv*(#1/\xint_c_iv)\relax
2898     \endcsname #1.%
2899 }%
2900 \expandafter\def\csname XINT_FL_fac_smallloop_1\endcsname #1.#2.%
2901 {%
2902     \XINT_FL_fac_addzeros #2.100000001!.{2.#1.}{#2}%
2903 }%
2904 \expandafter\def\csname XINT_FL_fac_smallloop_2\endcsname #1.#2.%
2905 {%
2906     \XINT_FL_fac_addzeros #2.100000002!.{3.#1.}{#2}%
2907 }%
2908 \expandafter\def\csname XINT_FL_fac_smallloop_1\endcsname #1.#2.%
2909 {%

```

```

2910 \XINT_FL_fac_addzeros #2.1000000006!.{4.#1.}{#2}%
2911 }%
2912 \expandafter\def\csname XINT_FL_fac_smallloop_0\endcsname #1.#2.%
2913 {%
2914 \XINT_FL_fac_addzeros #2.1000000024!.{5.#1.}{#2}%
2915 }%
2916 \def\XINT_FL_fac_addzeros #1.%
2917 {%
2918 \ifnum #1=\xint_c_viii \expandafter\XINT_FL_fac_addzeros_exit\fi
2919 \expandafter\XINT_FL_fac_addzeros
2920 \the\numexpr #1-\xint_c_viii.1000000000!%
2921 }%

```

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.

```

2922 \def\XINT_FL_fac_addzeros_exit #1.#2.#3#4{\XINT_FL_fac_smallloop_loop #3#21;![ -#4]}%
2923 \def\XINT_FL_fac_smallloop_loop #1.#2.%
2924 {%
2925 \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2926 \expandafter\XINT_FL_fac_smallloop_loop
2927 \the\numexpr #1+\xint_c_iv\expandafter.%
2928 \the\numexpr #2\expandafter.\romannumeral0\XINT_FL_fac_smallloop_mul #1!%
2929 }%
2930 \def\XINT_FL_fac_smallloop_mul #1!%
2931 {%
2932 \expandafter\XINT_FL_fac_mul
2933 \the\numexpr
2934 \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2935 }%[[
2936 \def\XINT_FL_fac_loop_exit #1!#2]#3{#3#2]}%
2937 \def\XINT_FL_fac_mul 1#1!%
2938 {\expandafter\XINT_FL_fac_mul_a\the\numexpr\XINT_FL_fac_smallmul 10!{#1}}%
2939 \def\XINT_FL_fac_mul_a #1-#2%
2940 {%
2941 \if#21\xint_afterfi{\expandafter\space\xint_gob_til_exclam}\else
2942 \expandafter\space\fi #11;!%
2943 }%
2944 \def\XINT_FL_fac_minimulwc_a #1#2#3#4#5!#6#7#8#9%
2945 {%
2946 \XINT_FL_fac_minimulwc_b {#1#2#3#4}{#5}{#6#7#8#9}%
2947 }%
2948 \def\XINT_FL_fac_minimulwc_b #1#2#3#4!#5%
2949 {%
2950 \expandafter\XINT_FL_fac_minimulwc_c
2951 \the\numexpr \xint_c_x^ix+#5+#2*#4!{#1}{#2}{#3}{#4}%
2952 }%
2953 \def\XINT_FL_fac_minimulwc_c 1#1#2#3#4#5#6!#7%
2954 {%
2955 \expandafter\XINT_FL_fac_minimulwc_d {#1#2#3#4#5}#7{#6}%
2956 }%
2957 \def\XINT_FL_fac_minimulwc_d #1#2#3#4#5%

```

```

2958 {%
2959   \expandafter\XINT_FL_fac_minimulwc_e
2960   \the\numexpr \xint_c_x^ix+#1+#2*#5+#3*#4!{#2}{#4}%
2961 }%
2962 \def\XINT_FL_fac_minimulwc_e 1#1#2#3#4#5#6!#7#8#9%
2963 {%
2964   1#6#9\expandafter!%
2965   \the\numexpr\expandafter\XINT_FL_fac_smallmul
2966   \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#7*#8!%
2967 }%
2968 \def\XINT_FL_fac_smallmul 1#1!#21#3!%
2969 {%
2970   \xint_gob_til_sc #3\XINT_FL_fac_smallmul_end;%
2971   \XINT_FL_fac_minimulwc_a #2!#3!{#1}{#2}%
2972 }%

```

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.)

```

2973 \def\XINT_FL_fac_smallmul_end;\XINT_FL_fac_minimulwc_a #1!;!#2#3[#4]%
2974 {%
2975   \ifnum #2=\xint_c_
2976     \expandafter\xint_firstoftwo\else
2977     \expandafter\xint_secondoftwo
2978   \fi
2979   {-2\relax[#4]}%
2980   {1#2\expandafter!\expandafter-\expandafter1\expandafter
2981     [\the\numexpr #4+\xint_c_viii]}%
2982 }%

```

8.81 `\xintFloatPFactorial`, `\XINTinFloatPFactorial`

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

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

```

2983 \def\xintFloatPFactorial {\romannumeral0\xintfloatpfactorial}%
2984 \def\xintfloatpfactorial #1{\XINT_flpfac_chkopt \xintfloat #1\xint:}%
2985 \def\XINTinFloatPFactorial {\romannumeral0\XINTinfloatpfactorial}%
2986 \def\XINTinfloatpfactorial #1{\XINT_flpfac_chkopt \XINTinfloat #1\xint:}%
2987 \def\XINT_flpfac_chkopt #1#2%
2988 {%
2989   \ifx [#2\expandafter\XINT_flpfac_opt
2990     \else\expandafter\XINT_flpfac_noopt
2991   \fi
2992   #1#2%
2993 }%
2994 \def\XINT_flpfac_noopt #1#2\xint:#3%
2995 {%
2996   \expandafter\XINT_FL_pfac_fork

```

```

2997 \the\numexpr \xintNum{#2}\expandafter.%
2998 \the\numexpr \xintNum{#3}.\xint_c_i{\XINTdigits}{#1[\XINTdigits]]}%
2999 }%
3000 \def\XINT_flpfac_opt #1[\xint:#2]%
3001 {%
3002 \expandafter\XINT_flpfac_opt_b\the\numexpr #2.#1%
3003 }%
3004 \def\XINT_flpfac_opt_b #1.#2#3#4%
3005 {%
3006 \expandafter\XINT_FL_pfac_fork
3007 \the\numexpr \xintNum{#3}\expandafter.%
3008 \the\numexpr \xintNum{#4}.\xint_c_i{#1}{#2[#1]]}%
3009 }%
3010 \def\XINT_FL_pfac_fork #1#2.#3#4.%
3011 {%
3012 \unless\ifnum #1#2<#3#4 \xint_dothis\XINT_FL_pfac_one\fi
3013 \if-#3\xint_dothis\XINT_FL_pfac_neg \fi
3014 \if-#1\xint_dothis\XINT_FL_pfac_zero\fi
3015 \ifnum #3#4>\xint_c_x^viii_mone\xint_dothis\XINT_FL_pfac_outofrange\fi
3016 \xint_orthat \XINT_FL_pfac_increaseP #1#2.#3#4.%
3017 }%
3018 \def\XINT_FL_pfac_outofrange #1.#2.#3#4#5%
3019 {%
3020 #5{\XINT_signalcondition{InvalidOperation}
3021 {pfactorial second arg too big: 99999999 < #2}{0[0]}}%
3022 }%
3023 \def\XINT_FL_pfac_one #1.#2.#3#4#5{#5{1[0]}}%
3024 \def\XINT_FL_pfac_zero #1.#2.#3#4#5{#5{0[0]}}%
3025 \def\XINT_FL_pfac_neg -#1.-#2.%
3026 {%
3027 \ifnum #1>\xint_c_x^viii\xint_dothis\XINT_FL_pfac_outofrange\fi
3028 \xint_orthat {%
3029 \ifodd\numexpr#2-#1\relax\xint_afterfi{\expandafter-\romannumeral`&&@}\fi
3030 \expandafter\XINT_FL_pfac_increaseP}%
3031 \the\numexpr #2-\xint_c_i\expandafter.\the\numexpr#1-\xint_c_i.%
3032 }%

```

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.

```

3033 \def\XINT_FL_pfac_increaseP #1.#2.#3#4%
3034 {%
3035 \expandafter\XINT_FL_pfac_a
3036 \the\numexpr \xint_c_viii*((\xint_c_iv+#4+\expandafter
3037 \XINT_FL_fac_countdigits\the\numexpr (#2-#1-\xint_c_i)%
3038 /\ifnum #2>\xint_c_x^iv #3\else(#3*\xint_c_ii)\fi\relax
3039 87654321\Z)/\xint_c_viii).#1.#2.%
3040 }%

```

```

3041 \def\XINT_FL_pfac_a #1.#2.#3.%
3042 {%
3043     \expandafter\XINT_FL_pfac_b\the\numexpr \xint_c_i+#2\expandafter.%
3044     \the\numexpr#3\expandafter.%
3045     \romannumeral0\XINT_FL_pfac_addzeroes #1.1000000001!1;![-#1]%
3046 }%
3047 \def\XINT_FL_pfac_addzeroes #1.%
3048 {%
3049     \ifnum #1=\xint_c_viii \expandafter\XINT_FL_pfac_addzeroes_exit\fi
3050     \expandafter\XINT_FL_pfac_addzeroes\the\numexpr #1-\xint_c_viii.100000000!%
3051 }%
3052 \def\XINT_FL_pfac_addzeroes_exit #1.{ }%
3053 \def\XINT_FL_pfac_b #1.%
3054 {%
3055     \ifnum #1>9999 \xint_dothis\XINT_FL_pfac_vbigloop \fi
3056     \ifnum #1>463 \xint_dothis\XINT_FL_pfac_bigloop \fi
3057     \ifnum #1>98 \xint_dothis\XINT_FL_pfac_medloop \fi
3058     \xint_orthat\XINT_FL_pfac_smallloop #1.%
3059 }%
3060 \def\XINT_FL_pfac_smallloop #1.#2.%
3061 {%
3062     \ifcase\numexpr #2-#1\relax
3063         \expandafter\XINT_FL_pfac_end_
3064     \or \expandafter\XINT_FL_pfac_end_i
3065     \or \expandafter\XINT_FL_pfac_end_ii
3066     \or \expandafter\XINT_FL_pfac_end_iii
3067     \else\expandafter\XINT_FL_pfac_smallloop_a
3068     \fi #1.#2.%
3069 }%
3070 \def\XINT_FL_pfac_smallloop_a #1.#2.%
3071 {%
3072     \expandafter\XINT_FL_pfac_smallloop_b
3073     \the\numexpr #1+\xint_c_iv\expandafter.%
3074     \the\numexpr #2\expandafter.%
3075     \romannumeral0\expandafter\XINT_FL_fac_mul
3076     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
3077 }%
3078 \def\XINT_FL_pfac_smallloop_b #1.%
3079 {%
3080     \ifnum #1>98 \expandafter\XINT_FL_pfac_medloop \else
3081         \expandafter\XINT_FL_pfac_smallloop \fi #1.%
3082 }%
3083 \def\XINT_FL_pfac_medloop #1.#2.%
3084 {%
3085     \ifcase\numexpr #2-#1\relax
3086         \expandafter\XINT_FL_pfac_end_
3087     \or \expandafter\XINT_FL_pfac_end_i
3088     \or \expandafter\XINT_FL_pfac_end_ii
3089     \else\expandafter\XINT_FL_pfac_medloop_a
3090     \fi #1.#2.%
3091 }%
3092 \def\XINT_FL_pfac_medloop_a #1.#2.%

```

```

3093 {%
3094   \expandafter\XINT_FL_pfac_medloop_b
3095   \the\numexpr #1+\xint_c_iii\expandafter.%
3096   \the\numexpr #2\expandafter.%
3097   \romannumeral0\expandafter\XINT_FL_fac_mul
3098   \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
3099 }%
3100 \def\XINT_FL_pfac_medloop_b #1.%
3101 {%
3102   \ifnum #1>463 \expandafter\XINT_FL_pfac_bigloop \else
3103               \expandafter\XINT_FL_pfac_medloop \fi #1.%
3104 }%
3105 \def\XINT_FL_pfac_bigloop #1.#2.%
3106 {%
3107   \ifcase\numexpr #2-#1\relax
3108     \expandafter\XINT_FL_pfac_end_
3109   \or \expandafter\XINT_FL_pfac_end_i
3110   \else\expandafter\XINT_FL_pfac_bigloop_a
3111   \fi #1.#2.%
3112 }%
3113 \def\XINT_FL_pfac_bigloop_a #1.#2.%
3114 {%
3115   \expandafter\XINT_FL_pfac_bigloop_b
3116   \the\numexpr #1+\xint_c_ii\expandafter.%
3117   \the\numexpr #2\expandafter.%
3118   \romannumeral0\expandafter\XINT_FL_fac_mul
3119   \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
3120 }%
3121 \def\XINT_FL_pfac_bigloop_b #1.%
3122 {%
3123   \ifnum #1>9999 \expandafter\XINT_FL_pfac_vbigloop \else
3124               \expandafter\XINT_FL_pfac_bigloop \fi #1.%
3125 }%
3126 \def\XINT_FL_pfac_vbigloop #1.#2.%
3127 {%
3128   \ifnum #2=#1
3129     \expandafter\XINT_FL_pfac_end_
3130   \else\expandafter\XINT_FL_pfac_vbigloop_a
3131   \fi #1.#2.%
3132 }%
3133 \def\XINT_FL_pfac_vbigloop_a #1.#2.%
3134 {%
3135   \expandafter\XINT_FL_pfac_vbigloop
3136   \the\numexpr #1+\xint_c_i\expandafter.%
3137   \the\numexpr #2\expandafter.%
3138   \romannumeral0\expandafter\XINT_FL_fac_mul
3139   \the\numexpr\xint_c_x^viii+#1!%
3140 }%
3141 \def\XINT_FL_pfac_end_iii #1.#2.%
3142 {%
3143   \expandafter\XINT_FL_fac_out
3144   \romannumeral0\expandafter\XINT_FL_fac_mul

```

```

3145 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
3146 }%
3147 \def\XINT_FL_pfac_end_ii #1.#2.%
3148 {%
3149 \expandafter\XINT_FL_fac_out
3150 \romannumeral0\expandafter\XINT_FL_fac_mul
3151 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
3152 }%
3153 \def\XINT_FL_pfac_end_i #1.#2.%
3154 {%
3155 \expandafter\XINT_FL_fac_out
3156 \romannumeral0\expandafter\XINT_FL_fac_mul
3157 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
3158 }%
3159 \def\XINT_FL_pfac_end_ #1.#2.%
3160 {%
3161 \expandafter\XINT_FL_fac_out
3162 \romannumeral0\expandafter\XINT_FL_fac_mul
3163 \the\numexpr \xint_c_x^viii+#1!%
3164 }%

```

8.82 \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.

```

3165 \def\xintFloatBinomial {\romannumeral0\xintfloatbinomial}%
3166 \def\xintfloatbinomial #1{\XINT_flbinom_chkopt \xintfloat #1\xint:}%
3167 \def\XINTinFloatBinomial {\romannumeral0\XINTinfloatbinomial}%
3168 \def\XINTinfloatbinomial #1{\XINT_flbinom_chkopt \XINTinfloat #1\xint:}%
3169 \def\XINT_flbinom_chkopt #1#2%
3170 {%
3171 \ifx [#2\expandafter\XINT_flbinom_opt
3172 \else\expandafter\XINT_flbinom_noopt
3173 \fi #1#2%
3174 }%
3175 \def\XINT_flbinom_noopt #1#2\xint:#3%
3176 {%
3177 \expandafter\XINT_FL_binom_a
3178 \the\numexpr\xintNum{#2}\expandafter.\the\numexpr\xintNum{#3}.\XINTdigits.#1%
3179 }%
3180 \def\XINT_flbinom_opt #1[\xint:#2]#3#4%
3181 {%
3182 \expandafter\XINT_FL_binom_a
3183 \the\numexpr\xintNum{#3}\expandafter.\the\numexpr\xintNum{#4}\expandafter.%
3184 \the\numexpr #2.#1%
3185 }%

```



```

3186 \def\XINT_FL_binom_a #1.#2.%
3187 {%
3188     \expandafter\XINT_FL_binom_fork \the\numexpr #1-#2.#2.#1.%
3189 }%
3190 \def\XINT_FL_binom_fork #1#2.#3#4.#5#6.%
3191 {%
3192     \if-#5\xint_dothis \XINT_FL_binom_neg\fi
3193     \if-#1\xint_dothis \XINT_FL_binom_zero\fi
3194     \if-#3\xint_dothis \XINT_FL_binom_zero\fi
3195     \if0#1\xint_dothis \XINT_FL_binom_one\fi
3196     \if0#3\xint_dothis \XINT_FL_binom_one\fi
3197     \ifnum #5#6>\xint_c_x^viii_mone \xint_dothis\XINT_FL_binom_toobig\fi
3198     \ifnum #1#2>#3#4 \xint_dothis\XINT_FL_binom_ab \fi
3199     \xint_orthat\XINT_FL_binom_aa
3200     #1#2.#3#4.#5#6.%
3201 }%
3202 \def\XINT_FL_binom_neg #1.#2.#3.#4.#5%
3203 {%
3204     #5[#4]{\XINT_signalcondition{InvalidOperation}
3205             {binomial with first arg negative: #3}}{\0[0]}}%
3206 }%
3207 \def\XINT_FL_binom_toobig #1.#2.#3.#4.#5%
3208 {%
3209     #5[#4]{\XINT_signalcondition{InvalidOperation}
3210             {binomial with first arg too big: 99999999 < #3}}{\0[0]}}%
3211 }%
3212 \def\XINT_FL_binom_one #1.#2.#3.#4.#5{#5[#4]{1[0]}}%
3213 \def\XINT_FL_binom_zero #1.#2.#3.#4.#5{#5[#4]{0[0]}}%
3214 \def\XINT_FL_binom_aa #1.#2.#3.#4.#5%
3215 {%
3216     #5[#4]{\xintDiv{\XINT_FL_pfac_increaseP
3217                 #2.#3.\xint_c_iv{#4+\xint_c_i}{\XINTinfloat[#4+\xint_c_iii]}}%
3218             {\XINT_FL_fac_fork_b
3219             #1.\xint_c_iv{#4+\xint_c_i}\XINT_FL_fac_out{\XINTinfloat[#4+\xint_c_iii]}}}%
3220 }%
3221 \def\XINT_FL_binom_ab #1.#2.#3.#4.#5%
3222 {%
3223     #5[#4]{\xintDiv{\XINT_FL_pfac_increaseP
3224                 #1.#3.\xint_c_iv{#4+\xint_c_i}{\XINTinfloat[#4+\xint_c_iii]}}%
3225             {\XINT_FL_fac_fork_b
3226             #2.\xint_c_iv{#4+\xint_c_i}\XINT_FL_fac_out{\XINTinfloat[#4+\xint_c_iii]}}}%
3227 }%

```

8.83 \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.

```

3228 \def\xintFloatSqrt      {\romannumeral0\xintfloatsqrt }%
3229 \def\xintfloatsqrt      #1{\XINT_flsqrt_chkopt \xintfloat #1\xint:}%
3230 \def\XINTinFloatSqrt    {\romannumeral0\XINTinfloatsqrt }%
3231 \def\XINTinfloatsqrt    #1{\XINT_flsqrt_chkopt \XINTinfloat #1\xint:}%
3232 \def\XINT_flsqrt_chkopt #1#2%
3233 {%
3234   \ifx [#2\expandafter\XINT_flsqrt_opt
3235     \else\expandafter\XINT_flsqrt_noopt
3236   \fi #1#2%
3237 }%
3238 \def\XINT_flsqrt_noopt #1#2\xint:%
3239 {%
3240   \expandafter\XINT_FL_sqrt_a
3241     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.#1%
3242 }%
3243 \def\XINT_flsqrt_opt #1[\xint:#2]%#3%
3244 {%
3245   \expandafter\XINT_flsqrt_opt_a\the\numexpr #2.#1%
3246 }%
3247 \def\XINT_flsqrt_opt_a #1.#2#3%
3248 {%
3249   \expandafter\XINT_FL_sqrt_a\romannumeral0\XINTinfloat[#1]{#3}#1.#2%
3250 }%
3251 \def\XINT_FL_sqrt_a #1%
3252 {%
3253   \xint_UDzerominusfork
3254   #1-\XINT_FL_sqrt_iszero
3255   0#1\XINT_FL_sqrt_isneg
3256   0-\XINT_FL_sqrt_pos #1}%
3257 \krof
3258 }%[
3259 \def\XINT_FL_sqrt_iszero #1]#2.#3{#3[#2]{0[0]}}%
3260 \def\XINT_FL_sqrt_isneg #1]#2.#3%
3261 {%
3262   #3[#2]{\XINT_signalcondition{InvalidOperation}
3263     {Square root of negative: -#1]}}{0[0]}}%
3264 }%

3265 \def\XINT_FL_sqrt_pos #1[#2]#3.%
3266 {%
3267   \expandafter\XINT_flsqrt
3268   \the\numexpr #3\ifodd #2 \xint_dothis {+\xint_c_iii.(#2+\xint_c_i).0}\fi
3269   \xint_orthat {+\xint_c_ii.#2.}}#100.#3.%
3270 }%

3271 \def\XINT_flsqrt #1.#2.%

```

```

3272 {%
3273     \expandafter\XINT_flsqrt_a
3274     \the\numexpr #2/\xint_c_ii-(#1-\xint_c_i)/\xint_c_ii.#1.%
3275 }%

3276 \def\XINT_flsqrt_a #1.#2.#3#4.#5.%
3277 {%
3278     \expandafter\XINT_flsqrt_b
3279     \the\numexpr (#2-\xint_c_i)/\xint_c_ii\expandafter.%
3280     \romannumeral0\XINT_sqrt_start #2.#4#3.#5.#2.#4#3.#5.#1.%
3281 }%

3282 \def\XINT_flsqrt_b #1.#2#3%
3283 {%
3284     \expandafter\XINT_flsqrt_c
3285     \romannumeral0\xintiisub
3286     {\XINT_dsx_addzeros {#1}#2;}%
3287     {\xintiiDivRound{\XINT_dsx_addzeros {#1}#3;}%
3288         {\XINT_dbl#2\xint_bye2345678\xint_bye*\xint_c_ii\relax}}.%
3289 }%

3290 \def\XINT_flsqrt_c #1.#2.%
3291 {%
3292     \expandafter\XINT_flsqrt_d
3293     \romannumeral0\XINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..%
3294 }%

3295 \def\XINT_flsqrt_d #1.#2#3.%
3296 {%
3297     \ifnum #2=\xint_c_v
3298     \expandafter\XINT_flsqrt_f\else\expandafter\XINT_flsqrt_finish\fi
3299     #2#3.#1.%
3300 }%

3301 \def\XINT_flsqrt_finish #1#2.#3.#4.#5.#6.#7.#8{#8[#6]{#3#1[#7]}}%

3302 \def\XINT_flsqrt_f 5#1.%
3303     {\expandafter\XINT_flsqrt_g\romannumeral0\xintinum{#1}\relax.}%
3304 \def\XINT_flsqrt_g #1#2#3.{\if\relax#2\xint_dothis{\XINT_flsqrt_h #1}\fi
3305     \xint_orthat{\XINT_flsqrt_finish 5.}}%
3306 \def\XINT_flsqrt_h #1{\ifnum #1<\xint_c_iii\xint_dothis{\XINT_flsqrt_again}\fi
3307     \xint_orthat{\XINT_flsqrt_finish 5.}}%

3308 \def\XINT_flsqrt_again #1.#2.%
3309 {%
3310     \expandafter\XINT_flsqrt_again_a\the\numexpr #2+\xint_c_viii.%
3311 }%

```

```

3312 \def\XINT_flsqrt_again_a #1.#2.#3.%
3313 {%
3314     \expandafter\XINT_flsqrt_b
3315     \the\numexpr (#1-\xint_c_i)/\xint_c_ii\expandafter.%
3316     \romannumeral0\XINT_sqrt_start #1.#2000000000.#3.%
3317                                     #1.#2000000000.#3.%
3318 }%

```

8.84 \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].

```

3319 \def\xintFloatE    {\romannumeral0\xintfloate }%
3320 \def\xintfloate #1{\XINT_floate_chkopt #1\xint:}%
3321 \def\XINT_floate_chkopt #1%
3322 {%
3323     \ifx [#1\expandafter\XINT_floate_opt
3324         \else\expandafter\XINT_floate_noopt
3325     \fi #1%
3326 }%
3327 \def\XINT_floate_noopt #1\xint:%
3328 {%
3329     \expandafter\XINT_floate_post
3330     \romannumeral0\XINTinfloat[\XINTdigits]{#1}\XINTdigits.%
3331 }%
3332 \def\XINT_floate_opt [\xint:#1]%
3333 {%
3334     \expandafter\XINT_floate_opt_a\the\numexpr #1.%
3335 }%
3336 \def\XINT_floate_opt_a #1.#2%
3337 {%
3338     \expandafter\XINT_floate_post
3339     \romannumeral0\XINTinfloat[#1]{#2}#1.%
3340 }%
3341 \def\XINT_floate_post #1%
3342 {%
3343     \xint_UDzerominusfork
3344     #1-\XINT_floate_zero
3345     0#1\XINT_floate_neg
3346     0-\XINT_floate_pos
3347     \krof #1%
3348 }%[
3349 \def\XINT_floate_zero #1]#2.#3{ 0.e0}%
3350 \def\XINT_floate_neg-{\expandafter-\romannumeral0\XINT_floate_pos}%

3351 \def\XINT_floate_pos #1#2[#3]#4.#5%
3352 {%
3353     \expandafter\XINT_float_pos_done\the\numexpr#3+#4+#5-\xint_c_i.#1.#2;%
3354 }%

```

```

3355 \def\XINTinFloatE {\romannumeral0\XINTinfloate }%
3356 \def\XINTinfloate
3357   {\expandafter\XINT_infloate\romannumeral0\XINTinfloat[\XINTdigits]]}%
3358 \def\XINT_infloate #1[#2]#3%
3359   {\expandafter\XINT_infloate_end\the\numexpr #3+#2.{#1}}%
3360 \def\XINT_infloate_end #1.#2{ #2[#1]}%

```

8.85 \XINTinFloatMod

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

```

3361 \def\XINTinFloatMod {\romannumeral0\XINTinfloatmod [\XINTdigits]]}%
3362 \def\XINTinfloatmod [#1]#2#3%
3363 {%
3364   \XINTinfloat[#1]{\xintMod
3365     {\romannumeral0\XINTinfloat[#1]{#2}}}%
3366     {\romannumeral0\XINTinfloat[#1]{#3}}}%
3367 }%

```

8.86 \XINTinFloatDivFloor

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

```

3368 \def\XINTinFloatDivFloor {\romannumeral0\XINTinfloatdivfloor [\XINTdigits]]}%
3369 \def\XINTinfloatdivfloor [#1]#2#3%
3370 {%
3371   \xintdivfloor
3372     {\romannumeral0\XINTinfloat[#1]{#2}}}%
3373     {\romannumeral0\XINTinfloat[#1]{#3}}}%
3374 }%

```

8.87 \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.

Breaking change at 1.4 as output format is not comma separated anymore. Attention also that it uses \expanded.

No time now at the time of completion of the big 1.4 rewrite of xintexpr to test whether code efficiency here can be improved to expand the second item of output.

```

3375 \def\XINTinFloatDivMod {\romannumeral0\XINTinfloatdivmod [\XINTdigits]]}%
3376 \def\XINTinfloatdivmod [#1]#2#3%
3377 {%
3378   \expandafter\XINT_infloatdivmod
3379   \romannumeral0\xintdivmod
3380     {\romannumeral0\XINTinfloat[#1]{#2}}}%
3381     {\romannumeral0\XINTinfloat[#1]{#3}}}%
3382   {#1}%
3383 }%
3384 \def\XINT_infloatdivmod #1#2#3{\expanded{{#1}{\XINTinFloat[#3]{#2}}}}}%

```

8.88 \xintifFloatInt

1.3a for ifint() function in \xintfloatexpr.

```
3385 \def\xintifFloatInt {\romannumeral0\xintiffloatint}%
3386 \def\xintiffloatint #1{\expandafter\XINT_iffloatint
3387     \romannumeral0\xintrez{\XINTinFloat[\XINTdigits]{#1}}}%
3388 \def\XINT_iffloatint #1#2/1[#3]%
3389 {%
3390   \if 0#1\xint_dothis\xint_stop_atfirstoftwo\fi
3391   \ifnum#3<\xint_c_\xint_dothis\xint_stop_atsecondoftwo\fi
3392   \xint_orthat\xint_stop_atfirstoftwo
3393 }%
```

8.89 \xintFloatIsInt

1.3d for isint() function in \xintfloatexpr.

```
3394 \def\xintFloatIsInt {\romannumeral0\xintfloatisint}%
3395 \def\xintfloatisint #1{\expandafter\XINT_iffloatint
3396     \romannumeral0\xintrez{\XINTinFloat[\XINTdigits]{#1}}10}%

```

8.90 \XINTinFloatdigits, \XINTinFloatSqrtdigits, \XINTinFloatFacdigits, \XINTiLogTendigits

For \xintNewExpr matters, mainly.

At 1.3e I add \XINTinFloatSdigits and use it at various places. I also modified \XINTinFloatFac to use S(hort) output format.

Also added \XINTiLogTendigits.

This whole stuff moved over from xintexpr.sty at 1.4

```
3397 \def\XINTinFloatdigits {\XINTinFloat [\XINTdigits]]}%
3398 \def\XINTinFloatSdigits {\XINTinFloatS [\XINTdigits]]}%
3399 \def\XINTinFloatSqrtdigits {\XINTinFloatSqrt[\XINTdigits]]}%
3400 \def\XINTinFloatFacdigits {\XINTinFloatFac [\XINTdigits]]}%
3401 \def\XINTFloatiLogTendigits{\XINTFloatiLogTen[\XINTdigits]]}%

```

8.91 (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.

```

3402 \def\XINTinRandomFloatS{\romannumeral0\XINTinrandomfloatS}%
3403 \def\XINTinRandomFloatSdigits{\XINTinRandomFloatS[\XINTdigits]}%
3404 \def\XINTinrandomfloatS[#1]%
3405 {%
3406   \expandafter\XINT_inrandomfloatS\the\numexpr\xint_c_-(#1)\xint:
3407 }%
3408 \def\XINT_inrandomfloatS-#1\xint:
3409 {%
3410   \expandafter\XINT_inrandomfloatS_a
3411   \romannumeral0\xintrandomdigits{#1}[-#1]%
3412 }%

```

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.

```

3413 \def\XINT_inrandomfloatS_a#1%
3414 {%
3415   \if#10\xint_dothis{\XINT_inrandomfloatS_b}\fi
3416   \xint_orthat{ #1}%
3417 }%[
3418 \def\XINT_inrandomfloatS_b#1%
3419 {%
3420   \if#1[\xint_dothis{\XINT_inrandomfloatS_zero}\fi% ]
3421   \if#10\xint_dothis{\XINT_inrandomfloatS_b}\fi
3422   \xint_orthat{ #1}%
3423 }%[
3424 \def\XINT_inrandomfloatS_zero#1{ 0[0]}%

```

8.92 (WIP) \XINTinRandomFloatSixteen

1.3b. Support for grand() function.

```

3425 \def\XINTinRandomFloatSixteen%
3426 {%
3427   \romannumeral0\expandafter\XINT_inrandomfloatS_a
3428   \romannumeral`&&\expandafter\XINT_eightrandomdigits
3429   \romannumeral`&&\XINT_eightrandomdigits[-16]%
3430 }%

```

8.93 \PoorManLogBaseTen

1.3f. Code originally in poormanlog v0.4 got transferred here. It produces the logarithm in base 10 with an error (believed to be at most) about 1 unit in the 9th (i.e. last) fractional digit. Testing seems to indicate error at most 2 units.

```

3431 \def\PoorManLogBaseTen{\romannumeral0\poormanlogbaseten}%
3432 \def\poormanlogbaseten #1%
3433   {\expandafter\PML@logbaseten\romannumeral0\XINTinfloat[9]{#1}}%
3434 \def\PML@logbaseten#1[#2]%
3435 {%

```

```

3436 \xintiiadd{\xintDSx{-9}}{\the\numexpr#2+8\relax}}{\the\numexpr\PML@#1.}%
3437 [-9]%
3438 }%

```

8.94 \PoorManPowerOfTen

1.3f. Transferred from poormanlog v0.4. Produces the $10^{\#1}$ with 9 digits of float precision, with an error (believed to be) at most 2 units in the last place. Of course for this the input must be precise enough to have 9 fractional digits of **fixed point** precision.

Attention that this breaks with low level Number too big error if integral part of argument exceeds TeX bound on integers. Indeed some \numexpr is used in the code to subtract 8... but anyway xintfrac allows for scientific exponents only integers within TeX bounds, so even if it did not break here it would quickly elsewhere.

```

3439 \def\PoorManPowerOfTen{\the\numexpr\poormanpoweroften}%
3440 \def\poormanpoweroften #1%
3441   {\expandafter\PML@powoften\romannumeral0\xintra{#1}}%
3442 \def\PML@powoften#1%
3443 {%
3444   \xint_UDzerominusfork
3445   #1-\PML@powoften@zero
3446   0#1\PML@powoften@neg
3447   0-\PML@powoften@pos
3448   \krof #1%
3449 }%
3450 \def\PML@powoften@zero 0{1\relax}%/1[0]
3451 \def\PML@powoften@pos#1[#2]%
3452 {%
3453   \expandafter\PML@powoften@pos@a\romannumeral0\xintround{9}{#1[#2]}.%
3454 }%
3455 \def\PML@powoften@pos@a#1.#2.{\PML@Pa#2.\expandafter[\the\numexpr-8+#1]}%
3456 \def\PML@powoften@neg#1[#2]%
3457 {%
3458   \expandafter\PML@powoften@neg@a\romannumeral0\xintround{9}{#1[#2]}.%
3459 }%
3460 \def\PML@powoften@neg@a#1.#2.%
3461 {\ifnum#2=\xint_c_ \xint_afterfi{1\relax/1[#1]}\else
3462   \expandafter\expandafter\expandafter
3463   \PML@Pa\expandafter\xint_gobble_i\the\numexpr20000000000-#2.%
3464   \expandafter[\the\numexpr-9+#1\expandafter]\fi
3465 }%

```

8.95 \PoorManPower

1.3f. This code originally in poormanlog v0.4 transferred here. It does #1 to the power #2.

```

3466 \def\PoorManPower#1#2%
3467 {%
3468   \PoorManPowerOfTen{\xintMul{#2}{\PoorManLogBaseTen{#1}}}%
3469 }%

```


8.96 Support macros for natural logarithm and exponential [xintexpr](#) functions

At 1.3f, the poormanlog v0.04 extension to xintfrac.sty got transferred here. These macros from xintlog.sty 1.3e got transferred here too.

```

3470 \def\xintLog#1{\xintMul{\PoorManLogBaseTen{#1}}{23025850923[-10]}}%
3471 \def\XINTinFloatLog#1{\XINTinFloatMul{\PoorManLogBaseTen{#1}}{23025850923[-10]}}%
3472 \def\xintExp#1{\PoorManPowerOfTen{\xintMul{#1}{434294481903[-12]}}}%
3473 \def\XINTinFloatExp#1{\PoorManPowerOfTen{\XINTinFloatMul{#1}{434294481903[-12]}}}%
3474 \let\XINTinFloatMaxof\XINT_Maxof
3475 \let\XINTinFloatMinof\XINT_Minof
3476 \let\XINTinFloatSum\XINT_Sum
3477 \let\XINTinFloatPrd\XINT_Prd
3478 \XINT_restorecatcodes_endinput%
```

9 Package **xintseries** implementation

.1	Catcodes, ε -TeX and reload detection . .	274	.7	\xintRationalSeries	277
.2	Package identification	275	.8	\xintRationalSeriesX	278
.3	\xintSeries	275	.9	\xintFxpPtPowerSeries	279
.4	\xintiSeries	275	.10	\xintFxpPtPowerSeriesX	280
.5	\xintPowerSeries	276	.11	\xintFloatPowerSeries	280
.6	\xintPowerSeriesX	277	.12	\xintFloatPowerSeriesX	282

The commenting is currently (2020/02/19) very sparse.

9.1 Catcodes, ε -TeX and reload detection

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

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2 \catcode13=5      % ^^M
3 \endlinechar=13 %
4 \catcode123=1     % {
5 \catcode125=2     % }
6 \catcode64=11     % @
7 \catcode35=6      % #
8 \catcode44=12     % ,
9 \catcode45=12     % -
10 \catcode46=12    % .
11 \catcode58=12    % :
12 \let\z\endgroup
13 \expandafter\let\expandafter\x\csname ver@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 [2020/02/19 v1.4a 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 {%

```

```

128   \fi \XINT_powseries_exit_ii  #6{#7}%
129 }%
130 \def\XINT_powseries_exit_ii #1#2#3#4#5#6%
131 {%
132   \xintmul{\xintPow {#5}{#6}}{#4}%
133 }%

```

9.6 \xintPowerSeriesX

Same as `\xintPowerSeries` except for the initial expansion of the `x` parameter. Modified in 1.06 to give the indices first to a `\numexpr` rather than expanding twice. I just use `\the\numexpr` and maintain the previous code after that. 1.08a adds the forgotten optimization following that previous change.

```

134 \def\xintPowerSeriesX {\romannumeral0\xintpowerseriesx }%
135 \def\xintpowerseriesx #1#2%
136 {%
137   \expandafter\XINT_powseriesx\expandafter
138   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
139 }%
140 \def\XINT_powseriesx #1#2#3#4%
141 {%
142   \ifnum #2<#1
143     \xint_afterfi { 0/1[0]}%
144   \else
145     \xint_afterfi
146     {\expandafter\XINT_powseriesx_pre\expandafter
147      {\romannumeral`&&@#4}{#1}{#2}{#3}%
148     }%
149   \fi
150 }%
151 \def\XINT_powseriesx_pre #1#2#3#4%
152 {%
153   \XINT_powseries_loop_i {#4}{#3}{#2}{#3}{#4}{#1}%
154 }%

```

9.7 \xintRationalSeries

This computes $F(a) + \dots + F(b)$ on the basis of the value of $F(a)$ and the ratios $F(n)/F(n-1)$. As in `\xintPowerSeries` we use an iterative scheme which has the great advantage to avoid denominator build-up. This makes exact computations possible with exponential type series, which would be completely inaccessible to `\xintSeries`. #1=a, #2=b, #3=F(a), #4=ratio function Modified in 1.06 to give the indices first to a `\numexpr` rather than expanding twice. I just use `\the\numexpr` and maintain the previous code after that. 1.08a adds the forgotten optimization following that previous change.

```

155 \def\xintRationalSeries {\romannumeral0\xintratseries }%
156 \def\xintratseries #1#2%
157 {%
158   \expandafter\XINT_ratseries\expandafter
159   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
160 }%
161 \def\XINT_ratseries #1#2#3#4%

```

```

162 {%
163   \ifnum #2<#1
164     \xint_afterfi { 0/1[0]}%
165   \else
166     \xint_afterfi
167     {\XINT_ratseries_loop {#2}{1}{#1}{#4}{#3}}%
168   \fi
169 }%
170 \def\XINT_ratseries_loop #1#2#3#4%
171 {%
172   \ifnum #1>#3 \else\XINT_ratseries_exit_i\fi
173   \expandafter\XINT_ratseries_loop\expandafter
174   {\the\numexpr #1-1\expandafter}\expandafter
175   {\romannumeral0\xintadd {1}{\xintMul {#2}{#4{#1}}}}{#3}{#4}%
176 }%
177 \def\XINT_ratseries_exit_i\fi #1#2#3#4#5#6#7#8%
178 {%
179   \fi \XINT_ratseries_exit_ii #6%
180 }%
181 \def\XINT_ratseries_exit_ii #1#2#3#4#5%
182 {%
183   \XINT_ratseries_exit_iii #5%
184 }%
185 \def\XINT_ratseries_exit_iii #1#2#3#4%
186 {%
187   \xintmul{#2}{#4}%
188 }%

```

9.8 \xintRationalSeriesX

a,b,initial,ratiofunction,x

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

```

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

```

```

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

```

9.9 \xintFxpPowerSeries

I am not two happy with this piece of code. Will make it more economical another day. Modified in 1.06 to give the indices first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that. 1.08a: forgot last time some optimization from the change to \numexpr.

```

210 \def\xintFxpPowerSeries {\romannumeral0\xintfxptpowerseries }%
211 \def\xintfxptpowerseries #1#2%
212 {%
213 \expandafter\XINT_fppowseries\expandafter
214 {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
215 }%
216 \def\XINT_fppowseries #1#2#3#4#5%
217 {%
218 \ifnum #2<#1
219 \xint_afterfi { 0}%
220 \else
221 \xint_afterfi
222 {\expandafter\XINT_fppowseries_loop_pre\expandafter
223 {\romannumeral0\xinttrunc {#5}{\xintPow {#4}{#1}}}%
224 {#1}{#4}{#2}{#3}{#5}%
225 }%
226 \fi
227 }%
228 \def\XINT_fppowseries_loop_pre #1#2#3#4#5#6%
229 {%
230 \ifnum #4>#2 \else\XINT_fppowseries_dont_i \fi
231 \expandafter\XINT_fppowseries_loop_i\expandafter
232 {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
233 {\romannumeral0\xintitrunc {#6}{\xintMul {#5{#2}}{#1}}}%
234 {#1}{#3}{#4}{#5}{#6}%
235 }%
236 \def\XINT_fppowseries_dont_i \fi\expandafter\XINT_fppowseries_loop_i
237 {\fi \expandafter\XINT_fppowseries_dont_ii }%
238 \def\XINT_fppowseries_dont_ii #1#2#3#4#5#6#7{\xinttrunc {#7}{#2[-#7]}}%
239 \def\XINT_fppowseries_loop_i #1#2#3#4#5#6#7%
240 {%
241 \ifnum #5>#1 \else \XINT_fppowseries_exit_i \fi
242 \expandafter\XINT_fppowseries_loop_ii\expandafter
243 {\romannumeral0\xinttrunc {#7}{\xintMul {#3}{#4}}}%
244 {#1}{#4}{#2}{#5}{#6}{#7}%
245 }%
246 \def\XINT_fppowseries_loop_ii #1#2#3#4#5#6#7%
247 {%
248 \expandafter\XINT_fppowseries_loop_i\expandafter

```

```

249     {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
250     {\romannumeral0\xintiadd {#4}{\xintiTrunc {#7}{\xintMul {#6{#2}}{#1}}}%
251     {#1}{#3}{#5}{#6}{#7}%
252 }%
253 \def\XINT_fppowseries_exit_i\fi\expandafter\XINT_fppowseries_loop_ii
254     {\fi \expandafter\XINT_fppowseries_exit_ii }%
255 \def\XINT_fppowseries_exit_ii #1#2#3#4#5#6#7%
256 {%
257     \xinttrunc {#7}
258     {\xintiadd {#4}{\xintiTrunc {#7}{\xintMul {#6{#2}}{#1}}}{-#7}}%
259 }%

```

9.10 \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, ε -T _E X and reload detection . . .	283	.16	<code>\xintiGctoF</code>	295
.2	Package identification	284	.17	<code>\xintCtoCv</code> , <code>\xintCstoCv</code>	296
.3	<code>\xintCFrac</code>	284	.18	<code>\xintiCstoCv</code>	297
.4	<code>\xintGCfrac</code>	285	.19	<code>\xintGctoCv</code>	297
.5	<code>\xintGGCFrac</code>	287	.20	<code>\xintiGctoCv</code>	299
.6	<code>\xintGctoGCx</code>	288	.21	<code>\xintFtoCv</code>	300
.7	<code>\xintFtoCs</code>	288	.22	<code>\xintFtoCCv</code>	300
.8	<code>\xintFtoCx</code>	289	.23	<code>\xintCntoF</code>	300
.9	<code>\xintFtoC</code>	289	.24	<code>\xintGcntoF</code>	301
.10	<code>\xintFtoGC</code>	290	.25	<code>\xintCntoCs</code>	302
.11	<code>\xintFGtoC</code>	290	.26	<code>\xintCntoGC</code>	302
.12	<code>\xintFtoCC</code>	291	.27	<code>\xintGcntoGC</code>	303
.13	<code>\xintCtoF</code> , <code>\xintCstoF</code>	292	.28	<code>\xintCstoGC</code>	304
.14	<code>\xintiCstoF</code>	293	.29	<code>\xintGctoGC</code>	304
.15	<code>\xintGctoF</code>	293			

The commenting is currently (2020/02/19) 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, ε -T_EX and reload detection

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

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

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2   \catcode13=5      % ^^M
3   \endlinechar=13 %
4   \catcode123=1     % {
5   \catcode125=2     % }
6   \catcode64=11     % @
7   \catcode35=6      % #
8   \catcode44=12     % ,
9   \catcode45=12     % -
10  \catcode46=12     % .
11  \catcode58=12     % :
12  \let\z\endgroup
13  \expandafter\let\expandafter\x\csname ver@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 [2020/02/19 v1.4a 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\xintrawwithzeros {#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%
```



```

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\xintrawwithzeros {#2}\Z {#1}%
282 }%
283 \def\XINT_ftcx_A #1/#2\Z
284 {%
285   \expandafter\XINT_ftcx_B\romannumeral0\xintiidivision {#1}{#2}{#2}%
286 }%
287 \def\XINT_ftcx_B #1#2%
288 {%
289   \XINT_ftcx_C #2.{#1}%
290 }%
291 \def\XINT_ftcx_C #1%
292 {%
293   \xint_gob_til_zero #1\XINT_ftcx_integer 0\XINT_ftcx_D #1%
294 }%
295 \def\XINT_ftcx_integer 0\XINT_ftcx_D 0#1.#2#3#4{ #2}%
296 \def\XINT_ftcx_D #1.#2#3#4{\XINT_ftcx_loop_a {#1}{#3}{#1}{#2#4}{#4}}%
297 \def\XINT_ftcx_loop_a
298 {%
299   \expandafter\XINT_ftcx_loop_d\romannumeral0\XINT_div_prepare
300 }%
301 \def\XINT_ftcx_loop_d #1#2%
302 {%
303   \XINT_ftcx_loop_e #2.{#1}%
304 }%
305 \def\XINT_ftcx_loop_e #1%
306 {%
307   \xint_gob_til_zero #1\xint_ftcx_loop_exit0\XINT_ftcx_loop_f #1%
308 }%
309 \def\XINT_ftcx_loop_f #1.#2#3#4#5%
310 {%
311   \XINT_ftcx_loop_a {#1}{#3}{#1}{#4#2#5}{#5}%
312 }%
313 \def\xint_ftcx_loop_exit0\XINT_ftcx_loop_f #1.#2#3#4#5{ #4#2}%

```

10.9 \xintFtoC

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

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

10.10 \xintFtoGC

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

10.11 \xintFGtoC

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

```
318 \def\xintFGtoC {\romannumeral0\xintfgtoc}%
319 \def\xintfgtoc#1%
320 {%
321   \expandafter\XINT_fgtc_a\romannumeral0\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\xintraawithzeros {#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\xintraawithzeros {#2}.#1%
545 }%
546 \def\xINT_gctf_loop_g #1/#2.#3#4#5#6%
547 {%
548     \expandafter\xINT_gctf_loop_h\expandafter
549     {\romannumeral0\xINT_mul_fork #1\xint:#6\xint:}%
550     {\romannumeral0\xINT_mul_fork #1\xint:#5\xint:}%
551     {\romannumeral0\xINT_mul_fork #2\xint:#4\xint:}%
552     {\romannumeral0\xINT_mul_fork #2\xint:#3\xint:}%
553 }%
554 \def\xINT_gctf_loop_h #1#2%
555 {%

```

```

556 \expandafter\XINT_gctf_loop_i\expandafter {\expandafter{#2}{#1}}%
557 }%
558 \def\XINT_gctf_loop_i #1#2%
559 {%
560 \expandafter\XINT_gctf_loop_j\expandafter {\expandafter{#2}{#1}}%
561 }%
562 \def\XINT_gctf_loop_j #1#2%
563 {%
564 \expandafter\XINT_gctf_loop_a\expandafter {#2}{#1}%
565 }%
566 \def\XINT_gctf_end #1.#2#3#4#5{\xintrawithzeros {#2/#3}}% 1.09b removes [0]

```

10.16 \xintiGctoF

```

567 \def\xintiGctoF {\romannumeral0\xintigctoF}%
568 \def\xintigctoF #1%
569 {%
570 \expandafter\XINT_igctf_prep \romannumeral`&&@#1+!/ %
571 }%
572 \def\XINT_igctf_prep
573 {%
574 \XINT_igctf_loop_a 1001%
575 }%
576 \def\XINT_igctf_loop_a #1#2#3#4#5+%
577 {%
578 \expandafter\XINT_igctf_loop_b
579 \romannumeral`&&@#5.{#1}{#2}{#3}{#4}%
580 }%
581 \def\XINT_igctf_loop_b #1.#2#3#4#5%
582 {%
583 \expandafter\XINT_igctf_loop_c\expandafter
584 {\romannumeral0\xintiiadd {#5}{\XINT_mul_fork #1\xint:#3\xint:}}%
585 {\romannumeral0\xintiiadd {#4}{\XINT_mul_fork #1\xint:#2\xint:}}%
586 {#2}{#3}%
587 }%
588 \def\XINT_igctf_loop_c #1#2%
589 {%
590 \expandafter\XINT_igctf_loop_f\expandafter {\expandafter{#2}{#1}}%
591 }%
592 \def\XINT_igctf_loop_f #1#2#3#4/%
593 {%
594 \xint_gob_til_exclam #4\XINT_igctf_end!%
595 \expandafter\XINT_igctf_loop_g
596 \romannumeral`&&@#4.{#2}{#3}{#1}%
597 }%
598 \def\XINT_igctf_loop_g #1.#2#3%
599 {%
600 \expandafter\XINT_igctf_loop_h\expandafter
601 {\romannumeral0\XINT_mul_fork #1\xint:#3\xint:}%
602 {\romannumeral0\XINT_mul_fork #1\xint:#2\xint:}%
603 }%
604 \def\XINT_igctf_loop_h #1#2%
605 {%
606 \expandafter\XINT_igctf_loop_i\expandafter {#2}{#1}%

```

```

607 }%
608 \def\XINT_igctf_loop_i #1#2#3#4%
609 {%
610   \XINT_igctf_loop_a {#3}{#4}{#1}{#2}%
611 }%
612 \def\XINT_igctf_end #1.#2#3#4#5{\xintrawwithzeros {#4/#5}}% 1.09b removes [0]

```

10.17 \xintCtoCv, \xintCstoCv

1.09m uses \xintCSVtoList on the argument of \xintCstoCv to allow spaces also before the commas. The original \xintCstoCv code became the one of the new \xintCtoF dealing with a braced rather than comma separated list.

```

613 \def\xintCstoCv {\romannumeral0\xintcstocv }%
614 \def\xintcstocv #1%
615 {%
616   \expandafter\XINT_ctcv_prep\romannumeral0\xintcsvtolist{#1}!%
617 }%
618 \def\xintCtoCv {\romannumeral0\xintctocv }%
619 \def\xintctocv #1%
620 {%
621   \expandafter\XINT_ctcv_prep\romannumeral`&&@#1!%
622 }%
623 \def\XINT_ctcv_prep
624 {%
625   \XINT_ctcv_loop_a {}1001%
626 }%
627 \def\XINT_ctcv_loop_a #1#2#3#4#5#6%
628 {%
629   \xint_gob_til_exclam #6\XINT_ctcv_end!%
630   \expandafter\XINT_ctcv_loop_b
631   \romannumeral0\xintrawwithzeros {#6}#{#2}{#3}{#4}{#5}{#1}%
632 }%
633 \def\XINT_ctcv_loop_b #1/#2.#3#4#5#6%
634 {%
635   \expandafter\XINT_ctcv_loop_c\expandafter
636   {\romannumeral0\XINT_mul_fork #2\xint:#4\xint:}%
637   {\romannumeral0\XINT_mul_fork #2\xint:#3\xint:}%
638   {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#6\xint:}%
639     {\XINT_mul_fork #1\xint:#4\xint:}}%
640   {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#5\xint:}%
641     {\XINT_mul_fork #1\xint:#3\xint:}}%
642 }%
643 \def\XINT_ctcv_loop_c #1#2%
644 {%
645   \expandafter\XINT_ctcv_loop_d\expandafter {\expandafter{#2}{#1}}%
646 }%
647 \def\XINT_ctcv_loop_d #1#2%
648 {%
649   \expandafter\XINT_ctcv_loop_e\expandafter {\expandafter{#2}#1}%
650 }%
651 \def\XINT_ctcv_loop_e #1#2%
652 {%
653   \expandafter\XINT_ctcv_loop_f\expandafter{#2}#1%

```



```

654 }%
655 \def\XINT_ctcv_loop_f #1#2#3#4#5%
656 {%
657     \expandafter\XINT_ctcv_loop_g\expandafter
658     {\romannumeral0\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 \xintiGctoCv

```

768 \def\xintiGctoCv {\romannumeral0\xintigctocv}%
769 \def\xintigctocv #1%
770 {%
771     \expandafter\XINT_igctcv_prep \romannumeral`&&@#1+!/%%
772 }%
773 \def\XINT_igctcv_prep
774 {%
775     \XINT_igctcv_loop_a {}1001%
776 }%
777 \def\XINT_igctcv_loop_a #1#2#3#4#5#6+%
778 {%
779     \expandafter\XINT_igctcv_loop_b
780     \romannumeral`&&@#6.{#2}{#3}{#4}{#5}{#1}%
781 }%
782 \def\XINT_igctcv_loop_b #1.#2#3#4#5%
783 {%
784     \expandafter\XINT_igctcv_loop_c\expandafter
785     {\romannumeral0\xintiiadd {#5}{\XINT_mul_fork #1\xint:#3\xint:}}%
786     {\romannumeral0\xintiiadd {#4}{\XINT_mul_fork #1\xint:#2\xint:}}%
787     {{#2}{#3}}%
788 }%
789 \def\XINT_igctcv_loop_c #1#2%
790 {%
791     \expandafter\XINT_igctcv_loop_f\expandafter {\expandafter{#2}{#1}}%
792 }%
793 \def\XINT_igctcv_loop_f #1#2#3#4/%
794 {%
795     \xint_gob_til_exclam #4\XINT_igctcv_end_a!%
796     \expandafter\XINT_igctcv_loop_g
797     \romannumeral`&&@#4.#1#2{#3}%
798 }%
799 \def\XINT_igctcv_loop_g #1.#2#3#4#5%
800 {%
801     \expandafter\XINT_igctcv_loop_h\expandafter
802     {\romannumeral0\XINT_mul_fork #1\xint:#5\xint:}%
803     {\romannumeral0\XINT_mul_fork #1\xint:#4\xint:}%
804     {{#2}{#3}}%

```

```

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

```

10.21 \xintFtoCv

Still uses \xinticstocv \xintFtoCs rather than \xintctocv \xintFtoC.

```

823 \def\xintFtoCv {\romannumeral0\xintftocv }%
824 \def\xintftocv #1%
825 {%
826     \xinticstocv {\xintFtoCs {#1}}%
827 }%

```

10.22 \xintFtoCCv

```

828 \def\xintFtoCCv {\romannumeral0\xintftoccv }%
829 \def\xintftoccv #1%
830 {%
831     \xintigctocv {\xintFtoCC {#1}}%
832 }%

```

10.23 \xintCntoF

Modified in 1.06 to give the N first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that.

```

833 \def\xintCntoF {\romannumeral0\xintcntof }%
834 \def\xintcntof #1%
835 {%
836     \expandafter\XINT_cntf\expandafter {\the\numexpr #1}%
837 }%
838 \def\XINT_cntf #1#2%
839 {%
840     \ifnum #1>\xint_c_
841         \xint_afterfi {\expandafter\XINT_cntf_loop\expandafter
842             {\the\numexpr #1-1\expandafter}\expandafter
843             {\romannumeral`&&@#2{#1}}{#2}}%
844     \else

```

```

845     \xint_afterfi
846     {\ifnum #1=\xint_c_
847         \xint_afterfi {\expandafter\space \romannumeral`&&#2{0}}}%
848         \else \xint_afterfi { }% 1.09m now returns nothing.
849     \fi}%
850 \fi
851 }%
852 \def\xINT_cntf_loop #1#2#3%
853 {%
854     \ifnum #1>\xint_c_ \else \XINT_cntf_exit \fi
855     \expandafter\xINT_cntf_loop\expandafter
856     {\the\numexpr #1-1\expandafter }\expandafter
857     {\romannumeral0\xintadd {\xintDiv {1[0]}\{#2}\{#3{#1}}}%
858     {#3}%
859 }%
860 \def\xINT_cntf_exit \fi
861     \expandafter\xINT_cntf_loop\expandafter
862     #1\expandafter #2#3%
863 {%
864     \fi\xint_gobble_ii #2%
865 }%

```

10.24 \xintGCntoF

Modified in 1.06 to give the N argument first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that.

```

866 \def\xintGCntoF {\romannumeral0\xintgcntof }%
867 \def\xintgcntof #1%
868 {%
869     \expandafter\xINT_gcntf\expandafter {\the\numexpr #1}%
870 }%
871 \def\xINT_gcntf #1#2#3%
872 {%
873     \ifnum #1>\xint_c_
874         \xint_afterfi {\expandafter\xINT_gcntf_loop\expandafter
875             {\the\numexpr #1-1\expandafter}\expandafter
876             {\romannumeral`&&#2{#1}\{#2}\{#3}}}%
877     \else
878         \xint_afterfi
879         {\ifnum #1=\xint_c_
880             \xint_afterfi {\expandafter\space\romannumeral`&&#2{0}}}%
881             \else \xint_afterfi { }% 1.09m now returns nothing rather than 0/1[0]
882         \fi}%
883     \fi
884 }%
885 \def\xINT_gcntf_loop #1#2#3#4%
886 {%
887     \ifnum #1>\xint_c_ \else \XINT_gcntf_exit \fi
888     \expandafter\xINT_gcntf_loop\expandafter
889     {\the\numexpr #1-1\expandafter }\expandafter
890     {\romannumeral0\xintadd {\xintDiv {#4{#1}\{#2}\{#3{#1}}}%
891     {#3}{#4}%

```

```

892 }%
893 \def\XINT_gcntf_exit \fi
894   \expandafter\XINT_gcntf_loop\expandafter
895   #1\expandafter #2#3#4%
896 {%
897   \fi\xint_gobble_ii #2%
898 }%

```

10.25 \xintCntoCs

Modified in 1.09m: added spaces after the commas in the produced list. Moreover the coefficients are not braced anymore. A slight induced limitation is that the macro argument should not contain some explicit comma (cf. `\XINT_cntcs_exit_b`), hence `\xintCntoCs {\macro,}` with `\def\macro,#1{<stuff>}` would crash. Not a very serious limitation, I believe.

```

899 \def\xintCntoCs {\romannumeral0\xintcntocs }%
900 \def\xintcntocs #1%
901 {%
902   \expandafter\XINT_cntcs\expandafter {\the\numexpr #1}%
903 }%
904 \def\XINT_cntcs #1#2%
905 {%
906   \ifnum #1<0
907     \xint_afterfi { }% 1.09i: a 0/1[0] was here, now the macro returns nothing
908   \else
909     \xint_afterfi {\expandafter\XINT_cntcs_loop\expandafter
910                   {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
911                   {\romannumeral`&&@#2{#1}}{#2}}% produced coeff not braced
912   \fi
913 }%
914 \def\XINT_cntcs_loop #1#2#3%
915 {%
916   \ifnum #1>-\xint_c_i \else \XINT_cntcs_exit \fi
917   \expandafter\XINT_cntcs_loop\expandafter
918   {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
919   {\romannumeral`&&@#3{#1}, #2}{#3}% space added, 1.09m
920 }%
921 \def\XINT_cntcs_exit \fi
922   \expandafter\XINT_cntcs_loop\expandafter
923   #1\expandafter #2#3%
924 {%
925   \fi\XINT_cntcs_exit_b #2%
926 }%
927 \def\XINT_cntcs_exit_b #1,{}% romannumeral stopping space already there

```

10.26 \xintCntoGC

Modified in 1.06 to give the N first to a `\numexpr` rather than expanding twice. I just use `\the\numexpr` and maintain the previous code after that.

1.09m maintains the braces, as the coeff are allowed to be fraction and the slash can not be naked in the GC format, contrarily to what happens in `\xintCntoCs`. Also the separators given to `\xintGctoGCx` may then fetch the coefficients as argument, as they are braced.

```

928 \def\xintCntoGC {\romannumeral0\xintcntogc }%
929 \def\xintcntogc #1%
930 {%
931     \expandafter\XINT_cntgc\expandafter {\the\numexpr #1}%
932 }%
933 \def\XINT_cntgc #1#2%
934 {%
935     \ifnum #1<0
936         \xint_afterfi { }% 1.09i there was as strange 0/1[0] here, removed
937     \else
938         \xint_afterfi {\expandafter\XINT_cntgc_loop\expandafter
939             {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
940             {\expandafter{\romannumeral`&&@#2{#1}}{#2}}}%
941     \fi
942 }%
943 \def\XINT_cntgc_loop #1#2#3%
944 {%
945     \ifnum #1>-\xint_c_i \else \XINT_cntgc_exit \fi
946     \expandafter\XINT_cntgc_loop\expandafter
947     {\the\numexpr #1-\xint_c_i\expandafter }\expandafter
948     {\expandafter{\romannumeral`&&@#3{#1}}+1/#2}{#3}%
949 }%
950 \def\XINT_cntgc_exit \fi
951     \expandafter\XINT_cntgc_loop\expandafter
952     #1\expandafter #2#3%
953 {%
954     \fi\XINT_cntgc_exit_b #2%
955 }%
956 \def\XINT_cntgc_exit_b #1+1/{ }%

```

10.27 \xintGCntoGC

Modified in 1.06 to give the N first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that.

```

957 \def\xintGCntoGC {\romannumeral0\xintgcntogc }%
958 \def\xintgcntogc #1%
959 {%
960     \expandafter\XINT_gcntgc\expandafter {\the\numexpr #1}%
961 }%
962 \def\XINT_gcntgc #1#2#3%
963 {%
964     \ifnum #1<0
965         \xint_afterfi { }% 1.09i now returns nothing
966     \else
967         \xint_afterfi {\expandafter\XINT_gcntgc_loop\expandafter
968             {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
969             {\expandafter{\romannumeral`&&@#2{#1}}{#2}{#3}}}%
970     \fi
971 }%
972 \def\XINT_gcntgc_loop #1#2#3#4%
973 {%
974     \ifnum #1>-\xint_c_i \else \XINT_gcntgc_exit \fi

```

```

975 \expandafter\XINT_gcntgc_loop_b\expandafter
976 {\expandafter{\romannumeral`&&@#4{#1}}/#2}{#3{#1}}{#1}{#3}{#4}%
977 }%
978 \def\XINT_gcntgc_loop_b #1#2#3%
979 {%
980 \expandafter\XINT_gcntgc_loop\expandafter
981 {\the\numexpr #3-\xint_c_i \expandafter}\expandafter
982 {\expandafter{\romannumeral`&&@#2}+#1}%
983 }%
984 \def\XINT_gcntgc_exit \fi
985 \expandafter\XINT_gcntgc_loop_b\expandafter #1#2#3#4#5%
986 {%
987 \fi\XINT_gcntgc_exit_b #1%
988 }%
989 \def\XINT_gcntgc_exit_b #1/{ }%

```

10.28 \xintCstoGC

```

990 \def\xintCstoGC {\romannumeral0\xintcstogc }%
991 \def\xintcstogc #1%
992 {%
993 \expandafter\XINT_cstc_prep \romannumeral`&&@#1,!,%
994 }%
995 \def\XINT_cstc_prep #1,{\XINT_cstc_loop_a {{#1}}}%
996 \def\XINT_cstc_loop_a #1#2,%
997 {%
998 \xint_gob_til_exclam #2\XINT_cstc_end!%
999 \XINT_cstc_loop_b {#1}{#2}%
1000 }%
1001 \def\XINT_cstc_loop_b #1#2{\XINT_cstc_loop_a {#1+1/{#2}}}%
1002 \def\XINT_cstc_end!\XINT_cstc_loop_b #1#2{ #1}%

```

10.29 \xintGctoGC

```

1003 \def\xintGctoGC {\romannumeral0\xintgctogc }%
1004 \def\xintgctogc #1%
1005 {%
1006 \expandafter\XINT_gctgc_start \romannumeral`&&@#1+!/,%
1007 }%
1008 \def\XINT_gctgc_start {\XINT_gctgc_loop_a {}}%
1009 \def\XINT_gctgc_loop_a #1#2+#3/%
1010 {%
1011 \xint_gob_til_exclam #3\XINT_gctgc_end!%
1012 \expandafter\XINT_gctgc_loop_b\expandafter
1013 {\romannumeral`&&@#2}{#3}{#1}%
1014 }%
1015 \def\XINT_gctgc_loop_b #1#2%
1016 {%
1017 \expandafter\XINT_gctgc_loop_c\expandafter
1018 {\romannumeral`&&@#2}{#1}%
1019 }%
1020 \def\XINT_gctgc_loop_c #1#2#3%
1021 {%
1022 \XINT_gctgc_loop_a {#3{#2}+#1}/}%

```


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

```
1023 }%
1024 \def\XINT_gctgc_end!\expandafter\XINT_gctgc_loop_b
1025 {%
1026     \expandafter\XINT_gctgc_end_b
1027 }%
1028 \def\XINT_gctgc_end_b #1#2#3{ #3{#1}}%
1029 \XINT_restorecatcodes_endinput%
```

11 Package **xintexpr** implementation

This is release 1.4a of 2020/02/19.

Contents

11.1	READ ME! Important warnings and explanations relative to the status of the code source at the time of the 1.4 release	308
11.2	Old comments	309
11.3	Catcodes, ε -TeX and reload detection	310
11.4	Package identification	311
11.5	<code>\xintDigits*</code> , <code>\xintSetDigits*</code>	311
11.6	Support for output and transform of nested braced contents as core data type	312
11.6.1	Bracketed list rendering with prettifying of leaves from nested braced contents	312
11.6.2	Braced contents rendering via a TeX alignment with prettifying of leaves	312
11.6.3	Transforming all leaves within nested braced contents	313
11.7	Top level user TeX interface: <code>\xinteval</code> , <code>\xintfloateval</code> , <code>\xintiieval</code>	314
11.7.1	<code>\xintexpr</code> , <code>\xintexpr</code> , <code>\xintfloatexpr</code> , <code>\xintiexpr</code>	314
11.7.2	<code>\XINT_expr_wrap</code> , <code>\XINT_iexpr_wrap</code> , <code>\XINT_flexpr_wrap</code>	316
11.7.3	<code>\XINTexprprint</code> , <code>\XINTiexprprint</code> , <code>\XINTflexprprint</code>	316
11.7.4	<code>\xintthe</code> , <code>\xintthealign</code> , <code>\xinttheexpr</code> , <code>\xinttheiexpr</code> , <code>\xintthefloatexpr</code> , <code>\xinttheiexpr</code>	317
11.7.5	<code>\thexintexpr</code> , <code>\thexintiexpr</code> , <code>\thexintfloatexpr</code> , <code>\thexintiexpr</code>	317
11.7.6	<code>\xintbareeval</code> , <code>\xintbarefloateval</code> , <code>\xintbareieval</code>	317
11.7.7	<code>\xintthebareeval</code> , <code>\xintthebarefloateval</code> , <code>\xintthebareieval</code>	318
11.7.8	<code>\xinteval</code> , <code>\xintiieval</code> , <code>\xintfloateval</code> , <code>\xintiieval</code>	318
11.7.9	<code>\xintboolexpr</code> , <code>\XINT_boolexpr_print</code> , <code>\xinttheboolexpr</code> , <code>\thexintboolexpr</code>	318
11.7.10	<code>\xintifboolexpr</code> , <code>\xintifboolfloatexpr</code> , <code>\xintifbooliexpr</code>	318
11.7.11	<code>\xintifsgnexpr</code> , <code>\xintifsgnfloatexpr</code> , <code>\xintifsgniexpr</code>	318
11.7.12	Small bits we have to put somewhere	319
11.8	Hooks into the numeric parser for usage by the <code>\xintdeffunc</code> symbolic parser	320
11.9	<code>\XINT_expr_getnext</code> : fetch some value then an operator and present them to last waiter with the found operator precedence, then the operator, then the value	320
11.10	<code>\XINT_expr_scan_nbr_or_func</code> : parsing the integer or decimal number or hexa-decimal number or function name or variable name or special hacky things	322
11.10.1	Integral part (skipping zeroes)	323
11.10.2	Fractional part	325
11.10.3	Scientific notation	326
11.10.4	Hexadecimal numbers	327
11.10.5	<code>\XINT_expr_scanfunc</code> : collecting names of functions and variables	329
11.10.6	<code>\XINT_expr_func</code> : dispatch to variable replacement or to function execution	330
11.11	<code>\XINT_expr_op_`</code> : launch function or pseudo-function, or evaluate variable and insert operator of multiplication in front of parenthesized contents	330
11.12	<code>\XINT_expr_op_</code> : replace a variable by its value and then fetch next operator	331
11.13	<code>\XINT_expr_getop</code> : fetch the next operator or closing parenthesis or end of expression	332
11.14	Expansion spanning; opening and closing parentheses	334
11.15	The comma as binary operator	336
11.16	The minus as prefix operator of variable precedence level	337
11.17	The <code>*</code> as Python-like «unpacking» prefix operator	338
11.18	Infix operators	338
11.18.1	<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>'or'</code> , <code>'xor'</code> , and <code>'mod'</code>	338
11.18.2	<code>..</code> , <code>..[</code> , and <code>]</code> as infix operators	341

11.18.3	Support macros for .., ..[and]..	342
11.19	Square brackets [] both as a container and a Python slicer	345
11.19.1	[...] as «oneple» constructor	345
11.19.2	[...] brackets and : operator for NumPy-like slicing and item indexing syntax	346
11.19.3	Macro layer implementing indexing and slicing	348
11.20	Support for raw A/B[N]	352
11.21	? as two-way and ?? as three-way «short-circuit» conditionals	352
11.22	! as postfix factorial operator	353
11.23	User defined variables	354
11.23.1	\xintdefvar, \xintdefiivar, \xintdeffloatvar	354
11.23.2	\xintunassignvar	356
11.24	Support for dummy variables	357
11.24.1	\xintnewdummy	357
11.24.2	\xintensuredummy, \xintrestorevariable	358
11.24.3	Checking (without expansion) that a symbolic expression contains correctly nested parentheses	359
11.24.4	Fetching balanced expressions E1, E2 and a variable name Name from E1, Name=E2)	360
11.24.5	Fetching a balanced expression delimited by a semi-colon	360
11.24.6	Low-level support for omit and abort keywords, the break() function, the n++ construct and the semi-colon as used in the syntax of seq(), add(), mul(), iter(), rseq(), iterr(), rrseq(), subsm(), subsn(), ndseq(), ndmap()	360
11.24.7	Reserved dummy variables @, @1, @2, @3, @4, @@, @@(1), ..., @@@, @@@(1), ... for recursions	362
11.25	Pseudo-functions involving dummy variables and generating scalars or sequences	363
11.25.1	Comments	363
11.25.2	subs(): substitution of one variable	365
11.25.3	subsm(): simultaneous independent substitutions	365
11.25.4	subsn(): leaner syntax for nesting (possibly dependent) substitutions	366
11.25.5	seq(): sequences from assigning values to a dummy variable	368
11.25.6	iter()	368
11.25.7	add(), mul()	370
11.25.8	rseq()	371
11.25.9	iterr()	372
11.25.10	rrseq()	373
11.26	Pseudo-functions related to N-dimensional hypercubic lists	374
11.26.1	ndseq()	374
11.26.2	ndmap()	375
11.26.3	ndfillraw()	377
11.27	Other pseudo-functions: bool(), togl(), protect(), qraw(), qint(), qfrac(), qfloat(), qrand(), random(), rbit()	377
11.28	Regular built-in functions: num(), reduce(), preduce(), abs(), sgn(), frac(), floor(), ceil(), sqr(), ?(), !(), not(), odd(), even(), isint(), isone(), factorial(), sqrt(), sqrtr(), inv(), round(), trunc(), float(), sfloat(), ilog10(), divmod(), mod(), binomial(), pfactorial(), randrange(), quo(), rem(), gcd(), lcm(), max(), min(), `+`(), `*`(), all(), any(), xor(), len(), first(), last(), reversed(), if(), ifint(), ifone(), ifsgn(), nuple() and unpack()	378
11.29	User declared functions	392
11.29.1	\xintdeffunc, \xintdefiifunc, \xintdeffloatfunc	392
11.29.2	\xintdefufunc, \xintdefiifunc, \xintdeffloatufunc	395
11.29.3	\xintunassignexprfunc, \xintunassigniifunc, \xintunassignfloatexprfunc	396
11.29.4	\xintNewFunction	396
11.29.5	Mysterious stuff	398

11.29.6	<code>\XINT_expr_redefinemacros</code>	409
11.29.7	<code>\xintNewExpr</code> , <code>\xintNewIExpr</code> , <code>\xintNewFloatExpr</code> , <code>\xintNewIIExpr</code>	410
11.29.8	<code>\ifxintexprsafecatcodes</code> , <code>\xintexprSafeCatcodes</code> , <code>\xintexprRestoreCatcodes</code>	412

11.1 READ ME! Important warnings and explanations relative to the status of the code source at the time of the 1.4 release

At release 1.4 the `csname` encapsulation of intermediate evaluations during parsing of expressions is dropped, and `xintexpr` requires the `\expanded` primitive. This means that there is no more impact on the string pool. And as internal storage now uses simply core `\TeX{}` syntax with braces rather than comma separated items inside a `csname` dummy control sequence, it became possible to let the [...] syntax be associated to a true internal type of «tuple» or «list».

The output of `\xintexpr` (after `\romannumeral0` or `\romannumeral-`0` triggered expansion or double expansion) is thus modified at 1.4. It now looks like this:

`\XINTfstop \XINTexprprint .{<number>}` in simplest case

`\XINTfstop \XINTexprprint .{...}...{...}` in general case

where ... stands for nested braces ultimately ending in `{<num. rep.>}` leaves. The `<num. rep.>` stands for some internal representation of numeric data. It may be empty, and currently as well as probably in future uses only catcode 12 tokens (no spaces currently).

`{}}` corresponds (in input as in output) to `[]`. The external TeX braces also serve as set-theoretical braces. The comma is concatenation, so for example `[]`, `[]` will become `{{{}}}`, or rather `{{}}` if sub-unit of something else.

The associated vocabulary is explained in the user manual and we avoid too much duplication here. `xintfrac` numerical macros receiving an empty argument usually handle it as being 0, but this is not the case of the `xintcore` macros supporting `\xintiexpr`, they usually break if exercised on some empty argument.

The above expansion result uses thus only normal catcodes are output, i.e. the backslash, regular braces, and catcode 12 characters (scientific notation is internally converted to raw `xintfrac` representation `[N]`).

Additional data may be located before the dot; this is the case only for `\xintfloatexpr` currently. As `xintexpr` actually defines three parsers `\xintexpr`, `\xintiexpr` and `\xintfloatexpr` but tries to share as much code as possible, some overhead is induced to fit all into the same mold.

`\XINTfstop` stops `\romannumeral-`0` (or 0) type spanned expansion, and is invariant under `\edef`, but simply disappears in typesetting context. It is thus now legal to use `\xintexpr` directly in typesetting flow.

`\XINTexprprint` is `\protected`.

The f-expansion of an `\xintexpr <expression>\relax` is a complete expansion, i.e. one whose result remains invariant under `\edef`. But if exposed to finitely many expansion steps (at least two) there is a «blinking» `\noexpand` upfront depending on parity of number of steps.

`\xintthe\xintexpr <expression>\relax` or `\xinteval{<expression>}` serve as formerly to deliver the explicit digits, or more exactly some prettifying view of the actual `<internal number representation>`. For example `\xintthe\xintboolexpr` will (this is tentative) use `True` and `False` in output.

Nested contents like this

`{{1}{{2}{3}}{{4}{5}{6}}}{9}}`

will get delivered using nested square brackets like that

`1, [2, 3, [4, 5, 6]], 9`

and as conversely `\xintexpr 1, [2, 3, [4, 5, 6]], 9\relax` expands to

`\XINTfstop \XINTexprprint .{1}{{2}{3}}{{4}{5}{6}}{9}}`

we obtain the gratifying result that

`\xinteval{1, [2, 3, [4, 5, 6]], 9}`

expands to

1, [2, 3, [4, 5, 6]], 9

See user manual for explanations on the plasticity of `\xintexpr` syntax regarding functions with multiple arguments, and the 1.4 «unpacking» Python-like `*` prefix operator.

I have suppressed (from the public dtx) many big chunks of comments. Some became obsolete and need to be updated, others are currently of value only to the author as a historical record.

ATTENTION! As the removal process itself took too much time, I ended up leaving as is many comments which are obsoleted and wrong to various degrees after the 1.4 release. Precedence levels of operators have all been doubled to make room for new constructs

Even comments added during 1.4 developement may now be obsolete because the preparation of 1.4 took a few weeks and that's enough of duration to provide the author many chances to contradict in the code what has been already commented upon.

Thus don't believe (fully) anything which is said here!



Warning: in text below and also in left-over old comments I may refer to «until» and «op» macros; due to the change of data storage at 1.4, I needed to refactor a bit the way expansion is controlled, and the situation now is mainly governed by «op», «exec», «check-» and «checkp» macros the latter three replacing the two «until_a» and «until_b» of former code. This allows to diminish the number of times an accumulated result will be grabbed in order to propagate expansion to its right. Formerly this was not an issue because such things were only a single token! I do not describe here how this is all articulated but it is not hard to see it from the code (the hardest thing in all such matter was in 2013 to actually write how the expansion would be intially launched becasue to do that one basically has to understand the mechanism in its whole and such things are not easy to develop piecemeal). Another thing to keep in mind is that operators in truth have a left precedence (i.e. the precedence they show to operators arising earlier) and a right precedence (which determines how they react to operators coming after them from the right). Only the first one is usually encapsulated in a chardef, the second one is most of the times identical to the first one and if not it is only virtual but implemented via `\ifcase` of `\ifnum` branching. A final remark is that some things are achieved by special «op» macros, which are a favorite tool to hack into the normal regular flow of things, via injection of special syntax elements. I did not rename these macros for avoiding too large git diffs, and besides the nice thing is that the 1.4 refactoring minimally had to modify them, and all hacky things using them kept on working with not a single modification. And a post-scriptum is that advanced features crucially exploit injecting sub-`\xintexpr`-essions, as all is expandable there is no real «context» (only a minimal one) which one would have to perhaps store and restore and doing this sub-expression injection is rather cheap and efficient operation.

11.2 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.

11.3 Catcodes, ε -TeX and reload detection

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

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

```

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

```

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

```

11.4 Package identification

\XINT_Cmp alias for *\xintiiCmp* needed for some forgotten reason related to *\xintNewExpr* (FIX THIS!)

```

56 \XINT_providespackage
57 \ProvidesPackage{xintexpr}%
58 [2020/02/19 v1.4a Expandable expression parser (JFB)]%
59 \catcode\! 11
60 \let\XINT_Cmp \xintiiCmp
61 \def\XINTfstop{\noexpand\XINTfstop}%

```

11.5 *\xintDigits**, *\xintSetDigits**

1.3f

```

62 \def\xintDigits {\futurelet\XINT_token\xintDigitss}%
63 \def\xintDigitss #1={\afterassignment\xintDigitsss\mathchardef\XINTdigits=}%
64 \def\xintDigitsss#1{\ifx*\XINT_token\expandafter\xintreloadxinttrig\fi}%
65 \let\xintfracSetDigits\xintSetDigits
66 \def\xintSetDigits#1#1{\if\relax\detokenize{#1}\relax
67     \else\afterassignment\xintreloadxinttrig\fi
68     \xintfracSetDigits}%

```

11.6 Support for output and transform of nested braced contents as core data type

New at 1.4, of course. The former `\csname.=...\endcsname` encapsulation technique made very difficult implementation of nested structures.

11.6.1 Bracketed list rendering with prettifying of leaves from nested braced contents

1.4

```

69 \def\XINT:expr:toblistwith#1#2%
70 {%
71     {\expandafter\XINT:expr:toblist_checkempty
72       \expanded{\noexpand#1!\expandafter}\detokenize{#2}^}%
73 }%
74 \def\XINT:expr:toblist_checkempty #1!#2%
75 {%
76     \if ^#2\expandafter\xint_gob_til_^ \else\expandafter\XINT:expr:toblist_a\fi
77     #1!#2%
78 }%
79 \catcode`< 1 \catcode`> 2 \catcode`{ 12 \catcode`} 12
80 \def\XINT:expr:toblist_a #1!#2%
81 <%
82     \if{#2\xint_dothis<[\XINT:expr:toblist_a>\fi
83     \xint_orthat\XINT:expr:toblist_b #1#2%
84 >%
85 \def\XINT:expr:toblist_b #1!#2}%
86 <%
87     \if\relax#2\relax\xintexprEmptyItem\else#1<#2>\fi\XINT:expr:toblist_c #1!}%
88 >%
89 \def\XINT:expr:toblist_c #1!#2%
90 <%
91     \if ^#2\xint_dothis<\xint_gob_til_^>\fi
92     \if{#2\xint_dothis<, \XINT:expr:toblist_a>\fi
93     \xint_orthat<]\XINT:expr:toblist_c>#1#2%
94 >%
95 \catcode`{ 1 \catcode`} 2 \catcode`< 12 \catcode`> 12

```

11.6.2 Braced contents rendering via a T_EX alignment with prettifying of leaves

1.4.

Breaking change at 1.4a as helper macros were renamed and their meanings refactored: no more `\xintexpralignntab` nor `\xintexpralignninnercomma` or `\xintexpralignnoutercomma` but `\xintexpralignninnersep`, etc... .

```

96 \catcode`& 4
97 \protected\def\xintexpralignbegin      {\halign\bgroup\tabskip2ex\hfil##&&&\hfil\cr}%
98 \protected\def\xintexpralignend        {\crcr\egroup}%
99 \protected\def\xintexpralignlinesep    {\cr}%
100 \protected\def\xintexpralignleftbracket {[%
101 \protected\def\xintexpralignrightbracket {]}%
102 \protected\def\xintexpralignleftsep    {&}%
103 \protected\def\xintexpralignrightsep   {&}%

```



```

104 \protected\def\xintexpraligninnersep    {,&}%
105 \catcode`& 7
106 \def\XINT:expr:toalignwith#1#2%
107 {%
108     {\expandafter\XINT:expr:toalign_checkempty
109     \expanded{\noexpand#1!\expandafter}\detokenize{#2}^\expandafter}%
110     \xintexpralignend
111 }%
112 \def\XINT:expr:toalign_checkempty #1!#2%
113 {%
114     \if ^#2\expandafter\xint_gob_til_^\else\expandafter\XINT:expr:toalign_a\fi
115     #1!#2%
116 }%
117 \catcode`< 1 \catcode`> 2 \catcode`\{ 12 \catcode`\} 12
118 \def\XINT:expr:toalign_a #1{#2%
119 <%
120     \if{#2\xint_dothis<\xintexpralignleftbracket\XINT:expr:toalign_a>\fi
121     \xint_orthat<\xintexpralignleftsep\XINT:expr:toalign_b>#1#2%
122 >%
123 \def\XINT:expr:toalign_b #1!#2}%
124 <%
125     \if\relax#2\relax\xintexprEmptyItem\else#1<#2>\fi\XINT:expr:toalign_c #1!}%
126 >%
127 \def\XINT:expr:toalign_c #1{#2%
128 <%
129     \if ^#2\xint_dothis<\xint_gob_til_^\fi
130     \if {#2\xint_dothis<\xintexpraligninnersep\XINT:expr:toalign_A>\fi
131     \xint_orthat<\xintexpralignrightsep\xintexpralignrightbracket\XINT:expr:toalign_C>#1#2%
132 >%
133 \def\XINT:expr:toalign_A #1{#2%
134 <%
135     \if{#2\xint_dothis<\xintexpralignleftbracket\XINT:expr:toalign_A>\fi
136     \xint_orthat\XINT:expr:toalign_b #1#2%
137 >%
138 \def\XINT:expr:toalign_C #1{#2%
139 <%
140     \if ^#2\xint_dothis<\xint_gob_til_^\fi
141     \if {#2\xint_dothis<\xintexpralignlinesep\XINT:expr:toalign_a>\fi
142     \xint_orthat<\xintexpralignrightbracket\XINT:expr:toalign_C>#1#2%
143 >%
144 \catcode`\{ 1 \catcode`\} 2 \catcode`< 12 \catcode`> 12

```

11.6.3 Transforming all leaves within nested braced contents

1.4. Leaves must be of catcode 12... This is currently not a constraint (or rather not a new constraint) for xintexpr because formerly anyhow all data went through csname encapsulation and extraction via string.

In order to share code with the functioning of universal functions, which will be allowed to transform a number into an ople, the applied macro is supposed to apply one level of bracing to its output. Thus to apply this with an xintfrac macro such as \xintiRound{0} one needs first to define a wrapper which will expand it into braces :

```
\def\foo#1{{\xintiRound{0}}{#1}}
```

As the things will expand inside expanded, propagating expansion is not an issue.

This code is used by `\xintiexpr` and `\xintfloatexpr` in case of optional argument and by the «Universal functions».

```

145 \def\XINT:expr:mapwithin#1#2%
146 {%
147     {\expandafter\XINT:expr:mapwithin_checkempty
148       \expanded{\noexpand#1!\expandafter}\detokenize{#2}^}}%
149 }%
150 \def\XINT:expr:mapwithin_checkempty #1!#2%
151 {%
152     \if ^#2\expandafter\xint_gob_til_^else\expandafter\XINT:expr:mapwithin_a\fi
153     #1!#2%
154 }%
155 \begingroup % should I check lccode s generally if corrupted context at load?
156 \catcode`[ 1 \catcode`] 2 \lccode`[{ \lccode`]`{
157 \catcode`< 1 \catcode`> 2 \catcode`{ 12 \catcode`} 12
158 \lowercase<\endgroup
159 \def\XINT:expr:mapwithin_a #1{#2%
160 <%
161     \if{#2\xint_dothis<[\iffalse]\fi\XINT:expr:mapwithin_a>\fi%
162     \xint_orthat\XINT:expr:mapwithin_b #1#2%
163 >%
164 \def\XINT:expr:mapwithin_b #1!#2}%
165 <%
166     #1<#2>\XINT:expr:mapwithin_c #1!}%
167 >%
168 \def\XINT:expr:mapwithin_c #1{#2%
169 <%
170     \if ^#2\xint_dothis<\xint_gob_til_^>\fi
171     \if{#2\xint_dothis<\XINT:expr:mapwithin_a>\fi%
172     \xint_orthat<[\iffalse]\fi\XINT:expr:mapwithin_c>#1#2%
173 >%
174 >% back to normal catcodes

```

11.7 Top level user T_EX interface: `\xinteval`, `\xintfloateval`, `\xintiieval`

11.7.1	<code>\xintexpr</code> , <code>\xintiexpr</code> , <code>\xintfloatexpr</code> , <code>\xintiiepr</code>	314
11.7.2	<code>\XINT_expr_wrap</code> , <code>\XINT_iiexpr_wrap</code> , <code>\XINT_flexpr_wrap</code>	316
11.7.3	<code>\XINTexprprint</code> , <code>\XINTiieprprint</code> , <code>\XINTflexprprint</code>	316
11.7.4	<code>\xintthe</code> , <code>\xintthealign</code> , <code>\xinttheexpr</code> , <code>\xinttheiexpr</code> , <code>\xintthefloatexpr</code> , <code>\xint-</code> <code>theiiepr</code>	317
11.7.5	<code>\thexintexpr</code> , <code>\thexintiexpr</code> , <code>\thexintfloatexpr</code> , <code>\thexintiiepr</code>	317
11.7.6	<code>\xintbareeval</code> , <code>\xintbarefloateval</code> , <code>\xintbareiieval</code>	317
11.7.7	<code>\xintthebareeval</code> , <code>\xintthebarefloateval</code> , <code>\xintthebareiieval</code>	318
11.7.8	<code>\xinteval</code> , <code>\xintiieval</code> , <code>\xintfloateval</code> , <code>\xintiieval</code>	318
11.7.9	<code>\xintboolexpr</code> , <code>\XINT_boolexpr_print</code> , <code>\xinttheboolexpr</code> , <code>\thexintboolexpr</code> . . .	318
11.7.10	<code>\xintifboolexpr</code> , <code>\xintifboolfloatexpr</code> , <code>\xintifbooliiepr</code>	318
11.7.11	<code>\xintifsgnexpr</code> , <code>\xintifsgnfloatexpr</code> , <code>\xintifsgniiepr</code>	318
11.7.12	Small bits we have to put somewhere	319

11.7.1 `\xintexpr`, `\xintiexpr`, `\xintfloatexpr`, `\xintiiepr`

`\xintiexpr` and `\xintfloatexpr` have an optional argument since 1.1.

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

The 1.4a version of optional argument [D] for \xintiexpr accepts a negative D, with same meaning as \xintTrunc from xintfrac.sty.

```

175 \def\xintexpr      {\romannumeral0\xintexpro      }%
176 \def\xintiexpr     {\romannumeral0\xintiexpro      }%
177 \def\xintfloatexpr {\romannumeral0\xintfloatexpro }%
178 \def\xintiexpro    {\romannumeral0\xintiexpro      }%
179 \def\xintexpro     {\expandafter\XINT_expr_wrap\romannumeral0\xintbareeval }%
180 \def\xintiexpro    {\expandafter\XINT_iexpr_wrap\romannumeral0\xintbareieeval }%
181 \def\xintiexpro #1%
182 {%
183   \ifx [#1\expandafter\XINT_iexpr_withopt\else\expandafter\XINT_iexpr_noopt
184   \fi #1%
185 }%
186 \def\XINT_iexpr_noopt
187 {%
188   \expandafter\XINT_iexpr_iiround\romannumeral0\xintbareeval
189 }%
190 \def\XINT_iexpr_iiround
191 {%
192   \expandafter\XINT_expr_wrap
193   \expanded
194   \XINT:NEhook:x:mapwithin\XINT:expr:mapwithin{\XINTbracediRoundzero}%
195 }%
196 \def\XINTbracediRoundzero#1{{\xintiRound{0}}{#1}}}%
197 \def\XINT_iexpr_withopt [#1]%
198 {%
199   \expandafter\XINT_iexpr_round
200   \the\numexpr \xint_zapspace #1 \xint_gobble_i\expandafter.%
201   \romannumeral0\xintbareeval
202 }%
203 \def\XINT_iexpr_round #1.%
204 {%
205   \ifnum#1=\xint_c_\xint_dothis{\XINT_iexpr_iiround}\fi
206   \xint_orthat{\XINT_iexpr_round_a #1.%}
207 }%
208 \def\XINT_iexpr_round_a #1.%
209 {%
210   \expandafter\XINT_expr_wrap
211   \expanded
212   \XINT:NEhook:x:mapwithin\XINT:expr:mapwithin{\XINTbracedRound{#1}}%
213 }%
214 \def\XINTbracedRound#1#2{{\xintRound{#1}}{#2}}}%
215 \def\xintfloatexpro #1%
216 {%
217   \ifx [#1\expandafter\XINT_flexpr_withopt\else\expandafter\XINT_flexpr_noopt
218   \fi #1%
219 }%
220 \def\XINT_flexpr_noopt
221 {%
222   \expandafter\XINT_flexpr_wrap\the\numexpr\XINTdigits\expandafter.%
223   \romannumeral0\xintbarefloateval

```

```

224 }%
225 \def\XINT_flexpr_withopt [#1]%
226 {%
227     \expandafter\XINT_flexpr_withopt_a
228     \the\numexpr\xint_zapspaces #1 \xint_gobble_i\expandafter.%
229     \romannumeral0\xintbarefloateval
230 }%
231 \def\XINT_flexpr_withopt_a #1#2.%
232 {%
233     \expandafter\XINT_flexpr_withopt_b\the\numexpr\if#1-\XINTdigits\fi#1#2.%
234 }%
235 \def\XINT_flexpr_withopt_b #1.%
236 {%
237     \expandafter\XINT_flexpr_wrap
238     \the\numexpr#1\expandafter.%
239     \expanded
240     \XINT:NEhook:x:mapwithin\XINT:expr:mapwithin{\XINTbracedinFloat[#1]}%
241 }%
242 \def\XINTbracedinFloat[#1]#2{{\XINTinFloat[#1]{#2}}}%

```

11.7.2 \XINT_expr_wrap, \XINT_iexpr_wrap, \XINT_flexpr_wrap

1.3e removes some leading space tokens which served nothing. There is no \XINT_iexpr_wrap, because \XINT_expr_wrap is used directly.

```

243 \def\XINT_expr_wrap    {\XINTfstop\XINTexprprint.}%
244 \def\XINT_iexpr_wrap  {\XINTfstop\XINTiexprprint.}%
245 \def\XINT_flexpr_wrap {\XINTfstop\XINTflexprprint}%

```

11.7.3 \XINTexprprint, \XINTiexprprint, \XINTflexprprint

Comments currently under reconstruction.

1.4: The reason for \expanded is to ensure \xintthe mechanism does expand completely in two steps, now that the print helper macros are not f-expandable anymore.

It is possible that the expression gave an empty object (e.g. \xintexpr [4][1]\relax). Thus we do not use \romannumeral`^^@ trigger else twice expansion of \xinttheexpr will propagate beyond empty expression. Thus \romannumeral ended via a chardef zero token.

```

246 \protected\def\XINTexprprint.%
247     {\XINT:NEhook:x:toblist\XINT:expr:toblistwith\xintexprPrintOne}%
248 \let\xintexprPrintOne\xintFracToSci
249 \def\xintexprEmptyItem{[]}%
250 \protected\def\XINTiexprprint.%
251     {\XINT:NEhook:x:toblist\XINT:expr:toblistwith\xintiexprPrintOne}%
252 \let\xintiexprPrintOne\xint_firstofone
253 \protected\def\XINTflexprprint #1.%
254     {\XINT:NEhook:x:toblist\XINT:expr:toblistwith{\xintfloatexprPrintOne{#1}}}%
255 \def\xintfloatexprPrintOne#1%
256     {\romannumeral0\XINT_pfloat_opt [\xint:#1]}% bad direct jump
257 \protected\def\XINTboolexprprint.%
258     {\XINT:NEhook:x:toblist\XINT:expr:toblistwith\xintboolexprPrintOne}%
259 \def\xintboolexprPrintOne#1{\xintiifNotZero{#1}{True}{False}}%

```

11.7.4 `\xintthe`, `\xintthealign`, `\xinttheexpr`, `\xinttheiexpr`, `\xintthefloatexpr`, `\xinttheiexpr`

The reason why `\xinttheiexpr` et `\xintthefloatexpr` are handled differently is that they admit an optional argument which acts via a custom «printing» stage.

We exploit here that `\expanded` expands forward until finding an implicit or explicit brace, and that this expansion overrules `\protected` macros, forcing them to expand, similarly as `\romannumeral` expands `\protected` macros, and contrarily to what happens *within* the actual `\expanded` scope. I discovered this fact by testing (with `pdftex`) and I don't know where this is documented apart from the source code of the relevant engines. This is useful to us because there are contexts where we will want to apply a complete expansion before printing, but in purely numerical context this is not needed (if I converted correctly after dropping at 1.4 the `\csname` governed expansions; however I rely at various places on the fact that the `xint` macros are f-expandable, so I have tried to not use zillions of `\expanded` all over the place), hence it is not needed to add the expansion overhead by default. But the `\expanded` here will allow `\xintNewExpr` to create macro with suitable modification or the printing step, via some hook rather than having to duplicate all macros here with some new «NE» meaning (aliasing does not work or causes big issues due to desire to support `\xinteval` also in «NE» context as sub-constituent. The `\XINT:NEhook:x:toblist` is something else which serves to achieve this support of *sub* `\xinteval`, it serves nothing for the actual produced macros. For `\xintdeffunc`, things are simpler, but still we support the `[N]` optional argument of `\xintiexpr` and `\xintfloatexpr`, which required some work...

```
260 \def\xintthe      #1{\expanded\expandafter\xint_gobble_i\romannumeral`&&@#1}%
261 \def\xintthealign #1{\expandafter\xintexpralignbegin
262                     \expanded\expandafter\XINT:expr:toalignwith
263                     \romannumeral0\expandafter\expandafter\expandafter\expandafter
264                     \expandafter\expandafter\expandafter\xint_gob_andstop_ii
265                     \expandafter\xint_gobble_i\romannumeral`&&@#1}%
266 \def\xinttheexpr
267   {\expanded\expandafter\XINTexprprint\expandafter.\romannumeral0\xintbareeval}%
268 \def\xinttheiexpr
269   {\expanded\expandafter\xint_gobble_i\romannumeral`&&\xintiexpr}%
270 \def\xintthefloatexpr
271   {\expanded\expandafter\xint_gobble_i\romannumeral`&&\xintfloatexpr}%
272 \def\xinttheiexpr
273   {\expanded\expandafter\XINTiexprprint\expandafter.\romannumeral0\xintbareiieval}%

```

11.7.5 `\thexintexpr`, `\thexintiexpr`, `\thexintfloatexpr`, `\thexintiexpr`

New with 1.2h. I have been for the last three years very strict regarding macros with `\xint` or `\XINT`, but well.

1.4. Definitely I don't like those. Don't use them, I will remove one day!

```
274 \let\thexintexpr    \xinttheexpr
275 \let\thexintiexpr    \xinttheiexpr
276 \let\thexintfloatexpr\xintthefloatexpr
277 \let\thexintiexpr    \xinttheiexpr

```

11.7.6 `\xintbareeval`, `\xintbarefloateval`, `\xintbareiieval`

At 1.4 added one expansion step via `_start` macros. Triggering is expected to be via either `\romannumeral`^^@` or `\romannumeral0` is also ok

```
278 \def\xintbareeval    {\XINT_expr_start  }%

```

```
279 \def\xintbarefloateval{\XINT_flexpr_start}%
280 \def\xintbareiieval {\XINT_iiexpr_start}%
```

11.7.7 \xintthebareeval, \xintthebarefloateval, \xintthebareiieval

For matters of \XINT_NewFunc

```
281 \def\xintthebareeval      {\romannumeral0\expandafter\xint_stop_atfirstofone\romannumeral0\xintbareeval}%
282 \def\xintthebarefloateval {\romannumeral0\expandafter\xint_stop_atfirstofone\romannumeral0\xintbarefloat}%
283 \def\xintthebareiieval    {\romannumeral0\expandafter\xint_stop_atfirstofone\romannumeral0\xintbareiieval}%
```

11.7.8 \xinteval, \xintieval, \xintfloateval, \xintiieval

```
284 \def\xinteval #1%
285   {\expanded\expandafter\XINTexprprint\expandafter.\romannumeral0\xintbareeval#1\relax}%
286 \def\xintieval #1%
287   {\expanded\expandafter\xint_gobble_i\romannumeral`&&\xintiexpr#1\relax}%
288 \def\xintfloateval #1%
289   {\expanded\expandafter\xint_gobble_i\romannumeral`&&\xintfloatexpr#1\relax}%
290 \def\xintiieval #1%
291   {\expanded\expandafter\XINTiexprprint\expandafter.\romannumeral0\xintbareiieval#1\relax}%
```

11.7.9 \xintboolexpr, \XINT_boolexpr_print, \xinttheboolexpr, \thexintboolexpr

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

Attention, the conversion to 1 or 0 is done only by the print macro. Perhaps I should force it also inside raw result.

```
292 \def\xintboolexpr
293 {%
294   \romannumeral0\expandafter\XINT_boolexpr_done\romannumeral0\xintexpr
295 }%
296 \def\XINT_boolexpr_done #1.{\XINTfstop\XINTboolexprprint.}%
297 \def\xinttheboolexpr
298 {%
299   \expanded\expandafter\XINTboolexprprint\expandafter.\romannumeral0\xintbareeval
300 }%
301 \let\thexintboolexpr\xinttheboolexpr
```

11.7.10 \xintifboolexpr, \xintifboolfloatexpr, \xintifbooliiexpr

They do not accept comma separated expressions input.

```
302 \def\xintifboolexpr      #1{\romannumeral0\xintiiifnotzero {\xinttheexpr #1\relax}}%
303 \def\xintifboolfloatexpr #1{\romannumeral0\xintiiifnotzero {\xintthefloatexpr #1\relax}}%
304 \def\xintifbooliiexpr    #1{\romannumeral0\xintiiifnotzero {\xinttheiiexpr #1\relax}}%
```

11.7.11 \xintifsgnexpr, \xintifsgnfloatexpr, \xintifsgniiexpr

1.3d.

They do not accept comma separated expressions.

```
305 \def\xintifsgnexpr      #1{\romannumeral0\xintiiifsgn {\xinttheexpr #1\relax}}%
306 \def\xintifsgnfloatexpr #1{\romannumeral0\xintiiifsgn {\xintthefloatexpr #1\relax}}%
307 \def\xintifsgniiexpr    #1{\romannumeral0\xintiiifsgn {\xinttheiiexpr #1\relax}}%
```

11.7.12 Small bits we have to put somewhere

Some renaming and modifications here with release 1.2 to switch from using chains of `\romannumeral-0` in order to gather numbers, possibly hexadecimal, to using a `\csname` governed expansion. In this way no more limit at 5000 digits, and besides this is a logical move because the `\xintexpr` parser is already based on `\csname...\endcsname` storage of numbers as one token.

The limitation at 5000 digits didn't worry me too much because it was not very realistic to launch computations with thousands of digits... such computations are still slow with 1.2 but less so now. Chains or `\romannumeral` are still used for the gathering of function names and other stuff which I have half-forgotten because the parser does many things.

In the earlier versions we used the `lockscan` macro after a chain of `\romannumeral-0` had ended gathering digits; this uses has been replaced by direct processing inside a `\csname...\endcsname` and the macro is kept only for matters of dummy variables.

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

```
308 \def\xINT_embrace#1{{#1}}%
309 \def\xint_gob_til_! #1!{}% ! with catcode 11
310 \def\xintError:noopening
311 {%
312     \XINT_expandableerror{Extra ) found during balancing, e(X)it before the worst.}%
313 }%
```

\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!

```
314 \def\xintthecoords#1%
315     {\romannumeral`&&\expandafter\XINT_thecoords_a\romannumeral0#1}%
316 \def\XINT_thecoords_a #1#2.#3% #2.=\XINTfloatprint<digits>. etc...
317     {\expanded{\expandafter\XINT_thecoords_b\expanded#2.{#3},!,!,^}}%
318 \def\XINT_thecoords_b #1#2,#3#4,%
319     {\xint_gob_til_! #3\XINT_thecoords_c ! (#1#2, #3#4)\XINT_thecoords_b }%
320 \def\XINT_thecoords_c #1^{}%
```

\xintthespaceseparated 1.4a This is a utility macro which was distributed previously separately for usage with PSTricks `\listplot`

```
321 \def\xintthespaceseparated#1%
322     {\romannumeral`&&\expandafter\xintthespaceseparated_a\romannumeral0#1}%
323 \def\xintthespaceseparated_a #1#2.#3%
324     {\expanded{\expandafter\xintthespaceseparated_b\expanded#2.{#3},!,!,!,!,!,!,!,!,!,^}}%
325 \def\xintthespaceseparated_b #1,#2,#3,#4,#5,#6,#7,#8,#9,%
326     {\xint_gob_til_! #9\xintthespaceseparated_c !%}
```

```

327      #1#2#3#4#5#6#7#8#9%
328      \xintthespaceseparated_b}%
329 \def\xintthespaceseparated_c !#1!#2^{#1}%

```

11.8 Hooks into the numeric parser for usage by the \xintdeffunc symbolic parser

This is new with 1.3 and considerably refactored at 1.4. See «Mysterious stuff».

```

330 \def\xint:NEhook:unpack{\xint_stop_atfirstofone}%
331 \let\xint:NEhook:f:one:from:one\expandafter
332 \let\xint:NEhook:f:one:from:one:direct\empty
333 \let\xint:NEhook:f:one:from:two\expandafter
334 \let\xint:NEhook:f:one:from:two:direct\empty
335 \let\xint:NEhook:x:one:from:two\empty
336 \let\xint:NEhook:x:one:from:twoandone\empty
337 \let\xint:NEhook:f:one:and:opt:direct \empty
338 \let\xint:NEhook:f:tacitzeroifone:direct \empty
339 \let\xint:NEhook:f:iitacitzeroifone:direct \empty
340 \let\xint:NEhook:x:select:obey\empty
341 \let\xint:NEhook:x:listsell\empty
342 \let\xint:NEhook:f:reverse\empty
343 \def\xint:NEhook:f:from:delim:u #1#2^{#1#2}%
344 \def\xint:NEhook:f:noeval:from:braced:u#1#2^{#1{#2}}%
345 \let\xint:NEhook:branch\expandafter
346 \let\xint:NEhook:seqx\empty
347 \let\xint:NEhook:iter\expandafter
348 \let\xint:NEhook:opx\empty
349 \let\xint:NEhook:rseq\expandafter
350 \let\xint:NEhook:iterr\expandafter
351 \let\xint:NEhook:rrseq\expandafter
352 \let\xint:NEhook:x:toblist\empty
353 \let\xint:NEhook:x:mapwithin\empty
354 \let\xint:NEhook:x:ndmapx\empty

```

11.9 \XINT_expr_getnext: fetch some value then an operator and present them to last waiter with the found operator precedence, then the operator, then the value

Big change in 1.1, no attempt to detect braced stuff anymore as the [N] notation is implemented otherwise. Now, braces should not be used at all; one level removed, then \romannumeral-`0 expansion.

Refactored at 1.4 to put expansion of \XINT_expr_gettop after the fetched number, thus avoiding it to have to fetch it (which could happen then multiple times, it was not really important when it was only one token in pre-1.4 xintexpr).

Allow \xintexpr\relax at 1.4.

Refactored at 1.4 the articulation \XINT_expr_getnext/XINT_expr_func/XINT_expr_gettop. For some legacy reason the first token picked by getnext was soon turned to catcode 12 The next ones after the first were not a priori stringified but the first token was, and this made allowing things such as \xintexpr\relax, \xintexpr, \relax, [], 1+(), [:] etc... complicated and requiring each time specific measures.

```

355 \def\xint_expr_getnext #1%

```



```

356 {%
357   \expandafter\XINT_expr_put_op_first\romannumeral`&&%
358   \expandafter\XINT_expr_getnext_a\romannumeral`&&#1%
359 }%
360 \def\XINT_expr_put_op_first #1#2#3{\expandafter#2\expandafter#3\expandafter{#1}}%
361 \def\XINT_expr_getnext_a #1%
362 {%
363   \ifx\relax #1\xint_dothis\XINT_expr_foundprematureend\fi
364   \ifx\XINTfstop#1\xint_dothis\XINT_expr_subexpr\fi
365   \ifcat\relax#1\xint_dothis\XINT_expr_countetc\fi
366   \xint_orthat{}\XINT_expr_getnextfork #1%
367 }%
368 \def\XINT_expr_foundprematureend\XINT_expr_getnextfork #1{{}\xint_c_\relax}%
369 \def\XINT_expr_subexpr #1.#2%
370 {%
371   \expanded{\unexpanded{{#2}}\expandafter}\romannumeral`&&\XINT_expr_getop
372 }%

```

1.2 adds \ht, \dp, \wd and the eTeX font things. 1.4 avoids big nested \if's, simply for code readability

```

373 \def\XINT_expr_countetc\XINT_expr_getnextfork#1%
374 {%
375   \if0\ifx\count#11\fi
376   \ifx\dimen#11\fi
377   \ifx\numexpr#11\fi
378   \ifx\dimexpr#11\fi
379   \ifx\skip#11\fi
380   \ifx\glueexpr#11\fi
381   \ifx\fontdimen#11\fi
382   \ifx\ht#11\fi
383   \ifx\dp#11\fi
384   \ifx\wd#11\fi
385   \ifx\fontcharht#11\fi
386   \ifx\fontcharwd#11\fi
387   \ifx\fontchardp#11\fi
388   \ifx\fontcharic#11\fi 0\expandafter\XINT_expr_fetch_as_number\fi
389   \expandafter\XINT_expr_getnext_a\number #1%
390 }%
391 \def\XINT_expr_fetch_as_number
392   \expandafter\XINT_expr_getnext_a\number #1%
393 {%
394   \expanded{{{\number#1}}\expandafter}\romannumeral`&&\XINT_expr_getop
395 }%

```

This is a key component which is involved in:

- support for \xintdefunc via special handling of parameter character,
- support for skipping over ignored + signs,
- support for Python-like * «unpacking» unary operator (added at 1.4),
- support for [...] nutple constructor (1.4, formerly [...] by itself was like (...)),
- support for numbers starting with a decimal point,
- support for the minus as unary operator of variable precedence level,
- support for sub-expressions inside parenthesis (with possibly tacit multiplication)
- else starting the scan of explicit digits or letters for a number or a function name

```

396 \begingroup
397 \lccode`:=`#
398 \lowercase{\endgroup
399 \def\XINT_expr_getnextfork #1{%
400   \if#1;\xint_dothis {\XINT_expr_scan_macropar ;}\fi
401   \if#1+\xint_dothis \XINT_expr_getnext_a \fi
402   \if#1*\xint_dothis {{}\xint_c_ii^v 0}\fi
403   \if#1[\xint_dothis {{}\xint_c_ii^v \XINT_expr_itself_obracket}\fi
404   \if#1.\xint_dothis {\XINT_expr_startdec}\fi
405   \if#1-\xint_dothis {{}}{-}\fi
406   \if#1(\xint_dothis {{}\xint_c_ii^v (}\fi
407   \xint_orthat {\XINT_expr_scan_nbr_or_func #1}%
408 }}%
409 \def\XINT_expr_scan_macropar #1#2%
410 {%
411   \expandafter{\expandafter{\expandafter#1\expandafter
412     #2\expandafter}\expandafter}\romannumeral`&&\XINT_expr_getop
413 }%

```

11.10 \XINT_expr_scan_nbr_or_func: parsing the integer or decimal number or hexa-decimal number or function name or variable name or special hacky things

11.10.1	Integral part (skipping zeroes)	323
11.10.2	Fractional part	325
11.10.3	Scientific notation	326
11.10.4	Hexadecimal numbers	327
11.10.5	\XINT_expr_scanfunc: collecting names of functions and variables	329
11.10.6	\XINT_expr_func: dispatch to variable replacement or to function execution	330

1.2 release has replaced chains of \romannumeral-`0 by \csname governed expansion. Thus there is no more the limit at about 5000 digits for parsed numbers.

In order to avoid having to lock and unlock in succession to handle the scientific part and adjust the exponent according to the number of digits of the decimal part, the parsing of this decimal part counts on the fly the number of digits it encounters.

There is some slight annoyance with \xintiexpr which should never be given a [n] inside its \csname.<digits>\endcsname storage of numbers (because its arithmetic uses the ii macros which know nothing about the [N] notation). Hence if the parser has only seen digits when hitting something else than the dot or e (or E), it will not insert a [0]. Thus we very slightly compromise the efficiency of \xintexpr and \xintfloatexpr in order to be able to share the same code with \xintiexpr.

Indeed, the parser at this location is completely common to all, it does not know if it is working inside \xintexpr or \xintiexpr. On the other hand if a dot or a e (or E) is met, then the (common) parser has no scruples ending this number with a [n], this will provoke an error later if that was within an \xintiexpr, as soon as an arithmetic macro is used.

As the gathered numbers have no spaces, no pluses, no minuses, the only remaining issue is with leading zeroes, which are discarded on the fly. The hexadecimal numbers leading zeroes are stripped in a second stage by the \xintHexToDec macro.

With 1.2, \xinttheexpr . \relax does not work anymore (it did in earlier releases). There must be digits either before or after the decimal mark. Thus both \xinttheexpr 1.\relax and \xinttheexpr .1\relax are legal.

The ``` syntax is here used for special constructs like ``+`(..)`, ``*`(..)` where `+` or `*` will be treated as functions. Current implementation picks only one token (could have been braced stuff), here it will be `+` or `*`, and via `\XINT_expr_op_`` this then becomes a suitable `\XINT_{expr|iiexpr|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.

Attention at this location #1 was of catcode 12 in all versions prior to 1.4.

Besides using principally `\if` tests, we will assume anyhow that catcodes of digits are 12...

```

414 \catcode96 11 % `
415 \def\XINT_expr_scan_nbr_or_func #1%
416 {%
417   \if "#1\xint_dothis \XINT_expr_scanhex_I\fi
418   \if `#1\xint_dothis {\XINT_expr_onliteral_`}\fi
419   \ifnum \xint_c_ix<1\string#1 \xint_dothis \XINT_expr_startint\fi
420   \xint_orthat \XINT_expr_scanfunc #1%
421 }%
422 \def\XINT_expr_onliteral_` #1#2#3({{#2}\xint_c_ii^v `}%
423 \catcode96 12 % `
424 \def\XINT_expr_startint #1%
425 {%
426   \if #10\expandafter\XINT_expr_gobz_a\else\expandafter\XINT_expr_scanint_a\fi #1%
427 }%
428 \def\XINT_expr_scanint_a #1#2%
429   {\expanded\bgroup{{\iffalse}}}\fi #1% spare a \string
430   \expandafter\XINT_expr_scanint_main\romannumeral`&&@#2}%
431 \def\XINT_expr_gobz_a #1#2%
432   {\expanded\bgroup{{\iffalse}}}\fi
433   \expandafter\XINT_expr_gobz_scanint_main\romannumeral`&&@#2}%
434 \def\XINT_expr_startdec #1%
435   {\expanded\bgroup{{\iffalse}}}\fi
436   \expandafter\XINT_expr_scandec_a\romannumeral`&&@#1}%

```

11.10.1 Integral part (skipping zeroes)

1.2 has modified the code to give highest priority to digits, the accelerating impact is non-negligible. I don't think the doubled `\string` is a serious penalty.

```

437 \def\XINT_expr_scanint_main #1%
438 {%
439   \ifcat \relax #1\expandafter\XINT_expr_scanint_hit_cs \fi
440   \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanint_next\fi
441   #1\XINT_expr_scanint_again
442 }%
443 \def\XINT_expr_scanint_again #1%
444 {%
445   \expandafter\XINT_expr_scanint_main\romannumeral`&&@#1%
446 }%
447 \def\XINT_expr_scanint_hit_cs \ifnum#1\fi#2\XINT_expr_scanint_again
448 {%
449   \iffalse{{{\fi}}\expandafter}\romannumeral`&&@\XINT_expr_getop#2%
450 }%

```

With 1.2d the tacit multiplication in front of a variable name or function name is now done with a higher precedence, intermediate between the common one of $*$ and $/$ and the one of $^$. Thus $x/2y$ is like $x/(2y)$, but x^2y is like x^2*y and $2y!$ is not $(2y)!$ but $2*y!$.

Finally, 1.2d has moved away from the `_scan` macros all the business of the tacit multiplication in one unique place via `\XINT_expr_getop`. For this, the ending token is not first given to `\string` as was done earlier before handing over back control to `\XINT_expr_getop`. Earlier we had to identify the catcode 11 ! signaling a sub-expression here. With no `\string` applied we can do it in `\XINT_expr_getop`. As a corollary of this displacement, parsing of big numbers should be a tiny bit faster now.

Extended for 1.2l to ignore underscore character `_` if encountered within digits; so it can serve as separator for better readability.

It is not obvious at 1.4 to support `[]` for three things: packing, slicing, ... and raw `xintfrac` syntax `A/B[N]`. The only good way would be to actually really separate completely `\xintexpr`, `\xintfloatexpr` and `\xintiexpr` code which would allow to handle both `/` and `[]` from `A/B[N]` as we handle `e` and `E`. But triplicating the code is something I need to think about. It is not possible as in pre 1.4 to consider `[]` only as an operator of same precedence as multiplication and division which was the way we did this, but we can use the technique of fake operators. Thus we intercept hitting a `[` here, which is not too much of a problem as anyhow we dropped temporarily `3*[1,2,3]+5` syntax so we don't have to worry that `3[1,2,3]` should do tacit multiplication. I think only way in future will be to really separate the code of the three parsers (or drop entirely support for `A/B[N]`; as 1.4 has modified output of `\xinteval` to not use this notation this is not too dramatic).

Anyway we find a way to inject here the former handling of `[N]`, which will use a delimited macro to directly fetch until the closing `]`. We do still need some fake operator because `A/B[N]` is (A/B) times 10^N and the `/B` is allowed to be missing. We hack this using the `which` which is not used currently as operator elsewhere in the syntax and need to hook into `\XINT_expr_getop_b`. No finally I use the null char. It must be of catcode 12.

```

451 \def\XINT_expr_scanint_next #1\XINT_expr_scanint_again
452 {%
453   \if    [#1\xint_dothis\XINT_expr_rawxintfrac\fi
454   \if    _#1\xint_dothis\XINT_expr_scanint_again\fi
455   \if    e#1\xint_dothis{[\the\numexpr0\XINT_expr_scanexp_a +}\fi
456   \if    E#1\xint_dothis{[\the\numexpr0\XINT_expr_scanexp_a +}\fi
457   \if    .#1\xint_dothis{\XINT_expr_startdec_a .}\fi
458   \xint_orthat
459   {\iffalse{{\fi}}\expandafter}\romannumeral`&&\XINT_expr_getop#1}%
460 }%
461 \def\XINT_expr_rawxintfrac
462 {%
463   \iffalse{{\fi}}\expandafter}\csname XINT_expr_precedence_&&\endcsname&&%
464 }%
465 \def\XINT_expr_gobz_scanint_main #1%
466 {%
467   \ifcat \relax #1\expandafter\XINT_expr_gobz_scanint_hit_cs\fi
468   \ifnum\xint_c_x<1\string#1 \else\expandafter\XINT_expr_gobz_scanint_next\fi
469   #1\XINT_expr_scanint_again
470 }%
471 \def\XINT_expr_gobz_scanint_again #1%
472 {%
473   \expandafter\XINT_expr_gobz_scanint_main\romannumeral`&&#1%
474 }%
475 \def\XINT_expr_gobz_scanint_hit_cs\ifnum#1\fi#2\XINT_expr_scanint_again

```

```

476 {%
477     0\iffalse{{{\fi}}\expandafter}\romannumeral`&&\XINT_expr_getop#2%
478 }%
479 \def\XINT_expr_gobz_scanint_next #1\XINT_expr_scanint_again
480 {%
481     \if [#1\xint_dothis{\expandafter0\XINT_expr_rawxintfrac}\fi
482     \if _#1\xint_dothis\XINT_expr_gobz_scanint_again\fi
483     \if e#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
484     \if E#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
485     \if .#1\xint_dothis{\XINT_expr_gobz_startdec_a .}\fi
486     \if 0#1\xint_dothis\XINT_expr_gobz_scanint_again\fi
487     \xint_orthat
488     {0\iffalse{{{\fi}}\expandafter}\romannumeral`&&\XINT_expr_getop#1}%
489 }%

```

11.10.2 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 ?

```

490 \def\XINT_expr_startdec_a .#1%
491 {%
492     \expandafter\XINT_expr_scandec_a\romannumeral`&&#1%
493 }%
494 \def\XINT_expr_scandec_a #1%
495 {%
496     \if .#1\xint_dothis{\iffalse{{{\fi}}\expandafter}%
497         \romannumeral`&&\XINT_expr_getop..}\fi
498     \xint_orthat {\XINT_expr_scandec_main 0.#1}%
499 }%
500 \def\XINT_expr_gobz_startdec_a .#1%
501 {%
502     \expandafter\XINT_expr_gobz_scandec_a\romannumeral`&&#1%
503 }%
504 \def\XINT_expr_gobz_scandec_a #1%
505 {%
506     \if .#1\xint_dothis
507     {0\iffalse{{{\fi}}\expandafter}\romannumeral`&&\XINT_expr_getop..}\fi
508     \xint_orthat {\XINT_expr_gobz_scandec_main 0.#1}%
509 }%
510 \def\XINT_expr_scandec_main #1.#2%
511 {%
512     \ifcat \relax #2\expandafter\XINT_expr_scandec_hit_cs\fi
513     \ifnum\xint_c_ix<1\string#2 \else\expandafter\XINT_expr_scandec_next\fi
514     #2\expandafter\XINT_expr_scandec_again\the\numexpr #1-\xint_c_i.%
515 }%
516 \def\XINT_expr_scandec_again #1.#2%
517 {%
518     \expandafter\XINT_expr_scandec_main
519     \the\numexpr #1\expandafter.\romannumeral`&&#2%

```

```

520 }%
521 \def\XINT_expr_scandec_hit_cs\ifnum#1\fi
522   #2\expandafter\XINT_expr_scandec_again\the\numexpr#3-\xint_c_i.%
523 {%
524   [#3]\iffalse{ {\fi}}\expandafter\romannumeral`&&\XINT_expr_getop#2%
525 }%
526 \def\XINT_expr_scandec_next #1#2\the\numexpr#3-\xint_c_i.%
527 {%
528   \if _#1\xint_dothis{\XINT_expr_scandec_again#3.}\fi
529   \if e#1\xint_dothis{[\the\numexpr#3\XINT_expr_scanexp_a +}\fi
530   \if E#1\xint_dothis{[\the\numexpr#3\XINT_expr_scanexp_a +}\fi
531   \xint_orthat
532   {[#3]\iffalse{ {\fi}}\expandafter\romannumeral`&&\XINT_expr_getop#1}%
533 }%
534 \def\XINT_expr_gobz_scandec_main #1.#2%
535 {%
536   \ifcat \relax #2\expandafter\XINT_expr_gobz_scandec_hit_cs\fi
537   \ifnum\xint_c_ix<1|string#2 \else\expandafter\XINT_expr_gobz_scandec_next\fi
538   \if0#2\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
539   {\expandafter\XINT_expr_gobz_scandec_main}%
540   {#2\expandafter\XINT_expr_scandec_again}\the\numexpr#1-\xint_c_i.%
541 }%
542 \def\XINT_expr_gobz_scandec_hit_cs \ifnum#1\fi\if0#2#3\xint_c_i.%
543 {%
544   0[0]\iffalse{ {\fi}}\expandafter\romannumeral`&&\XINT_expr_getop#2%
545 }%
546 \def\XINT_expr_gobz_scandec_next\if0#1#2\fi #3\numexpr#4-\xint_c_i.%
547 {%
548   \if _#1\xint_dothis{\XINT_expr_gobz_scandec_main #4.}\fi
549   \if e#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
550   \if E#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
551   \xint_orthat
552   {0[0]\iffalse{ {\fi}}\expandafter\romannumeral`&&\XINT_expr_getop#1}%
553 }%

```

11.10.3 Scientific notation

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

```

554 \def\XINT_expr_scanexp_a #1#2%
555 {%
556   #1\expandafter\XINT_expr_scanexp_main\romannumeral`&&#2%
557 }%
558 \def\XINT_expr_scanexp_main #1%
559 {%
560   \ifcat \relax #1\expandafter\XINT_expr_scanexp_hit_cs\fi
561   \ifnum\xint_c_ix<1|string#1 \else\expandafter\XINT_expr_scanexp_next\fi
562   #1\XINT_expr_scanexp_again
563 }%
564 \def\XINT_expr_scanexp_again #1%
565 {%

```

```

566 \expandafter\XINT_expr_scanexp_main_b\romannumeral`&&@#1%
567 }%
568 \def\XINT_expr_scanexp_hit_cs\ifnum#1\fi#2\XINT_expr_scanexp_again
569 {%
570 ]\iffalse{{{\fi}}\expandafter}\romannumeral`&&\XINT_expr_getop#2%
571 }%
572 \def\XINT_expr_scanexp_next #1\XINT_expr_scanexp_again
573 {%
574 \if _#1\xint_dothis \XINT_expr_scanexp_again \fi
575 \if +#1\xint_dothis {\XINT_expr_scanexp_a +}\fi
576 \if -#1\xint_dothis {\XINT_expr_scanexp_a -}\fi
577 \xint_orthat
578 {\iffalse{{{\fi}}\expandafter}\romannumeral`&&\XINT_expr_getop#1}%
579 }%
580 \def\XINT_expr_scanexp_main_b #1%
581 {%
582 \ifcat \relax #1\expandafter\XINT_expr_scanexp_hit_cs_b\fi
583 \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanexp_next_b\fi
584 #1\XINT_expr_scanexp_again_b
585 }%
586 \def\XINT_expr_scanexp_hit_cs_b\ifnum#1\fi#2\XINT_expr_scanexp_again_b
587 {%
588 ]\iffalse{{{\fi}}\expandafter}\romannumeral`&&\XINT_expr_getop#2%
589 }%
590 \def\XINT_expr_scanexp_again_b #1%
591 {%
592 \expandafter\XINT_expr_scanexp_main_b\romannumeral`&&@#1%
593 }%
594 \def\XINT_expr_scanexp_next_b #1\XINT_expr_scanexp_again_b
595 {%
596 \if _#1\xint_dothis\XINT_expr_scanexp_again\fi
597 \xint_orthat
598 {\iffalse{{{\fi}}\expandafter}\romannumeral`&&\XINT_expr_getop#1}%
599 }%

```

11.10.4 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.

```

600 \def\XINT_expr_hex_in #1.#2#3;%
601 {%
602 \expanded{{{\if#2>%
603 \xintHexToDec{#1}%
604 \else
605 \xintiiMul{\xintiiPow{625}{\xintLength{#3}}}{\xintHexToDec{#1#3}}}%
606 [\the\numexpr-4*\xintLength{#3}]}%
607 \fi}}\expandafter}\romannumeral`&&\XINT_expr_getop
608 }%

```

```

609 \def\XINT_expr_scanhex_I #1% #1="
610 {%
611     \expandafter\XINT_expr_hex_in\expanded\bgroup\XINT_expr_scanhexI_a
612 }%
613 \def\XINT_expr_scanhexI_a #1%
614 {%
615     \ifcat #1\relax\xint_dothis{.;>\iffalse{\fi}#1}\fi
616     \xint_orthat {\XINT_expr_scanhexI_aa #1}%
617 }%
618 \def\XINT_expr_scanhexI_aa #1%
619 {%
620     \if\ifnum`#1>`/
621         \ifnum`#1>`9
622         \ifnum`#1>`@
623         \ifnum`#1>`F
624         0\else1\fi\else0\fi\else1\fi\else0\fi 1%
625         \expandafter\XINT_expr_scanhexI_b
626     \else
627         \if _#1\xint_dothis{\expandafter\XINT_expr_scanhexI_bgob}\fi
628         \if .#1\xint_dothis{\expandafter\XINT_expr_scanhex_transition}\fi
629         \xint_orthat {\xint_afterfi {.>\iffalse{\fi}}}%
630     \fi
631     #1%
632 }%
633 \def\XINT_expr_scanhexI_b #1#2%
634 {%
635     #1\expandafter\XINT_expr_scanhexI_a\romannumeral`&&@#2%
636 }%
637 \def\XINT_expr_scanhexI_bgob #1#2%
638 {%
639     \expandafter\XINT_expr_scanhexI_a\romannumeral`&&@#2%
640 }%
641 \def\XINT_expr_scanhex_transition .#1%
642 {%
643     \expandafter.\expandafter.\expandafter
644     \XINT_expr_scanhexII_a\romannumeral`&&@#1%
645 }%
646 \def\XINT_expr_scanhexII_a #1%
647 {%
648     \ifcat #1\relax\xint_dothis{;>\iffalse{\fi}#1}\fi
649     \xint_orthat {\XINT_expr_scanhexII_aa #1}%
650 }%
651 \def\XINT_expr_scanhexII_aa #1%
652 {%
653     \if\ifnum`#1>`/
654         \ifnum`#1>`9
655         \ifnum`#1>`@
656         \ifnum`#1>`F
657         0\else1\fi\else0\fi\else1\fi\else0\fi 1%
658         \expandafter\XINT_expr_scanhexII_b
659     \else
660         \if _#1\xint_dothis{\expandafter\XINT_expr_scanhexII_bgob}\fi

```



```

661      \xint_orthat{\xint_afterfi {;\iffalse{\fi}}}%
662      \fi
663      #1%
664 }%
665 \def\xINT_expr_scanhexII_b #1#2%
666 {%
667      #1\expandafter\xINT_expr_scanhexII_a\romannumeral`&&@#2%
668 }%
669 \def\xINT_expr_scanhexII_bgob #1#2%
670 {%
671      \expandafter\xINT_expr_scanhexII_a\romannumeral`&&@#2%
672 }%

```

11.10.5 \XINT_expr_scanfunc: collecting names of functions and variables

At 1.4 the first token left over in string has not been submitted to `\string`. We also know it is not a control sequence. So we can test `catcode` to identify if operator is found. And it is allowed to hit some operator such as a closing parenthesis we will then insert the «nil» value (which however can cause breakage of arithmetic operations, although `xintfrac.sty` converts empty to 0).

The `@` causes a problem because it must work with both `catcode 11` or `12`.

The `_` can be used internally for starting variables but it will have `catcode 11` then.

There was prior to 1.4 solely the dispatch in `\XINT_expr_scanfunc_b` but now we do it immediately and issue `\XINT_expr_func` only in certain cases.

But we have to be careful that `!(...)` and `?(...)` are part of the syntax and genuine functions. Because we now do earlier to `getop` we must filter them out.

```

673 \def\xINT_expr_scanfunc #1%
674 {%
675      \if 1\ifcat a#10\fi\if @#10\fi\if !#10\fi\if ?#10\fi 1%
676      \expandafter\xint_firstoftwo
677      \else\expandafter\xint_secondoftwo
678      \fi
679      {\expandafter{\expandafter}\romannumeral`&&\XINT_expr_getop#1}%
680      {\expandafter\xINT_expr_func\expanded\bgob#1\xINT_expr_scanfunc_a}%
681 }%
682 \def\xINT_expr_scanfunc_a #1%
683 {%
684      \expandafter\xINT_expr_scanfunc_b\romannumeral`&&@#1%
685 }%

```

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.

```

686 \def\XINT_expr_scanfunc_b #1%
687 {%
688   \ifcat \relax#1\xint_dothis{\iffalse{\fi}(_#1)\fi
689   \if (#1\xint_dothis{\iffalse{\fi}(`)\fi
690   \if 1\ifcat a#10\fi
691     \ifnum\xint_c_ix<1\string#1 0\fi
692     \if @#10\fi
693     \if _#10\fi
694     1%
695     \xint_dothis{\iffalse{\fi}(_#1)\fi
696   \xint_orthat {#1\XINT_expr_scanfunc_a}%
697 }%

```

11.10.6 `\XINT_expr_func`: dispatch to variable replacement or to function execution

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_ii^v` 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_ii^v #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_ii^v` is there because `_op_`` must know in which parser it works. Dispendious for `_`. Hence I modify for 1.2d.

```

698 \def\XINT_expr_func #1(#2{\if _#2\xint_dothis{\XINT_expr_op_{#1}}\fi
699   \xint_orthat{{#1}\xint_c_ii^v #2}}%

```

11.11 `\XINT_expr_op_``: launch function or pseudo-function, or evaluate variable and insert operator of multiplication in front of parenthesized contents

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 "onlital" is also used to disambiguate a variable followed by an opening parenthesis from a function and then apply tacit multiplication. 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 `onlital_<name>` associated macro. This used to be decided much earlier at the time of `\XINT_expr_func`.

The advantage of 1.2c code is that the same name can be used for a variable or a function.

```

700 \def\XINT_tmpa #1#2#3{%
701   \def #1##1%
702   {%
703     \ifcsname XINT_#3_func_##1\endcsname
704     \csname XINT_#3_func_##1\expandafter\endcsname
705     \romannumeral`&&\expandafter#2%
706   \else
707     \ifcsname XINT_expr_onlital_##1\endcsname
708     \csname XINT_expr_onlital_##1\expandafter\expandafter\expandafter
709     \endcsname
710   \else
711     \csname XINT_#3_func_\XINT_expr_unknown_function {##1}%
712     \expandafter\endcsname
713     \romannumeral`&&\expandafter\expandafter\expandafter#2%
714   \fi
715 \fi
716 }%
717 }%
718 \def\XINT_expr_unknown_function #1%
719   {\XINT_expandableerror{"#1" is unknown as function. (I)nsert correct name:}}%
720 \def\XINT_expr_func_ #1#2#3{#1#2{{0}}}%
721 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
722   \expandafter\XINT_tmpa
723   \csname XINT_#1_op_\` \expandafter\endcsname
724   \csname XINT_#1_oparen\endcsname
725   {#1}%
726 }%

```

11.12 `\XINT_expr_op__`: replace a variable by its value and then fetch next operator

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`.

At 1.4 `\XINT_expr_getop` is launched with accumulated result on its left. But the `omit` and `abort` keywords are implemented via dummy variables which rely on possibility to modify upstream tokens. If we did here something such as `_var_#1\expandafter\endcsname\romannumeral`^^@\XINT_expr_getop` the premature expansion of `getop` would break things. Thus we revert to former code which put `\XINT_expr_getop` (call it `_legacy`) in front of variable expansion (in `xintexpr < 1.4` this expanded to a single token so the overhead was not serious).

Abusing variables to manipulate token stream is a bit bad, usually I prefer functions for this (such as the `break()` function) but then I have define 3 macros for the 3 parsers.

The situation here is not satisfactory. But 1.4 has to be released now.

```

727 \def\XINT_expr_op__ #1% op__ with two _'s
728 {%
729   \ifcsname XINT_expr_var_#1\endcsname

```

```

730 \expandafter\expandafter\expandafter\XINT_expr_getop_legacy
731 \csname XINT_expr_var_#1\expandafter\endcsname
732 \else
733 \expandafter\expandafter\expandafter\XINT_expr_getop_legacy
734 \csname XINT_expr_var_\XINT_expr_unknown_variable {#1}%
735 \expandafter\endcsname
736 \fi
737 }%
738 \def\XINT_expr_unknown_variable #1%
739 {\XINT_expandableerror {"#1" is unknown as a variable. (I)nsert correct one:}}%
740 \def\XINT_expr_var_{\0}%
741 \let\XINT_flexpr_op__ \XINT_expr_op__
742 \let\XINT_iexpr_op__ \XINT_expr_op__
743 \def\XINT_expr_getop_legacy #1%
744 {%
745 \expanded{\unexpanded{{#1}}\expandafter}\romannumeral`&&\XINT_expr_getop
746 }%

```

11.13 \XINT_expr_getop: fetch the next operator or closing parenthesis or end of expression

Release 1.1 implements multi-character operators.

1.2d adds tacit multiplication also in front of variable or functions names starting with a letter, not only a @ or a _ as was already the case. This is for (x+y)z situations. It also applies higher precedence in cases like x/2y or x/2@, or x/2max(3,5), or x/2\xintexpr 3\relax.

In fact, finally I decide that all sorts of tacit multiplication will always use the higher precedence.

Indeed I hesitated somewhat: with the current code one does not know if \XINT_expr_getop as invoked after a closing parenthesis or because a number parsing ended, and I felt distinguishing the two was unneeded extra stuff. This means cases like (a+b)/(c+d)(e+f) will first multiply the last two parenthesized terms.

The ! starting a sub-expression must be distinguished from the post-fix ! for factorial, thus we must not do a too early \string. In versions < 1.2c, the catcode 11 ! had to be identified in all branches of the number or function scans. Here it is simply treated as a special case of a letter.

1.2q adds tacit multiplication in cases such as (1+1)3 or 5!7!

1.4 has simplified coding here as \XINT_expr_getop expansion happens at a time when a fetched value has already being stored.

```

747 \def\XINT_expr_getop #1%
748 {%
749 \expandafter\XINT_expr_getop_a\romannumeral`&&#1%
750 }%
751 \catcode`* 11
752 \def\XINT_expr_getop_a #1%
753 {%
754 \ifx \relax #1\xint_dothis\xint_firstofthree\fi
755 \ifcat \relax #1\xint_dothis\xint_secondofthree\fi
756 \ifnum\xint_c_ix<1\string#1 \xint_dothis\xint_secondofthree\fi
757 \if :#1\xint_dothis \xint_thirdofthree\fi
758 \if _#1\xint_dothis \xint_secondofthree\fi
759 \if @#1\xint_dothis \xint_secondofthree\fi
760 \if (#1\xint_dothis \xint_secondofthree\fi %)

```

```

761 \ifcat a#1\xint_dothis \xint_secondofthree\fi
762 \xint_orthat \xint_thirdofthree
763 {\XINT_expr_foundend}%

```

tacit multiplication with higher precedence.

```

764 {\XINT_expr_precedence_*** *#1}%
765 {\expandafter\XINT_expr_getop_b \string#1}%
766}%
767\catcode`* 12

```

\relax is a place holder here.

```

768\def\XINT_expr_foundend {\xint_c_ \relax}%

```

? 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.

null char is used as hack to implement A/B[N] raw input at 1.4. See also \XINT_expr_scanint_c.

```

769\def\XINT_expr_getop_b #1%
770{%
771 \if &&@#1\xint_dothis{\csname XINT_expr_precedence_&&\endcsname&&}\fi
772 \if '#1\xint_dothis{\XINT_expr_binopwrld }\fi
773 \if ?#1\xint_dothis{\XINT_expr_precedence_? ?}\fi
774 \xint_orthat {\XINT_expr_scanop_a #1}%
775}%
776\def\XINT_expr_binopwrld #1'%
777{%
778 \expandafter\XINT_expr_foundop_a
779 \csname XINT_expr_itself_\xint_zapspaces #1 \xint_gobble_i\endcsname
780}%
781\def\XINT_expr_scanop_a #1#2%
782{%
783 \expandafter\XINT_expr_scanop_b\expandafter#1\romannumeral`&&@#2%
784}%
785\def\XINT_expr_scanop_b #1#2%
786{%
787 \ifcat#2\relax\xint_dothis{\XINT_expr_foundop_a #1#2}\fi
788 \ifcsname XINT_expr_itself_#1#2\endcsname
789 \xint_dothis
790 {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#2\endcsname}\fi
791 \xint_orthat {\XINT_expr_foundop_a #1#2}%
792}%
793\def\XINT_expr_scanop_c #1#2%
794{%
795 \expandafter\XINT_expr_scanop_d\expandafter#1\romannumeral`&&@#2%
796}%
797\def\XINT_expr_scanop_d #1#2%
798{%
799 \ifcat#2\relax \xint_dothis{\XINT_expr_foundop #1#2}\fi
800 \ifcsname XINT_expr_itself_#1#2\endcsname
801 \xint_dothis

```

```

802      {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#2\endcsname }\fi
803 \xint_orthat {\csname XINT_expr_precedence_#1\endcsname #1#2}%
804 }%
805 \def\XINT_expr_foundop_a #1%
806 {%
807   \ifcsname XINT_expr_precedence_#1\endcsname
808     \csname XINT_expr_precedence_#1\expandafter\endcsname
809     \expandafter #1%
810   \else
811     \xint_afterfi{\XINT_expr_getop\romannumeral0%
812     \XINT_expandableerror
813     {"#1" is unknown as operator. (I)nsert one:} }%<<deliberate space
814   \fi
815 }%
816 \def\XINT_expr_foundop #1{\csname XINT_expr_precedence_#1\endcsname #1}%

```

11.14 Expansion spanning; opening and closing parentheses

These comments apply to all definitions coming next relative to execution of operations from parsing of syntax.

Refactored (and unified) at 1.4. In particular the 1.4 scheme uses `op`, `exec`, `check-`, and `checkp`. Formerly it was `until_a` (`check-`) and `until_b` (now split into `checkp` and `exec`).

This way neither `check-` nor `checkp` have to grab the accumulated number so far (top of stack if you like) and besides one never has to go back to `check-` from `checkp` (and neither from `check-`).

Prior to 1.4, accumulated intermediate results were stored as one token, but now we have to use `\expanded` to propagate expansion beyond possibly arbitrary long braced nested data. With the 1.4 refactoring we do this only once and only grab a second time the data if we actually have to act upon it.

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.

Exceptionally, the `check-` is here abbreviated to `check`.

```

817 \catcode`) 11
818 \def\XINT_tmpa #1#2#3#4#5#6%
819 {%
820   \def#1% start
821   {%
822     \expandafter#2\romannumeral`&&\XINT_expr_getnext
823   }%
824   \def#2##1% check
825   {%
826     \xint_UDsignfork
827     ##1{\expandafter#3\romannumeral`&&@#4}%
828     -{#3##1}%
829   \krof
830   }%
831   \def#3##1##2% checkp
832   {%
833     \ifcase ##1%
834       \expandafter\XINT_expr_done
835     \or\expandafter#5%
836     \else

```

```

837         \expandafter#3\romannumeral`&&@\csname XINT_#6_op_##2\expandafter\endcsname
838     \fi
839 }%
840 \def#5%
841 {%
842     \XINT_expandableerror
843     {An extra ) has been removed. Hit Return, fingers crossed.}%
844     \expandafter#2\romannumeral`&&@\expandafter\XINT_expr_put_op_first
845     \romannumeral`&&@\XINT_expr_getop_legacy
846 }%
847 }%
848 \let\XINT_expr_done\space
849 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
850     \expandafter\XINT_tmpa
851     \csname XINT_#1_start\expandafter\endcsname
852     \csname XINT_#1_check\expandafter\endcsname
853     \csname XINT_#1_checkp\expandafter\endcsname
854     \csname XINT_#1_op_-xii\expandafter\endcsname
855     \csname XINT_#1_extra_\endcsname
856     {#1}%
857 }%

```

Here also we take some shortcuts relative to general philosophy and have no explicit exec macro.

```

858 \def\XINT_tmpa #1#2#3#4#5#6#7%
859 {%
860     \def #1##1% op_(
861     {%
862         \expandafter #4\romannumeral`&&@\XINT_expr_getnext
863     }%
864     \def #2##1% op_)
865     {%
866         \expanded{\unexpanded{\XINT_expr_put_op_first{##1}}\expandafter}\romannumeral`&&@\XINT_expr_geto
867     }%
868     \def #3% oparen
869     {%
870         \expandafter #4\romannumeral`&&@\XINT_expr_getnext
871     }%
872     \def #4##1% check-
873     {%
874         \xint_UDsignfork
875         ##1{\expandafter#5\romannumeral`&&@#6}%
876         -{#5##1}%
877         \krof
878     }%
879     \def #5##1##2% checkp
880     {%
881         \ifcase ##1\expandafter\XINT_expr_missing_
882         \or \csname XINT_#7_op_##2\expandafter\endcsname
883         \else
884             \expandafter #5\romannumeral`&&@\csname XINT_#7_op_##2\expandafter\endcsname
885         \fi
886     }%

```

```

887 }%
888 \def\XINT_expr_missing_
889   {\XINT_expandableerror{Sorry to report a missing ) at the end of this journey.}%
890   \xint_c_ \XINT_expr_done }%
891 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
892   \expandafter\XINT_tmpa
893   \csname XINT_#1_op_\expandafter\endcsname
894   \csname XINT_#1_op_\expandafter\endcsname
895   \csname XINT_#1_oparen\expandafter\endcsname
896   \csname XINT_#1_check_\expandafter\endcsname
897   \csname XINT_#1_checkp_\expandafter\endcsname
898   \csname XINT_#1_op_-xii\endcsname
899   {#1}%
900 }%
901 \let\XINT_expr_precedence_\xint_c_i
902 \catcode`) 12

```

11.15 The comma as binary operator

New with 1.09a. Refactored at 1.4.

```

903 \def\XINT_tmpa #1#2#3#4#5#6%
904 {%
905   \def #1##1% \XINT_expr_op_,
906   {%
907     \expanded{\unexpanded{#2{##1}}\expandafter}%
908     \romannumeral`&&\expandafter#3\romannumeral`&&\XINT_expr_getnext
909   }%
910   \def #2##1##2##3##4{##2##3{##1##4}}% \XINT_expr_exec_,
911   \def #3##1% \XINT_expr_check_,
912   {%
913     \xint_UDsignfork
914     ##1{\expandafter#4\romannumeral`&&@#5}%
915     -{#4##1}%
916     \krof
917   }%
918   \def #4##1##2% \XINT_expr_checkp_,
919   {%
920     \ifnum ##1>\xint_c_iii
921       \expandafter#4%
922       \romannumeral`&&\csname XINT_#6_op_##2\expandafter\endcsname
923     \else
924       \expandafter##1\expandafter##2%
925     \fi
926   }%
927 }%
928 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
929 \expandafter\XINT_tmpa
930   \csname XINT_#1_op_\expandafter\endcsname
931   \csname XINT_#1_exec_\expandafter\endcsname
932   \csname XINT_#1_check_\expandafter\endcsname
933   \csname XINT_#1_checkp_\expandafter\endcsname
934   \csname XINT_#1_op_-xii\endcsname {#1}%

```



```
935 }%
936 \expandafter\let\csname XINT_expr_precedence_,\endcsname\xint_c_iii
```

11.16 The minus as prefix operator of variable precedence level

Inherits the precedence level of the previous infix operator. Refactored at 1.4

```
937 \def\XINT_tmpb #1#2#3#4#5#6#7%
938 {%
939   \def #1% \XINT_expr_op_-<level>
940   {%
941     \expandafter #2\romannumeral`&&\expandafter#3%
942     \romannumeral`&&\XINT_expr_getnext
943   }%
944   \def #2##1##2##3% \XINT_expr_exec_-<level>
945   {%
946     \expandafter ##1\expandafter ##2\expandafter
947     {%
948       \romannumeral`&&\XINT:NEhook:f:one:from:one
949       {\romannumeral`&&#7##3}%
950     }%
951   }%
952   \def #3##1% \XINT_expr_check_-<level>
953   {%
954     \xint_UDsignfork
955     ##1{\expandafter #4\romannumeral`&&#1}%
956     -{#4##1}%
957     \krof
958   }%
959   \def #4##1##2% \XINT_expr_checkp_-<level>
960   {%
961     \ifnum ##1>#5%
962       \expandafter #4%
963       \romannumeral`&&\csname XINT_#6_op_##2\expandafter\endcsname
964     \else
965       \expandafter ##1\expandafter ##2%
966     \fi
967   }%
968 }%
969 \def\XINT_tmpa #1#2#3%
970 {%
971   \expandafter\XINT_tmpb
972   \csname XINT_#1_op_-#3\expandafter\endcsname
973   \csname XINT_#1_exec_-#3\expandafter\endcsname
974   \csname XINT_#1_check_-#3\expandafter\endcsname
975   \csname XINT_#1_checkp_-#3\expandafter\endcsname
976   \csname xint_c_#3\endcsname {#1}#2%
977 }%
```

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.

```
978 \xintApplyInline{\XINT_tmpa {expr}\xintOpp}{\xii}{\xiv}{\xvi}{\xviii}}%
```

```
979 \xintApplyInline{\XINT_tmpa {flexpr}\xintOpp}{\xii}{xiv}{xvi}{xviii}}%
980 \xintApplyInline{\XINT_tmpa {iiexpr}\xintiiOpp}{\xii}{xiv}{xvi}{xviii}}%
```

11.17 The * as Python-like «unpacking» prefix operator

New with 1.4. Prior to 1.4 the internal data structure was the one of \csname encapsulated comma separated numbers. No hierarchical structure was (easily) possible. At 1.4, we can use TeX braces because there is no detokenization to catcode 12.

```
981 \def\XINT_tmpa#1#2#3%
982 {%
983   \def#1##1{\expandafter#2\romannumeral`&&\XINT_expr_getnext}%
984   \def#2##1##2%
985   {%
986     \ifnum ##1>\xint_c_xx
987       \expandafter #2%
988       \romannumeral`&&\csname XINT_#3_op_##2\expandafter\endcsname
989     \else
990       \expandafter##1\expandafter##2\romannumeral0\expandafter\XINT:NEhook:unpack
991     \fi
992   }%
993 }%
994 \xintFor* #1 in {{expr}{flexpr}{iiexpr}}:
995   {\expandafter\XINT_tmpa\csname XINT_#1_op_0\expandafter\endcsname
996     \csname XINT_#1_until_unpack\endcsname {#1}}%
```

11.18 Infix operators

11.18.1	&&, , <, >, ==, <=, >=, !=, //, /:, +, -, *, /, ^, **, 'and', 'or', 'xor', and 'mod'	338
11.18.2	.., ..[, and].. as infix operators	341
11.18.3	Support macros for .., ..[and]..	342

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.

```
997 \catcode`& 12
998 \xintFor* #1 in {{==}{<=}{>=}{!=}{&&}{||}{**}{//}{/:}{..}{..[]}{.}{}{}}%
999   \do {\expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}}%
1000 \catcode`& 7
1001 \expandafter\let\csname XINT_expr_precedence_***\endcsname \xint_c_xvi
```

11.18.1 &&, ||, <, >, ==, <=, >=, !=, //, /:, +, -, *, /, ^, **, 'and', 'or', 'xor', and 'mod'

Usage of & and | is deprecated and only && and || should be used.

```
1002 \def\XINT_expr_defbin_c #1#2#3#4#5#6#7#8#9%
1003 {%
1004   \def #1##1% \XINT_expr_op_<op>
1005   {%
1006     \expanded{\unexpanded{#2{##1}}\expandafter}%
1007     \romannumeral`&&\expandafter#3\romannumeral`&&\XINT_expr_getnext
1008   }%
1009   \def #2##1##2##3##4% \XINT_expr_exec_<op>
```

```

1010 {%
1011   \expandafter##2\expandafter##3\expandafter
1012   {%9{\romannumeral`&&@#6##1##4}}%
1013 }%
1014 \def #3##1% \XINT_expr_check-_{<op>
1015 {%
1016   \xint_UDsignfork
1017   ##1{\expandafter#4\romannumeral`&&@#5}%
1018   -{#4##1}%
1019   \krof
1020 }%
1021 \def #4##1##2% \XINT_expr_checkp_{<op>
1022 {%
1023   \ifnum ##1>#7%
1024     \expandafter#4%
1025     \romannumeral`&&@\csname XINT_#8_op_##2\expandafter\endcsname
1026   \else
1027     \expandafter ##1\expandafter ##2%
1028   \fi
1029 }%
1030 }%
1031 \def\XINT_expr_defbin_b #1#2#3#4#5%
1032 {%
1033   \expandafter\XINT_expr_defbin_c
1034   \csname XINT_#1_op_#2\expandafter\endcsname
1035   \csname XINT_#1_exec_#2\expandafter\endcsname
1036   \csname XINT_#1_check-_{#2\expandafter\endcsname
1037   \csname XINT_#1_checkp_#2\expandafter\endcsname
1038   \csname XINT_#1_op_-#4\expandafter\endcsname
1039   \csname #5\expandafter\endcsname
1040   \csname XINT_expr_precedence_#2\endcsname
1041   {%1}{\romannumeral`&&@\XINT:NEhook:f:one:from:two}%
1042   \expandafter % done 3 times but well
1043   \let\csname XINT_expr_precedence_#2\expandafter\endcsname
1044   \csname xint_c_#3\endcsname
1045 }%
1046 \catcode`& 12
1047 \XINT_expr_defbin_b {expr} {||} {vi}{xii} {xintOR}%
1048 \XINT_expr_defbin_b {flexpr}{||} {vi}{xii} {xintOR}%
1049 \XINT_expr_defbin_b {iiexpr}{||} {vi}{xii} {xintOR}%
1050 \XINT_expr_defbin_b {expr} {&&} {viii}{xii} {xintAND}%
1051 \XINT_expr_defbin_b {flexpr}{&&} {viii}{xii} {xintAND}%
1052 \XINT_expr_defbin_b {iiexpr}{&&} {viii}{xii} {xintAND}%
1053 \XINT_expr_defbin_b {expr} {xor}{vi}{xii} {xintXOR}%
1054 \XINT_expr_defbin_b {flexpr}{xor}{vi}{xii} {xintXOR}%
1055 \XINT_expr_defbin_b {iiexpr}{xor}{vi}{xii} {xintXOR}%
1056 \XINT_expr_defbin_b {expr} < {x}{xii} {xintLt}%
1057 \XINT_expr_defbin_b {flexpr} < {x}{xii} {xintLt}%
1058 \XINT_expr_defbin_b {iiexpr} < {x}{xii} {xintiLt}%
1059 \XINT_expr_defbin_b {expr} > {x}{xii} {xintGt}%
1060 \XINT_expr_defbin_b {flexpr} > {x}{xii} {xintGt}%
1061 \XINT_expr_defbin_b {iiexpr} > {x}{xii} {xintiGt}%

```

```

1062 \XINT_expr_defbin_b {expr} {==} {x}{xii} {xintEq}%
1063 \XINT_expr_defbin_b {flexpr}{==} {x}{xii} {xintEq}%
1064 \XINT_expr_defbin_b {iiexpr}{==} {x}{xii} {xintiiEq}%
1065 \XINT_expr_defbin_b {expr} {<=} {x}{xii} {xintLtorEq}%
1066 \XINT_expr_defbin_b {flexpr}{<=} {x}{xii} {xintLtorEq}%
1067 \XINT_expr_defbin_b {iiexpr}{<=} {x}{xii} {xintiiLtorEq}%
1068 \XINT_expr_defbin_b {expr} {>=} {x}{xii} {xintGtorEq}%
1069 \XINT_expr_defbin_b {flexpr}{>=} {x}{xii} {xintGtorEq}%
1070 \XINT_expr_defbin_b {iiexpr}{>=} {x}{xii} {xintiiGtorEq}%
1071 \XINT_expr_defbin_b {expr} {!=} {x}{xii} {xintNotEq}%
1072 \XINT_expr_defbin_b {flexpr}{!=} {x}{xii} {xintNotEq}%
1073 \XINT_expr_defbin_b {iiexpr}{!=} {x}{xii} {xintiiNotEq}%
1074 \XINT_expr_defbin_b {expr} {//} {xiv}{xiv}{xintDivFloor}% CHANGED IN 1.2p!
1075 \XINT_expr_defbin_b {flexpr}{//} {xiv}{xiv}{XINTinFloatDivFloor}% "
1076 \XINT_expr_defbin_b {iiexpr}{//} {xiv}{xiv}{xintiiDivFloor}% "
1077 \XINT_expr_defbin_b {expr} {/:} {xiv}{xiv}{xintMod}% "
1078 \XINT_expr_defbin_b {flexpr}{/:} {xiv}{xiv}{XINTinFloatMod}% "
1079 \XINT_expr_defbin_b {iiexpr}{/:} {xiv}{xiv}{xintiiMod}% "
1080 \XINT_expr_defbin_b {expr} + {xii}{xii} {xintAdd}%
1081 \XINT_expr_defbin_b {flexpr} + {xii}{xii} {XINTinFloatAdd}%
1082 \XINT_expr_defbin_b {iiexpr} + {xii}{xii} {xintiiAdd}%
1083 \XINT_expr_defbin_b {expr} - {xii}{xii} {xintSub}%
1084 \XINT_expr_defbin_b {flexpr} - {xii}{xii} {XINTinFloatSub}%
1085 \XINT_expr_defbin_b {iiexpr} - {xii}{xii} {xintiiSub}%
1086 \XINT_expr_defbin_b {expr} * {xiv}{xiv}{xintMul}%
1087 \XINT_expr_defbin_b {flexpr} * {xiv}{xiv}{XINTinFloatMul}%
1088 \XINT_expr_defbin_b {iiexpr} * {xiv}{xiv}{xintiiMul}%
1089 \XINT_expr_defbin_b {expr} / {xiv}{xiv}{xintDiv}%
1090 \XINT_expr_defbin_b {flexpr} / {xiv}{xiv}{XINTinFloatDiv}%
1091 \XINT_expr_defbin_b {iiexpr} / {xiv}{xiv}{xintiiDivRound}% CHANGED IN 1.1!
1092 \XINT_expr_defbin_b {expr} ^ {xviii}{xviii} {xintPow}%
1093 \XINT_expr_defbin_b {flexpr} ^ {xviii}{xviii} {XINTinFloatPowerH}%
1094 \XINT_expr_defbin_b {iiexpr} ^ {xviii}{xviii} {xintiiPow}%
1095 \xintFor #1 in {and,or,xor,mod} \do
1096 {%
1097   \expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}%
1098 }%
1099 \expandafter\let\csname XINT_expr_precedence_and\expandafter\endcsname
1100   \csname XINT_expr_precedence_&\endcsname
1101 \expandafter\let\csname XINT_expr_precedence_or\expandafter\endcsname
1102   \csname XINT_expr_precedence_|\endcsname
1103 \expandafter\let\csname XINT_expr_precedence_mod\expandafter\endcsname
1104   \csname XINT_expr_precedence_/\endcsname
1105 \xintFor #1 in {expr, flexpr, iiexpr} \do
1106 {%
1107   \expandafter\let\csname XINT_#1_op_and\expandafter\endcsname
1108     \csname XINT_#1_op_&\endcsname
1109   \expandafter\let\csname XINT_#1_op_or\expandafter\endcsname
1110     \csname XINT_#1_op_|\endcsname
1111   \expandafter\let\csname XINT_#1_op_mod\expandafter\endcsname
1112     \csname XINT_#1_op_/\endcsname
1113 }%

```

```

1114 \expandafter\let\csname XINT_expr_precedence_=\expandafter\endcsname
1115         \csname XINT_expr_precedence_==\endcsname
1116 \expandafter\let\csname XINT_expr_precedence_&\expandafter\endcsname
1117         \csname XINT_expr_precedence_&\endcsname
1118 \expandafter\let\csname XINT_expr_precedence_|\expandafter\endcsname
1119         \csname XINT_expr_precedence_|\endcsname
1120 \expandafter\let\csname XINT_expr_precedence_**\expandafter\endcsname
1121         \csname XINT_expr_precedence_^\endcsname
1122 \xintFor #1 in {expr, flexpr, iexpr} \do
1123 {%
1124     \expandafter\let\csname XINT_#1_op_=\expandafter\endcsname
1125         \csname XINT_#1_op_==\endcsname
1126     \expandafter\let\csname XINT_#1_op_&\expandafter\endcsname
1127         \csname XINT_#1_op_&\endcsname
1128     \expandafter\let\csname XINT_#1_op_|\expandafter\endcsname
1129         \csname XINT_#1_op_|\endcsname
1130     \expandafter\let\csname XINT_#1_op_**\expandafter\endcsname
1131         \csname XINT_#1_op_^\endcsname
1132 }%
1133 \catcode`& 7

```

11.18.2 .., ..[, and].. as infix operators

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

```

1134 \def\XINT_expr_defbin_b #1#2#3#4%
1135 {%
1136     \expandafter\XINT_expr_defbin_c
1137     \csname XINT_#1_op_#2\expandafter\endcsname
1138     \csname XINT_#1_exec_#2\expandafter\endcsname
1139     \csname XINT_#1_check_#2\expandafter\endcsname
1140     \csname XINT_#1_checkp_#2\expandafter\endcsname
1141     \csname XINT_#1_op_#4\expandafter\endcsname
1142     \expandafter{\expandafter}%
1143     \csname XINT_expr_precedence_#2\endcsname
1144     {#1}{\expandafter}% REVOIR
1145     \expandafter
1146     \let\csname XINT_expr_precedence_#2\expandafter\endcsname
1147         \csname xint_c_#3\endcsname
1148 }%
1149 \XINT_expr_defbin_b {expr}  {...}{vi}{xii}%
1150 \XINT_expr_defbin_b {flexpr}{...}{vi}{xii}%
1151 \XINT_expr_defbin_b {iexpr}{...}{vi}{xii}%
1152 \def\XINT_expr_defbin_c #1#2#3#4#5#6#7#8#9%
1153 {%
1154     \def #1#1% \XINT_expr_op_<op>
1155     {%
1156         \expanded{\unexpanded{#2{##1}}\expandafter}%
1157         \romannumeral`&&\expandafter#3\romannumeral`&&\XINT_expr_getnext
1158     }%
1159     \def #2##1##2##3##4% \XINT_expr_exec_<op>
1160     {%
1161         \expandafter##2\expandafter##3\expanded{{#9#6##1##4}}%

```

```

1162 }%
1163 \def #3##1% \XINT_expr_check-_{<op>
1164 {%
1165   \xint_UDsignfork
1166   ##1{\expandafter#4\romannumeral`&&@#5}%
1167   -{#4##1}%
1168   \krof
1169 }%
1170 \def #4##1##2% \XINT_expr_checkp_{<op>
1171 {%
1172   \ifnum ##1>#7%
1173     \expandafter#4%
1174     \romannumeral`&&\csname XINT_#8_op_##2\expandafter\endcsname
1175   \else
1176     \expandafter ##1\expandafter ##2%
1177   \fi
1178 }%
1179 }%
1180 \def\XINT_expr_defbin_b #1#2#3#4#5#6%
1181 {%
1182   \expandafter\XINT_expr_defbin_c
1183   \csname XINT_#1_op_#2\expandafter\endcsname
1184   \csname XINT_#1_exec_#2\expandafter\endcsname
1185   \csname XINT_#1_check_#2\expandafter\endcsname
1186   \csname XINT_#1_checkp_#2\expandafter\endcsname
1187   \csname XINT_#1_op_#4\expandafter\endcsname
1188   \csname #5\expandafter\endcsname
1189   \csname XINT_expr_precedence_#2\endcsname {#1}#6%
1190   \expandafter\let
1191   \csname XINT_expr_precedence_#2\expandafter\endcsname
1192   \csname xint_c_#3\endcsname
1193 }%
1194 \XINT_expr_defbin_b {expr} {...} {vi}{xii}{xintSeq:tl:x}\XINT:NEhook:x:one:from:two
1195 \XINT_expr_defbin_b {flexpr}{...} {vi}{xii}{xintSeq:tl:x}\XINT:NEhook:x:one:from:two
1196 \XINT_expr_defbin_b {iiexpr}{...} {vi}{xii}{xintiSeq:tl:x}\XINT:NEhook:x:one:from:two
1197 \XINT_expr_defbin_b {expr} {...} {vi}{xii}{xintSeqB:tl:x}\XINT:NEhook:x:one:from:twoandone
1198 \XINT_expr_defbin_b {flexpr}{...} {vi}{xii}{xintSeqB:tl:x}\XINT:NEhook:x:one:from:twoandone
1199 \XINT_expr_defbin_b {iiexpr}{...} {vi}{xii}{xintiSeqB:tl:x}\XINT:NEhook:x:one:from:twoandone

```

11.18.3 Support macros for .., ..[and]..

\xintSeq:tl:x 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.

```

1200 \def\xintSeq:tl:x #1#2%
1201 {%
1202   \expandafter\XINT_Seq:tl:x
1203   \the\numexpr \xintiCeil{#1}\expandafter.\the\numexpr \xintiFloor{#2}%.
1204 }%
1205 \def\XINT_Seq:tl:x #1.#2.%
1206 {%

```

```

1207 \ifnum #2=#1 \xint_dothis\XINT_Seq:tl:x_z\fi
1208 \ifnum #2<#1 \xint_dothis\XINT_Seq:tl:x_n\fi
1209 \xint_orthat\XINT_Seq:tl:x_p
1210 #1.#2.%
1211 }%
1212 \def\XINT_Seq:tl:x_z #1.#2.{\#1/1[0]}}%
1213 \def\XINT_Seq:tl:x_p #1.#2.%
1214 {%
1215     {\#1/1[0]}\ifnum #1=#2 \XINT_Seq:tl:x_e\fi
1216     \expandafter\XINT_Seq:tl:x_p \the\numexpr #1+\xint_c_i.#2.%
1217 }%
1218 \def\XINT_Seq:tl:x_n #1.#2.%
1219 {%
1220     {\#1/1[0]}\ifnum #1=#2 \XINT_Seq:tl:x_e\fi
1221     \expandafter\XINT_Seq:tl:x_n \the\numexpr #1-\xint_c_i.#2.%
1222 }%
1223 \def\XINT_Seq:tl:x_e#1#2.#3.{#1}%

\xintiSeq:tl:x
1224 \def\xintiSeq:tl:x #1#2%
1225 {%
1226     \expandafter\XINT_iiSeq:tl:x
1227     \the\numexpr \xintiCeil{#1}\expandafter.\the\numexpr \xintiFloor{#2}.%
1228 }%
1229 \def\XINT_iiSeq:tl:x #1.#2.%
1230 {%
1231     \ifnum #2=#1 \xint_dothis\XINT_iiSeq:tl:x_z\fi
1232     \ifnum #2<#1 \xint_dothis\XINT_iiSeq:tl:x_n\fi
1233     \xint_orthat\XINT_iiSeq:tl:x_p
1234     #1.#2.%
1235 }%
1236 \def\XINT_iiSeq:tl:x_z #1.#2.{\#1}}%
1237 \def\XINT_iiSeq:tl:x_p #1.#2.%
1238 {%
1239     {\#1}\ifnum #1=#2 \XINT_Seq:tl:x_e\fi
1240     \expandafter\XINT_iiSeq:tl:x_p \the\numexpr #1+\xint_c_i.#2.%
1241 }%
1242 \def\XINT_iiSeq:tl:x_n #1.#2.%
1243 {%
1244     {\#1}\ifnum #1=#2 \XINT_Seq:tl:x_e\fi
1245     \expandafter\XINT_iiSeq:tl:x_n \the\numexpr #1-\xint_c_i.#2.%
1246 }%

```

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`.

\xintSeqA, \xintiSeqA

```

1247 \def\xintSeqA          {\expandafter\XINT_SeqA\romannumeral0\xintraw}%
1248 \def\xintiSeqA         #1{\expandafter\XINT_iiSeqA\romannumeral`&&@#1;}%
1249 \def\XINT_SeqA         #1]#2{\expandafter\XINT_SeqA_a\romannumeral0\xintraw {#2}#1]]}%
1250 \def\XINT_iiSeqA#1]#2{\expandafter\XINT_SeqA_a\romannumeral`&&@#2;#1;}%
1251 \def\XINT_SeqA_a       #1{\xint_UDzerominusfork

```

```

1252                                #1-{z}%
1253                                0#1{n}%
1254                                0-{p}%
1255                                \krof #1}%

```

\xintSeqB:tl:x At 1.4, delayed expansion of start and step done here and not before, for matters of \xintdefunc and «NEhooks».

The float variant at 1.4 is made identical to the exact variant. I.e. stepping is exact and comparison to the range limit too. But recall that a/b input will be converted to a float. To handle 1/3 step for example still better to use \xintexpr 1..1/3..10\relax for example inside the \xintfloateval.

```

1256 \def\xintSeqB:tl:x #1{\expandafter\xintSeqB:tl:x\romannumeral`&&\xintSeqA#1}%
1257 \def\xintSeqB:tl:x #1{\csname XINT_SeqB#1:tl:x\endcsname}%
1258 \def\xintSeqBz:tl:x #1]#2]#3{{#2}}}%
1259 \def\xintSeqBp:tl:x #1]#2]#3{\expandafter\xintSeqBp:tl:x_a\romannumeral0\xintraw{#3}#2]#1}%
1260 \def\xintSeqBp:tl:x_a #1]#2]#3}%
1261 {%
1262     \xintifCmp{#1}{{#2}}}%
1263     {{{#2}}}{{#2}}\expandafter\xintSeqBp:tl:x_b\romannumeral0\xintadd{#3}{{#2}}#1]#3}%
1264 }%
1265 \def\xintSeqBp:tl:x_b #1]#2]#3}%
1266 {%
1267     \xintifCmp{#1}{{#2}}}%
1268     {{#1}}\expandafter\xintSeqBp:tl:x_b\romannumeral0\xintadd{#3}{{#1}}#2]#3}{{#1}}}%
1269 }%
1270 \def\xintSeqBn:tl:x #1]#2]#3{\expandafter\xintSeqBn:tl:x_a\romannumeral0\xintraw{#3}#2]#1}%
1271 \def\xintSeqBn:tl:x_a #1]#2]#3}%
1272 {%
1273     \xintifCmp{#1}{{#2}}}%
1274     {{{#2}}}\expandafter\xintSeqBn:tl:x_b\romannumeral0\xintadd{#3}{{#2}}#1]#3}{{#2}}}%
1275 }%
1276 \def\xintSeqBn:tl:x_b #1]#2]#3}%
1277 {%
1278     \xintifCmp{#1}{{#2}}}%
1279     {{{#1}}}{{#1}}\expandafter\xintSeqBn:tl:x_b\romannumeral0\xintadd{#3}{{#1}}#2]#3}%
1280 }%

```

\xintiiSeqB:tl:x

```

1281 \def\xintiiSeqB:tl:x #1{\expandafter\xintiiSeqB:tl:x\romannumeral`&&\xintiiSeqA#1}%
1282 \def\xintiiSeqB:tl:x #1{\csname XINT_iiSeqB#1:tl:x\endcsname}%
1283 \def\xintiiSeqBz:tl:x #1;#2;#3{{#2}}}%
1284 \def\xintiiSeqBp:tl:x #1;#2;#3{\expandafter\xintiiSeqBp:tl:x_a\romannumeral`&&#3;#2;#1;}%
1285 \def\xintiiSeqBp:tl:x_a #1;#2;#3;%
1286 {%
1287     \xintiiifCmp{#1}{{#2}}}%
1288     {{{#2}}}{{#2}}\expandafter\xintiiSeqBp:tl:x_b\romannumeral0\xintiiadd{#3}{{#2}};#1;#3;%
1289 }%
1290 \def\xintiiSeqBp:tl:x_b #1;#2;#3;%
1291 {%
1292     \xintiiifCmp{#1}{{#2}}}%
1293     {{#1}}\expandafter\xintiiSeqBp:tl:x_b\romannumeral0\xintiiadd{#3}{{#1}};#2;#3}{{#1}}}%

```



```

1294 }%
1295 \def\XINT_iiSeqBn:tl:x #1;#2;#3{\expandafter\XINT_iiSeqBn:tl:x_a\romannumeral`&&#3;#2;#1;}%
1296 \def\XINT_iiSeqBn:tl:x_a #1;#2;#3;%
1297 {%
1298     \xintiiifCmp{#1}{#2}%
1299     {{#2}\expandafter\XINT_iiSeqBn:tl:x_b\romannumeral0\xintiiadd{#3}{#2};#1;#3;}}{{#2}}}%
1300 }%
1301 \def\XINT_iiSeqBn:tl:x_b #1;#2;#3;%
1302 {%
1303     \xintiiifCmp{#1}{#2}%
1304     {{{#1}}{{#1}\expandafter\XINT_iiSeqBn:tl:x_b\romannumeral0\xintiiadd{#3}{#1};#2;#3;}}}%
1305 }%

```

11.19 Square brackets [] both as a container and a Python slicer

Refactored at 1.4

The architecture allows to implement separately a «left» and a «right» precedence and this is crucial.

11.19.1	[...] as «oneple» constructor	345
11.19.2	[...] brackets and : operator for NumPy-like slicing and item indexing syntax	346
11.19.3	Macro layer implementing indexing and slicing	348

11.19.1 [...] as «oneple» constructor

In the definition of \XINT_expr_op_obracket the parameter is trash {}. The [is intercepted by the getnextfork and handled via the \xint_c_ii^v highest precedence trick to get op_obracket executed.

```

1306 \def\XINT_expr_itself_obracket{obracket}%
1307 \catcode`] 11 \catcode`[ 11
1308 \def\XINT_tmpa #1#2#3#4#5#6%
1309 {%
1310     \def #1##1%
1311     {%
1312         \expandafter#3\romannumeral`&&\XINT_expr_getnext
1313     }%
1314     \def #2##1% op_]
1315     {%
1316         \expanded{\unexpanded{\XINT_expr_put_op_first{{#1}}}\expandafter}%
1317         \romannumeral`&&\XINT_expr_getop
1318     }%
1319     \def #3##1% until_cbracket_a
1320     {%
1321         \xint_UDsignfork
1322         ##1{\expandafter#4\romannumeral`&&#5}% #5 = op_-xii
1323         -{#4##1}%
1324     \krof
1325     }%
1326     \def #4##1##2% until_cbracket_b
1327     {%
1328         \ifcase #1\expandafter\XINT_expr_missing_]
1329         \or \expandafter\XINT_expr_missing_]

```

```

1330     \or \expandafter#2%
1331     \else
1332     \expandafter #4%
1333     \romannumeral`&&@\csname XINT_#6_op_##2\expandafter\endcsname
1334     \fi
1335 }%
1336}%
1337\xintFor #1 in {expr,flexpr,iiexpr} \do {%
1338    \expandafter\XINT_tmpa
1339    \csname XINT_#1_op_obracket\expandafter\endcsname
1340    \csname XINT_#1_op_]\expandafter\endcsname
1341    \csname XINT_#1_until_cbracket_a\expandafter\endcsname
1342    \csname XINT_#1_until_cbracket_b\expandafter\endcsname
1343    \csname XINT_#1_op_-xii\endcsname
1344    {#1}%
1345}%
1346\def\XINT_expr_missing_
1347  {\XINT_expandableerror{Ooops, looks like we are missing a ] here. Goodbye!}%
1348  \xint_c_ \XINT_expr_done}%
1349\let\XINT_expr_precedence_]\xint_c_ii

```

11.19.2 [...] brackets and : operator for NumPy-like slicing and item indexing syntax

The opening bracket [for the nutple constructor is filtered out by \XINT_expr_getnextfork and becomes «obracket» which behaves with precedence level 2. For the [...] Python slicer on the other hand, a real operator [is defined with precedence level 4 (it must be higher than precedence level of commas) on its right and maximal precedence on its left.

Important: although slicing and indexing shares many rules with Python/NumPy there are some significant differences: in particular there can not be any out-of-range error generated, slicing applies also to «oples» and not only to «nutple», and nested lists do not have to have their leaves at a constant depth. See the user manual.

Currently, NumPy-like nested (basic) slicing is implemented, i.e [a:b, c:d, N, e:f, M] type syntax with Python rules regarding negative integers. This is parsed as an expression and can arise from expansion or contain calculations.

Currently stepping, Ellipsis, and simultaneous multi-index extracting are not yet implemented. There are some subtle things here with possibility of variables been passed by reference.

```

1350\def\XINT_tmpa #1#2#3#4#5#6%
1351{%
1352    \def #1##1% \XINT_expr_op_[
1353    {%
1354        \expanded{\unexpanded{#2{##1}}\expandafter}%
1355        \romannumeral`&&@\expandafter#3\romannumeral`&&@\XINT_expr_getnext
1356    }%
1357    \def #2##1##2##3##4% \XINT_expr_exec_
1358    {%
1359        \expandafter\XINT_expr_put_op_first
1360        \expanded
1361        {%
1362            {\XINT:NEhook:x:lists\XINT_ListSel_top ##1__##4&({##1}\expandafter}%
1363            \expandafter
1364        }%
1365        \romannumeral`&&@\XINT_expr_getop

```

```

1366 }%
1367 \def #3##1% \XINT_expr_check-_]
1368 {%
1369   \xint_UDsignfork
1370   ##1{\expandafter#4\romannumeral`&&@#5}%
1371   -{#4##1}%
1372   \krof
1373 }%
1374 \def #4##1##2% \XINT_expr_checkp_]
1375 {%
1376   \ifcase ##1\XINT_expr_missing_]
1377   \or \XINT_expr_missing_]
1378   \or \expandafter##1\expandafter##2%
1379   \else \expandafter#4%
1380   \romannumeral`&&@\csname XINT_#6_op_##2\expandafter\endcsname
1381   \fi
1382 }%
1383 }%
1384 \let\XINT_expr_precedence_ \xint_c_xx
1385 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1386 \expandafter\XINT_tmpa
1387   \csname XINT_#1_op_[\expandafter\endcsname
1388   \csname XINT_#1_exec_]\expandafter\endcsname
1389   \csname XINT_#1_check-_\expandafter\endcsname
1390   \csname XINT_#1_checkp_]\expandafter\endcsname
1391   \csname XINT_#1_op_-xii\endcsname
1392   {#1}%
1393 }%
1394 \catcode` ] 12 \catcode` [ 12

```

At 1.4 the `getnext`, `scanint`, `scanfunc`, `getop` chain got revisited to trigger automatic insertion of the `nil` variable if needed, without having in situations like here to define operators to support «`:`» or «`:`»». And as we want to implement nested slicing à la NumPy, we would have had to handle also «`:`», for example. Thus here we simply have to define the sole operator «`:`» and it will be some sort of inert joiner preparing a slicing spec.

```

1395 \def\XINT_tmpa #1#2#3#4#5#6%
1396 {%
1397   \def #1##1% \XINT_expr_op_:
1398   {%
1399     \expanded{\unexpanded{#2{##1}}\expandafter}%
1400     \romannumeral`&&@\expandafter#3\romannumeral`&&@\XINT_expr_getnext
1401   }%
1402   \def #2##1##2##3##4% \XINT_expr_exec_:
1403   {%
1404     ##2##3{:#1{0};##4:_}%
1405   }%
1406   \def #3##1% \XINT_expr_check-_:
1407   {\xint_UDsignfork
1408     ##1{\expandafter#4\romannumeral`&&@#5}%
1409     -{#4##1}%
1410     \krof
1411   }%

```

```

1412 \def #4##1##2% \XINT_expr_checkp_:
1413 {%
1414 \ifnum ##1>\XINT_expr_precedence_:
1415 \expandafter #4\romannumeral`&&@%
1416 \csname XINT_#6_op_##2\expandafter\endcsname
1417 \else
1418 \expandafter##1\expandafter##2%
1419 \fi
1420 }%
1421 }%
1422 \let\XINT_expr_precedence_:\xint_c_vi
1423 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1424 \expandafter\XINT_tmpa
1425 \csname XINT_#1_op_:\expandafter\endcsname
1426 \csname XINT_#1_exec_:\expandafter\endcsname
1427 \csname XINT_#1_check_:\expandafter\endcsname
1428 \csname XINT_#1_checkp_:\expandafter\endcsname
1429 \csname XINT_#1_op_-xii\endcsname {#1}%
1430 }%

```

11.19.3 Macro layer implementing indexing and slicing

xintexpr applies slicing not only to «objects» (which can be passed as arguments to functions) but also to «oples».

Our «nlists» are not necessarily regular N-dimensional arrays à la NumPy. Leaves can be at arbitrary depths. If we were handling regular «ndarrays», we could proceed a bit differently.

For the related explanations, refer to the user manual.

Notice that currently the code uses f-expandable (and not using \expanded) macros \xintApply, \xintApplyUnbraced, \xintKeep, \xintTrim, \xintNthOne from [xinttools](#).

But the whole expansion happens inside an \expanded context, so possibly some gain could be achieved with x-expandable variants (xintexpr < 1.4 had an \xintKeep:x:csv).

I coded \xintApply:x and \xintApplyUnbraced:x in [xinttools](#), Brief testing indicated they were perhaps a bit better for 5x5x5x5 and 15x15x15x15 arrays of 8 digits numbers and for 30x30x15 with 16 digits numbers: say 1% gain... this seems to raise to between 4% and 5% for 400x400 array of 1 digit...

Currently sticking with old macros.

```

1431 \def\XINT_ListSel_deeper #1%
1432 {%
1433 \if :#1\xint_dothis\XINT_ListSel_slice_next\fi
1434 \xint_orthat {\XINT_ListSel_extract_next {#1}}%
1435 }%
1436 \def\XINT_ListSel_slice_next #1(%
1437 {%
1438 \xintApply{\XINT_ListSel_recurse{:#1}}%
1439 }%
1440 \def\XINT_ListSel_extract_next #1(%
1441 {%
1442 \xintApplyUnbraced{\XINT_ListSel_recurse{#1}}%
1443 }%
1444 \def\XINT_ListSel_recurse #1#2%
1445 {%
1446 \XINT_ListSel_check #2__#1({#2}\expandafter\empty\empty

```

```

1447 }%
1448 \def\XINT_ListSel_check{\expandafter\XINT_ListSel_check_a \string}%
1449 \def\XINT_ListSel_check_a #1%
1450 {%
1451     \if #1\bgroup\xint_dothis\XINT_ListSel_check_is_ok\fi
1452     \xint_orthat\XINT_ListSel_check_leaf
1453 }%
1454 \def\XINT_ListSel_check_leaf #1\expandafter{\expandafter}%
1455 \def\XINT_ListSel_check_is_ok
1456 {%
1457     \expandafter\XINT_ListSel_check_is_ok_a\expandafter{\string}%
1458 }%
1459 \def\XINT_ListSel_check_is_ok_a #1__#2%
1460 {%
1461     \if :#2\xint_dothis{\XINT_ListSel_slice}\fi
1462     \xint_orthat {\XINT_ListSel_nthone {#2}}%
1463 }%
1464 \def\XINT_ListSel_top #1#2%
1465 {%
1466     \if _\noexpand#2%
1467         \expandafter\XINT_ListSel_top_one_or_none\string#1.\else
1468         \expandafter\XINT_ListSel_top_at_least_two\fi
1469 }%
1470 \def\XINT_ListSel_top_at_least_two #1__{\XINT_ListSel_top_ople}%
1471 \def\XINT_ListSel_top_one_or_none #1%
1472 {%
1473     \if #1_\xint_dothis\XINT_ListSel_top_nil\fi
1474     \if #1.\xint_dothis\XINT_ListSel_top_nutple_a\fi
1475     \if #1\bgroup\xint_dothis\XINT_ListSel_top_nutple\fi
1476     \xint_orthat\XINT_ListSel_top_number
1477 }%
1478 \def\XINT_ListSel_top_nil #1\expandafter#2\expandafter{\fi\expandafter}%
1479 \def\XINT_ListSel_top_nutple
1480 {%
1481     \expandafter\XINT_ListSel_top_nutple_a\expandafter{\string}%
1482 }%
1483 \def\XINT_ListSel_top_nutple_a #1_#2#3(#4%
1484 {%
1485     \fi\if :#2\xint_dothis{{\XINT_ListSel_slice #3(#4}}\fi
1486     \xint_orthat {\XINT_ListSel_nthone {#2}#3(#4}%
1487 }%
1488 \def\XINT_ListSel_top_number #1_{\fi\XINT_ListSel_top_ople}%
1489 \def\XINT_ListSel_top_ople #1%
1490 {%
1491     \if :#1\xint_dothis\XINT_ListSel_slice\fi
1492     \xint_orthat {\XINT_ListSel_nthone {#1}}%
1493 }%
1494 \def\XINT_ListSel_slice #1%
1495 {%
1496     \expandafter\XINT_ListSel_slice_a \expandafter{\romannumeral0\xintnum{#1}}%
1497 }%
1498 \def\XINT_ListSel_slice_a #1#2;#3#4%

```

```

1499 {%
1500   \if _#4\expandafter\XINT_ListSel_s_b
1501     \else\expandafter\XINT_ListSel_slice_b\fi
1502   #1;#3%
1503 }%
1504 \def\XINT_ListSel_s_b #1#2;#3#4%
1505 {%
1506   \if &#4\expandafter\XINT_ListSel_s_last\fi
1507   \XINT_ListSel_s_c #1{#1#2}{#4}%
1508 }%
1509 \def\XINT_ListSel_s_last\XINT_ListSel_s_c #1#2#3(#4%
1510 {%
1511   \if-#1\expandafter\xintKeep\else\expandafter\xintTrim\fi {#2}{#4}%
1512 }%
1513 \def\XINT_ListSel_s_c #1#2#3(#4%
1514 {%
1515   \expandafter\XINT_ListSel_deeper
1516   \expanded{\unexpanded{#3}(\expandafter)\expandafter{%
1517     \romannumeral0%
1518     \if-#1\expandafter\xintkeep\else\expandafter\xinttrim\fi {#2}{#4}}%
1519 }%

```

`\xintNthElt` from `xinttools` (knowingly) strips one level of braces when fetching `kth` «item» from `{v1}...{vN}`. If we expand `{\xintNthElt{k}{v1}...{vN}}` (notice external braces):

- if `k` is out of range we end up with `{}`
- if `k` is in range and the `kth` braced item was `{}` we end up with `{}`
- if `k` is in range and the `kth` braced item was `{17}` we end up with `{17}`

Problem is that individual numbers such as 17 are stored `{{17}}`. So we must have one more brace pair and in the first two cases we end up with `{{}}`. But in the first case we should end up with the empty ogle `{}`, not the empty bracketed ogle `{{}}`.

I have thus added `\xintNthOne` to `xinttools` which does not strip brace pair from an extracted item.

Attention: `\XINT_nthonepy_a` does no expansion on second argument. But here arguments are either numerical or already expanded. Normally.

```

1520 \def\XINT_ListSel_nthone #1#2%
1521 {%
1522   \if &#2\expandafter\XINT_ListSel_nthone_last\fi
1523   \XINT_ListSel_nthone_a {#1}{#2}%
1524 }%
1525 \def\XINT_ListSel_nthone_a #1#2(#3%
1526 {%
1527   \expandafter\XINT_ListSel_deeper
1528   \expanded{\unexpanded{#2}(\expandafter)\expandafter{%
1529     \romannumeral0\expandafter\XINT_nthonepy_a\the\numexpr\xintNum{#1}.{#3}}%
1530 }%
1531 \def\XINT_ListSel_nthone_last\XINT_ListSel_nthone_a #1#2(%#3%
1532 {%
1533   \romannumeral0\expandafter\XINT_nthonepy_a\the\numexpr\xintNum{#1}.%{#3}
1534 }%

```

The macros here are basically f-expandable and use the f-expandable `\xintKeep` and `\xintTrim`. Prior to xint 1.4, there was here an x-expandable `\xintKeep:x:csv` dealing with comma separated items, for time being we make do with our f-expandable toolkit.

```

1535 \def\XINT_ListSel_slice_b #1;#2_#3%
1536 {%
1537     \if &#3\expandafter\XINT_ListSel_slice_last\fi
1538     \expandafter\XINT_ListSel_slice_c \expandafter{\romannumeral0\xintnum{#2}};#1;{#3}%
1539 }%
1540 \def\XINT_ListSel_slice_last\expandafter\XINT_ListSel_slice_c #1;#2;#3(%#4
1541 {%
1542     \expandafter\XINT_ListSel_slice_last_c #1;#2;%{#4}
1543 }%
1544 \def\XINT_ListSel_slice_last_c #1;#2;#3%
1545 {%
1546     \romannumeral0\XINT_ListSel_slice_d #2;#1;{#3}%
1547 }%
1548 \def\XINT_ListSel_slice_c #1;#2;#3(%#4
1549 {%
1550     \expandafter\XINT_ListSel_deeper
1551     \expanded{\unexpanded{#3}(\expandafter\expandafter{%
1552         \romannumeral0\XINT_ListSel_slice_d #2;#1;{#4}}}%
1553 }%
1554 \def\XINT_ListSel_slice_d #1#2;#3#4;%
1555 {%
1556     \xint_UDsignsfork
1557         #1#3\XINT_ListSel_N:N
1558         #1-\XINT_ListSel_N:P
1559         -#3\XINT_ListSel_P:N
1560         --\XINT_ListSel_P:P
1561     \krof #1#2;#3#4;%
1562 }%
1563 \def\XINT_ListSel_P:P #1;#2;#3%
1564 {%
1565     \unless\ifnum #1<#2 \expandafter\xint_gob_andstop_iii\fi
1566     \xintkeep{#2-#1}{\xintTrim{#1}{#3}}%
1567 }%
1568 \def\XINT_ListSel_N:N #1;#2;#3%
1569 {%
1570     \expandafter\XINT_ListSel_N:N_a
1571     \the\numexpr #2-#1\expandafter;\the\numexpr#1+\xintLength{#3};{#3}%
1572 }%
1573 \def\XINT_ListSel_N:N_a #1;#2;#3%
1574 {%
1575     \unless\ifnum #1>\xint_c_ \expandafter\xint_gob_andstop_iii\fi
1576     \xintkeep{#1}{\xintTrim{\ifnum#2<\xint_c_\xint_c_\else#2\fi}{#3}}%
1577 }%
1578 \def\XINT_ListSel_N:P #1;#2;#3%
1579 {%
1580     \expandafter\XINT_ListSel_N:P_a
1581     \the\numexpr #1+\xintLength{#3};#2;{#3}%
1582 }%
1583 \def\XINT_ListSel_N:P_a #1#2;%
1584     {\if -#1\expandafter\XINT_ListSel_O:P\fi\XINT_ListSel_P:P #1#2;}%
1585 \def\XINT_ListSel_O:P\XINT_ListSel_P:P #1;{\XINT_ListSel_P:P 0;}%
1586 \def\XINT_ListSel_P:N #1;#2;#3%

```

```

1587 {%
1588     \expandafter\XINT_ListSel_P:N_a
1589     \the\numexpr #2+\xintLength{#3};#1;{#3}%
1590 }%
1591 \def\XINT_ListSel_P:N_a #1#2;#3;%
1592     {\if -#1\expandafter\XINT_ListSel_P:O\fi\XINT_ListSel_P:P #3;#1#2;}%
1593 \def\XINT_ListSel_P:O\XINT_ListSel_P:P #1;#2;{\XINT_ListSel_P:P #1;0;}%

```

11.20 Support for raw A/B[N]

Releases earlier than 1.1 required the use of braces around A/B[N] input. The [N] is now implemented directly. *BUT* this uses a delimited macro! thus N is not allowed to be itself an expression (I could add it...). \xintE, \xintiiE, and \XINTinFloatE all put #2 in a \numexpr. But attention to the fact that \numexpr stops at spaces separating digits: \the\numexpr 3 + 7 9\relax gives 109\relax !! Hence we have to be careful.

\numexpr will not handle catcode 11 digits, but adding a \detokenize will suddenly make illicit for N to rely on macro expansion.

At 1.4, [is already overloaded and it is not easy to support this. We do this by a kludge maintaining more or less former (very not efficient) way but using \$ sign which is free for time being. No, finally I use the null character, should be safe enough! (I hesitated about using R with catcode 12).

As for ? operator we needed to hack into \XINT_expr_getop_b for intercepting that pseudo operator. See also \XINT_expr_scanint_c (\XINT_expr_rawxintfrac).

```

1594 \catcode0 11
1595 \let\XINT_expr_precedence_&&@ \xint_c_xiv
1596 \def\XINT_expr_op_&&@ #1#2]%
1597 {%
1598     \expandafter\XINT_expr_put_op_first
1599     \expanded{{{ \xintE#1{ \xint_zapspaces #2 \xint_gobble_i}}}%
1600     \expandafter}\romannumeral`&&@\XINT_expr_getop
1601 }%
1602 \def\XINT_iiexpr_op_&&@ #1#2]%
1603 {%
1604     \expandafter\XINT_expr_put_op_first
1605     \expanded{{{ \xintiiE#1{ \xint_zapspaces #2 \xint_gobble_i}}}%
1606     \expandafter}\romannumeral`&&@\XINT_expr_getop
1607 }%
1608 \def\XINT_flexpr_op_&&@ #1#2]%
1609 {%
1610     \expandafter\XINT_expr_put_op_first
1611     \expanded{{{ \XINTinFloatE#1{ \xint_zapspaces #2 \xint_gobble_i}}}%
1612     \expandafter}\romannumeral`&&@\XINT_expr_getop
1613 }%
1614 \catcode0 12

```

11.21 ? as two-way and ?? as three-way «short-circuit» conditionals

Comments undergoing reconstruction.

```

1615 \let\XINT_expr_precedence_? \xint_c_xx
1616 \catcode`- 11
1617 \def\XINT_expr_op_? {\XINT_expr_op__? \XINT_expr_op_-xii}%

```



```

1618 \def\XINT_flexpr_op_{\XINT_expr_op_{\XINT_flexpr_op_-xii}%
1619 \def\XINT_iiexpr_op_{\XINT_expr_op_{\XINT_iiexpr_op_-xii}%
1620 \catcode`- 12
1621 \def\XINT_expr_op_{#1#2#3%
1622     {\XINT_expr_op_{_a #3!\xint_bye\XINT_expr_exec_{#1}{#2}{#3}}%
1623 \def\XINT_expr_op_{_a #1{\expandafter\XINT_expr_op_{_b\detokenize{#1}}}%
1624 \def\XINT_expr_op_{_b #1%
1625     {\if ?#1\expandafter\XINT_expr_op_{_c}else\expandafter\xint_bye\fi }%
1626 \def\XINT_expr_op_{_c #1{\xint_gob_til_! #1\XINT_expr_op_{_? !\xint_bye}%
1627 \def\XINT_expr_op_{_? !\xint_bye\xint_bye\XINT_expr_exec_{\XINT_expr_exec_{_?}}%
1628 \catcode`- 11
1629 \def\XINT_expr_exec_{_? #1#2%
1630 {%
1631     \expandafter\XINT_expr_check-_after?\expandafter#1%
1632     \romannumeral`&&\expandafter\XINT_expr_getnext\romannumeral0\xintiiifnotzero#2%
1633 }%
1634 \def\XINT_expr_exec_{_? #1#2#3%
1635 {%
1636     \expandafter\XINT_expr_check-_after?\expandafter#1%
1637     \romannumeral`&&\expandafter\XINT_expr_getnext\romannumeral0\xintiiifsgn#2%
1638 }%
1639 \def\XINT_expr_check-_after? #1{%
1640 \def\XINT_expr_check-_after? ##1##2%
1641 {%
1642     \xint_UDsignfork
1643     ##2{##1}%
1644     #1{##2}%
1645     \krof
1646 }}\expandafter\XINT_expr_check-_after?\string -%
1647 \catcode`- 12

```

11.22 ! as postfix factorial operator

```

1648 \let\XINT_expr_precedence_! \xint_c_xx
1649 \def\XINT_expr_op_! #1%
1650 {%
1651     \expandafter\XINT_expr_put_op_first
1652     \expanded{{\romannumeral`&&\XINT:NEhook:f:one:from:one
1653     {\romannumeral`&&\xintFac#1}}\expandafter}\romannumeral`&&\XINT_expr_getop
1654 }%
1655 \def\XINT_flexpr_op_! #1%
1656 {%
1657     \expandafter\XINT_expr_put_op_first
1658     \expanded{{\romannumeral`&&\XINT:NEhook:f:one:from:one
1659     {\romannumeral`&&\XINTinFloatFac#1}}\expandafter}\romannumeral`&&\XINT_expr_getop
1660 }%
1661 \def\XINT_iiexpr_op_! #1%
1662 {%
1663     \expandafter\XINT_expr_put_op_first
1664     \expanded{{\romannumeral`&&\XINT:NEhook:f:one:from:one
1665     {\romannumeral`&&\xintiiFac#1}}\expandafter}\romannumeral`&&\XINT_expr_getop
1666 }%

```

11.23 User defined variables

11.23.1	<code>\xintdefvar</code> , <code>\xintdefiivar</code> , <code>\xintdeffloatvar</code>	354
11.23.2	<code>\xintunassignvar</code>	356

11.23.1 `\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.

Note that the optional argument [P] as usable with `\xintfloatexpr` is **not** supported by `\xintdeffloatvar`. One must do `\xintdeffloatvar foo = \xintfloatexpr[16] blabla \relax` to achieve the effect.

1.4 (2020/01/27). The expression will be fetched up to final semi-colon in a manner allowing inner semi-colons as used in the `iter()`, `rseq()`, `subsm()`, `subsn()` etc... syntax. They don't need to be hidden within a braced pair anymore.

TODO: prior to 1.4 a variable «value» was passed along as a single token. Now it is managed, like everything else, as explicit braced contents. But most of the code is ready for passing it along again as a single (braced, now) token again, because all needed `\expanded/\unexpanded` things are in place. However this is «most of the code». I am really eager to get 1.4 released now, because I can't devote more time in immediate future. It is too late to engage into an umpteenth deep refactoring at a time where things work and many new features were added and most aspects of inner working got adapted. However in future it could be that variables holding large data will be managed much faster.

```

1667 \catcode`* 11
1668 \def\xINT_expr_defvar_one #1#2%
1669 {%
1670   \XINT_global
1671   \expandafter\edef\csname XINT_expr_varvalue_#1\endcsname {#2}%
1672   \XINT_expr_defvar_one_b {#1}%
1673 }%
1674 \def\xINT_expr_defvar_one_b #1%
1675 {%
1676   \XINT_global
1677   \expandafter\edef\csname XINT_expr_var_#1\endcsname

```

```

1678      {{\expandafter\noexpand\csname XINT_expr_varvalue_#1\endcsname}}}%
1679 \XINT_global
1680 \expandafter\edef\csname XINT_expr_onliteral_#1\endcsname
1681   {\noexpand\expandafter\XINT_expr_precedence_***
1682    \noexpand\expandafter *\expandafter
1683    \noexpand\csname XINT_expr_var_#1\endcsname{}}%
1684 \ifxintverbose\xintMessage{xintexpr}{Info}
1685   {Variable "#1" \ifxintglobaldefs globally \fi
1686    defined with value \csname XINT_expr_varvalue_#1\endcsname.}%
1687 \fi
1688 }%
1689 \catcode`* 12
1690 \catcode`~ 13
1691 \catcode`: 12
1692 \def\XINT_expr_defvar_getname #1:#2~%
1693 {%
1694   \endgroup
1695   \def\XINT_defvar_tmpa{#1}\edef\XINT_defvar_tmpc{\xintCSVLength{#1}}%
1696 }%
1697 \def\XINT_expr_defvar #1#2%
1698 {%
1699   \def\XINT_defvar_tmpa{#2}%
1700   \expandafter\XINT_expr_defvar_a\expandafter#1\romannumeral\XINT_expr_fetch_to_semicolon
1701 }%
1702 \def\XINT_expr_defvar_a #1#2%
1703 {%
1704   \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..).

The \XINT_expr_defvar_getname closes the group opened here.

```

1705 \begingroup\lccode`~: \lowercase{\let~}\empty
1706 \edef\XINT_defvar_tmpa{\XINT_defvar_tmpa}%
1707 \edef\XINT_defvar_tmpa{\xint_zapspaces_o\XINT_defvar_tmpa}%
1708 \expandafter\XINT_expr_defvar_getname
1709   \detokenize\expandafter{\XINT_defvar_tmpa}:~%
1710 \ifcase\XINT_defvar_tmpc\space
1711   \xintMessage {xintexpr}{Error}
1712   {Aborting: not allowed to declare variable with empty name.}%
1713 \or
1714   \XINT_global
1715   \expandafter\edef\csname XINT_expr_varvalue_\XINT_defvar_tmpa\endcsname
1716     {\romannumeral0#1#2\relax}%
1717   \XINT_expr_defvar_one_b\XINT_defvar_tmpa
1718 \else
1719   \edef\XINT_defvar_tmpb{\romannumeral0#1#2\relax}%
1720   \edef\XINT_defvar_tmpd{\expandafter\xintLength\expandafter{\XINT_defvar_tmpb}}%
1721   \let\XINT_defvar_tmpe\empty
1722   \if1\XINT_defvar_tmpd
1723     \def\XINT_defvar_tmpe{unpacked }%
1724     \oodef\XINT_defvar_tmpb{\expandafter\xint_firstofone\XINT_defvar_tmpb}%

```

```

1725     \edef\XINT_defvar_tmpd{\expandafter\xintLength\expandafter{\XINT_defvar_tmpb}}%
1726     \fi
1727     \ifnum\XINT_defvar_tmpe=\XINT_defvar_tmpd\space
1728     \xintAssignArray\xintCSVtoList\XINT_defvar_tmpe\to\XINT_defvar_tmpevar
1729     \xintAssignArray\xintApply\XINT_embrace{\XINT_defvar_tmpe}\to\XINT_defvar_tmpeval
1730     \def\XINT_defvar_tmpe{1}%
1731     \xintloop
1732     \expandafter\XINT_expr_defvar_one
1733     \csname XINT_defvar_tmpevar\XINT_defvar_tmpe\expandafter\endcsname
1734     \csname XINT_defvar_tmpeval\XINT_defvar_tmpe\endcsname
1735     \ifnum\XINT_defvar_tmpe<\XINT_defvar_tmpe\space
1736     \edef\XINT_defvar_tmpe{\the\numexpr\XINT_defvar_tmpe+1}%
1737     \repeat
1738     \xintRelaxArray\XINT_defvar_tmpevar
1739     \xintRelaxArray\XINT_defvar_tmpeval
1740     \else
1741     \xintMessage {xintexpr}{Error}
1742     {Aborting: mismatch between number of variables (\XINT_defvar_tmpe)
1743     and number of \XINT_defvar_tmpe values (\XINT_defvar_tmpe).}%
1744     \fi
1745     \fi
1746     \let\XINT_defvar_tmpe\empty
1747     \let\XINT_defvar_tmpeb\empty
1748     \let\XINT_defvar_tmpe\empty
1749     \let\XINT_defvar_tmpe\empty
1750 }%
1751 \catcode`~ 3
1752 \catcode`: 11

```

This SafeCatcodes is mainly in the hope that semi-colon ending the expression can still be sanitized.

```

1753 \def\xintdefvar      {\xintexprSafeCatcodes\xintdefvar_a}%
1754 \def\xintdefiivar    {\xintexprSafeCatcodes\xintdefiivar_a}%
1755 \def\xintdeffloatvar {\xintexprSafeCatcodes\xintdeffloatvar_a}%
1756 \def\xintdefvar_a    #1={\XINT_expr_defvar\xintthebareeval    {#1}}%
1757 \def\xintdefiivar_a  #1={\XINT_expr_defvar\xintthebareiieval  {#1}}%
1758 \def\xintdeffloatvar_a #1={\XINT_expr_defvar\xintthebarefloateval {#1}}%

```

11.23.2 \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_``.

```

1759 \def\xintunassignvar #1{%
1760     \edef\XINT_unvar_tmpe{#1}%
1761     \edef\XINT_unvar_tmpe {\xint_zapspaces_o\XINT_unvar_tmpe}%
1762     \ifcsname XINT_expr_var_\XINT_unvar_tmpe\endcsname
1763     \ifnum\expandafter\xintLength\expandafter{\XINT_unvar_tmpe}=\@ne
1764     \expandafter\xintnewdummy\XINT_unvar_tmpe

```

```

1765     \else
1766         \XINT_global\expandafter
1767         \let\csname XINT_expr_varvalue_\XINT_unvar_tmpa\endcsname\xint_undefined
1768         \XINT_global\expandafter
1769         \let\csname XINT_expr_var_\XINT_unvar_tmpa\endcsname\xint_undefined
1770         \XINT_global\expandafter
1771         \let\csname XINT_expr_onliteral_\XINT_unvar_tmpa\endcsname\xint_undefined
1772         \ifxintverbose\xintMessage {xintexpr}{Info}
1773         {Variable \XINT_unvar_tmpa\space has been
1774         \ifxintglobaldefs globally \fi ``unassigned''.}%
1775     \fi
1776 \fi
1777 \else
1778     \xintMessage {xintexpr}{Warning}
1779     {Error: there was no such variable \XINT_unvar_tmpa\space to unassign.}%
1780 \fi
1781 }%
```

11.24 Support for dummy variables

11.24.1	<code>\xintnewdummy</code>	357
11.24.2	<code>\xintensuredummy</code> , <code>\xintrestorevariable</code>	358
11.24.3	Checking (without expansion) that a symbolic expression contains correctly nested parentheses	359
11.24.4	Fetching balanced expressions E1, E2 and a variable name Name from E1, Name=E2) .	360
11.24.5	Fetching a balanced expression delimited by a semi-colon	360
11.24.6	Low-level support for omit and abort keywords, the <code>break()</code> function, the <code>n++</code> construct and the semi-colon as used in the syntax of <code>seq()</code> , <code>add()</code> , <code>mul()</code> , <code>iter()</code> , <code>rseq()</code> , <code>iterr()</code> , <code>rrseq()</code> , <code>subsm()</code> , <code>subsn()</code> , <code>ndseq()</code> , <code>ndmap()</code>	360
11.24.7	Reserved dummy variables <code>@</code> , <code>@1</code> , <code>@2</code> , <code>@3</code> , <code>@4</code> , <code>@@</code> , <code>@@(1)</code> , <code>...</code> , <code>@@@</code> , <code>@@@(1)</code> , <code>...</code> for recursions	362

11.24.1 `\xintnewdummy`

Comments under reconstruction.

1.4 adds multi-letter names as usable dummy variables!

```

1782 \catcode`* 11
1783 \def\XINT_expr_makedummy #1%
1784 {%
1785     \edef\XINT_tmpa{\xint_zapspace #1 \xint_gobble_i}%
1786     \ifcsname XINT_expr_var_\XINT_tmpa\endcsname
1787         \XINT_global
1788         \expandafter\let\csname XINT_expr_var_\XINT_tmpa\old\expandafter\endcsname
1789         \csname XINT_expr_var_\XINT_tmpa\expandafter\endcsname
1790     \fi
1791     \ifcsname XINT_expr_onliteral_\XINT_tmpa\endcsname
1792         \XINT_global
1793         \expandafter\let\csname XINT_expr_onliteral_\XINT_tmpa\old\expandafter\endcsname
1794         \csname XINT_expr_onliteral_\XINT_tmpa\expandafter\endcsname
1795     \fi
1796     \expandafter\XINT_global
1797     \expanded
```

```

1798 {\edef\expandafter\noexpand
1799     \csname XINT_expr_var_\XINT_tmpa\endcsname ##1\relax !\XINT_tmpa##2}%
1800     {{##2}##1\relax !\XINT_tmpa{##2}}}%
1801 \expandafter\XINT_global
1802 \expanded
1803 {\edef\expandafter\noexpand
1804     \csname XINT_expr_onliteral_\XINT_tmpa\endcsname ##1\relax !\XINT_tmpa##2}%
1805     {\XINT_expr_precedence_*** *{##2}({##1\relax !\XINT_tmpa{##2}})%
1806 }%
1807 \xintApplyUnbraced \XINT_expr_makedummy {abcdefghijklmnopqrstuvwxyz}%
1808 \xintApplyUnbraced \XINT_expr_makedummy {ABCDEFGHIJKLMNOPQRSTUVWXYZ}%
1809 \def\xintnewdummy #1{%
1810     \XINT_expr_makedummy{#1}%
1811     \ifxintverbose\xintMessage {xintexpr}{Info}%
1812     {\XINT_tmpa\space now
1813     \ifxintglobaldefs globally \fi usable as dummy variable.}%
1814     \fi
1815 }%
1816 % \begin{macrocode}
1817 % Je ne définis pas de onliteral for them (it only serves for allowing
1818 % tacit multiplication if variable name is in front of an opening
1819 % parenthesis).
1820 %
1821 % The |nil| variable was need in |xint < 1.4| (with some other meaning)
1822 % in places the syntax could not allow emptiness, such as |,,|, and
1823 % other things, but at |1.4| meaning as changed.
1824 %
1825 % The other variables are new with |1.4|.
1826 % Don't use the |None|, it is tentative, and may be input as |[ ]|.
1827 % \begin{macrocode}
1828 \def\XINT_expr_var_nil{{{}}}%
1829 \def\XINT_expr_var_None{{{}}}% ? tentative
1830 \def\XINT_expr_var_false{{{0}}}% Maple, TeX
1831 \def\XINT_expr_var_true{{{1}}}%
1832 \def\XINT_expr_var_False{{{0}}}% Python
1833 \def\XINT_expr_var_True{{{1}}}%
1834 \catcode`* 12

```

11.24.2 \xintensuredummy, \xintrestorevariable

1.3e \xintensuredummy differs from \xintnewdummy only in the informational message... Attention that this is not meant to be nested.

1.4 fixes that the message mentioned non-existent \xintrestoredummy (real name was \xintrestorelettervar and renames the latter to \xintrestorevariable as it applies also to multi-letter names.

```

1835 \def\xintensuredummy #1{%
1836     \XINT_expr_makedummy{#1}%
1837     \ifxintverbose\xintMessage {xintexpr}{Info}%
1838     {\XINT_tmpa\space now
1839     \ifxintglobaldefs globally \fi usable as dummy variable.&&}
1840     Issue \string\xintrestorevariable{\XINT_tmpa} to restore former meaning.}%
1841     \fi

```

```

1842 }%
1843 \def\xintrestorevariablesilently #1{%
1844   \edef\XINT_tmpa{\xint_zapspaces #1 \xint_gobble_i}%
1845   \ifcsname XINT_expr_var_\XINT_tmpa/old\endcsname
1846     \XINT_global
1847     \expandafter\let\csname XINT_expr_var_\XINT_tmpa\expandafter\endcsname
1848       \csname XINT_expr_var_\XINT_tmpa/old\expandafter\endcsname
1849   \fi
1850   \ifcsname XINT_expr_onliteral_\XINT_tmpa/old\endcsname
1851     \XINT_global
1852     \expandafter\let\csname XINT_expr_onliteral_\XINT_tmpa\expandafter\endcsname
1853       \csname XINT_expr_onliteral_\XINT_tmpa/old\expandafter\endcsname
1854   \fi
1855 }%
1856 \def\xintrestorevariable #1{%
1857   \xintrestorevariablesilently {#1}%
1858   \ifxintverbose\xintMessage {xintexpr}{Info}%
1859     {\XINT_tmpa\space
1860     \ifxintglobaldefs globally \fi restored to its earlier status, if any.}%
1861   \fi
1862 }%

```

11.24.3 Checking (without expansion) that a symbolic expression contains correctly nested parentheses

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. Call it as:

`\XINT_isbalanced_a \relax #1(\xint_bye)\xint_bye`

This is legacy f-expandable code not using `\expanded` even at 1.4.

```

1863 \def\XINT_isbalanced_a #1({\XINT_isbalanced_b #1)\xint_bye }%
1864 \def\XINT_isbalanced_b #1)#2%
1865   {\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

1866 \def\XINT_isbalanced_error #1)\xint_bye {\xint_c_mone}%

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

1867 \def\XINT_isbalanced_c\xint_bye\XINT_isbalanced_error #1%
1868   {\xint_bye #1\XINT_isbalanced_yes\xint_bye\XINT_isbalanced_d #1}%

   #1 is \xint_bye, there was never ( nor ) in original #1, hence OK.

1869 \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.

1870 \def\XINT_isbalanced_d #1)#2%
1871   {\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.

1872 \def\XINT_isbalanced_no\xint_bye #1\xint_bye\xint_bye {\xint_c_i }%

```

11.24.4 Fetching balanced expressions E1, E2 and a variable name Name from E1, Name=E2)

Multi-letter dummy variables added at 1.4.

```

1873 \def\XINT_expr_fetch_E_comma_V_equal_E_a #1#2,%
1874 {%
1875     \ifcase\XINT_isbalanced_a \relax #1#2(\xint_bye)\xint_bye
1876         \expandafter\XINT_expr_fetch_E_comma_V_equal_E_c
1877         \or\expandafter\XINT_expr_fetch_E_comma_V_equal_E_b
1878         \else\expandafter\xintError:noopening
1879     \fi {#1#2},%
1880}%
1881 \def\XINT_expr_fetch_E_comma_V_equal_E_b #1,%
1882     {\XINT_expr_fetch_E_comma_V_equal_E_a {#1},}%
1883 \def\XINT_expr_fetch_E_comma_V_equal_E_c #1,#2#3=%
1884 {%
1885     \expandafter\XINT_expr_fetch_E_comma_V_equal_E_d\expandafter
1886     {\expanded{{\xint_zapspaces #2#3 \xint_gobble_i}}{#1}}}%
1887}%
1888 \def\XINT_expr_fetch_E_comma_V_equal_E_d #1#2#3)%
1889 {%
1890     \ifcase\XINT_isbalanced_a \relax #2#3(\xint_bye)\xint_bye
1891         \or\expandafter\XINT_expr_fetch_E_comma_V_equal_E_e
1892         \else\expandafter\xintError:noopening
1893     \fi
1894     {#1}{#2#3}%
1895}%
1896 \def\XINT_expr_fetch_E_comma_V_equal_E_e #1#2{\XINT_expr_fetch_E_comma_V_equal_E_d {#1}{#2}}}%

```

11.24.5 Fetching a balanced expression delimited by a semi-colon

1.4. For subsn() leaner syntax of nested substitutions.

Will also serve to \xintdeffunc, to not have to hide inner semi-colons in for example an iter() from \xintdeffunc.

Adding brace removal protection for no serious reason, anyhow the xintexpr parsers always removes braces when moving forward, but well.

Trigger by \romannumeral\XINT_expr_fetch_to_semicolon upfront.

```

1897 \def\XINT_expr_fetch_to_semicolon {\XINT_expr_fetch_to_semicolon_a {} \empty}%
1898 \def\XINT_expr_fetch_to_semicolon_a #1#2;%
1899 {%
1900     \ifcase\XINT_isbalanced_a \relax #1#2(\xint_bye)\xint_bye
1901         \xint_dothis{\expandafter\XINT_expr_fetch_to_semicolon_c}%
1902         \or\xint_dothis{\expandafter\XINT_expr_fetch_to_semicolon_b}%
1903         \else\expandafter\xintError:noopening
1904     \fi\xint_orthat{} \expandafter{#2}{#1}%
1905}%
1906 \def\XINT_expr_fetch_to_semicolon_b #1#2{\XINT_expr_fetch_to_semicolon_a {#2#1;} \empty}%
1907 \def\XINT_expr_fetch_to_semicolon_c #1#2{\xint_c_{#2#1}}}%

```

11.24.6 Low-level support for omit and abort keywords, the break() function, the n++ construct and the semi-colon as used in the syntax of seq(), add(), mul(), iter(), rseq(), iterr(), rrseq(), subsm(), subsn(), ndseq(), ndmap()

There is some clever play simply based on setting suitable precedence levels combined with special meanings given to op macros.

The special !? internal operator is a helper for omit and abort keywords in list generators.

Prior to 1.4 support for +[, *[, ...,]+,]*, had some elements here.

The n++ construct 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.

```
1908 \expandafter\def\csname XINT_expr_itself_++\endcsname {++}%
1909 \expandafter\def\csname XINT_expr_itself_++\endcsname {++}%
1910 \expandafter\let\csname XINT_expr_precedence_++\endcsname \xint_c_i
1911 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1912     \expandafter\def\csname XINT_#1_op_++\endcsname ##1##2\relax
1913     {\expandafter\XINT_expr_foundend
1914         \expanded{{+{\XINT:NEhook:f:one:from:one:direct\xintiCeil##1}}}%
1915     }%
1916 }%
```

The break() function break is a true function, the parsing via expansion of the enclosed material proceeds via _oparen macros as with any other function.

```
1917 \catcode`? 3
1918 \def\XINT_expr_func_break #1#2#3{#1#2{?#3}}%
1919 \catcode`? 11
1920 \let\XINT_flexpr_func_break \XINT_expr_func_break
1921 \let\XINT_iiexpr_func_break \XINT_expr_func_break
```

The omit and abort keywords Comments are currently undergoing reconstruction.

```
1922 \edef\XINT_expr_var_omit #1\relax !{1\string !?!\relax !}%
1923 \edef\XINT_expr_var_abort #1\relax !{1\string !?^\relax !}%
1924 \def\XINT_expr_itself_!? {!}%
1925 \def\XINT_expr_op_!? #1#2\relax{\XINT_expr_foundend{#2}}%
1926 \let\XINT_iiexpr_op_!? \XINT_expr_op_!?
1927 \let\XINT_flexpr_op_!? \XINT_expr_op_!?
1928 \let\XINT_expr_precedence_!? \xint_c_iv
```

The semi-colon Obsolete comments undergoing re-construction

```
1929 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1930     \expandafter\def\csname XINT_#1_op_;\endcsname {\xint_c_i ;}%
1931 }%
1932 \expandafter\let\csname XINT_expr_precedence_;\endcsname\xint_c_i
1933 \expandafter\def\csname XINT_expr_itself_;\endcsname {}%
1934 \expandafter\let\csname XINT_expr_precedence_;\endcsname\xint_c_i
```

11.24.7 Reserved dummy variables @, @1, @2, @3, @4, @@, @@(1), ..., @@@, @@@(1), ... for recursions

Comments currently under reconstruction.

1.4 breaking change: @ and @1 behave differently and one can not use @ in place of @1 in iterr() and rrseq(). Formerly @ and @1 had the same definition.

Brace stripping in \XINT_expr_func_@@ is prevented by some ending 0 or other token see iterr() and rrseq() code.

For the record, the ~ and ? have catcode 3 in this code.

```

1935 \catcode`* 11
1936 \def\XINT_expr_var_@ #1~#2{{#2}#1~{#2}}%
1937 \def\XINT_expr_onliteral_@ #1~#2{\XINT_expr_precedence_*** *{#2}({#1~{#2}}}%
1938 \expandafter
1939 \def\csname XINT_expr_var_@1\endcsname #1~#2{{#2}#1~{#2}}%
1940 \expandafter
1941 \def\csname XINT_expr_var_@2\endcsname #1~#2#3{{#3}#1~{#2}{#3}}%
1942 \expandafter
1943 \def\csname XINT_expr_var_@3\endcsname #1~#2#3#4{{#4}#1~{#2}{#3}{#4}}%
1944 \expandafter
1945 \def\csname XINT_expr_var_@4\endcsname #1~#2#3#4#5{{#5}#1~{#2}{#3}{#4}{#5}}%
1946 \expandafter\def\csname XINT_expr_onliteral_@1\endcsname #1~#2%
1947         {\XINT_expr_precedence_*** *{#2}({#1~{#2}}}%
1948 \expandafter\def\csname XINT_expr_onliteral_@2\endcsname #1~#2#3%
1949         {\XINT_expr_precedence_*** *{#3}({#1~{#2}{#3}}}%
1950 \expandafter\def\csname XINT_expr_onliteral_@3\endcsname #1~#2#3#4%
1951         {\XINT_expr_precedence_*** *{#4}({#1~{#2}{#3}{#4}}}%
1952 \expandafter\def\csname XINT_expr_onliteral_@4\endcsname #1~#2#3#4#5%
1953         {\XINT_expr_precedence_*** *{#5}({#1~{#2}{#3}{#4}{#5}}}%
1954 \catcode`* 12
1955 \catcode`? 3
1956 \def\XINT_expr_func_@@ #1#2#3#4~#5?%
1957 {%
1958     \expandafter#1\expandafter#2\expandafter{\expandafter{%
1959         \romannumeral0\xintntheltnoexpand{\xintNum#3}{#5}}#4~#5?%
1960 }}%
1961 \def\XINT_expr_func_@@@ #1#2#3#4~#5~#6?%
1962 {%
1963     \expandafter#1\expandafter#2\expandafter{\expandafter{%
1964         \romannumeral0\xintntheltnoexpand{\xintNum#3}{#6}}#4~#5~#6?%
1965 }}%
1966 \def\XINT_expr_func_@@@@ #1#2#3#4~#5~#6~#7?%
1967 {%
1968     \expandafter#1\expandafter#2\expandafter{\expandafter{%
1969         \romannumeral0\xintntheltnoexpand{\xintNum#3}{#7}}#4~#5~#6~#7?%
1970 }}%
1971 \let\XINT_flexpr_func_@@\XINT_expr_func_@@
1972 \let\XINT_flexpr_func_@@@\XINT_expr_func_@@@
1973 \let\XINT_flexpr_func_@@@@\XINT_expr_func_@@@@
1974 \def\XINT_iiexpr_func_@@ #1#2#3#4~#5?%
1975 {%
1976     \expandafter#1\expandafter#2\expandafter{\expandafter{%
1977         \romannumeral0\xintntheltnoexpand{\xint_firstofone#3}{#5}}#4~#5?%

```

```

1978 }%
1979 \def\XINT_iiexpr_func_@@@ #1#2#3#4~#5~#6?%
1980 {%
1981     \expandafter#1\expandafter#2\expandafter{\expandafter{%
1982         \romannumeral0\xintntheltnoexpand{\xint_firstofone#3}{#6}}}\#4~#5~#6?%
1983 }%
1984 \def\XINT_iiexpr_func_@@@ #1#2#3#4~#5~#6~#7?%
1985 {%
1986     \expandafter#1\expandafter#2\expandafter{\expandafter{%
1987         \romannumeral0\xintntheltnoexpand{\xint_firstofone#3}{#7}}}\#4~#5~#6~#7?%
1988 }%
1989 \catcode`? 11

```

11.25 Pseudo-functions involving dummy variables and generating scalars or sequences

11.25.1	Comments	363
11.25.2	subs(): substitution of one variable	365
11.25.3	subsm(): simultaneous independent substitutions	365
11.25.4	subsn(): leaner syntax for nesting (possibly dependent) substitutions	366
11.25.5	seq(): sequences from assigning values to a dummy variable	368
11.25.6	iter()	368
11.25.7	add(), mul()	370
11.25.8	rseq()	371
11.25.9	iterr()	372
11.25.10	rrseq()	373

11.25.1 Comments

Comments added 2020/01/16.

The mechanism for «seq» is the following. When the parser encounters «seq», which means it parsed these letters and encountered (from expansion) an opening parenthesis, the `\XINT_expr_func` mechanism triggers the «`» operator which realizes that «seq» is a pseudo-function (there is no `_func_seq`) and thus spans the `\XINT_expr_onliteral_seq` macro (currently this means however that the knowledge of which parser we are in is lost, see comments of `\XINT_expr_op`` code). The latter will use delimited macros and parenthesis check to fetch (without any expansion), the symbolic expression `ExprSeq` to evaluate, the `Name` (now possibly multi-letter) of the variable and the expression `ExprValues` to evaluate which will give the values to assign to the dummy variable `Name`. It then positions upstream `ExprValues` suitably terminated (see next) and after it `{{Name}{ExprSeq}}`. Then it inserts a second call to the «`» operator with now «seqx» as argument hence the appropriate «{,fl,ii}expr_func_seqx» macros gets executed. The general way function macros work is that first all their arguments are evaluated via a call not to `\xintbare{,float,ii}eval` but to the suitable `\XINT_{expr,flexpr,iiexpr}_oparen` core macro which does almost same excepts it expects a final closing parenthesis (of course allowing nested parenthesis in-between) and stops there. Here, this closing parenthesis got positioned deliberately with a `\relax` after it, so the parser, which always after having gathered a value looks ahead to find the next operator, thinks it has hit the end of the expression and as result inserts a `\xint_c_` (i.e. `\z@`) token for precedence level and a dummy `\relax` token (place-holder for a non-existing operator). Generally speaking «func_foo» macros expect to be executed with three parameters `#1#2#3`, `#1` = precedence, `#2` = operator, `#3` = values (call it «args») i.e. the fully evaluated list of all its arguments. The special «func_seqx» and cousins know that the first two tokens are trash and they now proceed forward, having thus lying before them upstream the values to loop over, now fully evaluated, and `{{Name}{ExprSeq}}`.

It then positions appropriately ExprSeq inside a sub-expression and after it, following suitable delimiter, Name and the evaluated values to assign to Name.

Dummy variables are essentially simply delimited macros where the delimiter is the variable name preceded by a \relax token and a catcode 11 exclamation point. Thus the various «subsx», «seqx», «iterx» position the tokens appropriately and launch suitable loops.

All of this nests well, inner «seq»'s (or more often in practice «subs»'s) being allowed to refer to the dummy variables used by outer «seq»'s because the outer «seq»'s have the values to assign to their variables evaluated first and their ExprSeq evaluated last. For inner dummy variables to be able to refer to outer dummy variables the author must be careful of course to not use in the implementation braces { and } which would break dummy variables to fetch values beyond the closing brace.

The above «seq» mechanism was done around June 15-25th 2014 at the time of the transition from 1.09n to 1.1 but already in October 2014 I made a note that I had a hard time to understand it again:

« [START OF YEAR 2014 COMMENTS]

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_fetch_E_comma_V_equal_E_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_ii^v in seq_f is for the recurrent thing that we don't know in what type of expressions we are, hence we must move back up, with some loss of efficiency (superfluous check for minus sign, etc...). But the code manages simultaneously expr, flexpr and iiexpr.

[END OF YEAR 2014 OLD COMMENTS]»

On Jeudi 16 janvier 2020 à 15:13:32 I finally did the documentation as above.

The case of «iter», «rseq», «iterr», «rrseq» differs slightly because the initial values need evaluation. This is done by genuine functions \XINT_<parser>_func_iter etc... (there was no \XINT_<parser>_func_seq). The trick is via the semi-colon ; which is a genuine operator having the precedence of a closing parenthesis and whose action is only to stop expansion. Thus this first step of gathering the initial values is done as part of the regular expansion job of the parser not using delimited macros and the ; can be hidden in braces {} because the three parsers when moving forward remove one level of braces always. Thus \XINT_<parser>_func_seq simply hand over to \XINT_allexpr_iter which will then trigger the fetching without expansion of ExprIter, Name=ExprValues as described previously for «seq».

With 1.4, multi-letter names for dummy variables are allowed.

Also there is the additional 1.4 ambition to make the whole thing parsable by \xintNewExpr/\xintdeffunc. This is done by checking if all is numerical, because the omit, abort and break() mechanisms have no translation into macros, and the only solution for symbolic material is to simply keep it as is, so that expansion will again activate the xintexpr parsers. At 1.4 this approach is fine although the initial goals of \xintNewExpr/\xintdeffunc was to completely replace the parsers (whose storage method hit the string pool formerly) by macros. Now that 1.4 does not impact the string pool we can make \xintdeffunc much more powerful but it will not be a construct using only xintfrac macros, it will still be partially the \xintexpr etc... parsers in such cases.

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.

Refactored at 1.4 using \expanded rather than \csname.

And support for multi-letter variables, which means function declarations can now use multi-letter variables !

11.25.2 subs(): substitution of one variable

```
1990 \def\XINT_expr_onliteral_subs
1991 {%
1992   \expandafter\XINT_allexpr_subsx_f
1993   \romannumeral`&&\XINT_expr_fetch_E_comma_V_equal_E_a {%}
1994 }%
1995 \def\XINT_allexpr_subsx_f #1#2{\xint_c_ii^v`{subsx}#2)\relax #1}%
1996 \def\XINT_expr_func_subsx #1#2{\XINT_allexpr_subsx \xintbareeval}%
1997 \def\XINT_flexpr_func_subsx #1#2{\XINT_allexpr_subsx \xintbarefloateval}%
1998 \def\XINT_iexpr_func_subsx #1#2{\XINT_allexpr_subsx \xintbareieval}%

```

#2 is the value to assign to the dummy variable #3 is the dummy variable name (possibly multi-letter), #4 is the expression to evaluate

```
1999 \def\XINT_allexpr_subsx #1#2#3#4%
2000 {%
2001   \expandafter\XINT_expr_put_op_first
2002   \expanded
2003   \bgroup\romannumeral0#1#4\relax \iffalse\relax !#3{#2}{\fi
2004   \expandafter}\romannumeral`&&\XINT_expr_getop
2005 }%

```

11.25.3 subsm(): simultaneous independent substitutions

New with 1.4. Globally the var1=expr1; var2=expr2; var2=expr3;... part can arise from expansion, except that once a semi-colon has been found (from expansion) the varK= thing following it must be there. And as for subs() the final parenthesis must be there from the start.

```
2006 \def\XINT_expr_onliteral_subsm
2007 {%
2008   \expandafter\XINT_allexpr_subsm_f
2009   \romannumeral`&&\XINT_expr_fetch_E_comma_V_equal_E_a {%}
2010 }%
2011 \def\XINT_allexpr_subsm_f #1#2{\xint_c_ii^v`{subsmx}#2)\relax #1}%
2012 \def\XINT_expr_func_subsmx
2013 {%
2014   \expandafter\XINT_allexpr_subsmx\expandafter\xintbareeval
2015   \expanded\bgroup{\iffalse}\fi\XINT_allexpr_subsm_A\XINT_expr_oparen
2016 }%
2017 \def\XINT_flexpr_func_subsmx
2018 {%
2019   \expandafter\XINT_allexpr_subsmx\expandafter\xintbarefloateval
2020   \expanded\bgroup{\iffalse}\fi\XINT_allexpr_subsm_A\XINT_flexpr_oparen
2021 }%
2022 \def\XINT_iexpr_func_subsmx
2023 {%
2024   \expandafter\XINT_allexpr_subsmx\expandafter\xintbareieval

```

```

2025 \expanded\bgroup{\iffalse}\fi\XINT_allexpr_subsm_A\XINT_iiexpr_oparen
2026 }%
2027 \def\XINT_allexpr_subsm_A #1#2#3%
2028 {%
2029 \ifx#2\xint_c_
2030 \expandafter\XINT_allexpr_subsm_done
2031 \else
2032 \expandafter\XINT_allexpr_subsm_B
2033 \fi #1%
2034 }%
2035 \def\XINT_allexpr_subsm_B #1#2#3#4=%
2036 {%
2037 {#2}\relax !\xint_zapspaces#3#4 \xint_gobble_i
2038 \expandafter\XINT_allexpr_subsm_A\expandafter#1\romannumeral`&&@#1%
2039 }%

#1 = \xintbareeval, or \xintbarefloateval or \xintbareiieval #2 = evaluation of last variable
assignment

2040 \def\XINT_allexpr_subsm_done #1#2{{#2}\iffalse{{\fi}}}%

#1 = \xintbareeval or \xintbarefloateval or \xintbareiieval #2 = {value1}\relax !var2{value2}...\relax
!varN{valueN} (value's may be oples) #3 = {var1} #4 = the expression to evaluate

2041 \def\XINT_allexpr_subsmx #1#2#3#4%
2042 {%
2043 \expandafter\XINT_expr_put_op_first
2044 \expanded
2045 \bgroup\romannumeral0#1#4\relax \iffalse\relax !#3#2{\fi
2046 \expandafter}\romannumeral`&&@\XINT_expr_getop
2047 }%

```

11.25.4 subsn(): leaner syntax for nesting (possibly dependent) substitutions

New with 1.4. 2020/01/24

```

2048 \def\XINT_expr_onliteral_subsn
2049 {%
2050 \expandafter\XINT_allexpr_subsn_f
2051 \romannumeral`&&@\XINT_expr_fetch_E_comma_V_equal_E_a {}%
2052 }%
2053 \def\XINT_allexpr_subsn_f #1{\XINT_allexpr_subsn_g #1}%

#1 = Name1
#2 = Expression in all variables which is to evaluate
#3 = all the stuff after Name1 = and up to final parenthesis

2054 \def\XINT_allexpr_subsn_g #1#2#3%
2055 {%
2056 \expandafter\XINT_allexpr_subsn_h
2057 \expanded\bgroup{\iffalse}\fi\expandafter\XINT_allexpr_subsn_B
2058 \romannumeral\XINT_expr_fetch_to_semicolon #1=#3;\hbox=;;^{#2}%
2059 }%
2060 \def\XINT_allexpr_subsn_B #1{\XINT_allexpr_subsn_C #1\vbox}%

```

```

2061 \def\XINT_allexpr_subsn_C #1#2=#3\vbox
2062 {%
2063   \ifx\hbox#1\iffalse{\fi}\expandafter\else
2064   {\xint_zapspaces #1#2 \xint_gobble_i}};\unexpanded{{#3}}}%
2065   \expandafter\XINT_allexpr_subsn_B
2066   \romannumeral\expandafter\XINT_expr_fetch_to_semicolon\fi
2067 }%
2068 \def\XINT_allexpr_subsn_h
2069 {%
2070   \xint_c_ii^v `{subsnx}\romannumeral0\xintreverseorder
2071 }%
2072 \def\XINT_expr_func_subsnx #1#2#3#4#5;#6%
2073 {%
2074   \xint_gob_til_ ^ #6\XINT_allexpr_subsnx_H ^%
2075   \expandafter\XINT_allexpr_subsnx\expandafter
2076   \xintbareeval\romannumeral0\xintbareeval #5\relax !#4{#3}\xintundefined
2077   {\relax !#4{#3}\relax !#6}%
2078 }%
2079 \def\XINT_iiexpr_func_subsnx #1#2#3#4#5;#6%
2080 {%
2081   \xint_gob_til_ ^ #6\XINT_allexpr_subsnx_H ^%
2082   \expandafter\XINT_allexpr_subsnx\expandafter
2083   \xintbareiieval\romannumeral0\xintbareiieval #5\relax !#4{#3}\xintundefined
2084   {\relax !#4{#3}\relax !#6}%
2085 }%
2086 \def\XINT_flexpr_func_subsnx #1#2#3#4#5;#6%
2087 {%
2088   \xint_gob_til_ ^ #6\XINT_allexpr_subsnx_H ^%
2089   \expandafter\XINT_allexpr_subsnx\expandafter
2090   \xintbarefloateval\romannumeral0\xintbarefloateval #5\relax !#4{#3}\xintundefined
2091   {\relax !#4{#3}\relax !#6}%
2092 }%
2093 \def\XINT_allexpr_subsnx #1#2!#3\xintundefined#4#5;#6%
2094 {%
2095   \xint_gob_til_ ^ #6\XINT_allexpr_subsnx_I ^%
2096   \expandafter\XINT_allexpr_subsnx\expandafter
2097   #1\romannumeral0#1#5\relax !#4{#2}\xintundefined
2098   {\relax !#4{#2}\relax !#6}%
2099 }%
2100 \def\XINT_allexpr_subsnx_H ^#1\romannumeral0#2#3!#4\xintundefined #5#6%
2101 {%
2102   \expandafter\XINT_allexpr_subsnx_J\romannumeral0#2#6#5%
2103 }%
2104 \def\XINT_allexpr_subsnx_I ^#1\romannumeral0#2#3\xintundefined #4#5%
2105 {%
2106   \expandafter\XINT_allexpr_subsnx_J\romannumeral0#2#5#4%
2107 }%
2108 \def\XINT_allexpr_subsnx_J #1#2^%
2109 {%
2110   \expandafter\XINT_expr_put_op_first
2111   \expanded{\unexpanded{{#1}}\expandafter}\romannumeral`&&\XINT_expr_getop
2112 }%

```


11.25.5 seq(): sequences from assigning values to a dummy variable

In seq_f, the #2 is the ExprValues expression which needs evaluation to provide the values to the dummy variable and #1 is {Name}{ExprSeq} where Name is the name of dummy variable and {ExprSeq} the expression which will have to be evaluated.

```

2113 \def\XINT_allexpr_seq_f #1#2{\xint_c_ii^v `{seqx}#2)\relax #1}%
2114 \def\XINT_expr_onliteral_seq
2115 {\expandafter\XINT_allexpr_seq_f\romannumeral`&&\XINT_expr_fetch_E_comma_V_equal_E_a {}}%
2116 \def\XINT_expr_func_seqx #1#2{\XINT:NEhook:seqx\XINT_allexpr_seqx\xintbareeval}%
2117 \def\XINT_flexpr_func_seqx #1#2{\XINT:NEhook:seqx\XINT_allexpr_seqx\xintbarefloateval}%
2118 \def\XINT_iexpr_func_seqx #1#2{\XINT:NEhook:seqx\XINT_allexpr_seqx\xintbareiieval}%
2119 \def\XINT_allexpr_seqx #1#2#3#4%
2120 {%
2121     \expandafter\XINT_expr_put_op_first
2122     \expanded \bgroup {\iffalse}\fi\XINT_expr_seq:_b {#1#4\relax !#3}#2^%
2123     \XINT_expr_cb_and_getop
2124 }%
2125 \def\XINT_expr_cb_and_getop{\iffalse{\fi\expandafter}\romannumeral`&&\XINT_expr_getop}%

```

Comments undergoing reconstruction.

```

2126 \catcode`? 3
2127 \def\XINT_expr_seq:_b #1#2%
2128 {%
2129     \ifx +#2\xint_dothis\XINT_expr_seq:_Ca\fi
2130     \ifx !#2!\xint_dothis\XINT_expr_seq:_noop\fi
2131     \ifx ^#2\xint_dothis\XINT_expr_seq:_end\fi
2132     \xint_orthat{\XINT_expr_seq:_c}{#2}{#1}%
2133 }%
2134 \def\XINT_expr_seq:_noop #1{\XINT_expr_seq:_b}%
2135 \def\XINT_expr_seq:_end #1#2{\iffalse{\fi}}%
2136 \def\XINT_expr_seq:_c #1#2{\expandafter\XINT_expr_seq:_d\romannumeral0#2{{#1}}{{#2}}}%
2137 \def\XINT_expr_seq:_d #1{\ifx ^#1\xint_dothis\XINT_expr_seq:_abort\fi
2138     \ifx ?#1\xint_dothis\XINT_expr_seq:_break\fi
2139     \ifx !#1\xint_dothis\XINT_expr_seq:_omit\fi
2140     \xint_orthat{\XINT_expr_seq:_goon {#1}}}%
2141 \def\XINT_expr_seq:_abort #1!#2^{\iffalse{\fi}}%
2142 \def\XINT_expr_seq:_break #1!#2^{\iffalse{\fi}}%
2143 \def\XINT_expr_seq:_omit #1!#2#\expandafter\XINT_expr_seq:_b\xint_gobble_i}%
2144 \def\XINT_expr_seq:_goon #1!#2#{\iffalse{\fi}\expandafter\XINT_expr_seq:_b\xint_gobble_i}%
2145 \def\XINT_expr_seq:_Ca #1#2#3{\XINT_expr_seq:_Cc#3.#2}%
2146 \def\XINT_expr_seq:_Cb #1{\expandafter\XINT_expr_seq:_Cc\the\numexpr#1+\xint_c_i}%
2147 \def\XINT_expr_seq:_Cc #1.#2{\expandafter\XINT_expr_seq:_D\romannumeral0#2{{#1}}{{#1}}{{#2}}}%
2148 \def\XINT_expr_seq:_D #1{\ifx ^#1\xint_dothis\XINT_expr_seq:_abort\fi
2149     \ifx ?#1\xint_dothis\XINT_expr_seq:_break\fi
2150     \ifx !#1\xint_dothis\XINT_expr_seq:_omit\fi
2151     \xint_orthat{\XINT_expr_seq:_Goon {#1}}}%
2152 \def\XINT_expr_seq:_Omit #1!#2#\expandafter\XINT_expr_seq:_Cb\xint_gobble_i}%
2153 \def\XINT_expr_seq:_Goon #1!#2#{\iffalse{\fi}\expandafter\XINT_expr_seq:_Cb\xint_gobble_i}%

```

11.25.6 iter()

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.

To understand the tokens which are presented to \XINT_allexpr_iter it is needed to check elsewhere in the source code how the ; hack is done.

The #2 in \XINT_allexpr_iter is \xint_c_i from the ; hack. Formerly (xint < 1.4) there was no such token. The change is motivated to using ; also in subsm() syntax.

```

2154 \def\XINT_expr_func_iter    {\XINT_allexpr_iter \xintbareeval    }%
2155 \def\XINT_flexpr_func_iter {\XINT_allexpr_iter \xintbarefloateval }%
2156 \def\XINT_iiexpr_func_iter {\XINT_allexpr_iter \xintbareiieval    }%
2157 \def\XINT_allexpr_iter #1#2#3#4%
2158 {%
2159     \expandafter\XINT_expr_iterx
2160     \expandafter#1\expanded{\unexpanded{#{4}}\expandafter}%
2161     \romannumeral`&&\XINT_expr_fetch_E_comma_V_equal_E_a {}%
2162 }%
2163 \def\XINT_expr_iterx #1#2#3#4%
2164 {%
2165     \XINT:NEhook:iter\XINT_expr_itery\romannumeral0#1(#{4})\relax {#2}#3#1%
2166 }%
2167 \def\XINT_expr_itery #1#2#3#4#5%
2168 {%
2169     \expandafter\XINT_expr_put_op_first
2170     \expanded \bgroup {\iffalse}\fi
2171     \XINT_expr_iter:_b {#5#4}\relax !#3}#1^~{#2}\XINT_expr_cb_and_getop
2172 }%
2173 \def\XINT_expr_iter:_b #1#2%
2174 {%
2175     \ifx +#2\xint_dothis\XINT_expr_iter:_Ca\fi
2176     \ifx !#2!\xint_dothis\XINT_expr_iter:_noop\fi
2177     \ifx ^#2\xint_dothis\XINT_expr_iter:_end\fi
2178     \xint_orthat{\XINT_expr_iter:_c}{#2}{#1}%
2179 }%
2180 \def\XINT_expr_iter:_noop #1{\XINT_expr_iter:_b }%
2181 \def\XINT_expr_iter:_end  #1#2~#3{#3\iffalse{\fi}}%
2182 \def\XINT_expr_iter:_c #1#2{\expandafter\XINT_expr_iter:_d\romannumeral0#2{#{1}}{#2}}%
2183 \def\XINT_expr_iter:_d #1{\ifx ^#1\xint_dothis\XINT_expr_iter:_abort\fi
2184     \ifx ?#1\xint_dothis\XINT_expr_iter:_break\fi
2185     \ifx !#1\xint_dothis\XINT_expr_iter:_omit\fi
2186     \xint_orthat{\XINT_expr_iter:_goon {#1}}}%
2187 \def\XINT_expr_iter:_abort #1!#2^~#3{#3\iffalse{\fi}}%
2188 \def\XINT_expr_iter:_break #1!#2^~#3{#1\iffalse{\fi}}%
2189 \def\XINT_expr_iter:_omit  #1!#2#{\expandafter\XINT_expr_iter:_b\xint_gobble_i}%
2190 \def\XINT_expr_iter:_goon  #1!#2#{\XINT_expr_iter:_goon_a {#1}}%
2191 \def\XINT_expr_iter:_goon_a #1#2#3~#4{\XINT_expr_iter:_b #3~{#1}}%
2192 \def\XINT_expr_iter:_Ca #1#2#3{\XINT_expr_iter:_Cc#3.{#2}}%
2193 \def\XINT_expr_iter:_Cb #1{\expandafter\XINT_expr_iter:_Cc\the\numexpr#1+\xint_c_i.}%
2194 \def\XINT_expr_iter:_Cc #1.#2{\expandafter\XINT_expr_iter:_D\romannumeral0#2{#{1}}{#1}{#2}}%
2195 \def\XINT_expr_iter:_D #1{\ifx ^#1\xint_dothis\XINT_expr_iter:_abort\fi
2196     \ifx ?#1\xint_dothis\XINT_expr_iter:_break\fi

```

```

2197             \ifx !#1\xint_dothis\xINT_expr_iter:_Omit\fi
2198             \xint_orthat{\XINT_expr_iter:_Goon {#1}}}%
2199 \def\xINT_expr_iter:_Omit #1!#2#\expandafter\xINT_expr_iter:_Cb\xint_gobble_i}%
2200 \def\xINT_expr_iter:_Goon #1!#2#\XINT_expr_iter:_Goon_a {#1}}%
2201 \def\xINT_expr_iter:_Goon_a #1#2#3~#4{\XINT_expr_iter:_Cb #3~{#1}}%

```

11.25.7 add(), mul()

Comments under reconstruction.

These were a bit anomalous as they did not implement omit and abort keyword and the break() function (and per force then neither the n++ syntax).

At 1.4 they are simply mapped to using adequately iter(). Thus, there is small loss in efficiency, but supporting omit, abort and break is important. Using dedicated macros here would have caused also slight efficiency drop. Simpler to remove the old approach.

```

2202 \def\xINT_expr_onliteral_add
2203 {\expandafter\xINT_allexpr_add_f\romannumeral`&&@\XINT_expr_fetch_E_comma_V_equal_E_a {}}%
2204 \def\xINT_allexpr_add_f #1#2{\xint_c_ii^v `{opx}#2)\relax #1{+}{0}}%
2205 \def\xINT_expr_onliteral_mul
2206 {\expandafter\xINT_allexpr_mul_f\romannumeral`&&@\XINT_expr_fetch_E_comma_V_equal_E_a {}}%
2207 \def\xINT_allexpr_mul_f #1#2{\xint_c_ii^v `{opx}#2)\relax #1{*}{1}}%
2208 \def\xINT_expr_func_opx {\XINT:NEhook:opx \XINT_allexpr_opx \xintbareeval }%
2209 \def\xINT_flexpr_func_opx {\XINT:NEhook:opx \XINT_allexpr_opx \xintbarefloateval}%
2210 \def\xINT_iexpr_func_opx {\XINT:NEhook:opx \XINT_allexpr_opx \xintbareieval }%

```

1.4a In case of usage of omit (did I not test it? obviously I didn't as neither omit nor abort could work; and break neither), 1.4 code using (#6) syntax caused a (somewhat misleading) «missing » error message which originated in the #6. This is non-obvious problem (perhaps explained why prior to 1.4 I had not added support for omit and break() to add() and mul()...

Allowing () is not enough as it would have to be 0 or 1 depending on whether we are using add() or mul(). Hence the somewhat complicated detour (relying on precise way var_omit and var_abort work) via \XINT_allexpr_opx_ifnotomitted.

\break() has special meaning here as it is used as last operand, not as last value. The code is very unsatisfactory and inefficient but this is hotfix for 1.4a.

```

2211 \def\xINT_allexpr_opx #1#2#3#4#5#6#7#8%
2212 {%
2213     \expandafter\xINT_expr_put_op_first
2214     \expanded \bgroup {\iffalse}\fi
2215     \XINT_expr_iter:_b {#1%
2216     \expandafter\xINT_allexpr_opx_ifnotomitted
2217     \romannumeral0#1#6\relax#7@\relax !#5}#4^~{{#8}}\XINT_expr_cb_and_getop
2218 }%
2219 \def\xINT_allexpr_opx_ifnotomitted #1%
2220 {%
2221     \ifx !#1\xint_dothis{@\relax}\fi
2222     \ifx ^#1\xint_dothis{\XINTfstop. ^\relax}\fi
2223     \if ?\xintFirstItem{#1}\xint_dothis{\XINT_allexpr_opx_break{#1}}\fi
2224     \xint_orthat{\XINTfstop.{#1}}}%
2225 }%
2226 \def\xINT_allexpr_opx_break #1#2\relax
2227 {%
2228     break(\expandafter\xINTfstop\expandafter.\expandafter{\xint_gobble_i#1}#2)\relax
2229 }%

```

11.25.8 rseq()

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 (and leaving a \xint_c_i left-over token since 1.4). 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.

```

2230 \def\xINT_expr_func_rseq {\XINT_allexpr_rseq \xintbareeval}%
2231 \def\xINT_flexpr_func_rseq {\XINT_allexpr_rseq \xintbarefloateval}%
2232 \def\xINT_iexpr_func_rseq {\XINT_allexpr_rseq \xintbareiieval}%
2233 \def\xINT_allexpr_rseq #1#2#3#4%
2234 {%
2235 \expandafter\xINT_expr_rseqx
2236 \expandafter #1\expanded{\unexpanded{{#4}}\expandafter}%
2237 \romannumeral &&@\XINT_expr_fetch_E_comma_V_equal_E_a {%
2238}%
2239 \def\xINT_expr_rseqx #1#2#3#4%
2240 {%
2241 \XINT:NEhook:rseq \XINT_expr_rseqy\romannumeral0#1(#4)\relax {#2}#3#1%
2242}%
2243 \def\xINT_expr_rseqy #1#2#3#4#5%
2244 {%
2245 \expandafter\xINT_expr_put_op_first
2246 \expanded \bgroup {\iffalse}\fi
2247 #2%
2248 \XINT_expr_rseq:_b {#5#4\relax !#3}#1^~{#2}\XINT_expr_cb_and_getop
2249}%
2250 \def\xINT_expr_rseq:_b #1#2%
2251 {%
2252 \ifx +#2\xint_dothis\xINT_expr_rseq:_Ca\fi
2253 \ifx !#2!\xint_dothis\xINT_expr_rseq:_noop\fi
2254 \ifx ^#2\xint_dothis\xINT_expr_rseq:_end\fi
2255 \xint_orthat{\XINT_expr_rseq:_c}{#2}{#1}%
2256}%
2257 \def\xINT_expr_rseq:_noop #1{\XINT_expr_rseq:_b}%
2258 \def\xINT_expr_rseq:_end #1#2~#3{\iffalse{\fi}}%
2259 \def\xINT_expr_rseq:_c #1#2{\expandafter\xINT_expr_rseq:_d\romannumeral0#2{{#1}}{{#2}}%
2260 \def\xINT_expr_rseq:_d #1{\ifx ^#1\xint_dothis\xINT_expr_rseq:_abort\fi
2261 \ifx ?#1\xint_dothis\xINT_expr_rseq:_break\fi
2262 \ifx !#1\xint_dothis\xINT_expr_rseq:_omit\fi
2263 \xint_orthat{\XINT_expr_rseq:_goon {#1}}}%
2264 \def\xINT_expr_rseq:_abort #1!#2^~#3{\iffalse{\fi}}%
2265 \def\xINT_expr_rseq:_break #1!#2^~#3{#1\iffalse{\fi}}%
2266 \def\xINT_expr_rseq:_omit #1!#2#3{\expandafter\xINT_expr_rseq:_b\xint_gobble_i}%
2267 \def\xINT_expr_rseq:_goon #1!#2#3{\XINT_expr_rseq:_goon_a {#1}}%
2268 \def\xINT_expr_rseq:_goon_a #1#2#3~#4{#1\xINT_expr_rseq:_b #3~{#1}}%
2269 \def\xINT_expr_rseq:_Ca #1#2#3{\XINT_expr_rseq:_Cc#3.{#2}}%
2270 \def\xINT_expr_rseq:_Cb #1{\expandafter\xINT_expr_rseq:_Cc\the\numexpr#1+\xint_c_i.}%
2271 \def\xINT_expr_rseq:_Cc #1.#2{\expandafter\xINT_expr_rseq:_D\romannumeral0#2{{#1}}{{#1}}{{#2}}%
2272 \def\xINT_expr_rseq:_D #1{\ifx ^#1\xint_dothis\xINT_expr_rseq:_abort\fi
2273 \ifx ?#1\xint_dothis\xINT_expr_rseq:_break\fi
2274 \ifx !#1\xint_dothis\xINT_expr_rseq:_omit\fi
2275 \xint_orthat{\XINT_expr_rseq:_Goon {#1}}}%

```

```
2276 \def\XINT_expr_rseq:_Omit #1!#2#{\expandafter\XINT_expr_rseq:_Cb\xint_gobble_i}%
2277 \def\XINT_expr_rseq:_Goon #1!#2#{\XINT_expr_rseq:_Goon_a {#1}}%
2278 \def\XINT_expr_rseq:_Goon_a #1#2#3~#4{#1\XINT_expr_rseq:_Cb #3~{#1}}%
```

11.25.9 iterr()

ATTENTION! at 1.4 the @ and @1 are not synonymous anymore. One *must* use @1 in iterr() context.

```
2279 \def\XINT_expr_func_iterr {\XINT_allexpr_iterr \xintbareeval }%
2280 \def\XINT_flexpr_func_iterr {\XINT_allexpr_iterr \xintbarefloateval }%
2281 \def\XINT_iexpr_func_iterr {\XINT_allexpr_iterr \xintbareiieval }%
2282 \def\XINT_allexpr_iterr #1#2#3#4%
2283 {%
2284   \expandafter\XINT_expr_iterrx
2285   \expandafter #1\expanded{{\xintRevWithBraces{#4}}\expandafter}%
2286   \romannumeral`&&\XINT_expr_fetch_E_comma_V_equal_E_a {}%
2287 }%
2288 \def\XINT_expr_iterrx #1#2#3#4%
2289 {%
2290   \XINT:NEhook:iterr\XINT_expr_iterr\romannumeral0#1(#4)\relax {#2}#3#1%
2291 }%
2292 \def\XINT_expr_iterr #1#2#3#4#5%
2293 {%
2294   \expandafter\XINT_expr_put_op_first
2295   \expanded \bgroup {\iffalse}\fi
2296   \XINT_expr_iterr:_b {#5#4\relax !#3}#1^~#20?\XINT_expr_cb_and_getop
2297 }%
2298 \def\XINT_expr_iterr:_b #1#2%
2299 {%
2300   \ifx +#2\xint_dothis\XINT_expr_iterr:_Ca\fi
2301   \ifx !#2!\xint_dothis\XINT_expr_iterr:_noop\fi
2302   \ifx ^#2\xint_dothis\XINT_expr_iterr:_end\fi
2303   \xint_orthat{\XINT_expr_iterr:_c}{#2}{#1}%
2304 }%
2305 \def\XINT_expr_iterr:_noop #1{\XINT_expr_iterr:_b }%
2306 \def\XINT_expr_iterr:_end #1#2~#3#4?{{#3}\iffalse{\fi}}%
2307 \def\XINT_expr_iterr:_c #1#2{\expandafter\XINT_expr_iterr:_d\romannumeral0#2{{#1}}{#2}}%
2308 \def\XINT_expr_iterr:_d #1{\ifx ^#1\xint_dothis\XINT_expr_iterr:_abort\fi
2309   \ifx ?#1\xint_dothis\XINT_expr_iterr:_break\fi
2310   \ifx !#1\xint_dothis\XINT_expr_iterr:_omit\fi
2311   \xint_orthat{\XINT_expr_iterr:_goon {#1}}}%
2312 \def\XINT_expr_iterr:_abort #1!#2^~#3?{{\iffalse{\fi}}}%
2313 \def\XINT_expr_iterr:_break #1!#2^~#3?{#1\iffalse{\fi}}%
2314 \def\XINT_expr_iterr:_omit #1!#2#{\expandafter\XINT_expr_iterr:_b\xint_gobble_i}%
2315 \def\XINT_expr_iterr:_goon #1!#2#{\XINT_expr_iterr:_goon_a{#1}}%
2316 \def\XINT_expr_iterr:_goon_a #1#2#3~#4?%
2317 {%
2318   \expandafter\XINT_expr_iterr:_b \expanded{\unexpanded{#3~}\xintTrim{-2}{#1#4}}0?%
2319 }%
2320 \def\XINT_expr_iterr:_Ca #1#2#3{\XINT_expr_iterr:_Cc#3.{#2}}%
2321 \def\XINT_expr_iterr:_Cb #1{\expandafter\XINT_expr_iterr:_Cc\the\numexpr#1+\xint_c_i.%
2322 \def\XINT_expr_iterr:_Cc #1.#2{\expandafter\XINT_expr_iterr:_D\romannumeral0#2{{#1}}{#1}{#2}}%
2323 \def\XINT_expr_iterr:_D #1{\ifx ^#1\xint_dothis\XINT_expr_iterr:_abort\fi
```

```

2324             \ifx ?#1\xint_dothis\XINT_expr_iterr:_break\fi
2325             \ifx !#1\xint_dothis\XINT_expr_iterr:_Omit\fi
2326             \xint_orthat{\XINT_expr_iterr:_Goon {#1}}}%
2327 \def\XINT_expr_iterr:_Omit #1!#2#{\expandafter\XINT_expr_iterr:_Cb\xint_gobble_i}%
2328 \def\XINT_expr_iterr:_Goon #1!#2#{\XINT_expr_iterr:_Goon_a{#1}}%
2329 \def\XINT_expr_iterr:_Goon_a #1#2#3~#4?%
2330 {%
2331     \expandafter\XINT_expr_iterr:_Cb \expanded{\unexpanded{#3~}\xintTrim{-2}{#1#4}}0?%
2332 }%

```

11.25.10 rrseq()

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. #2 = \xint_c_i and #3 are left-over trash.

```

2333 \def\XINT_expr_func_rrseq {\XINT_allexpr_rrseq \xintbareeval }%
2334 \def\XINT_flexpr_func_rrseq {\XINT_allexpr_rrseq \xintbarefloateval }%
2335 \def\XINT_iexpr_func_rrseq {\XINT_allexpr_rrseq \xintbareiieval }%
2336 \def\XINT_allexpr_rrseq #1#2#3#4%
2337 {%
2338     \expandafter\XINT_expr_rrseqx\expandafter#1\expanded
2339     {\unexpanded{#{#4}}{\xintRevWithBraces{#4}}\expandafter}%
2340     \romannumeral`&&\XINT_expr_fetch_E_comma_V_equal_E_a {}%
2341 }%
2342 \def\XINT_expr_rrseqx #1#2#3#4#5%
2343 {%
2344     \XINT:NEhook:rrseq\XINT_expr_rrseqy\romannumeral0#1(#5)\relax {#2}{#3}#4#1%
2345 }%
2346 \def\XINT_expr_rrseqy #1#2#3#4#5#6%
2347 {%
2348     \expandafter\XINT_expr_put_op_first
2349     \expanded \bgroup {\iffalse}\fi
2350     #2\XINT_expr_rrseq:_b {#6#5}\relax !#4}#1^~#30?\XINT_expr_cb_and_getop
2351 }%
2352 \def\XINT_expr_rrseq:_b #1#2%
2353 {%
2354     \ifx +#2\xint_dothis\XINT_expr_rrseq:_Ca\fi
2355     \ifx !#2!\xint_dothis\XINT_expr_rrseq:_noop\fi
2356     \ifx ^#2\xint_dothis\XINT_expr_rrseq:_end\fi
2357     \xint_orthat{\XINT_expr_rrseq:_c}{#2}{#1}%
2358 }%
2359 \def\XINT_expr_rrseq:_noop #1{\XINT_expr_rrseq:_b }%
2360 \def\XINT_expr_rrseq:_end #1#2~#3?{\iffalse{\fi}}%
2361 \def\XINT_expr_rrseq:_c #1#2{\expandafter\XINT_expr_rrseq:_d\romannumeral0#2{#{#1}}{#2}}%
2362 \def\XINT_expr_rrseq:_d #1{\ifx ^#1\xint_dothis\XINT_expr_rrseq:_abort\fi
2363     \ifx ?#1\xint_dothis\XINT_expr_rrseq:_break\fi
2364     \ifx !#1\xint_dothis\XINT_expr_rrseq:_omit\fi
2365     \xint_orthat{\XINT_expr_rrseq:_goon {#1}}}%
2366 \def\XINT_expr_rrseq:_abort #1!#2^~#3?{\iffalse{\fi}}%
2367 \def\XINT_expr_rrseq:_break #1!#2^~#3?{#1\iffalse{\fi}}%
2368 \def\XINT_expr_rrseq:_omit #1!#2#{\expandafter\XINT_expr_rrseq:_b\xint_gobble_i}%
2369 \def\XINT_expr_rrseq:_goon #1!#2#{\XINT_expr_rrseq:_goon_a {#1}}%

```

```

2370 \def\XINT_expr_rrseq:_goon_a #1#2#3~#4?%
2371 {%
2372     #1\expandafter\XINT_expr_rrseq:_b\expanded{\unexpanded{#3~}\xintTrim{-2}{#1#4}}0?%
2373 }%
2374 \def\XINT_expr_rrseq:_Ca #1#2#3{\XINT_expr_rrseq:_Cc#3.{#2}}%
2375 \def\XINT_expr_rrseq:_Cb #1{\expandafter\XINT_expr_rrseq:_Cc\the\numexpr#1+\xint_c_i.}%
2376 \def\XINT_expr_rrseq:_Cc #1.#2{\expandafter\XINT_expr_rrseq:_D\romannumeral0#2{{#1}}{{#1}}{{#2}}}%
2377 \def\XINT_expr_rrseq:_D #1{\ifx ^#1\xint_dothis\XINT_expr_rrseq:_abort\fi
2378     \ifx ?#1\xint_dothis\XINT_expr_rrseq:_break\fi
2379     \ifx !#1\xint_dothis\XINT_expr_rrseq:_Omit\fi
2380     \xint_orthat{\XINT_expr_rrseq:_Goon {#1}}}%
2381 \def\XINT_expr_rrseq:_Omit #1!#2#{\expandafter\XINT_expr_rrseq:_Cb\xint_gobble_i}%
2382 \def\XINT_expr_rrseq:_Goon #1!#2#{\XINT_expr_rrseq:_Goon_a {#1}}%
2383 \def\XINT_expr_rrseq:_Goon_a #1#2#3~#4?%
2384 {%
2385     #1\expandafter\XINT_expr_rrseq:_Cb\expanded{\unexpanded{#3~}\xintTrim{-2}{#1#4}}0?%
2386 }%
2387 \catcode`? 11

```

11.26 Pseudo-functions related to N-dimensional hypercubic lists

11.26.1 ndseq()

New with 1.4. 2020/01/23. It is derived from subsm() but instead of evaluating one expression according to one value per variable, it constructs a nested bracketed seq... this means the expression is parsed each time ! Anyway, proof of concept. Nota Bene : omit, abort, break() work !

```

2388 \def\XINT_expr_onliteral_ndseq
2389 {%
2390     \expandafter\XINT_allexpr_ndseq_f
2391     \romannumeral`&&\XINT_expr_fetch_E_comma_V_equal_E_a {}%
2392 }%
2393 \def\XINT_allexpr_ndseq_f #1#2{\xint_c_ii^v `{ndseqx}#2)\relax #1}%
2394 \def\XINT_expr_func_ndseqx
2395 {%
2396     \expandafter\XINT_allexpr_ndseqx\expandafter\xintbareeval
2397     \expandafter{\romannumeral0\expandafter\xint_gobble_i\string}%
2398     \expandafter\xintrevwithbraces
2399     \expanded\bgroup{\iffalse}\fi\XINT_allexpr_ndseq_A\XINT_expr_oparen
2400 }%
2401 \def\XINT_flexpr_func_ndseqx
2402 {%
2403     \expandafter\XINT_allexpr_ndseqx\expandafter\xintbarefloateval
2404     \expandafter{\romannumeral0\expandafter\xint_gobble_i\string}%
2405     \expandafter\xintrevwithbraces
2406     \expanded\bgroup{\iffalse}\fi\XINT_allexpr_ndseq_A\XINT_flexpr_oparen
2407 }%
2408 \def\XINT_iiexpr_func_ndseqx
2409 {%
2410     \expandafter\XINT_allexpr_ndseqx\expandafter\xintbareiieval
2411     \expandafter{\romannumeral0\expandafter\xint_gobble_i\string}%
2412     \expandafter\xintrevwithbraces

```

```

2413 \expanded\bgroup{\iffalse}\fi\XINT_allexpr_ndseq_A\XINT_iiexpr_oparen
2414 }%
2415 \def\XINT_allexpr_ndseq_A #1#2#3%
2416 {%
2417 \ifx#2\xint_c_
2418 \expandafter\XINT_allexpr_ndseq_C
2419 \else
2420 \expandafter\XINT_allexpr_ndseq_B
2421 \fi #1%
2422 }%
2423 \def\XINT_allexpr_ndseq_B #1#2#3#4=%
2424 {%
2425 {#2}{\xint_zapspace#3#4 \xint_gobble_i}%
2426 \expandafter\XINT_allexpr_ndseq_A\expandafter#1\romannumeral`&&@#1%
2427 }%

#1 = \xintbareeval, or \xintbarefloateval or \xintbareiieval #2 = values for last coordinate

2428 \def\XINT_allexpr_ndseq_C #1#2{{#2}\iffalse{{{ \fi}}}}%

#1 = \xintbareeval or \xintbarefloateval or \xintbareiieval #2 = {valuesN}...{values2}{var2}{values1}
#3 = {var1} #4 = the expression to evaluate

2429 \def\XINT_allexpr_ndseqx #1#2#3#4%
2430 {%
2431 \expandafter\XINT_expr_put_op_first
2432 \expanded
2433 \bgroup
2434 \romannumeral0#1\empty
2435 \expanded{\xintReplicate{\xintLength{#{3}#2}/2}{[seq(}%
2436 \unexpanded{#4}%
2437 \XINT_allexpr_ndseqx_a #2{#3}^^%
2438 }%
2439 \relax
2440 \iffalse{\fi\expandafter}\romannumeral`&&\XINT_expr_getop
2441 }%
2442 \def\XINT_allexpr_ndseqx_a #1#2%
2443 {%
2444 \xint_gob_til_^ #1\XINT_allexpr_ndseqx_e ^%
2445 \unexpanded{,#2=\XINTfstop.{#1})}\XINT_allexpr_ndseqx_a
2446 }%
2447 \def\XINT_allexpr_ndseqx_e ^#1\XINT_allexpr_ndseqx_a{%

```

11.26.2 ndmap()

New with 1.4. 2020/01/24.

```

2448 \def\XINT_expr_onliteral_ndmap #1,{\xint_c_ii^v `{ndmapx}\XINTfstop.{#1};}%
2449 \def\XINT_expr_func_ndmapx #1#2#3%
2450 {%
2451 \expandafter\XINT_allexpr_ndmapx
2452 \curname XINT_expr_func_\xint_zapspace #3 \xint_gobble_i\endcname
2453 \XINT_expr_oparen
2454 }%

```



```

2455 \def\XINT_flexpr_func_ndmapx #1#2#3%
2456 {%
2457     \expandafter\XINT_allexpr_ndmapx
2458     \csname XINT_flexpr_func_\xint_zapspaces #3 \xint_gobble_i\endcsname
2459     \XINT_flexpr_oparen
2460 }%
2461 \def\XINT_iiexpr_func_ndmapx #1#2#3%
2462 {%
2463     \expandafter\XINT_allexpr_ndmapx
2464     \csname XINT_iiexpr_func_\xint_zapspaces #3 \xint_gobble_i\endcsname
2465     \XINT_iiexpr_oparen
2466 }%
2467 \def\XINT_allexpr_ndmapx #1#2%
2468 {%
2469     \expandafter\XINT_expr_put_op_first
2470     \expanded\bgroup{\iffalse}\fi
2471     \expanded
2472     {\noexpand\XINT:NEhook:x:ndmapx
2473     \noexpand\XINT_allexpr_ndmapx_a
2474     \noexpand#1{}\expandafter}%
2475     \expanded\bgroup\expandafter\XINT_allexpr_ndmap_A
2476     \expandafter#2\romannumeral`&&@#2%
2477 }%
2478 \def\XINT_allexpr_ndmap_A #1#2#3%
2479 {%
2480     \ifx#3;%
2481         \expandafter\XINT_allexpr_ndmap_B
2482     \else
2483         \xint_afterfi{\XINT_allexpr_ndmap_C#2#3}%
2484     \fi #1%
2485 }%
2486 \def\XINT_allexpr_ndmap_B #1#2%
2487 {%
2488     {#2}\expandafter\XINT_allexpr_ndmap_A\expandafter#1\romannumeral`&&@#1%
2489 }%
2490 \def\XINT_allexpr_ndmap_C #1#2#3#4%
2491 {%
2492     {#4}^\relax\iffalse{{\fi}}{#1#2%
2493 }%
2494 \def\XINT_allexpr_ndmapx_a #1#2#3%
2495 {%
2496     \xint_gob_til_ ^ #3\XINT_allexpr_ndmapx_l ^%
2497     \XINT_allexpr_ndmapx_b #1{#2}{#3}%
2498 }%
2499 \def\XINT_allexpr_ndmapx_l ^#1\XINT_allexpr_ndmapx_b #2#3#4\relax
2500 {%
2501     #2\empty\xint_firstofone{#3}%
2502 }%
2503 \def\XINT_allexpr_ndmapx_b #1#2#3#4\relax
2504 {%
2505     {\iffalse}\fi\XINT_allexpr_ndmapx_c {#4\relax}{#1}{#2}{#3}%
2506 }%

```



```

2507 \def\XINT_allexpr_ndmapx_c #1#2#3#4%
2508 {%
2509     \xint_gob_til_ ^ #4\XINT_allexpr_ndmapx_e ^%
2510     \XINT_allexpr_ndmapx_a #2#{#3#{#4}}#1%
2511     \XINT_allexpr_ndmapx_c  {#1}#2#{#3}%
2512 }%
2513 \def\XINT_allexpr_ndmapx_e ^#1\XINT_allexpr_ndmapx_c
2514     {\iffalse{\fi}\xint_gobble_iii}%

```

11.26.3 ndfillraw()

New with 1.4. 2020/01/24. J'hésite à autoriser un #1 quelconque, ou plutôt à le wrapper dans un \xintbareval. Mais il faut alors distinguer les trois. De toute façon les variables ne marcheraient pas donc j'hésite à mettre un wrapper automatique. Mais ce n'est pas bien d'autoriser l'injection de choses quelconques.

Pour des choses comme ndfillraw(\xintRandomBit,[10,10]).

Je n'aime pas le nom !. Le changer. ndconst? Surtout je n'aime pas que dans le premier argument il faut rajouter explicitement si nécessaire \xintiexpr wrap.

```

2515 \def\XINT_expr_onliteral_ndfillraw #1,{\xint_c_ii^v `{ndfillrawx}\XINTfstop.{#{#1}},}%
2516 \def\XINT_expr_func_ndfillrawx #1#2#3%
2517 {%
2518     \expandafter#1\expandafter#2\expanded{{{XINT_allexpr_ndfillrawx_a #3}}}%
2519 }%
2520 \let\XINT_iiexpr_func_ndfillrawx\XINT_expr_func_ndfillrawx
2521 \let\XINT_flexpr_func_ndfillrawx\XINT_expr_func_ndfillrawx
2522 \def\XINT_allexpr_ndfillrawx_a #1#2%
2523 {%
2524     \expandafter\XINT_allexpr_ndfillrawx_b
2525     \romannumeral0\xintApply{\xintNum}{#2}^{\relax {#1}}%
2526 }%
2527 \def\XINT_allexpr_ndfillrawx_b #1#2\relax#3%
2528 {%
2529     \xint_gob_til_ ^ #1\XINT_allexpr_ndfillrawx_c ^%
2530     \xintReplicate{#1}{{XINT_allexpr_ndfillrawx_b #2\relax {#3}}}%
2531 }%
2532 \def\XINT_allexpr_ndfillrawx_c ^\xintReplicate #1#2%
2533 {%
2534     \expandafter\XINT_allexpr_ndfillrawx_d\xint_firstofone #2%
2535 }%
2536 \def\XINT_allexpr_ndfillrawx_d\XINT_allexpr_ndfillrawx_b \relax #1{#1}%

```

11.27 Other pseudo-functions: bool(), togl(), protect(), qraw(), qint(), qfrac(), qfloat(), qrand(), random(), rbit()

bool, togl and protect use delimited macros. They are not true functions, they turn off the parser to gather their "variable".

1.2. adds qint(), qfrac(), qfloat().

1.3c. adds qraw(). Useful to limit impact on T_EX memory from abuse of \csname's storage when generating many comma separated values from a loop.

1.3e. qfloat() keeps a short mantissa if possible.

They allow the user to hand over quickly a big number to the parser, spaces not immediately removed but should be harmless in general. The qraw() does no post-processing at all apart complete expansion, useful for comma-separated values, but must be obedient to (non really documented) expected format. Each uses a delimited macro, the closing parenthesis can not emerge from expansion.

1.3b. random(), grand() Function-like syntax but with no argument currently, so let's use fast parsing which requires though the closing parenthesis to be explicit.

Attention that qraw() which pre-supposes knowledge of internal storage model is fragile and may break at any release.

1.4 adds rbit(). Short for random bit.

```

2537 \def\XINT_expr_onliteral_bool #1)%
2538     {\expandafter\XINT_expr_put_op_first\expanded{{{xintBool{#1}}}\expandafter
2539     }\romannumeral`&&\XINT_expr_getop}%
2540 \def\XINT_expr_onliteral_togl #1)%
2541     {\expandafter\XINT_expr_put_op_first\expanded{{{xintToggle{#1}}}\expandafter
2542     }\romannumeral`&&\XINT_expr_getop}%
2543 \def\XINT_expr_onliteral_protect #1)%
2544     {\expandafter\XINT_expr_put_op_first\expanded{{{detokenize{#1}}}\expandafter
2545     }\romannumeral`&&\XINT_expr_getop}%
2546 \def\XINT_expr_onliteral_qint #1)%
2547     {\expandafter\XINT_expr_put_op_first\expanded{{{xintiNum{#1}}}\expandafter
2548     }\romannumeral`&&\XINT_expr_getop}%
2549 \def\XINT_expr_onliteral_qfrac #1)%
2550     {\expandafter\XINT_expr_put_op_first\expanded{{{xintRaw{#1}}}\expandafter
2551     }\romannumeral`&&\XINT_expr_getop}%
2552 \def\XINT_expr_onliteral_qfloat #1)%
2553     {\expandafter\XINT_expr_put_op_first\expanded{{{XINTinFloatSdigits{#1}}}\expandafter
2554     }\romannumeral`&&\XINT_expr_getop}%
2555 \def\XINT_expr_onliteral_qraw #1)%
2556     {\expandafter\XINT_expr_put_op_first\expanded{#{#1}\expandafter
2557     }\romannumeral`&&\XINT_expr_getop}%
2558 \def\XINT_expr_onliteral_random #1)%
2559     {\expandafter\XINT_expr_put_op_first\expanded{{{XINTinRandomFloatSdigits}}\expandafter
2560     }\romannumeral`&&\XINT_expr_getop}%
2561 \def\XINT_expr_onliteral_grand #1)%
2562     {\expandafter\XINT_expr_put_op_first\expanded{{{XINTinRandomFloatSixteen}}\expandafter
2563     }\romannumeral`&&\XINT_expr_getop}%
2564 \def\XINT_expr_onliteral_rbit #1)%
2565     {\expandafter\XINT_expr_put_op_first\expanded{{{xintRandBit}}\expandafter
2566     }\romannumeral`&&\XINT_expr_getop}%

```

11.28 Regular built-in functions: num(), reduce(), preduce(), abs(), sgn(), frac(), floor(), ceil(), sqr(), ?(), !(), not(), odd(), even(), isint(), isone(), factorial(), sqrt(), sqrtr(), inv(), round(), trunc(), float(), sfloat(), ilog10(), divmod(), mod(), binomial(), pfactorial(), randrange(), quo(), rem(), gcd(), lcm(), max(), min(), `+`(), `*`(), all(), any(), xor(), len(), first(), last(), reversed(), if(), ifint(), ifone(), ifsgn(), nuple() and unpack()

```

2567 \def\XINT:expr:f:one:and:opt #1#2#3!#4#5%
2568 {%

```

```

2569 \if\relax#3\relax\expandafter\xint_firstoftwo\else
2570 \expandafter\xint_secondoftwo\fi
2571 {#4}{#5[\xintNum{#2}]}{#1}%
2572 }%
2573 \def\xint:expr:f:tacitzeroifone #1#2#3!#4#5%
2574 {%
2575 \if\relax#3\relax\expandafter\xint_firstoftwo\else
2576 \expandafter\xint_secondoftwo\fi
2577 {#4{0}}{#5{\xintNum{#2}}}{#1}%
2578 }%
2579 \def\xint:expr:f:iitacitzeroifone #1#2#3!#4%
2580 {%
2581 \if\relax#3\relax\expandafter\xint_firstoftwo\else
2582 \expandafter\xint_secondoftwo\fi
2583 {#4{0}}{#4{#2}}{#1}%
2584 }%
2585 \def\xint:expr_func_num #1#2#3%
2586 {%
2587 \expandafter #1\expandafter #2\expandafter{%
2588 \romannumeral`&&\xint:NEhook:f:one:from:one
2589 {\romannumeral`&&\xintNum#3}}%
2590 }%
2591 \let\xint:flexpr_func_num\xint:expr_func_num
2592 \let\xint:iiexpr_func_num\xint:expr_func_num
2593 \def\xint:expr_func_reduce #1#2#3%
2594 {%
2595 \expandafter #1\expandafter #2\expandafter{%
2596 \romannumeral`&&\xint:NEhook:f:one:from:one
2597 {\romannumeral`&&\xintIrr#3}}%
2598 }%
2599 \let\xint:flexpr_func_reduce\xint:expr_func_reduce
2600 \def\xint:expr_func_preduce #1#2#3%
2601 {%
2602 \expandafter #1\expandafter #2\expandafter{%
2603 \romannumeral`&&\xint:NEhook:f:one:from:one
2604 {\romannumeral`&&\xintPIrr#3}}%
2605 }%
2606 \let\xint:flexpr_func_preduce\xint:expr_func_preduce
2607 \def\xint:expr_func_abs #1#2#3%
2608 {%
2609 \expandafter #1\expandafter #2\expandafter{%
2610 \romannumeral`&&\xint:NEhook:f:one:from:one
2611 {\romannumeral`&&\xintAbs#3}}%
2612 }%
2613 \let\xint:flexpr_func_abs\xint:expr_func_abs
2614 \def\xint:iiexpr_func_abs #1#2#3%
2615 {%
2616 \expandafter #1\expandafter #2\expandafter{%
2617 \romannumeral`&&\xint:NEhook:f:one:from:one
2618 {\romannumeral`&&\xintiAbs#3}}%
2619 }%
2620 \def\xint:expr_func_sgn #1#2#3%

```

```

2621 {%
2622   \expandafter #1\expandafter #2\expandafter{%
2623   \romannumeral`&&\XINT:NEhook:f:one:from:one
2624   {\romannumeral`&&\xintSgn#3}}%
2625 }%
2626 \let\XINT_flexpr_func_sgn\XINT_expr_func_sgn
2627 \def\XINT_iiexpr_func_sgn #1#2#3%
2628 {%
2629   \expandafter #1\expandafter #2\expandafter{%
2630   \romannumeral`&&\XINT:NEhook:f:one:from:one
2631   {\romannumeral`&&\xintiSgn#3}}%
2632 }%
2633 \def\XINT_expr_func_frac #1#2#3%
2634 {%
2635   \expandafter #1\expandafter #2\expandafter{%
2636   \romannumeral`&&\XINT:NEhook:f:one:from:one
2637   {\romannumeral`&&\xintTFrac#3}}%
2638 }%
2639 \def\XINT_flexpr_func_frac #1#2#3%
2640 {%
2641   \expandafter #1\expandafter #2\expandafter{%
2642   \romannumeral`&&\XINT:NEhook:f:one:from:one
2643   {\romannumeral`&&\XINTinFloatFracdigits#3}}%
2644 }%

```

no \XINT_iiexpr_func_frac

```

2645 \def\XINT_expr_func_floor #1#2#3%
2646 {%
2647   \expandafter #1\expandafter #2\expandafter{%
2648   \romannumeral`&&\XINT:NEhook:f:one:from:one
2649   {\romannumeral`&&\xintFloor#3}}%
2650 }%
2651 \let\XINT_flexpr_func_floor\XINT_expr_func_floor

```

The floor and ceil functions in \xintiexpr require protect(a/b) or, better, \qfrac(a/b); else the / will be executed first and do an integer rounded division.

```

2652 \def\XINT_iiexpr_func_floor #1#2#3%
2653 {%
2654   \expandafter #1\expandafter #2\expandafter{%
2655   \romannumeral`&&\XINT:NEhook:f:one:from:one
2656   {\romannumeral`&&\xintiFloor#3}}%
2657 }%
2658 \def\XINT_expr_func_ceil #1#2#3%
2659 {%
2660   \expandafter #1\expandafter #2\expandafter{%
2661   \romannumeral`&&\XINT:NEhook:f:one:from:one
2662   {\romannumeral`&&\xintCeil#3}}%
2663 }%
2664 \let\XINT_flexpr_func_ceil\XINT_expr_func_ceil
2665 \def\XINT_iiexpr_func_ceil #1#2#3%
2666 {%
2667   \expandafter #1\expandafter #2\expandafter{%
2668   \romannumeral`&&\XINT:NEhook:f:one:from:one

```

```

2669     {\romannumeral`&&@\xintiCeil#3}}%
2670 }%
2671 \def\XINT_expr_func_sqr #1#2#3%
2672 {%
2673     \expandafter #1\expandafter #2\expandafter{%
2674     \romannumeral`&&@\XINT:NEhook:f:one:from:one
2675     {\romannumeral`&&@\xintSqr#3}}%
2676 }%
2677 \def\XINTinFloatSqr#1{\XINTinFloatMul{#1}{#1}}%
2678 \def\XINT_flexpr_func_sqr #1#2#3%
2679 {%
2680     \expandafter #1\expandafter #2\expandafter{%
2681     \romannumeral`&&@\XINT:NEhook:f:one:from:one
2682     {\romannumeral`&&@\XINTinFloatSqr#3}}%
2683 }%
2684 \def\XINT_iiexpr_func_sqr #1#2#3%
2685 {%
2686     \expandafter #1\expandafter #2\expandafter{%
2687     \romannumeral`&&@\XINT:NEhook:f:one:from:one
2688     {\romannumeral`&&@\xintiiSqr#3}}%
2689 }%
2690 \def\XINT_expr_func_? #1#2#3%
2691 {%
2692     \expandafter #1\expandafter #2\expandafter{%
2693     \romannumeral`&&@\XINT:NEhook:f:one:from:one
2694     {\romannumeral`&&@\xintiiIsNotZero#3}}%
2695 }%
2696 \let\XINT_flexpr_func_? \XINT_expr_func_?
2697 \let\XINT_iiexpr_func_? \XINT_expr_func_?
2698 \def\XINT_expr_func_! #1#2#3%
2699 {%
2700     \expandafter #1\expandafter #2\expandafter{%
2701     \romannumeral`&&@\XINT:NEhook:f:one:from:one
2702     {\romannumeral`&&@\xintiiIsZero#3}}%
2703 }%
2704 \let\XINT_flexpr_func_! \XINT_expr_func_!
2705 \let\XINT_iiexpr_func_! \XINT_expr_func_!
2706 \def\XINT_expr_func_not #1#2#3%
2707 {%
2708     \expandafter #1\expandafter #2\expandafter{%
2709     \romannumeral`&&@\XINT:NEhook:f:one:from:one
2710     {\romannumeral`&&@\xintiiIsZero#3}}%
2711 }%
2712 \let\XINT_flexpr_func_not \XINT_expr_func_not
2713 \let\XINT_iiexpr_func_not \XINT_expr_func_not
2714 \def\XINT_expr_func_odd #1#2#3%
2715 {%
2716     \expandafter #1\expandafter #2\expandafter{%
2717     \romannumeral`&&@\XINT:NEhook:f:one:from:one
2718     {\romannumeral`&&@\xintOdd#3}}%
2719 }%
2720 \let\XINT_flexpr_func_odd\XINT_expr_func_odd

```

```

2721 \def\XINT_iiexpr_func_odd #1#2#3%
2722 {%
2723   \expandafter #1\expandafter #2\expandafter{%
2724     \romannumeral`&&\XINT:NEhook:f:one:from:one
2725     {\romannumeral`&&\xintiiOdd#3}}%
2726 }%
2727 \def\XINT_expr_func_even #1#2#3%
2728 {%
2729   \expandafter #1\expandafter #2\expandafter{%
2730     \romannumeral`&&\XINT:NEhook:f:one:from:one
2731     {\romannumeral`&&\xintEven#3}}%
2732 }%
2733 \let\XINT_flexpr_func_even\XINT_expr_func_even
2734 \def\XINT_iiexpr_func_even #1#2#3%
2735 {%
2736   \expandafter #1\expandafter #2\expandafter{%
2737     \romannumeral`&&\XINT:NEhook:f:one:from:one
2738     {\romannumeral`&&\xintiiEven#3}}%
2739 }%
2740 \def\XINT_expr_func_isint #1#2#3%
2741 {%
2742   \expandafter #1\expandafter #2\expandafter{%
2743     \romannumeral`&&\XINT:NEhook:f:one:from:one
2744     {\romannumeral`&&\xintIsInt#3}}%
2745 }%
2746 \def\XINT_flexpr_func_isint #1#2#3%
2747 {%
2748   \expandafter #1\expandafter #2\expandafter{%
2749     \romannumeral`&&\XINT:NEhook:f:one:from:one
2750     {\romannumeral`&&\xintFloatIsInt#3}}%
2751 }%
2752 \let\XINT_iiexpr_func_isint\XINT_expr_func_isint % ? perhaps rather always 1
2753 \def\XINT_expr_func_isone #1#2#3%
2754 {%
2755   \expandafter #1\expandafter #2\expandafter{%
2756     \romannumeral`&&\XINT:NEhook:f:one:from:one
2757     {\romannumeral`&&\xintIsOne#3}}%
2758 }%
2759 \let\XINT_flexpr_func_isone\XINT_expr_func_isone
2760 \def\XINT_iiexpr_func_isone #1#2#3%
2761 {%
2762   \expandafter #1\expandafter #2\expandafter{%
2763     \romannumeral`&&\XINT:NEhook:f:one:from:one
2764     {\romannumeral`&&\xintiiIsOne#3}}%
2765 }%
2766 \def\XINT_expr_func_factorial #1#2#3%
2767 {%
2768   \expandafter #1\expandafter #2\expandafter{\expandafter{%
2769     \romannumeral`&&\XINT:NEhook:f:one:and:opt:direct
2770     \XINT:expr:f:one:and:opt #3,! \xintFac\XINTinFloatFac
2771     }}%
2772 }%

```

```

2773 \def\XINT_flexpr_func_factorial #1#2#3%
2774 {%
2775     \expandafter #1\expandafter #2\expandafter{\expandafter{%
2776     \romannumeral`&&\XINT:NEhook:f:one:and:opt:direct
2777     \XINT:expr:f:one:and:opt#3,! \XINTinFloatFacdigits\XINTinFloatFac
2778     }}%
2779 }%
2780 \def\XINT_iiexpr_func_factorial #1#2#3%
2781 {%
2782     \expandafter #1\expandafter #2\expandafter{%
2783     \romannumeral`&&\XINT:NEhook:f:one:from:one
2784     {\romannumeral`&&\xintiiFac#3}}%
2785 }%
2786 \def\XINT_expr_func_sqrt #1#2#3%
2787 {%
2788     \expandafter #1\expandafter #2\expandafter{\expandafter{%
2789     \romannumeral`&&\XINT:NEhook:f:one:and:opt:direct
2790     \XINT:expr:f:one:and:opt #3,! \XINTinFloatSqrtdigits\XINTinFloatSqrt
2791     }}%
2792 }%
2793 \let\XINT_flexpr_func_sqrt\XINT_expr_func_sqrt
2794 \def\XINT_expr_func_sqrt_ #1#2#3%
2795 {%
2796     \expandafter #1\expandafter #2\expandafter{%
2797     \romannumeral`&&\XINT:NEhook:f:one:from:one
2798     {\romannumeral`&&\XINTinFloatSqrtdigits#3}}%
2799 }%
2800 \let\XINT_flexpr_func_sqrt_\XINT_expr_func_sqrt_
2801 \def\XINT_iiexpr_func_sqrt #1#2#3%
2802 {%
2803     \expandafter #1\expandafter #2\expandafter{%
2804     \romannumeral`&&\XINT:NEhook:f:one:from:one
2805     {\romannumeral`&&\xintiiSqrt#3}}%
2806 }%
2807 \def\XINT_iiexpr_func_sqrtr #1#2#3%
2808 {%
2809     \expandafter #1\expandafter #2\expandafter{%
2810     \romannumeral`&&\XINT:NEhook:f:one:from:one
2811     {\romannumeral`&&\xintiiSqrtR#3}}%
2812 }%
2813 \def\XINT_expr_func_inv #1#2#3%
2814 {%
2815     \expandafter #1\expandafter #2\expandafter{%
2816     \romannumeral`&&\XINT:NEhook:f:one:from:one
2817     {\romannumeral`&&\xintInv#3}}%
2818 }%
2819 \def\XINT_flexpr_func_inv #1#2#3%
2820 {%
2821     \expandafter #1\expandafter #2\expandafter{%
2822     \romannumeral`&&\XINT:NEhook:f:one:from:one
2823     {\romannumeral`&&\XINTinFloatInv#3}}%
2824 }%

```

```

2825 \def\XINT_expr_func_round #1#2#3%
2826 {%
2827     \expandafter #1\expandafter #2\expandafter{\expandafter{%
2828         \romannumeral`&&\XINT:NEhook:f:tacitzeroifone:direct
2829         \XINT:expr:f:tacitzeroifone #3,! \xintiRound\xintRound
2830     }}%
2831 }%
2832 \let\XINT_flexpr_func_round\XINT_expr_func_round
2833 \def\XINT_iiexpr_func_round #1#2#3%
2834 {%
2835     \expandafter #1\expandafter #2\expandafter{\expandafter{%
2836         \romannumeral`&&\XINT:NEhook:f:iitacitzeroifone:direct
2837         \XINT:expr:f:iitacitzeroifone #3,! \xintiRound
2838     }}%
2839 }%
2840 \def\XINT_expr_func_trunc #1#2#3%
2841 {%
2842     \expandafter #1\expandafter #2\expandafter{\expandafter{%
2843         \romannumeral`&&\XINT:NEhook:f:tacitzeroifone:direct
2844         \XINT:expr:f:tacitzeroifone #3,! \xintiTrunc\xintTrunc
2845     }}%
2846 }%
2847 \let\XINT_flexpr_func_trunc\XINT_expr_func_trunc
2848 \def\XINT_iiexpr_func_trunc #1#2#3%
2849 {%
2850     \expandafter #1\expandafter #2\expandafter{\expandafter{%
2851         \romannumeral`&&\XINT:NEhook:f:iitacitzeroifone:direct
2852         \XINT:expr:f:iitacitzeroifone #3,! \xintiTrunc
2853     }}%
2854 }%

```

Hesitation at 1.3e about using `\XINTinFloatSdigits` and `\XINTinFloatS`. Finally I add a `sfloat()` function. It helps for `xinttrig.sty`.

```

2855 \def\XINT_expr_func_float #1#2#3%
2856 {%
2857     \expandafter #1\expandafter #2\expandafter{\expandafter{%
2858         \romannumeral`&&\XINT:NEhook:f:one:and:opt:direct
2859         \XINT:expr:f:one:and:opt #3,! \XINTinFloatdigits\XINTinFloat
2860     }}%
2861 }%
2862 \let\XINT_flexpr_func_float\XINT_expr_func_float

```

`float_()` added at 1.4. Does not check for optional argument. Useful to transfer functions defined with `\xintdefunc` to functions usable in `\xintfloateval`. I hesitated briefly about notation but here we go. Unfortunately I will have to document it (contrarily to `sqrt_`).

No need to do same for `sfloat()` currently used in `xinttrig.sty` to go from float to expr, because `sfloat(x)` sees there is no optional argument.

Still I wonder if better would not be to have some function «`single()`» which signals to outer one it is a single argument? Must think about this. Too late now for 1.4.

```

2863 \def\XINT_expr_func_float_ #1#2#3%
2864 {%
2865     \expandafter #1\expandafter #2\expandafter{%
2866         \romannumeral`&&\XINT:NEhook:f:one:from:one

```



```

2867     {\romannumeral`&&\XINTinFloatdigits#3}}%
2868 }%
2869 \let\XINT_flexpr_func_float_\XINT_expr_func_float_
2870 \def\XINT_expr_func_sfloat #1#2#3%
2871 {%
2872     \expandafter #1\expandafter #2\expandafter{\expandafter{%
2873         \romannumeral`&&\XINT:NEhook:f:one:and:opt:direct
2874         \XINT:expr:f:one:and:opt #3,! \XINTinFloatSdigits\XINTinFloatS
2875     }}%
2876 }%
2877 \let\XINT_flexpr_func_sfloat\XINT_expr_func_sfloat
2878 % \XINT_iiexpr_func_sfloat not defined
2879 \expandafter\def\csname XINT_expr_func_ilog10\endcsname #1#2#3%
2880 {%
2881     \expandafter #1\expandafter #2\expandafter{\expandafter{%
2882         \romannumeral`&&\XINT:NEhook:f:one:and:opt:direct
2883         \XINT:expr:f:one:and:opt #3,! \xintiLogTen\XINTFloatiLogTen
2884     }}%
2885 }%
2886 \expandafter\def\csname XINT_flexpr_func_ilog10\endcsname #1#2#3%
2887 {%
2888     \expandafter #1\expandafter #2\expandafter{\expandafter{%
2889         \romannumeral`&&\XINT:NEhook:f:one:and:opt:direct
2890         \XINT:expr:f:one:and:opt #3,! \XINTFloatiLogTendigits\XINTFloatiLogTen
2891     }}%
2892 }%
2893 \expandafter\def\csname XINT_iiexpr_func_ilog10\endcsname #1#2#3%
2894 {%
2895     \expandafter #1\expandafter #2\expandafter{%
2896         \romannumeral`&&\XINT:NEhook:f:one:from:one
2897         {\romannumeral`&&\xintiiLogTen#3}}%
2898 }%
2899 \def\XINT_expr_func_divmod #1#2#3%
2900 {%
2901     \expandafter #1\expandafter #2\expandafter{\romannumeral`&&%
2902     \XINT:NEhook:f:one:from:two
2903     {\romannumeral`&&\xintDivMod #3}}%
2904 }%
2905 \def\XINT_flexpr_func_divmod #1#2#3%
2906 {%
2907     \expandafter #1\expandafter #2\expandafter{\romannumeral`&&%
2908     \XINT:NEhook:f:one:from:two
2909     {\romannumeral`&&\XINTinFloatDivMod #3}}%
2910 }%
2911 \def\XINT_iiexpr_func_divmod #1#2#3%
2912 {%
2913     \expandafter #1\expandafter #2\expandafter{\romannumeral`&&%
2914     \XINT:NEhook:f:one:from:two
2915     {\romannumeral`&&\xintiiDivMod #3}}%
2916 }%
2917 \def\XINT_expr_func_mod #1#2#3%
2918 {%

```

```

2919 \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
2920 \XINT:NEhook:f:one:from:two
2921 {\romannumeral`&&\xintMod#3}}%
2922 }%
2923 \def\XINT_flexpr_func_mod #1#2#3%
2924 {%
2925 \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
2926 \XINT:NEhook:f:one:from:two
2927 {\romannumeral`&&\XINTinFloatMod#3}}%
2928 }%
2929 \def\XINT_iiexpr_func_mod #1#2#3%
2930 {%
2931 \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
2932 \XINT:NEhook:f:one:from:two
2933 {\romannumeral`&&\xintiMod#3}}%
2934 }%
2935 \def\XINT_expr_func_binomial #1#2#3%
2936 {%
2937 \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
2938 \XINT:NEhook:f:one:from:two
2939 {\romannumeral`&&\xintBinomial #3}}%
2940 }%
2941 \def\XINT_flexpr_func_binomial #1#2#3%
2942 {%
2943 \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
2944 \XINT:NEhook:f:one:from:two
2945 {\romannumeral`&&\XINTinFloatBinomial #3}}%
2946 }%
2947 \def\XINT_iiexpr_func_binomial #1#2#3%
2948 {%
2949 \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
2950 \XINT:NEhook:f:one:from:two
2951 {\romannumeral`&&\xintiBinomial #3}}%
2952 }%
2953 \def\XINT_expr_func_pfactorial #1#2#3%
2954 {%
2955 \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
2956 \XINT:NEhook:f:one:from:two
2957 {\romannumeral`&&\xintPFactorial #3}}%
2958 }%
2959 \def\XINT_flexpr_func_pfactorial #1#2#3%
2960 {%
2961 \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
2962 \XINT:NEhook:f:one:from:two
2963 {\romannumeral`&&\XINTinFloatPFactorial #3}}%
2964 }%
2965 \def\XINT_iiexpr_func_pfactorial #1#2#3%
2966 {%
2967 \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
2968 \XINT:NEhook:f:one:from:two
2969 {\romannumeral`&&\xintiPFactorial #3}}%
2970 }%

```

```

2971 \def\XINT_expr_func_randrange #1#2#3%
2972 {%
2973     \expandafter #1\expandafter #2\expanded{{{
2974     \XINT:expr:randrange #3,!%
2975     }}}%
2976 }%
2977 \let\XINT_flexpr_func_randrange\XINT_expr_func_randrange
2978 \def\XINT_iiexpr_func_randrange #1#2#3%
2979 {%
2980     \expandafter #1\expandafter #2\expanded{{{
2981     \XINT:iiexpr:randrange #3,!%
2982     }}}%
2983 }%
2984 \def\XINT:expr:randrange #1#2#3!%
2985 {%
2986     \if\relax#3\relax\expandafter\xint_firstoftwo\else
2987         \expandafter\xint_secondoftwo\fi
2988     {\xintiiRandRange{\XINT:NEhook:f:one:from:one:direct\xintNum{#1}}}%
2989     {\xintiiRandRangeAtoB{\XINT:NEhook:f:one:from:one:direct\xintNum{#1}}%
2990         {\XINT:NEhook:f:one:from:one:direct\xintNum{#2}}}%
2991     }%
2992 }%
2993 \def\XINT:iiexpr:randrange #1#2#3!%
2994 {%
2995     \if\relax#3\relax\expandafter\xint_firstoftwo\else
2996         \expandafter\xint_secondoftwo\fi
2997     {\xintiiRandRange{#1}}%
2998     {\xintiiRandRangeAtoB{#1}{#2}}%
2999 }%
3000 \def\XINT_expr_func_quo #1#2#3%
3001 {%
3002     \expandafter #1\expandafter #2\expandafter{\romannumeral`&&%
3003     \XINT:NEhook:f:one:from:two
3004     {\romannumeral`&&\xintiQuo #3}}%
3005 }%
3006 \let\XINT_flexpr_func_quo\XINT_expr_func_quo
3007 \def\XINT_iiexpr_func_quo #1#2#3%
3008 {%
3009     \expandafter #1\expandafter #2\expandafter{\romannumeral`&&%
3010     \XINT:NEhook:f:one:from:two
3011     {\romannumeral`&&\xintiiQuo #3}}%
3012 }%
3013 \def\XINT_expr_func_rem #1#2#3%
3014 {%
3015     \expandafter #1\expandafter #2\expandafter{\romannumeral`&&%
3016     \XINT:NEhook:f:one:from:two
3017     {\romannumeral`&&\xintiRem #3}}%
3018 }%
3019 \let\XINT_flexpr_func_rem\XINT_expr_func_rem
3020 \def\XINT_iiexpr_func_rem #1#2#3%
3021 {%
3022     \expandafter #1\expandafter #2\expandafter{\romannumeral`&&%

```

```

3023 \XINT:NEhook:f:one:from:two
3024 {\romannumeral`&&\xintiiRem #3}}%
3025 }%
3026 \def\XINT_expr_func_gcd #1#2#3%
3027 {%
3028 \expandafter #1\expandafter #2\expandafter{\expandafter
3029 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_GCDof#3^}}%
3030 }%
3031 \let\XINT_flexpr_func_gcd\XINT_expr_func_gcd
3032 \def\XINT_iiexpr_func_gcd #1#2#3%
3033 {%
3034 \expandafter #1\expandafter #2\expandafter{\expandafter
3035 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_iiGCDof#3^}}%
3036 }%
3037 \def\XINT_expr_func_lcm #1#2#3%
3038 {%
3039 \expandafter #1\expandafter #2\expandafter{\expandafter
3040 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_LCMof#3^}}%
3041 }%
3042 \let\XINT_flexpr_func_lcm\XINT_expr_func_lcm
3043 \def\XINT_iiexpr_func_lcm #1#2#3%
3044 {%
3045 \expandafter #1\expandafter #2\expandafter{\expandafter
3046 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_iiLCMof#3^}}%
3047 }%
3048 \def\XINT_expr_func_max #1#2#3%
3049 {%
3050 \expandafter #1\expandafter #2\expandafter{\expandafter
3051 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_Maxof#3^}}%
3052 }%
3053 \def\XINT_iiexpr_func_max #1#2#3%
3054 {%
3055 \expandafter #1\expandafter #2\expandafter{\expandafter
3056 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_iiMaxof#3^}}%
3057 }%
3058 \def\XINT_flexpr_func_max #1#2#3%
3059 {%
3060 \expandafter #1\expandafter #2\expandafter{\expandafter
3061 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINTinFloatMaxof#3^}}%
3062 }%
3063 \def\XINT_expr_func_min #1#2#3%
3064 {%
3065 \expandafter #1\expandafter #2\expandafter{\expandafter
3066 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_Minof#3^}}%
3067 }%
3068 \def\XINT_iiexpr_func_min #1#2#3%
3069 {%
3070 \expandafter #1\expandafter #2\expandafter{\expandafter
3071 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_iiMinof#3^}}%
3072 }%
3073 \def\XINT_flexpr_func_min #1#2#3%
3074 {%

```

```

3075 \expandafter #1\expandafter #2\expandafter{\expandafter
3076 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINTinFloatMinof#3^}}%
3077 }%
3078 \expandafter
3079 \def\csname XINT_expr_func_+\endcsname #1#2#3%
3080 {%
3081 \expandafter #1\expandafter #2\expandafter{\expandafter
3082 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_Sum#3^}}%
3083 }%
3084 \expandafter
3085 \def\csname XINT_flexpr_func_+\endcsname #1#2#3%
3086 {%
3087 \expandafter #1\expandafter #2\expandafter{\expandafter
3088 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINTinFloatSum#3^}}%
3089 }%
3090 \expandafter
3091 \def\csname XINT_iiexpr_func_+\endcsname #1#2#3%
3092 {%
3093 \expandafter #1\expandafter #2\expandafter{\expandafter
3094 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_iiSum#3^}}%
3095 }%
3096 \expandafter
3097 \def\csname XINT_expr_func_*\endcsname #1#2#3%
3098 {%
3099 \expandafter #1\expandafter #2\expandafter{\expandafter
3100 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_Prd#3^}}%
3101 }%
3102 \expandafter
3103 \def\csname XINT_flexpr_func_*\endcsname #1#2#3%
3104 {%
3105 \expandafter #1\expandafter #2\expandafter{\expandafter
3106 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINTinFloatPrd#3^}}%
3107 }%
3108 \expandafter
3109 \def\csname XINT_iiexpr_func_*\endcsname #1#2#3%
3110 {%
3111 \expandafter #1\expandafter #2\expandafter{\expandafter
3112 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_iiPrd#3^}}%
3113 }%
3114 \def\XINT_expr_func_all #1#2#3%
3115 {%
3116 \expandafter #1\expandafter #2\expandafter{\expandafter
3117 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_ANDof#3^}}%
3118 }%
3119 \let\XINT_flexpr_func_all\XINT_expr_func_all
3120 \let\XINT_iiexpr_func_all\XINT_expr_func_all
3121 \def\XINT_expr_func_any #1#2#3%
3122 {%
3123 \expandafter #1\expandafter #2\expandafter{\expandafter
3124 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_ORof#3^}}%
3125 }%
3126 \let\XINT_flexpr_func_any\XINT_expr_func_any

```

```

3127 \let\XINT_iiexpr_func_any\XINT_expr_func_any
3128 \def\XINT_expr_func_xor #1#2#3%
3129 {%
3130     \expandafter #1\expandafter #2\expandafter{\expandafter
3131         {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_XORof#3^}}%
3132 }%
3133 \let\XINT_flexpr_func_xor\XINT_expr_func_xor
3134 \let\XINT_iiexpr_func_xor\XINT_expr_func_xor
3135 \def\XINT_expr_func_len #1#2#3%
3136 {%
3137     \expandafter#1\expandafter#2\expandafter{\expandafter{%
3138         \romannumeral`&&\XINT:NEhook:f:noeval:from:braced:u\xintLength#3^%
3139     }}%
3140 }%
3141 \let\XINT_flexpr_func_len \XINT_expr_func_len
3142 \let\XINT_iiexpr_func_len \XINT_expr_func_len
3143 \def\XINT_expr_func_first #1#2#3%
3144 {%
3145     \expandafter #1\expandafter #2\expandafter{%
3146         \romannumeral`&&\XINT:NEhook:f:noeval:from:braced:u\xintFirstOne#3^%
3147     }%
3148 }%
3149 \let\XINT_flexpr_func_first\XINT_expr_func_first
3150 \let\XINT_iiexpr_func_first\XINT_expr_func_first
3151 \def\XINT_expr_func_last #1#2#3%
3152 {%
3153     \expandafter #1\expandafter #2\expandafter{%
3154         \romannumeral`&&\XINT:NEhook:f:noeval:from:braced:u\xintLastOne#3^%
3155     }%
3156 }%
3157 \let\XINT_flexpr_func_last\XINT_expr_func_last
3158 \let\XINT_iiexpr_func_last\XINT_expr_func_last
3159 \def\XINT_expr_func_reversed #1#2#3%
3160 {%
3161     \expandafter #1\expandafter #2\expandafter{%
3162         \romannumeral`&&\XINT:NEhook:f:reverse\XINT_expr_reverse
3163         #3^^#3\xint:\xint:\xint:\xint:
3164         \xint:\xint:\xint:\xint:\xint_bye
3165     }%
3166 }%
3167 \def\XINT_expr_reverse #1#2%
3168 {%
3169     \if ^\noexpand#2%
3170         \expandafter\XINT_expr_reverse:_one_or_none\string#1.%
3171     \else
3172         \expandafter\XINT_expr_reverse:_at_least_two
3173     \fi
3174 }%
3175 \def\XINT_expr_reverse:_at_least_two #1^^{\XINT_revwbr_loop {}}%
3176 \def\XINT_expr_reverse:_one_or_none #1%
3177 {%
3178     \if #1\bgroup\xint_dothis\XINT_expr_reverse:_nutple\fi

```

```

3179 \if #1^\xint_dothis\xINT_expr_reverse:_nil\fi
3180 \xint_orthat\xINT_expr_reverse:_leaf
3181 }%
3182 \edef\xINT_expr_reverse:_nil #1\xint_bye{\noexpand\fi\space}%
3183 \def\xINT_expr_reverse:_leaf#1\fi #2\xint:#3\xint_bye{\fi\xint_gob_andstop_i#2}%
3184 \def\xINT_expr_reverse:_nutple%
3185 {%
3186 \expandafter\xINT_expr_reverse:_nutple_a\expandafter{\string}%
3187 }%
3188 \def\xINT_expr_reverse:_nutple_a #1^#2\xint:#3\xint_bye
3189 {%
3190 \fi\expandafter
3191 {\romannumeral0\xINT_revwbr_loop{}}#2\xint:#3\xint_bye}%
3192 }%
3193 \let\xINT_flexpr_func_reversed\xINT_expr_func_reversed
3194 \let\xINT_iiexpr_func_reversed\xINT_expr_func_reversed
3195 \def\xINT_expr_func_if #1#2#3%
3196 {%
3197 \expandafter #1\expandafter #2\expandafter{%
3198 \romannumeral`&&\XINT:NEhook:branch{\romannumeral`&&\xintiiifNotZero #3}}%
3199 }%
3200 \let\xINT_flexpr_func_if\xINT_expr_func_if
3201 \let\xINT_iiexpr_func_if\xINT_expr_func_if
3202 \def\xINT_expr_func_ifint #1#2#3%
3203 {%
3204 \expandafter #1\expandafter #2\expandafter{%
3205 \romannumeral`&&\XINT:NEhook:branch{\romannumeral`&&\xintifInt #3}}%
3206 }%
3207 \let\xINT_iiexpr_func_ifint\xINT_expr_func_ifint
3208 \def\xINT_flexpr_func_ifint #1#2#3%
3209 {%
3210 \expandafter #1\expandafter #2\expandafter{%
3211 \romannumeral`&&\XINT:NEhook:branch{\romannumeral`&&\xintifFloatInt #3}}%
3212 }%
3213 \def\xINT_expr_func_ifone #1#2#3%
3214 {%
3215 \expandafter #1\expandafter #2\expandafter{%
3216 \romannumeral`&&\XINT:NEhook:branch{\romannumeral`&&\xintifOne #3}}%
3217 }%
3218 \let\xINT_flexpr_func_ifone\xINT_expr_func_ifone
3219 \def\xINT_iiexpr_func_ifone #1#2#3%
3220 {%
3221 \expandafter #1\expandafter #2\expandafter{%
3222 \romannumeral`&&\XINT:NEhook:branch{\romannumeral`&&\xintiiifOne #3}}%
3223 }%
3224 \def\xINT_expr_func_ifsgn #1#2#3%
3225 {%
3226 \expandafter #1\expandafter #2\expandafter{%
3227 \romannumeral`&&\XINT:NEhook:branch{\romannumeral`&&\xintiiifSgn #3}}%
3228 }%
3229 \let\xINT_flexpr_func_ifsgn\xINT_expr_func_ifsgn
3230 \let\xINT_iiexpr_func_ifsgn\xINT_expr_func_ifsgn

```

```

3231 \def\XINT_expr_func_nuple #1#2#3{#1#2{{#3}}}%
3232 \let\XINT_flexpr_func_nuple\XINT_expr_func_nuple
3233 \let\XINT_iiexpr_func_nuple\XINT_expr_func_nuple
3234 \def\XINT_expr_func_unpack #1#2%#3%
3235     {\expandafter#1\expandafter#2\romannumeral0\XINT:NEhook:unpack}%
3236 \let\XINT_flexpr_func_unpack\XINT_expr_func_unpack
3237 \let\XINT_iiexpr_func_unpack\XINT_expr_func_unpack

```

11.29 User declared functions

It is possible that the author actually does understand at this time the `\xintNewExpr`/`\xintdeffunc` refactored code and mechanisms for the first time since 2014: past evolutions such as the 2018 1.3 refactoring were done a bit in the fog (although they did accomplish a crucial step).

The 1.4 version of function and macro definitions is much more powerful than 1.3 one. But the mechanisms such as «omit», «abort» and «break()» in `iter()` et al. can't be translated into much else than their actual code when they potentially have to apply to non-numeric only context. The 1.4 `\xintdeffunc` is thus apparently able to digest them but its pre-parsing benefits are limited compared to simply assigning such parts of an expression to a mock-function created by `\xintNewFunction` (which creates simply a TeX macro from its substitution expression in macro parameters and add syntactic sugar to let it appear to `\xintexpr` as a genuine «function» although nothing of the syntax has really been pre-parsed.)

At 1.4 fetching the expression up to final semi-colon is done using `\XINT_expr_fetch_to_semicolon`, hence semi-colons arising in the syntax do not need to be hidden inside braces.

11.29.1	<code>\xintdeffunc</code> , <code>\xintdefiifunc</code> , <code>\xintdeffloatfunc</code>	392
11.29.2	<code>\xintdefufunc</code> , <code>\xintdefiiufunc</code> , <code>\xintdeffloatufunc</code>	395
11.29.3	<code>\xintunassignexprfunc</code> , <code>\xintunassigniiexprfunc</code> , <code>\xintunassignfloatexprfunc</code>	396
11.29.4	<code>\xintNewFunction</code>	396
11.29.5	Mysterious stuff	398
11.29.6	<code>\XINT_expr_redefinemacros</code>	409
11.29.7	<code>\xintNewExpr</code> , <code>\xintNewIExpr</code> , <code>\xintNewFloatExpr</code> , <code>\xintNewIIExpr</code>	410
11.29.8	<code>\ifxintexprsafecatcodes</code> , <code>\xintexprSafeCatcodes</code> , <code>\xintexprRestoreCatcodes</code>	412

11.29.1 `\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.

1.2c (2015/11/13).

Function names first expanded then detokenized and cleaned of spaces.

1.2e (2015/11/21).

No `\detokenize` anymore on the function names. And `#1(#2)#3=#4` parameter pattern to avoid to have to worry if a `:` is there and it is active.

1.2f (2016/02/22).

La macro associée à la fonction ne débute plus par un `\romannumeral`, car de toute façon elle est pour emploi dans `\csname..\endcsname`.

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.

1.3e (??).

`\xintdefefunc` variant added for functions which will expand completely if used with numeric arguments in other function definitions. They can't be used for recursive definitions.

1.4 (2020/01/10).

Multi-letter variables can be used (with no prior declaration)

1.4 (2020/01/11).

The new internal data model has caused many worries initially (such as whether to allow functions with «ople» outputs in contrast to «numbers» or «nutples») but in the end all is simpler again and the refactoring of ? and ?? in function definitions allows to fuse inert functions (allowing recursive definitions) and expanding functions (expanding completely if with numeric arguments) into a single entity.

Thus the 1.3e `\xintdefefunc`, `\xintdefiiefunc`, `\xintdeffloatefunc` constructors of «expanding» functions are kept only as aliases of legacy `\xintdeffunc` et al. and deprecated.

A special situation is with functions of no variables. In that case it will be handled as an inert entity, else they would not be different from variables.

```

3238 \def\xINT_tmpa #1#2#3#4#5%
3239 {%
3240   \def #1##1(##2)##3={%
3241     \edef\xINT_deffunc_tmpa {##1}%
3242     \edef\xINT_deffunc_tmpa {\xint_zapspaces_o \xINT_deffunc_tmpa}%
3243     \def\xINT_deffunc_tmpe {0}%
3244     \edef\xINT_deffunc_tmpe {##2}%
3245     \edef\xINT_deffunc_tmpe {\xint_zapspaces_o\xINT_deffunc_tmpe}%
3246     \def\xINT_deffunc_tmpe {0}%
3247     \expandafter\xromannumeral\xINT_expr_fetch_to_semicolon
3248   }% end of \xintdeffunc_a definition
3249   \def#5##1{%
3250     \def\xINT_deffunc_tmpe{##1}%
3251     \ifnum\xintLength:f:csv{\xINT_deffunc_tmpe}>\xint_c_
3252       \xintFor #####1 in {\xINT_deffunc_tmpe}\do
3253       {%
3254         \xintifForFirst{\let\xINT_deffunc_tmpe\empty}{}%
3255         \def\xINT_deffunc_tmpef{#####1}%
3256         \if*\xintFirstItem{#####1}%
3257           \xintifForLast
3258           {%
3259             \def\xINT_deffunc_tmpef{1}%
3260             \edef\xINT_deffunc_tmpef{\xintTrim{1}{#####1}}%
3261           }%
3262           {%
3263             \edef\xINT_deffunc_tmpef{\xintTrim{1}{#####1}}%
3264             \xintMessage{xintexpr}{Error}
3265             {Only the last positional argument can be variadic. Trimmed #####1 to
3266             \xINT_deffunc_tmpef}%
3267           }%
3268         \fi

```

```

3269     \XINT_expr_makedummy{\XINT_deffunc_tmpf}%
3270     \edef\XINT_deffunc_tmpd{\XINT_deffunc_tmpd{\XINT_deffunc_tmpf}}%
3271     \edef\XINT_deffunc_tmpb {\the\numexpr\XINT_deffunc_tmpb+\xint_c_i}%
3272     \edef\XINT_deffunc_tmpc {subs(\unexpanded\expandafter{\XINT_deffunc_tmpc},%
3273                                   \XINT_deffunc_tmpf=#####\XINT_deffunc_tmpb)}%
3274     }%
3275     \fi

Place holder for comments. Logic at 1.4 is simplified here compared to earlier releases.

3276     \ifcase\XINT_deffunc_tmpb\space
3277     \expandafter\XINT_expr_defuserfunc_none\csname
3278     \else
3279     \expandafter\XINT_expr_defuserfunc\csname
3280     \fi
3281         XINT_#2_func_\XINT_deffunc_tmpa\expandafter\endcsname
3282     \csname XINT_#2_userfunc_\XINT_deffunc_tmpa\expandafter\endcsname
3283     \expandafter{\XINT_deffunc_tmpa}{#2}%
3284     \expandafter#3\csname XINT_#2_userfunc_\XINT_deffunc_tmpa\endcsname
3285         [\XINT_deffunc_tmpb]{\XINT_deffunc_tmpc}%
3286     \ifxintverbose\xintMessage {xintexpr}{Info}
3287         {Function \XINT_deffunc_tmpa\space for \string\xint #4 parser
3288         associated to \string\XINT_#2_userfunc_\XINT_deffunc_tmpa\space
3289         with \ifxintglobaldefs global \fi meaning \expandafter\meaning
3290         \csname XINT_#2_userfunc_\XINT_deffunc_tmpa\endcsname}%
3291     \fi
3292     \xintFor* ####1 in {\XINT_deffunc_tmpd}:{\xintrestorevariablesilently{####1}}%
3293     \xintexprRestoreCatcodes
3294 }% end of \xintdeffunc_b definition
3295 }%

3296 \def\xintdeffunc      {\xintexprSafeCatcodes\xintdeffunc_a}%
3297 \def\xintdefiifunc    {\xintexprSafeCatcodes\xintdefiifunc_a}%
3298 \def\xintdeffloatfunc {\xintexprSafeCatcodes\xintdeffloatfunc_a}%
3299 \XINT_tmpa\xintdeffunc_a {expr} \XINT_NewFunc {expr}\xintdeffunc_b
3300 \XINT_tmpa\xintdefiifunc_a {iiexpr}\XINT_NewIIFunc {iiexpr}\xintdefiifunc_b
3301 \XINT_tmpa\xintdeffloatfunc_a{flexpr}\XINT_NewFloatFunc{floatexpr}\xintdeffloatfunc_b
3302 \def\XINT_expr_defuserfunc_none #1#2#3#4%
3303 {%
3304     \XINT_global
3305     \def #1##1##2##3%
3306     {%
3307         \expandafter##1\expandafter##2\expanded{%
3308             {\XINT:NEhook:usernoargfunc\csname XINT_#4_userfunc_#3\endcsname}%
3309         }%
3310     }%
3311 }%

3312 \let\XINT:NEhook:usernoargfunc \empty
3313 \def\XINT_expr_defuserfunc #1#2#3#4%
3314 {%
3315     \if0\XINT_deffunc_tmpe
3316     \XINT_global
3317     \def #1##1##2##3%
3318     {%
3319         \expandafter ##1\expandafter##2\expanded\bgroup{\iffalse}\fi

```

```

3320      \XINT:NEhook:userfunc{XINT_#4_userfunc_#3}#2##3%
3321  }%
3322  \else
3323  \def #1##1{%
3324  \XINT_global\def #1####1####2####3%
3325  {%
3326      \expandafter #####1\expandafter####2\expanded\bgroup{\iffalse}\fi
3327      \XINT:NEhook:userfunc:argv{##1}{XINT_#4_userfunc_#3}#2####3%
3328  }}\expandafter#1\expandafter{\the\numexpr\XINT_deffunc_tmpb-1}%
3329  \fi
3330 }%
3331 \def\XINT:NEhook:userfunc #1#2#3{#2#3\iffalse{{\fi}}}%
3332 \def\XINT:NEhook:userfunc:argv #1#2#3#4%
3333   {\expandafter#3\expanded{\xintKeep{#1}{#4}{\xintTrim{#1}{#4}}}\iffalse{{\fi}}}%
3334 \let\xintdefefunc\xintdeffunc
3335 \let\xintdefiifunc\xintdefiifunc
3336 \let\xintdeffloatefunc\xintdeffloatfunc

```

11.29.2 \xintdefufunc, \xintdefiufunc, \xintdeffloatufunc

1.4

```

3337 \def\XINT_tmpa #1#2#3#4#5#6%
3338 {%
3339   \def #1##1(##2)##3={%
3340     \edef\XINT_defufunc_tmpa {##1}%
3341     \edef\XINT_defufunc_tmpa {\xint_zapspaces_o \XINT_defufunc_tmpa}%
3342     \edef\XINT_defufunc_tmpd {##2}%
3343     \edef\XINT_defufunc_tmpd {\xint_zapspaces_o\XINT_defufunc_tmpd}%
3344     \expandafter#5\romannumeral\XINT_expr_fetch_to_semicolon
3345   }% end of \xint_defufunc_a
3346   \def#5##1{%
3347     \def\XINT_defufunc_tmpe{##1}%
3348     \ifnum\xintLength:f:csv{\XINT_defufunc_tmpe}=\xint_c_i
3349       \expandafter#6%
3350     \else
3351       \xintMessage {xintexpr}{ERROR}
3352       {Universal functions must be functions of one argument only,
3353       but the declaration of \XINT_defufunc_tmpe\space
3354       has \xintLength:f:csv{\XINT_defufunc_tmpe} of them. Cancelled.}%
3355       \xintexprRestoreCatcodes
3356     \fi
3357   }% end of \xint_defufunc_b
3358   \def #6{%
3359     \XINT_expr_makedummy{\XINT_defufunc_tmpe}%
3360     \edef\XINT_defufunc_tmpe {subs(\unexpanded\expandafter{\XINT_defufunc_tmpe},%
3361                                   \XINT_defufunc_tmpe=#####1)}%
3362     \expandafter\XINT_expr_defuserufunc
3363     \csname XINT_#2_func_\XINT_defufunc_tmpe\expandafter\endcsname
3364     \csname XINT_#2_userufunc_\XINT_defufunc_tmpe\expandafter\endcsname
3365     \expandafter{\XINT_defufunc_tmpe}{#2}%
3366     \expandafter#3\csname XINT_#2_userufunc_\XINT_defufunc_tmpe\endcsname
3367     [1]{\XINT_defufunc_tmpe}%

```

```

3368 \ifxintverbose\xintMessage {xintexpr}{Info}
3369 {Universal function \XINT_defufunc_tmpa\space for \string\xint #4 parser
3370 associated to \string\XINT_#2_userufunc_\XINT_defufunc_tmpa\space
3371 with \ifxintglobaldefs global \fi meaning \expandafter\meaning
3372 \csname XINT_#2_userufunc_\XINT_defufunc_tmpa\endcsname}%
3373 \fi
3374 }% end of \xint_defufunc_c
3375 }%
3376 \def\xintdefufunc {\xintexprSafeCatcodes\xintdefufunc_a}%
3377 \def\xintdefiiufunc {\xintexprSafeCatcodes\xintdefiiufunc_a}%
3378 \def\xintdeffloatufunc {\xintexprSafeCatcodes\xintdeffloatufunc_a}%
3379 \XINT_tmpa\xintdefufunc_a {expr} \XINT_NewFunc {expr}%
3380 \xintdefufunc_b\xintdefufunc_c
3381 \XINT_tmpa\xintdefiiufunc_a {iiexpr}\XINT_NewIIFunc {iiexpr}%
3382 \xintdefiiufunc_b\xintdefiiufunc_c
3383 \XINT_tmpa\xintdeffloatufunc_a{fexpr}\XINT_NewFloatFunc{floatexpr}%
3384 \xintdeffloatufunc_b\xintdeffloatufunc_c
3385 \def\XINT_expr_defuserufunc #1#2#3#4%
3386 {%
3387 \XINT_global
3388 \def #1##1##2%##3%
3389 {%
3390 \expandafter ##1\expandafter##2\expanded
3391 \XINT:NEhook:userufunc{XINT_#4_userufunc_#3}#2%##3%
3392 }%
3393 }%
3394 \def\XINT:NEhook:userufunc #1{\XINT:expr:mapwithin}%

```

11.29.3 \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.

```

3395 \def\XINT_tmpa #1{\expandafter\def\csname xintunassign#1func\endcsname ##1{%
3396 \edef\XINT_unfunc_tmpa{##1}%
3397 \edef\XINT_unfunc_tmpa {\xint_zapspaces_o\XINT_unfunc_tmpa}%
3398 \XINT_global\expandafter
3399 \let\csname XINT_#1_func_\XINT_unfunc_tmpa\endcsname\xint_undefined
3400 \XINT_global\expandafter
3401 \let\csname XINT_#1_userfunc_\XINT_unfunc_tmpa\endcsname\xint_undefined
3402 \XINT_global\expandafter
3403 \let\csname XINT_#1_userufunc_\XINT_unfunc_tmpa\endcsname\xint_undefined
3404 \ifxintverbose\xintMessage {xintexpr}{Info}
3405 {Function \XINT_unfunc_tmpa\space for \string\xint #1 parser now
3406 \ifxintglobaldefs globally \fi undefined.}%
3407 \fi}}%
3408 \XINT_tmpa{expr}\XINT_tmpa{iiexpr}\XINT_tmpa{floatexpr}%

```

11.29.4 \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 pars-

ing is delayed until function execution. Hence the expression admits all constructs, contrarily to \xintNewExpr or \xintdefunc.

As the letters used for variables in \xintdefunc, #1, #2, etc... can not stand for non numeric «oples», because at time of function call f(a, b, c, ...) how to decide if #1 stands for a or a, b etc... ? Or course «a» can be packed and thus the macro function can handle #1 as a «nutple» and for this be defined with the * unpacking operator being applied to it.

```

3409 \def\xintNewFunction #1#2[#3]#4%
3410 {%
3411   \edef\XINT_newfunc_tmpa {#1}%
3412   \edef\XINT_newfunc_tmpa {\xint_zapspaces_o \XINT_newfunc_tmpa}%
3413   \def\XINT_newfunc_tmpb ##1##2##3##4##5##6##7##8##9{#4}%
3414   \begingroup
3415     \ifcase #3\relax
3416       \toks0{}%
3417     \or \toks0{##1}%
3418     \or \toks0{##1##2}%
3419     \or \toks0{##1##2##3}%
3420     \or \toks0{##1##2##3##4}%
3421     \or \toks0{##1##2##3##4##5}%
3422     \or \toks0{##1##2##3##4##5##6}%
3423     \or \toks0{##1##2##3##4##5##6##7}%
3424     \or \toks0{##1##2##3##4##5##6##7##8}%
3425     \else \toks0{##1##2##3##4##5##6##7##8##9}%
3426     \fi
3427     \expandafter
3428   \endgroup\expandafter
3429   \XINT_global\expandafter
3430   \def\csname XINT_expr_macrofunc_\XINT_newfunc_tmpa\expandafter\endcsname
3431   \the\toks0\expandafter{\XINT_newfunc_tmpb
3432     {\XINTfstop.{##1}}{\XINTfstop.{##2}}{\XINTfstop.{##3}}}%
3433     {\XINTfstop.{##4}}{\XINTfstop.{##5}}{\XINTfstop.{##6}}}%
3434     {\XINTfstop.{##7}}{\XINTfstop.{##8}}{\XINTfstop.{##9}}}%
3435   \expandafter\XINT_expr_newfunction
3436   \csname XINT_expr_func_\XINT_newfunc_tmpa\expandafter\endcsname
3437   \expandafter{\XINT_newfunc_tmpa}\xintbareeval
3438   \expandafter\XINT_expr_newfunction
3439   \csname XINT_iexpr_func_\XINT_newfunc_tmpa\expandafter\endcsname
3440   \expandafter{\XINT_newfunc_tmpa}\xintbareieeval
3441   \expandafter\XINT_expr_newfunction
3442   \csname XINT_flexpr_func_\XINT_newfunc_tmpa\expandafter\endcsname
3443   \expandafter{\XINT_newfunc_tmpa}\xintbarefloateval
3444   \ifxintverbose
3445     \xintMessage {xintexpr}{Info}
3446     {Function \XINT_newfunc_tmpa\space for the expression parsers is
3447       associated to \string\XINT_expr_macrofunc_\XINT_newfunc_tmpa\space
3448       with \ifxintglobaldefs global \fi meaning \expandafter\meaning
3449       \csname XINT_expr_macrofunc_\XINT_newfunc_tmpa\endcsname}%
3450   \fi
3451 }%
3452 \def\XINT_expr_newfunction #1#2#3%
3453 {%
3454   \XINT_global

```

```

3455 \def#1##1##2##3%
3456 {\expandafter ##1\expandafter ##2%
3457 \romannumeral0\XINT:NEhook:macrofunc
3458 #3{\csname XINT_expr_macrofunc_#2\endcsname##3}\relax
3459 }%
3460 }%
3461 \let\XINT:NEhook:macrofunc\empty

```

11.29.5 Mysterious stuff

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.

It is always too soon to try to comment and explain. In brief, this attempts to hack into the *purely numeric* `\xintexpr` parsers to transform them into *symbolic* parsers, allowing to do once and for all the parsing job and inherit a gigantic nested macro. Originally only f-expandable nesting. The initial motivation was that the `\csname` encapsulation impacted the string pool memory. Later this work proved to be the basis to provide support for implementing user-defined functions and it is now its main purpose.

Deep refactorings happened at 1.3 and 1.4.

At 1.3 the crucial idea of the «hook» macros was introduced, reducing considerably the preparatory work done by `\xintNewExpr`.

At 1.4 further considerable simplifications happened, and it is possible that the author currently does at long last understand the code!

The 1.3 code had serious complications with trying to identify would-be «list» arguments, distinguishing them from «single» arguments (things like parsing `#2+[[#1..#3]..#4][#5:#6]]*#7` and convert it to a single nested f-expandable macro...)

The conversion at 1.4 is both more powerful and simpler, due in part to the new storage model which from `\csname` encapsulated comma separated values up to 1.3f became simply a braced list of braced values, and also crucially due to the possibilities opened up by usage of `\expanded` primitive.

```

3462 \catcode`~ 12
3463 \def\XINT:NE:hashtilde#1~#2#3\relax{\unless\if !#21\fi}%
3464 \def\XINT:NE:hashash#1{%
3465 \def\XINT:NE:hashash##1#1##2##3\relax{\unless\if !##21\fi}%
3466 }\expandafter\XINT:NE:hashash\string#%
3467 \def\XINT:NE:unpack #1{%
3468 \def\XINT:NE:unpack ##1%
3469 {%
3470 \if0\XINT:NE:hashtilde ##1~!\relax
3471 \XINT:NE:hashash ##1#1!\relax 0\else
3472 \expandafter\XINT:NE:unpack:p\fi
3473 \xint_stop_atfirstofone{##1}%
3474 }}\expandafter\XINT:NE:unpack\string#%
3475 \def\XINT:NE:unpack:p#1#2%
3476 {{~romannumeral0~expandafter~xint_stop_atfirstofone~expanded{#2}}}%
3477 \def\XINT:NE:f:one:from:one #1{%
3478 \def\XINT:NE:f:one:from:one ##1%
3479 {%
3480 \if0\XINT:NE:hashtilde ##1~!\relax
3481 \XINT:NE:hashash ##1#1!\relax 0\else
3482 \xint_dothis\XINT:NE:f:one:from:one_a\fi

```

```

3483 \xint_orthat\XINT:NE:f:one:from:one_b
3484 ##1&&A%
3485 }}\expandafter\XINT:NE:f:one:from:one:string#%
3486 \def\XINT:NE:f:one:from:one_a\romannumeral`&&@#1#2&&A%
3487 {%
3488 \expandafter{\detokenize{\expandafter#1}#2}%
3489 }%
3490 \def\XINT:NE:f:one:from:one_b#1{%
3491 \def\XINT:NE:f:one:from:one_b\romannumeral`&&@##1##2&&A%
3492 {%
3493 \expandafter{\romannumeral`&&@%
3494 \if0\XINT:NE:hasilde ##2~!\relax
3495 \XINT:NE:hashash ##2#1!\relax 0\else
3496 \expandafter\string\fi
3497 ##1{##2}}%
3498 }}\expandafter\XINT:NE:f:one:from:one_b\string#%
3499 \def\XINT:NE:f:one:from:one:direct #1#2{\XINT:NE:f:one:from:one:direct_a #2&&A{#1}}%
3500 \def\XINT:NE:f:one:from:one:direct_a #1#2&&A#3%
3501 {%
3502 \if ##1\xint_dothis {\detokenize{#3}}\fi
3503 \if ~#1\xint_dothis {\detokenize{#3}}\fi
3504 \xint_orthat {#3}{#1#2}%
3505 }%
3506 \def\XINT:NE:f:one:from:two #1{%
3507 \def\XINT:NE:f:one:from:two ##1%
3508 {%
3509 \if0\XINT:NE:hasilde ##1~!\relax
3510 \XINT:NE:hashash ##1#1!\relax 0\else
3511 \xint_dothis\XINT:NE:f:one:from:two_a\fi
3512 \xint_orthat\XINT:NE:f:one:from:two_b ##1&&A%
3513 }}\expandafter\XINT:NE:f:one:from:two\string#%
3514 \def\XINT:NE:f:one:from:two_a\romannumeral`&&@#1#2&&A%
3515 {%
3516 \expandafter{\detokenize{\expandafter#1\expanded}{#2}}%
3517 }%
3518 \def\XINT:NE:f:one:from:two_b#1{%
3519 \def\XINT:NE:f:one:from:two_b\romannumeral`&&@##1##2##3&&A%
3520 {%
3521 \expandafter{\romannumeral`&&@%
3522 \if0\XINT:NE:hasilde ##2##3~!\relax
3523 \XINT:NE:hashash ##2##3#1!\relax 0\else
3524 \expandafter\string\fi
3525 ##1{##2}{##3}}%
3526 }}\expandafter\XINT:NE:f:one:from:two_b\string#%
3527 \def\XINT:NE:f:one:from:two:direct #1#2#3{\XINT:NE:two_fork #2&&A#3&&A#1{#2}{#3}}%
3528 \def\XINT:NE:two_fork #1#2&&A#3#4&&A{\XINT:NE:two_fork_nn#1#3}%
3529 \def\XINT:NE:two_fork_nn #1#2%
3530 {%
3531 \if #1##\xint_dothis\string\fi
3532 \if #1~\xint_dothis\string\fi
3533 \if #2##\xint_dothis\string\fi
3534 \if #2~\xint_dothis\string\fi

```



```

3535 \xint_orthat{}%
3536 }%
3537 \def\xint:NE:f:one:and:opt:direct#1{%
3538 \def\xint:NE:f:one:and:opt:direct##1!%
3539 {%
3540 \if0\xint:NE:hasilde ##1~!\relax
3541 \xint:NE:hashash ##1#1!\relax 0\else
3542 \xint_dothis\xint:NE:f:one:and:opt_a\fi
3543 \xint_orthat\xint:NE:f:one:and:opt_b ##1&&A%
3544 }}\expandafter\xint:NE:f:one:and:opt:direct\string#%
3545 \def\xint:NE:f:one:and:opt_a #1#2&&A#3#4%
3546 {%
3547 \detokenize{\romannumeral-`0\expandafter#1\expanded{#2}$XINT_expr_exclam#3#4}%$
3548 }%
3549 \def\xint:NE:f:one:and:opt_b\xint:expr:f:one:and:opt #1#2#3&&A#4#5%
3550 {%
3551 \if\relax#3\relax\expandafter\xint_firstoftwo\else
3552 \expandafter\xint_secondoftwo\fi
3553 {\xint:NE:f:one:from:one:direct#4}%
3554 {\expandafter\xint:NE:f:one:withopttoone\expandafter#5%
3555 \expanded{{\xint:NE:f:one:from:one:direct\xintNum{#2}}}}%
3556 {#1}%
3557 }%
3558 \def\xint:NE:f:one:withopttoone#1#2#3{\xint:NE:two_fork #2&&A#3&&A#1[#2]{#3}}%
3559 \def\xint:NE:f:tacitzeroifone:direct#1{%
3560 \def\xint:NE:f:tacitzeroifone:direct##1!%
3561 {%
3562 \if0\xint:NE:hasilde ##1~!\relax
3563 \xint:NE:hashash ##1#1!\relax 0\else
3564 \xint_dothis\xint:NE:f:one:and:opt_a\fi
3565 \xint_orthat\xint:NE:f:tacitzeroifone_b ##1&&A%
3566 }}\expandafter\xint:NE:f:tacitzeroifone:direct\string#%
3567 \def\xint:NE:f:tacitzeroifone_b\xint:expr:f:tacitzeroifone #1#2#3&&A#4#5%
3568 {%
3569 \if\relax#3\relax\expandafter\xint_firstoftwo\else
3570 \expandafter\xint_secondoftwo\fi
3571 {\xint:NE:f:one:from:two:direct#4{0}}%
3572 {\expandafter\xint:NE:f:one:from:two:direct\expandafter#5%
3573 \expanded{{\xint:NE:f:one:from:one:direct\xintNum{#2}}}}%
3574 {#1}%
3575 }%
3576 \def\xint:NE:f:iitacitzeroifone:direct#1{%
3577 \def\xint:NE:f:iitacitzeroifone:direct##1!%
3578 {%
3579 \if0\xint:NE:hasilde ##1~!\relax
3580 \xint:NE:hashash ##1#1!\relax 0\else
3581 \xint_dothis\xint:NE:f:iitacitzeroifone_a\fi
3582 \xint_orthat\xint:NE:f:iitacitzeroifone_b ##1&&A%
3583 }}\expandafter\xint:NE:f:iitacitzeroifone:direct\string#%
3584 \def\xint:NE:f:iitacitzeroifone_a #1#2&&A#3%
3585 {%
3586 \detokenize{\romannumeral-`0\expandafter#1\expanded{#2}$XINT_expr_exclam#3}%$

```



```

3587 }%
3588 \def\XINT:NE:f:iitacitzeroifone_b\XINT:expr:f:iitacitzeroifone #1#2#3&&A#4%
3589 {%
3590     \if\relax#3\relax\expandafter\xint_firstoftwo\else
3591         \expandafter\xint_secondoftwo\fi
3592     {\XINT:NE:f:one:from:two:direct#4{0}}}%
3593     {\XINT:NE:f:one:from:two:direct#4{#2}}}%
3594     {#1}}%
3595 }%
3596 \def\XINT:NE:x:one:from:two #1#2#3{\XINT:NE:x:one:from:two_fork #2&&A#3&&A#1{#2}{#3}}%
3597 \def\XINT:NE:x:one:from:two_fork #1{%
3598 \def\XINT:NE:x:one:from:two_fork ##1##2&&A##3##4&&A%
3599 {%
3600     \if0\XINT:NE:hashtilde ##1##3~!\relax\XINT:NE:hashash ##1##3#1!\relax 0%
3601     \else
3602         \expandafter\XINT:NE:x:one:from:two:p
3603     \fi
3604 }}\expandafter\XINT:NE:x:one:from:two_fork\string#%
3605 \def\XINT:NE:x:one:from:two:p #1#2#3%
3606     {\~expanded{\detokenize{\expandafter#1}\~expanded{{#2}{#3}}}}}%
3607 \def\XINT:NE:x:one:from:twoandone #1#2#3{\XINT:NE:x:one:from:twoandone_a #2#3&&A#1{#2}{#3}}%
3608 \def\XINT:NE:x:one:from:twoandone_a #1#2{\XINT:NE:x:one:from:twoandone_fork #1&&A#2&&A}%
3609 \def\XINT:NE:x:one:from:twoandone_fork #1{%
3610 \def\XINT:NE:x:one:from:twoandone_fork ##1##2&&A##3##4&&A##5##6&&A%
3611 {%
3612     \if0\XINT:NE:hashtilde ##1##3##5~!\relax\XINT:NE:hashash ##1##3##5#1!\relax 0%
3613     \else
3614         \expandafter\XINT:NE:x:one:from:two:p
3615     \fi
3616 }}\expandafter\XINT:NE:x:one:from:twoandone_fork\string#%
3617 \def\XINT:NE:x:listscl #1{%
3618 \def\XINT:NE:x:listscl ##1##2&%
3619 {%
3620     \if0\expandafter\XINT:NE:hashtilde\detokenize{##2}~!\relax
3621         \expandafter\XINT:NE:hashash\detokenize{##2}#1!\relax 0%
3622     \else
3623         \expandafter\XINT:NE:x:listscl:p
3624     \fi
3625     ##1##2&%
3626 }}\expandafter\XINT:NE:x:listscl\string#%
3627 \def\XINT:NE:x:listscl:p #1#2&(#3%
3628 {%
3629     \detokenize
3630     {%
3631         \expanded{\expandafter#1\expanded{#2$XINT_expr_tab({#3})\expandafter\empty\empty}}%$
3632     }%
3633 }%
3634 \def\XINT:NE:f:reverse #1{%
3635 \def\XINT:NE:f:reverse ##1^%
3636 {%
3637     \if0\expandafter\XINT:NE:hashtilde\detokenize\expandafter{\xint_gobble_i##1}~!\relax
3638     \expandafter\XINT:NE:hashash\detokenize{##1}#1!\relax 0%

```

```

3639 \else
3640 \expandafter\XINT:NE:f:reverse:p
3641 \fi
3642 ##1^%
3643 }}\expandafter\XINT:NE:f:reverse:string#%
3644 \def\XINT:NE:f:reverse:p #1^#2\xint_bye
3645 {%
3646 \detokenize
3647 {%
3648 \romannumeral0\expandafter\XINT:expr:f:reverse
3649 \expandafter{\expanded\expandafter{\xint_gobble_i#1}}}%
3650 }%
3651 }%
3652 \def\XINT:expr:f:reverse #1%
3653 {%
3654 \XINT_expr_reverse #1^^#1\xint:\xint:\xint:\xint:
3655 \xint:\xint:\xint:\xint:\xint_bye
3656 }%
3657 \def\XINT:NE:f:from:delim:u #1{%
3658 \def\XINT:NE:f:from:delim:u ##1##2^%
3659 {%
3660 \if0\expandafter\XINT:NE:hashtilde\detokenize{##2}~!\relax
3661 \expandafter\XINT:NE:hashash\detokenize{##2}#1!\relax 0%
3662 \expandafter##1%
3663 \else
3664 \xint_afterfi{\XINT:NE:f:from:delim:u:p##1\empty}%
3665 \fi
3666 ##2^%
3667 }}\expandafter\XINT:NE:f:from:delim:u:string#%
3668 \def\XINT:NE:f:from:delim:u:p #1#2^%
3669 {\detokenize{\expandafter#1}~expanded{#2}$XINT_expr_caret}%$
3670 \def\XINT:NE:f:noeval:from:braced:u #1{%
3671 \def\XINT:NE:f:noeval:from:braced:u ##1##2^%
3672 {%
3673 \if0\XINT:NE:hashtilde ##2~!\relax\XINT:NE:hashash ##2#1!\relax 0%
3674 \else
3675 \expandafter\XINT:NE:f:noeval:from:braced:u:p
3676 \fi
3677 ##1{##2}%
3678 }}\expandafter\XINT:NE:f:noeval:from:braced:u:string#%
3679 \def\XINT:NE:f:noeval:from:braced:u:p #1#2%
3680 {\detokenize{\expandafter#1}~expanded{{#2}}}%
3681 \catcode`- 11
3682 \def\XINT:NE:exec_? #1#2%
3683 {%
3684 \XINT:NE:exec_?_b #2&&A#1{#2}%
3685 }%
3686 \def\XINT:NE:exec_?_b #1{%
3687 \def\XINT:NE:exec_?_b ##1&&A%
3688 {%
3689 \if0\XINT:NE:hashtilde ##1~!\relax
3690 \XINT:NE:hashash ##1#1!\relax 0%

```

```

3691 \xint_dothis\XINT:NE:exec_?:x\fi
3692 \xint_orthat\XINT:NE:exec_?:p
3693 }}\expandafter\XINT:NE:exec_?_b\string%%
3694 \def\XINT:NE:exec_?:x #1#2#3%
3695 {%
3696 \expandafter\XINT_expr_check-_after?\expandafter#1%
3697 \romannumeral`&&\expandafter\XINT_expr_getnext\romannumeral0\xintiiifnotzero#3%
3698 }%
3699 \def\XINT:NE:exec_?:p #1#2#3#4#5%
3700 {%
3701 \csname XINT_expr_func_*If\expandafter\endcsname
3702 \romannumeral`&&@#2\XINTfstop.{#3},{#4},{#5})%
3703 }%
3704 \expandafter\def\csname XINT_expr_func_*If\endcsname #1#2#3%
3705 {%
3706 #1#2{\expanded{\~xintiiifNotZero#3}}}%
3707 }%
3708 \def\XINT:NE:exec_?? #1#2#3%
3709 {%
3710 \XINT:NE:exec_??_b #2&&A#1{#2}%
3711 }%
3712 \def\XINT:NE:exec_??_b #1{%
3713 \def\XINT:NE:exec_??_b ##1&&A%
3714 {%
3715 \if0\XINT:NE:hastilde ##1~!\relax
3716 \XINT:NE:hashash ##1#1!\relax 0%
3717 \xint_dothis\XINT:NE:exec_??:x\fi
3718 \xint_orthat\XINT:NE:exec_??:p
3719 }}\expandafter\XINT:NE:exec_??_b\string%%
3720 \def\XINT:NE:exec_??:x #1#2#3%
3721 {%
3722 \expandafter\XINT_expr_check-_after?\expandafter#1%
3723 \romannumeral`&&\expandafter\XINT_expr_getnext\romannumeral0\xintiiifsgn#3%
3724 }%
3725 \def\XINT:NE:exec_??:p #1#2#3#4#5#6%
3726 {%
3727 \csname XINT_expr_func_*IfSgn\expandafter\endcsname
3728 \romannumeral`&&@#2\XINTfstop.{#3},{#4},{#5},{#6})%
3729 }%
3730 \expandafter\def\csname XINT_expr_func_*IfSgn\endcsname #1#2#3%
3731 {%
3732 #1#2{\expanded{\~xintiiifSgn#3}}}%
3733 }%
3734 \catcode`- 12
3735 \def\XINT:NE:branch #1%
3736 {%
3737 \if0\XINT:NE:hastilde #1~!\relax 0\else
3738 \xint_dothis\XINT:NE:branch_a\fi
3739 \xint_orthat\XINT:NE:branch_b #1&&A%
3740 }%
3741 \def\XINT:NE:branch_a\romannumeral`&&@#1#2&&A%
3742 {%

```

```

3743 \expandafter{\detokenize{\expandafter#1\expanded}{#2}}%
3744 }%
3745 \def\XINT:NE:branch_b#1{%
3746 \def\XINT:NE:branch_b\romannumeral`&&@##1##2##3&&A%
3747 {%
3748 \expandafter{\romannumeral`&&@%
3749 \if0\XINT:NE:hashtilde ##2~!\relax
3750 \XINT:NE:hashash ##2#1!\relax 0\else
3751 \expandafter\string\fi
3752 ##1{##2}##3}%
3753 }}\expandafter\XINT:NE:branch_b\string#%
3754 \def\XINT:NE:seqx#1{%
3755 \def\XINT:NE:seqx\XINT_allexpr_seqx##1##2%
3756 {%
3757 \if 0\expandafter\XINT:NE:hashtilde\detokenize{##2}~!\relax
3758 \expandafter\XINT:NE:hashash \detokenize{##2}#1!\relax 0%
3759 \else
3760 \expandafter\XINT:NE:seqx:p
3761 \fi \XINT_allexpr_seqx{##1}{##2}%
3762 }}\expandafter\XINT:NE:seqx\string#%
3763 \def\XINT:NE:seqx:p\XINT_allexpr_seqx #1#2#3#4%
3764 {%
3765 \expandafter\XINT_expr_put_op_first
3766 \expanded {%
3767 {%
3768 \detokenize
3769 {%
3770 \expanded\bgroup
3771 \expanded
3772 {\unexpanded{\XINT_expr_seq:_b{#1#4\relax $XINT_expr_exclam #3}}}%
3773 #2$XINT_expr_caret}%
3774 }%
3775 }%
3776 \expandafter}\romannumeral`&&\XINT_expr_getop
3777 }%
3778 \def\XINT:NE:opx#1{%
3779 \def\XINT:NE:opx\XINT_allexpr_opx ##1##2##3##4##5##6##7##8%
3780 {%
3781 \if 0\expandafter\XINT:NE:hashtilde\detokenize{##4}~!\relax
3782 \expandafter\XINT:NE:hashash \detokenize{##4}#1!\relax 0%
3783 \else
3784 \expandafter\XINT:NE:opx:p
3785 \fi \XINT_allexpr_opx ##1{##2}{##3}{##4}% en fait ##2 = \xint_c_, ##3 = \relax
3786 }}\expandafter\XINT:NE:opx\string#%
3787 \def\XINT:NE:opx:p\XINT_allexpr_opx #1#2#3#4#5#6#7#8%
3788 {%
3789 \expandafter\XINT_expr_put_op_first
3790 \expanded {%
3791 {%
3792 \detokenize
3793 {%
3794 \expanded\bgroup

```

```

3795         \expanded{\unexpanded{\XINT_expr_iter:_b
3796             {#1\expandafter\XINT_allexpr_opx_ifnotomitted
3797                 \romannumeral0#1#6\relax#7@\relax $XINT_expr_exclam #5}}}%
3798             #4$XINT_expr_caret$XINT_expr_tilde{{#8}}}%$
3799     }%
3800 }%
3801 \expandafter\romannumeral`&&\XINT_expr_getop
3802 }%
3803 \def\XINT:NE:iter{\expandafter\XINT:NE:itery\expandafter}%
3804 \def\XINT:NE:itery#1{%
3805 \def\XINT:NE:itery\XINT_expr_itery##1##2%
3806 {%
3807     \if 0\expandafter\XINT:NE:hasilde\detokenize{##1##2}~!\relax
3808         \expandafter\XINT:NE:hashash \detokenize{##1##2}#1!\relax 0%
3809     \else
3810         \expandafter\XINT:NE:itery:p
3811     \fi \XINT_expr_itery{##1}{##2}%
3812 }}\expandafter\XINT:NE:itery\string#%
3813 \def\XINT:NE:itery:p\XINT_expr_itery #1#2#3#4#5%
3814 {%
3815     \expandafter\XINT_expr_put_op_first
3816     \expanded {%
3817         {%
3818             \detokenize
3819             {%
3820                 \expanded\bgroup
3821                 \expanded{\unexpanded{\XINT_expr_iter:_b {#5#4\relax $XINT_expr_exclam #3}}}%
3822                 #1$XINT_expr_caret$XINT_expr_tilde{#2}}}%$
3823             }%
3824         }%
3825     \expandafter\romannumeral`&&\XINT_expr_getop
3826 }%
3827 \def\XINT:NE:rseq{\expandafter\XINT:NE:rseqy\expandafter}%
3828 \def\XINT:NE:rseqy#1{%
3829 \def\XINT:NE:rseqy\XINT_expr_rseqy##1##2%
3830 {%
3831     \if 0\expandafter\XINT:NE:hasilde\detokenize{##1##2}~!\relax
3832         \expandafter\XINT:NE:hashash \detokenize{##1##2}#1!\relax 0%
3833     \else
3834         \expandafter\XINT:NE:rseqy:p
3835     \fi \XINT_expr_rseqy{##1}{##2}%
3836 }}\expandafter\XINT:NE:rseqy\string#%
3837 \def\XINT:NE:rseqy:p\XINT_expr_rseqy #1#2#3#4#5%
3838 {%
3839     \expandafter\XINT_expr_put_op_first
3840     \expanded {%
3841         {%
3842             \detokenize
3843             {%
3844                 \expanded\bgroup
3845                 \expanded{#2\unexpanded{\XINT_expr_rseq:_b {#5#4\relax $XINT_expr_exclam #3}}}%
3846                 #1$XINT_expr_caret$XINT_expr_tilde{#2}}}%$

```

```

3847     }%
3848 }%
3849 \expandafter}\romannumeral`&&@\XINT_expr_getop
3850 }%
3851 \def\XINT:NE:iterr{\expandafter\XINT:NE:iterr\expandafter}%
3852 \def\XINT:NE:iterr#1{%
3853 \def\XINT:NE:iterr\XINT_expr_iterr##1##2%
3854 {%
3855     \if 0\expandafter\XINT:NE:hasilde\detokenize{##1##2}~!\relax
3856     \expandafter\XINT:NE:hashash \detokenize{##1##2}#1!\relax 0%
3857     \else
3858     \expandafter\XINT:NE:iterr:p
3859     \fi \XINT_expr_iterr{##1}{##2}%
3860 }}\expandafter\XINT:NE:iterr\string#%
3861 \def\XINT:NE:iterr:p\XINT_expr_iterr #1#2#3#4#5%
3862 {%
3863     \expandafter\XINT_expr_put_op_first
3864     \expanded {%
3865     {%
3866         \detokenize
3867         {%
3868             \expanded\bgroup
3869             \expanded{\unexpanded{\XINT_expr_iterr:_b {#5#4\relax $XINT_expr_exclam #3}}%
3870                 #1$XINT_expr_caret$XINT_expr_tilde #20$XINT_expr_qmark}%
3871         }%
3872     }%
3873     \expandafter}\romannumeral`&&@\XINT_expr_getop
3874 }%
3875 \def\XINT:NE:rrseq{\expandafter\XINT:NE:rrseq\expandafter}%
3876 \def\XINT:NE:rrseq#1{%
3877 \def\XINT:NE:rrseq\XINT_expr_rrseq##1##2%
3878 {%
3879     \if 0\expandafter\XINT:NE:hasilde\detokenize{##1##2}~!\relax
3880     \expandafter\XINT:NE:hashash \detokenize{##1##2}#1!\relax 0%
3881     \else
3882     \expandafter\XINT:NE:rrseq:p
3883     \fi \XINT_expr_rrseq{##1}{##2}%
3884 }}\expandafter\XINT:NE:rrseq\string#%
3885 \def\XINT:NE:rrseq:p\XINT_expr_rrseq #1#2#3#4#5#6%
3886 {%
3887     \expandafter\XINT_expr_put_op_first
3888     \expanded {%
3889     {%
3890         \detokenize
3891         {%
3892             \expanded\bgroup
3893             \expanded{#2\unexpanded{\XINT_expr_rrseq:_b {#6#5\relax $XINT_expr_exclam #4}}%
3894                 #1$XINT_expr_caret$XINT_expr_tilde #30$XINT_expr_qmark}%
3895         }%
3896     }%
3897     \expandafter}\romannumeral`&&@\XINT_expr_getop
3898 }%

```

```

3899 \def\XINT:NE:x:toblist#1{%
3900 \def\XINT:NE:x:toblist\XINT:expr:toblistwith##1##2%
3901 {%
3902     \if 0\expandafter\XINT:NE:hasilde\detokenize{##2}~!\relax
3903         \expandafter\XINT:NE:hashash \detokenize{##2}#1!\relax 0%
3904     \else
3905         \expandafter\XINT:NE:x:toblist:p
3906     \fi \XINT:expr:toblistwith{##1}{##2}%
3907 }}\expandafter\XINT:NE:x:toblist\string#%
3908 \def\XINT:NE:x:toblist:p\XINT:expr:toblistwith #1#2{{\XINTfstop.{#2}}}%
3909 \def\XINT:NE:x:mapwithin#1{%
3910 \def\XINT:NE:x:mapwithin\XINT:expr:mapwithin ##1##2%
3911 {%
3912     \if 0\expandafter\XINT:NE:hasilde\detokenize{##2}~!\relax
3913         \expandafter\XINT:NE:hashash \detokenize{##2}#1!\relax 0%
3914     \else
3915         \expandafter\XINT:NE:x:mapwithin:p
3916     \fi \XINT:expr:mapwithin {##1}{##2}%
3917 }}\expandafter\XINT:NE:x:mapwithin\string#%
3918 \def\XINT:NE:x:mapwithin:p \XINT:expr:mapwithin #1#2%
3919 {%
3920     {%
3921         \detokenize
3922         {%
3923             \expanded
3924             {%
3925                 \expandafter\XINT:expr:mapwithin_checkempty
3926                 \expanded{\noexpand#1$XINT_expr_exclam\expandafter}%$
3927                 \detokenize\expandafter{\expanded{#2}}$XINT_expr_caret%$
3928             }%
3929         }%
3930     }%
3931 }%
3932 \def\XINT:NE:x:ndmapx#1{%
3933 \def\XINT:NE:x:ndmapx\XINT_allexpr_ndmapx_a ##1##2^%
3934 {%
3935     \if 0\expandafter\XINT:NE:hasilde\detokenize{##2}~!\relax
3936         \expandafter\XINT:NE:hashash \detokenize{##2}#1!\relax 0%
3937     \else
3938         \expandafter\XINT:NE:x:ndmapx:p
3939     \fi \XINT_allexpr_ndmapx_a ##1##2^%
3940 }}\expandafter\XINT:NE:x:ndmapx\string#%
3941 \def\XINT:NE:x:ndmapx:p #1#2#3^{\relax
3942 {%
3943     \detokenize
3944     {%
3945         \expanded{%
3946             \expandafter#1\expandafter#2\expanded{#3}$XINT_expr_caret\relax %$
3947         }%
3948     }%
3949 }%

```

Attention here that user function names may contain digits, so we don't use a \detokenize or ~

approach.

This syntax means that a function defined by \xintdeffunc never expands when used in another definition, so it can implement recursive definitions.

\XINT:NE:userfunc et al. added at 1.3e.

I added at \xintdeffunc, \xintdefiifunc, \xintdeffloatfunc at 1.3e to on the contrary expand if possible (i.e. if used only with numeric arguments) in another definition.

The \XINTusefunc uses \expanded. Its ancestor \xintExpandArgs (xinttools 1.3) had some more primitive f-expansion technique.

```

3950 \def\xintusenoargfunc #1%
3951 {%
3952     0\csname #1\endcsname
3953 }%
3954 \def\xint:NE:usenoargfunc\csname #1\endcsname
3955 {%
3956     ~romannumeral~\xintusenoargfunc{#1}%
3957 }%
3958 \def\xintusefunc #1%
3959 {%
3960     0\csname #1\expandafter\endcsname\expanded
3961 }%
3962 \def\xint:NE:usefunc #1#2#3%
3963 {%
3964     ~romannumeral~\xintusefunc{#1}{#3}\iffalse{\fi}%
3965 }%
3966 \def\xintusefunc #1%
3967 {%
3968     \expanded\expandafter\xint:expr:mapwithin\csname #1\expandafter\endcsname\expanded
3969 }%
3970 \def\xint:NE:usefunc #1#2#3%
3971 {%
3972     {{~\expanded~\xintusefunc{#1}{#3}}}%
3973 }%
3974 \def\xint:NE:userfunc #1{%
3975 \def\xint:NE:userfunc ##1##2##3%
3976 {%
3977     \if0\expandafter\xint:NE:hasilde\detokenize{##3}~!\relax
3978         \expandafter\xint:NE:hashash\detokenize{##3}#1!\relax 0%
3979         \expandafter\xint:NE:userfunc_x
3980     \else
3981         \expandafter\xint:NE:usefunc
3982     \fi {##1}{##2}{##3}%
3983 }}\expandafter\xint:NE:userfunc\string#%
3984 \def\xint:NE:userfunc_x #1#2#3{#2#3\iffalse{\fi}}%
3985 \def\xint:NE:userufunc #1{%
3986 \def\xint:NE:userufunc ##1##2##3%
3987 {%
3988     \if0\expandafter\xint:NE:hasilde\detokenize{##3}~!\relax
3989         \expandafter\xint:NE:hashash\detokenize{##3}#1!\relax 0%
3990         \expandafter\xint:NE:userufunc_x
3991     \else
3992         \expandafter\xint:NE:useufunc
3993     \fi {##1}{##2}{##3}%

```



```

3994 }}\expandafter\XINT:NE:userufunc\string#%
3995 \def\XINT:NE:userufunc_x #1{\XINT:expr:mapwithin}%
3996 \def\XINT:NE:macrofunc #1#2%
3997   {\expandafter\XINT:NE:macrofunc:a\string#1#2\empty&}%
3998 \def\XINT:NE:macrofunc:a#1\csname #2\endcsname#3&%
3999   {{\XINTusemacrofunc{#1}{#2}{#3}}}%
4000 \def\XINTusemacrofunc #1#2#3%
4001 {%
4002   \romannumeral0\expandafter\xint_stop_atfirstofone
4003   \romannumeral0#1\csname #2\endcsname#3\relax
4004 }%

```

11.29.6 \XINT_expr_redefinmacros

Completely refactored at 1.3.

Again refactored at 1.4. The availability of \expanded allows more powerful mechanisms and more importantly I better thought out the root problems caused by the handling of list operations in this context and this helped simplify considerably the code.

```

4005 \catcode`- 11
4006 \def\XINT_expr_redefinmacros {%
4007   \let\XINT:NEhook:unpack          \XINT:NE:unpack
4008   \let\XINT:NEhook:f:one:from:one   \XINT:NE:f:one:from:one
4009   \let\XINT:NEhook:f:one:from:one:direct \XINT:NE:f:one:from:one:direct
4010   \let\XINT:NEhook:f:one:from:two    \XINT:NE:f:one:from:two
4011   \let\XINT:NEhook:f:one:from:two:direct \XINT:NE:f:one:from:two:direct
4012   \let\XINT:NEhook:x:one:from:two    \XINT:NE:x:one:from:two
4013   \let\XINT:NEhook:x:one:from:twoandone \XINT:NE:x:one:from:twoandone
4014   \let\XINT:NEhook:f:one:and:opt:direct \XINT:NE:f:one:and:opt:direct
4015   \let\XINT:NEhook:f:tacitzeroifone:direct \XINT:NE:f:tacitzeroifone:direct
4016   \let\XINT:NEhook:f:iitacitzeroifone:direct \XINT:NE:f:iitacitzeroifone:direct
4017   \let\XINT:NEhook:x:listscl        \XINT:NE:x:listscl
4018   \let\XINT:NEhook:f:reverse        \XINT:NE:f:reverse
4019   \let\XINT:NEhook:f:from:delim:u   \XINT:NE:f:from:delim:u
4020   \let\XINT:NEhook:f:noeval:from:braced:u \XINT:NE:f:noeval:from:braced:u
4021   \let\XINT:NEhook:branch           \XINT:NE:branch
4022   \let\XINT:NEhook:seqx             \XINT:NE:seqx
4023   \let\XINT:NEhook:opx              \XINT:NE:opx
4024   \let\XINT:NEhook:rseq             \XINT:NE:rseq
4025   \let\XINT:NEhook:iter             \XINT:NE:iter
4026   \let\XINT:NEhook:rrseq            \XINT:NE:rrseq
4027   \let\XINT:NEhook:iterr            \XINT:NE:iterr
4028   \let\XINT:NEhook:x:toblist        \XINT:NE:x:toblist
4029   \let\XINT:NEhook:x:mapwithin      \XINT:NE:x:mapwithin
4030   \let\XINT:NEhook:x:ndmapx         \XINT:NE:x:ndmapx
4031   \let\XINT:NEhook:userfunc         \XINT:NE:userfunc
4032   \let\XINT:NEhook:userufunc        \XINT:NE:userufunc
4033   \let\XINT:NEhook:usernoargfunc    \XINT:NE:usernoargfunc
4034   \let\XINT:NEhook:macrofunc        \XINT:NE:macrofunc
4035   \def\XINTinRandomFloatSdigits{\XINTinRandomFloatSdigits }%
4036   \def\XINTinRandomFloatSixteen{\XINTinRandomFloatSixteen }%
4037   \def\xintiiRandRange{\xintiiRandRange }%
4038   \def\xintiiRandRangeAtoB{\xintiiRandRangeAtoB }%

```

```

4039 \def\xintRandBit{~xintRandBit}%
4040 \let\xINT_expr_exec_? \XINT:NE:exec_?
4041 \let\xINT_expr_exec_?? \XINT:NE:exec_??
4042 \def\xINT_expr_op_? {\XINT_expr_op_?{\XINT_expr_op_-xii\xINT_expr_oparen}}%
4043 \def\xINT_flexpr_op_?{\XINT_expr_op_?{\XINT_flexpr_op_-xii\xINT_flexpr_oparen}}%
4044 \def\xINT_iexpr_op_?{\XINT_expr_op_?{\XINT_iexpr_op_-xii\xINT_iexpr_oparen}}%
4045}%
4046\catcode`- 12

```

11.29.7 \xintNewExpr, \xintNewIExpr, \xintNewFloatExpr, \xintNewIIExpr

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.

As \XINT_NewExpr always execute \XINT_expr_redefineprints since 1.3e whether with \xintNewExpr or \XINT_NewFunc, it has been moved from argument to hardcoded in replacement text.

NO MORE \XINT_expr_redefineprints at 1.4 ! This allows better support for \xinteval, \xinttheexpr as sub-entities inside an \xintNewExpr. And the «cleaning» will remove the new \XINTfstop (detokenized from \meaning output), to maintain backwards compatibility with former behaviour that created macros expand to explicit digits and not an encapsulated result.

The #2#3 in clean stands for \noexpand\xINTfstop (where the actual scantoken-ized input uses \$ originally with catcode letter as the escape character).

```

4047 \def\xintNewExpr {\XINT_NewExpr\xint_firstofone\xintexpr \XINT_newexpr_clean}%
4048 \def\xintNewFloatExpr{\XINT_NewExpr\xint_firstofone\xintfloatexpr\xINT_newexpr_clean}%
4049 \def\xintNewIExpr {\XINT_NewExpr\xint_firstofone\xintiexpr \XINT_newexpr_clean}%
4050 \def\xintNewIIExpr {\XINT_NewExpr\xint_firstofone\xintiiexpr \XINT_newexpr_clean}%
4051 \def\xintNewBoolExpr {\XINT_NewExpr\xint_firstofone\xintboolexpr \XINT_newexpr_clean}%
4052 \def\xINT_newexpr_clean #1>#2#3{\noexpand\expanded\noexpand\xintNEprinthead}%
4053 \def\xintNEprinthead#1.#2{\expanded{\unexpanded{#1.}{#2}}}%

```

1.2c for \xintdeffunc, \xintdefiifunc, \xintdeffloatfunc.

At 1.3, NewFunc does not use anymore a comma delimited pattern for the arguments to the macro being defined.

At 1.4 we use \xintthebareeval, whose meaning now does not mean unlock from csname but firstofone to remove a level of braces This is involved in functioning of expr:userfunc and expr:userefunc

```

4054 \def\xINT_NewFunc {\XINT_NewExpr\xint_gobble_i\xintthebareeval\xINT_newfunc_clean}%
4055 \def\xINT_NewFloatFunc{\XINT_NewExpr\xint_gobble_i\xintthebarefloateval\xINT_newfunc_clean}%
4056 \def\xINT_NewIIFunc {\XINT_NewExpr\xint_gobble_i\xintthebareiieval\xINT_newfunc_clean}%
4057 \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.

```

4058 \def\xINT_NewExpr #1#2#3#4#5[#6]%
4059 {\%
4060 \begingroup
4061 \ifcase #6\relax
4062 \toks0 {\endgroup\xINT_global\def#4}%
4063 \or \toks0 {\endgroup\xINT_global\def#4##1}%
4064 \or \toks0 {\endgroup\xINT_global\def#4##1##2}%
4065 \or \toks0 {\endgroup\xINT_global\def#4##1##2##3}%

```

```

4066 \or \toks0 {\endgroup\XINT_global\def#4##1##2##3##4}%
4067 \or \toks0 {\endgroup\XINT_global\def#4##1##2##3##4##5}%
4068 \or \toks0 {\endgroup\XINT_global\def#4##1##2##3##4##5##6}%
4069 \or \toks0 {\endgroup\XINT_global\def#4##1##2##3##4##5##6##7}%
4070 \or \toks0 {\endgroup\XINT_global\def#4##1##2##3##4##5##6##7##8}%
4071 \or \toks0 {\endgroup\XINT_global\def#4##1##2##3##4##5##6##7##8##9}%
4072 \fi
4073 #1\xintexprSafeCatcodes
4074 \XINT_expr_redefinmacros
4075 \XINT_NewExpr_a #1#2#3#4%
4076 }%

```

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`.

Attention that at 1.4, there might be entire sub-xintexpressions embedded in detokenized form. They are re-tokenized and the main thing is that the parser should not mis-interpret catcode 11 characters as starting variable names. As some macros use `:` in their names, the retokenization must be done with `:` having catcode 11. To not break embedded non-evaluated sub-expressions, the `\XINT_expr_getop` was extended to intercept the `:` (alternative would have been to never inject any macro with `:` in its name... too late now). On the other hand the `!` is not used in the macro names potentially kept as is non expanded by the `\xintNewExpr/\xintdeffunc` process; it can thus be retokenized with catcode 12. But the «hooks» of `seq()`, `iter()`, etc... if deciding they can't evaluate immediately will inject a full sub-expression (possibly arbitrarily complicated) and append to it for its delayed expansion a catcode 11 `!` character (as well as possibly catcode 3 `~` and `?` and catcode 11 caret `^` and even catcode 7 `&`). The macros `\XINT_expr_tilde` etc... below serve for this injection (there are `*two*` successive `\scantokens` using different catcode regimes and these macros remain detokenized during the first pass!) and as consequence the final meaning may have characters such as `!` or and special catcodes depending on where they are located. It may thus not be possible to (easily) retokenize the meaning as printed in the log file if `\xintverbosetrue` was issued.

If a defined function is used in another expression it would thus break things if its meaning was included pre-expanded ; a mechanism exists which keeps only the name of the macro associated to the function (this name may contain digits by the way), when the macro can not be immediately fully expanded. Thus its meaning (with its possibly funny catcodes) is not exposed. And this gives opportunity to pre-expand its arguments before actually expanding the macro.

```

4077 \catcode`~ 3 \catcode`? 3
4078 \def\XINT_expr_tilde{~}\def\XINT_expr_qmark{?}% catcode 3
4079 \def\XINT_expr_caret{^}\def\XINT_expr_exclam{!}% catcode 11
4080 \def\XINT_expr_tab{&}% catcode 7
4081 \catcode`~ 13 \catcode`@ 14 \catcode`\% 6 \catcode`# 12 \catcode`$ 11 @ $
4082 \def\XINT_NewExpr_a %1%2%3%4%5@
4083 {@
4084 \def\XINT_tmpa %1%2%3%4%5%6%7%8%9{%5}%
4085 \def~{$noexpand$}%
4086 \catcode`: 11 \catcode`_ 11 \catcode`\@ 11
4087 \catcode`# 12 \catcode`~ 13 \escapechar 126
4088 \endlinechar -1 \everyeof {\noexpand }%
4089 \edef\XINT_tmpb
4090 {\scantokens\expandafter{\romannumeral`&&\expandafter
4091 %2\XINT_tmpa{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}\relax}%

```

```

4092     }@
4093     \escapechar 92 \catcode`# 6 \catcode`$ 0 @ $
4094     \edef\XINT_tmpa {%1%2%3%4%5%6%7%8%9@
4095         {\scantokens\expandafter{\expandafter%3\meaning\XINT_tmpb}}@
4096     \the\toks0\expandafter
4097         {\XINT_tmpa{%1}{%2}{%3}{%4}{%5}{%6}{%7}{%8}{%9}}@
4098     %1{\ifxintverbose
4099         \xintMessage{xintexpr}{Info}@
4100         {\string%4\space now with @
4101         \ifxintglobaldefs global \fi meaning \meaning%4}@
4102     \fi}@
4103 }@
4104 \catcode`% 14
4105 \XINT_setcatcodes % clean up to avoid surprises if something changes

```

11.29.8 \ifxintexprsafecatcodes, \xintexprSafeCatcodes, \xintexprRestoreCatcodes

1.3c (2018/06/17).

Added \ifxintexprsafecatcodes to allow nesting

```

4106 \newif\ifxintexprsafecatcodes
4107 \let\xintexprRestoreCatcodes\empty
4108 \def\xintexprSafeCatcodes
4109 {%
4110     \unless\ifxintexprsafecatcodes
4111         \edef\xintexprRestoreCatcodes {%
4112             \catcode59=\the\catcode59 % ;
4113             \catcode34=\the\catcode34 % "
4114             \catcode63=\the\catcode63 % ?
4115             \catcode124=\the\catcode124 % |
4116             \catcode38=\the\catcode38 % &
4117             \catcode33=\the\catcode33 % !
4118             \catcode93=\the\catcode93 % ]
4119             \catcode91=\the\catcode91 % [
4120             \catcode94=\the\catcode94 % ^
4121             \catcode95=\the\catcode95 % _
4122             \catcode47=\the\catcode47 % /
4123             \catcode41=\the\catcode41 % )
4124             \catcode40=\the\catcode40 % (
4125             \catcode42=\the\catcode42 % *
4126             \catcode43=\the\catcode43 % +
4127             \catcode62=\the\catcode62 % >
4128             \catcode60=\the\catcode60 % <
4129             \catcode58=\the\catcode58 % :
4130             \catcode46=\the\catcode46 % .
4131             \catcode45=\the\catcode45 % -
4132             \catcode44=\the\catcode44 % ,
4133             \catcode61=\the\catcode61 % =
4134             \catcode96=\the\catcode96 % `
4135             \catcode32=\the\catcode32\relax % space
4136             \noexpand\xintexprsafecatcodesfalse
4137         }%
4138     \fi

```

```

4139 \xintexprsafecatcodestrue
4140     \catcode59=12 % ;
4141     \catcode34=12 % "
4142     \catcode63=12 % ?
4143     \catcode124=12 % |
4144     \catcode38=4  % &
4145     \catcode33=12 % !
4146     \catcode93=12 % ]
4147     \catcode91=12 % [
4148     \catcode94=7  % ^
4149     \catcode95=8  % _
4150     \catcode47=12 % /
4151     \catcode41=12 % )
4152     \catcode40=12 % (
4153     \catcode42=12 % *
4154     \catcode43=12 % +
4155     \catcode62=12 % >
4156     \catcode60=12 % <
4157     \catcode58=12 % :
4158     \catcode46=12 % .
4159     \catcode45=12 % -
4160     \catcode44=12 % ,
4161     \catcode61=12 % =
4162     \catcode96=12 % `
4163     \catcode32=10 % space
4164 }%
4165 \let\XINT_tmpa\undefined \let\XINT_tmpb\undefined \let\XINT_tmpc\undefined
4166 \let\XINT_tmpe\undefined \let\XINT_tmpe\undefined
4167 \ifdefined\RequirePackage\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
4168 {\RequirePackage{xinttrig}}%
4169 \RequirePackage{xintlog}}%
4170 {\input xinttrig.sty
4171 \input xintlog.sty
4172 }%
4173 \XINT_restorecatcodes_endinput%

```

12 Package [xinttrig](#) implementation

Contents

12.1	Catcodes, ϵ -T _E X and reload detection	415
12.2	Library identification	415
12.3	Ensure used letters are dummy letters	416
12.4	<code>\xintreloadxinttrig</code>	416
12.5	Auxiliary variables (only temporarily needed, but left free to re-use)	416
12.5.1	<code>twoPi</code> , <code>threePiover2</code> , <code>Pi</code> , <code>Piover2</code>	416
12.5.2	<code>oneDegree</code> , <code>oneRadian</code>	416
12.5.3	Inverse factorial coefficients: <code>invfact2</code> , ..., <code>invfact44</code>	416
12.6	The sine and cosine series	417
12.6.1	<code>sin_aux()</code> , <code>cos_aux()</code>	417
12.6.2	Make <code>sin_aux()</code> and <code>cos_aux()</code> known to <code>\xintexpr</code>	419
12.6.3	<code>sin_()</code> , <code>cos_()</code>	419
12.7	Range reduction for sine and cosine using degrees	419
12.7.1	Core level macro <code>\XINT_mod_ccclx_i</code>	420
12.7.2	<code>sind_()</code> , <code>cosd_()</code> , and support macros <code>\xintSind</code> , <code>\xintCosd</code>	420
12.8	<code>sind()</code> , <code>cosd()</code>	424
12.9	<code>sin()</code> , <code>cos()</code>	425
12.10	<code>sinc()</code>	425
12.11	<code>tan()</code> , <code>tand()</code> , <code>cot()</code> , <code>cotd()</code>	425
12.12	<code>sec()</code> , <code>secd()</code> , <code>csc()</code> , <code>cscd()</code>	426
12.13	Core routine for inverse trigonometry	426
12.14	<code>asin()</code> , <code>asind()</code>	428
12.15	<code>acos()</code> , <code>acosd()</code>	428
12.16	<code>atan()</code> , <code>atand()</code>	428
12.17	<code>Arg()</code> , <code>atan2()</code> , <code>Argd()</code> , <code>atan2d()</code> , <code>pArg()</code> , <code>pArgd()</code>	429
12.18	Synonyms: <code>tg()</code> , <code>cotg()</code>	430
12.19	Let the functions be known to the <code>\xintexpr</code> parser	431

Comments under reconstruction.

The original was done in January 15 and 16, 2019. It provided `asin()` and `acos()` based on a Newton algorithm approach. Then during March 25-31 I revisited the code, adding more inverse trigonometrical functions (with a modified algorithm, quintically convergent), extending the precision range (so that the package reacts to the `\xintDigits` value at time of load, or reload), and replaced high level range reduction by some optimized lower level coding.

This led me next to improve upon the innards of `\xintdeffunc` and `\xintNewExpr`, and to add to `xintexpr` the `\xintdefefunc` macro (see user documentation).

Finally on April 5, 2019 I pushed further the idea of the algorithm for the arcsine function. The cost is at least the one of a combined `sin()/cos()` evaluation, surely this is not best approach for low precision, but I like the principle and its suitability to go into hundreds of digits if desired.

Almost all of the code remains written at high level, and in particular it is not easily feasible from this interface to execute computations with guard digits. Expect the last one or two digits to be systematically off.

Also, small floating-point inputs are handled quite sub-optimally both for the direct and inverse functions; substantial gains are possible. I added the `ilog10()` function too late to consider using it here with the high level interface.

12.1 Catcodes, ϵ -TeX and reload detection

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2 \catcode13=5 % ^^M
3 \endlinechar=13 %
4 \catcode123=1 % {
5 \catcode125=2 % }
6 \catcode64=11 % @
7 \catcode35=6 % #
8 \catcode44=12 % ,
9 \catcode45=12 % -
10 \catcode46=12 % .
11 \catcode58=12 % :
12 \catcode94=7 % ^
13 \def\z{\endgroup}%
14 \def\empty{}\def\space{ }\newlinechar10
15 \expandafter\let\expandafter\w\csname ver@xintexpr.sty\endcsname
16 \expandafter
17 \ifx\csname PackageInfo\endcsname\relax
18 \def\y#1#2{\immediate\write-1{Package #1 Info:^^J%
19 \space\space\space\space#2.}}%
20 \else
21 \def\y#1#2{\PackageInfo{#1}{#2}}%
22 \fi
23 \expandafter
24 \ifx\csname numexpr\endcsname\relax
25 \y{xinttrig}{numexpr not available, aborting input}%
26 \aftergroup\endinput
27 \else
28 \ifx\w\relax % xintexpr.sty not yet loaded.
29 \y{xinttrig}%
30 {Loading should be via \ifx\x\empty\string\usepackage{xintexpr.sty}
31 \else\string\input\space xintexpr.sty \fi
32 rather, aborting}%
33 \aftergroup\endinput
34 \fi
35 \fi
36 \z%
37 \catcode`_ 11 \XINT_setcatcodes \catcode`? 12

```

12.2 Library identification

```

38 \ifcsname xintlibver@trig\endcsname
39 \expandafter\xint_firstoftwo
40 \else
41 \expandafter\xint_secondoftwo
42 \fi
43 {\immediate\write-1{Reloading xinttrig library using Digits=\xinttheDigits.}}%
44 {\expandafter\gdef\csname xintlibver@trig\endcsname{2020/02/19 v1.4a}%
45 \XINT_providespackage
46 \ProvidesPackage{xinttrig}%
47 [2020/02/19 v1.4a Trigonometrical functions for xintexpr (JFB)]%
48 }%

```

12.3 Ensure used letters are dummy letters

```
49 \xintFor* #1 in {iDTVtuwxyzX}\do{\xintensuredummy{#1}}%
```

12.4 \xintreloadxinttrig

```
50 \def\xintreloadxinttrig
51   {\edef\XINT_restorecatcodes_now{\XINT_restorecatcodes}%
52    \XINT_setcatcodes\catcode`? 12
53    \input xinttrig.sty
54    \XINT_restorecatcodes_now}%
```

12.5 Auxiliary variables (only temporarily needed, but left free to re-use)

These variables don't have really private names but this does not matter because only their actual values will be stored in the functions defined next. Nevertheless they are not unassigned, and are left free to use as is.

12.5.1 twoPi, threePiover2, Pi, Piover2

We take them with 60 digits and force conversion to \xintDigits setting via "0 + " syntax.

```
55 \xintdeffloatvar twoPi      := 0 +
56   6.28318530717958647692528676655900576839433879875021164194989;%
57 \xintdeffloatvar threePiover2 := 0 +
58   4.71238898038468985769396507491925432629575409906265873146242;%
59 \xintdeffloatvar Pi        := 0 +
60   3.14159265358979323846264338327950288419716939937510582097494;%
61 \xintdeffloatvar Piover2   := 0 +
62   1.57079632679489661923132169163975144209858469968755291048747;%
```

12.5.2 oneDegree, oneRadian

```
63 \xintdeffloatvar oneDegree := 0 +
64   0.0174532925199432957692369076848861271344287188854172545609719;% Pi/180
65 \xintdeffloatvar oneRadian := 0 +
66   57.2957795130823208767981548141051703324054724665643215491602;% 180/Pi
```

12.5.3 Inverse factorial coefficients: invfact2, ..., invfact44

Pre-compute $1/n!$ for $n = 2, \dots, 44$

The following example (among many, see below) shows that we must be careful when pre-computing the $1/i!$.

Consider $35!=10333147966386144929666651337523200000000$.

With `\xintDigit:=26; \xintfloateval{35!}` obtains $1.0333147966386144929666651e40$

which is the correct rounding to 26 digits. But `\xintfloateval{1/35!}` obtains $9.6775929586318909920898167e-41$ which differs by 3ulps from the correct rounding of $1/35!$ to 26 places which is $9.6775929586318909920898164e-41$. The problem isn't in the factorial computations, but in the fact that the rounding of the inverse of a quantity which is itself a rounding is not necessarily the rounding of the exact inverse of the original.

Here is a little program to explore this phenomenon systematically:

```
\xintDigits:=55;%
\edef\tempNlist{\xintSeq{2}{39}}
\xintFor*#1in{\tempNlist}\do{% we precompute some rounding here to
```



```
% speed up things in the next double loop.
\expandafter\edef\csname invfact#1\endcsname {\xintfloatexpr 1/#1!\relax}%
}%
\xintFor*#1in{\xintSeq{4}{50}}\do{%
  \xintDigits:=#1;%
  \xintFor*#2in{\tempNlist}\do{%
    (D=#1, N=#2)
    % attention to != which is parsed as negation operator != followed by = (sigh...)
    \xintifboolfloatexpr{(1/#2!)==0+\csname invfact#2\endcsname}%
    {ok}
    {mismatch: \xintfloateval{1/#2!} vs (exact)
      \xintfloateval{0+\csname invfact#2\endcsname}}%
  }
\par
}%
}%
```

We can see that for D=16, the problem is there with N=22, 25, 26, 27, 28...and more. If we were to use 1/i! directly in the \xintdeffloatfunc of sin_aux(X) and cos_aux(X) we would have this problem.

If we use \xintexpr1/i!\relax encapsulation in the function declaration the rounding will be delayed to actual use of the function... which is bad, so we need it to happen now. We could use (0+\xintexpr1/i!\relax) inside the declaration of the sine and cosine series, which will give the expected result but for readability we use some temporary variables. We could use seq(0+\xintexpr1/i!\relax, i = 2..44) but opt for an rseq. The semi-colon must be braced to hide it from \xintdeffloatvar grabbing of the delimited argument.

1.4 update: use \xintfloatexpr with optional argument for the rounding rather than «0+x» method.

```
67 \xintdeffloatvar invfact\xintListWithSep{, invfact}{\xintSeq{2}{44}}%
68   := \xintfloatexpr [\XINTdigits] % force float rounding after exact evaluations
69   \xintexpr rseq(1/2{;};@/i, i=3..44)\relax % need to hide inner ; from \xintdeffloatvar
70   \relax;%
```

12.6 The sine and cosine series

12.6.1 sin_aux(), cos_aux()

Should I rather use successive divisions by (2n+1)(2n), or rather multiplication by their precomputed inverses, in a modified Horner scheme ? The \ifnum tests are executed at time of definition.

Criteria for truncated series using $\pi/4$, actually 0.79.

Small values of the variable X are very badly handled here because a much shorter truncation of the sine series should be used.

```
71 \xintdeffloatfunc sin_aux(X) := 1 - X(invfact3 - X(invfact5
72 \ifnum\XINTdigits>4
73   - X(invfact7
74 \ifnum\XINTdigits>6
75   - X(invfact9
76 \ifnum\XINTdigits>8
77   - X(invfact11
78 \ifnum\XINTdigits>10
79   - X(invfact13
80 \ifnum\XINTdigits>13
81   - X(invfact15
82 \ifnum\XINTdigits>15
83   - X(invfact17
```

[illegible]

Criteria on basis of $\pi/4$, we actually used 0.79 to choose the transition values and this makes them a bit less favourable at 24, 26, 29...and some more probably. Again this is very bad for small X.

```

111 \xintdeffloatfunc cos_aux(X) := 1 - X(invfact2 - X(invfact4
112 \ifnum\XINTdigits>3
113 - X(invfact6
114 \ifnum\XINTdigits>5
115 - X(invfact8
116 \ifnum\XINTdigits>7
117 - X(invfact10
118 \ifnum\XINTdigits>9
119 - X(invfact12
120 \ifnum\XINTdigits>12
121 - X(invfact14
122 \ifnum\XINTdigits>14
123 - X(invfact16
124 \ifnum\XINTdigits>17
125 - X(invfact18
126 \ifnum\XINTdigits>20
127 - X(invfact20
128 \ifnum\XINTdigits>23
129 - X(invfact22
130 \ifnum\XINTdigits>25
131 - X(invfact24

```

[illegible]

12.6.2 Make `sin_aux()` and `cos_aux()` known to `\xintexpr`

We need them shortly for the `asin()` in an `\xintexpr` variant. We short-circuit the high level interface as it will not be needed to add some `\xintFloat` wrapper.

```

153 \expandafter\let\csname XINT_expr_func_sin_aux\expandafter\endcsname
154         \csname XINT_flexpr_func_sin_aux\endcsname
155 \expandafter\let\csname XINT_expr_func_cos_aux\expandafter\endcsname
156         \csname XINT_flexpr_func_cos_aux\endcsname

```

12.6.3 `sin_()`, `cos_()`

Use this only between $-\pi/4$ and $\pi/4$

```
157 \xintdeffloatfunc sin_(x) := x * sin_aux(sqr(x));%
```

Use this only between $-\pi/4$ and $\pi/4$

```
158 \xintdeffloatfunc cos_(x) := cos_aux(sqr(x));%
```

12.7 Range reduction for sine and cosine using degrees

Notice that even when handling radians it is much better to convert to degrees and then do range reduction there, because this can be done in the fixed point sense. I lost 1h puzzled about some mismatch of my results with those of Maple (at 16 digits) near $-\pi$. Turns out that Maple probably adds π in the floating point sense causing catastrophic loss of digits when one is near $-\pi$. On the other hand my $\sin(x)$ function will first convert to degrees then add 180 without any loss of floating point precision, even for a result near zero, then convert back to radians and use the sine series.

12.7.1 Core level macro \XINT_mod_ccclx_i

input: \the\numexpr\XINT_mod_ccclx_i k.N. (delimited by dots)

output: (N times 10^k) modulo 360. (with a final dot)

Attention N must be non-negative (I could make it accept negative but the fact that numexpr / is not periodical in numerator adds overhead).

360 divides 9000 hence 10^k is 280 for k at least 3 and the additive group generated by it modulo 360 is the set of multiples of 40.

```

159 \def\XINT_mod_ccclx_i #1.% input <k>.<N>. k is a non-negative exponent
160 {%
161     \expandafter\XINT_mod_ccclx_e\the\numexpr
162     \expandafter\XINT_mod_ccclx_j\the\numexpr1\ifcase#1 \or0\or00\else000\fi.%
163 }%
164 \def\XINT_mod_ccclx_j 1#1.#2.% #2=N is a non-negative mantissa
165 {%
166     (\XINT_mod_ccclx_ja {++}#2#1\XINT_mod_ccclx_jb 0000000\relax
167 }%
168 \def\XINT_mod_ccclx_ja #1#2#3#4#5#6#7#8#9%
169 {%
170     #9+#8+#7+#6+#5+#4+#3+#2\xint_firstoftwo{+\XINT_mod_ccclx_ja{+#9+#8+#7}}{#1}%
171 }%
172 \def\XINT_mod_ccclx_jb #1\xint_firstoftwo#2#3{#1+0)*280\XINT_mod_ccclx_jc #1#3}%

```

Attention that \XINT_cclcx_e wants non negative input because \numexpr division is not periodical ...

```

173 \def\XINT_mod_ccclx_jc  +#1+#2+#3#4\relax{+80*(#3+#2+#1)+#3#2#1.}%
174 \def\XINT_mod_ccclx_e#1.{\expandafter\XINT_mod_ccclx_z\the\numexpr(#1+180)/360-1.#1.}%
175 \def\XINT_mod_ccclx_z#1.#2.{#2-360*#1.}%

```

12.7.2 sind(), cosd(), and support macros \xintSind, \xintCosd

sind() coded directly at macro level with a macro \xintSind (ATTENTION! it requires a positive argument) which will suitably use \XINT_flexpr_func_sin_ defined from \xintdeffloatfunc

```

176 \def\XINT_flexpr_func_sind_ #1#2#3%
177 {%
178     \expandafter #1\expandafter #2\expandafter{%
179     \romannumeral`&&\XINT:NEhook:f:one:from:one{\romannumeral`&&\xintSind#3}}%
180 }%

```

Must be f-expandable for nesting macros from \xintNewExpr

ATTENTION ONLY FOR POSITIVE ARGUMENTS

```

181 \def\XINT_expr_unlock{\expandafter\xint_firstofone\romannumeral`&&}%
182 \def\xintSind#1{\romannumeral`&&\expandafter\xintsind
183     \romannumeral0\XINTinfloatS[\XINTdigits]{#1}}%
184 \def\xintsind #1[#2#3]%
185 {%
186     \xint_UDsignfork
187     #2\XINT_sind
188     -\XINT_sind_int
189     \krof#2#3.#1..%<< attention extra dot

```

```

190 }%
191 \def\XINT_sind #1.#2.% NOT TO BE USED WITH VANISHING (OR NEGATIVE) #2.
192 {%
193     \expandafter\XINT_sind_a
194     \romannumeral0\xinttrunc{\XINTdigits}{#2[#1]}%
195 }%
196 \def\XINT_sind_a{\expandafter\XINT_sind_i\the\numexpr\XINT_mod_ccclx_i0.}%
197 \def\XINT_sind_int
198 {%
199     \expandafter\XINT_sind_i\the\numexpr\expandafter\XINT_mod_ccclx_i
200 }%
201 \def\XINT_sind_i #1.% range reduction inside [0, 360[
202 {%
203     \ifcase\numexpr#1/90\relax
204         \expandafter\XINT_sind_A
205     \or\expandafter\XINT_sind_B\the\numexpr-90+%
206     \or\expandafter\XINT_sind_C\the\numexpr-180+%
207     \or\expandafter\XINT_sind_D\the\numexpr-270+%
208     \else\expandafter\XINT_sind_E\the\numexpr-360+%
209     \fi#1.%
210 }%

```

#2 will be empty in the "integer branch". Notice that a single dot "." is valid as input to the xintfrac macros. During developing phase I did many silly mistakes due to wanting to use too low-level interface, e.g. I would use something like #2[-\XINTdigits] with #2 the fractional digits, but there maybe some leading zero and then xintfrac.sty will think the whole thing is zero due to the requirements of my own core format A[N]....

Multiplication is done exactly but anyway currently float multiplication goes via exact multiplication after rounding arguments ; as here integer part has at most three digits, doing exact multiplication will prove not only more accurate but probably faster.

```

211 \def\XINT_sind_A#1{%
212 \def\XINT_sind_A##1.##2.%
213 {%
214     \XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_sin_\expandafter
215     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{##1.##2}{#1}}}%
216 }%
217 }\expandafter
218 \XINT_sind_A\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%
219 \def\XINT_sind_B#1{\xint_UDsignfork#1\XINT_sind_B_n-\XINT_sind_B_p\krof #1}%
220 \def\XINT_tmpa#1{%
221 \def\XINT_sind_B_n-##1.##2.%
222 {%
223     \XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_cos_\expandafter
224     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{\xintSub{##1[0]}{.##2}}{#1}}}%
225 }%
226 \def\XINT_sind_B_p##1.##2.%
227 {%
228     \XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_cos_\expandafter
229     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{##1.##2}{#1}}}%
230 }%
231 }\expandafter
232 \XINT_tmpa\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%

```

```

233 \def\XINT_sind_C#1{\xint_UDsignfork#1\XINT_sind_C_n-\XINT_sind_C_p\krof #1}%
234 \def\XINT_tmpa#1{%
235 \def\XINT_sind_C_n-##1.##2.%
236 {%
237     \XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_sin\expandafter
238     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{\xintSub{##1[0]}.{##2}}#1}}%
239 }%
240 \def\XINT_sind_C_p##1.##2.%
241 {%
242     \xintiopp\XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_sin\expandafter
243     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{##1.##2}#1}}%
244 }%
245 }\expandafter
246 \XINT_tmpa\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%
247 \def\XINT_sind_D#1{\xint_UDsignfork#1\XINT_sind_D_n-\XINT_sind_D_p\krof #1}%
248 \def\XINT_tmpa#1{%
249 \def\XINT_sind_D_n-##1.##2.%
250 {%
251     \xintiopp\XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_cos\expandafter
252     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{\xintSub{##1[0]}.{##2}}#1}}%
253 }%
254 \def\XINT_sind_D_p##1.##2.%
255 {%
256     \xintiopp\XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_cos\expandafter
257     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{##1.##2}#1}}%
258 }%
259 }\expandafter
260 \XINT_tmpa\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%
261 \def\XINT_sind_E#1{%
262 \def\XINT_sind_E-##1.##2.%
263 {%
264     \xintiopp\XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_sin\expandafter
265     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{\xintSub{##1[0]}.{##2}}#1}}%
266 }%
267 }\expandafter
268 \XINT_sind_E\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%

```

The cosd_ auxiliary function

```

269 \def\XINT_flexpr_func_cosd_ #1#2#3%
270 {%
271     \expandafter #1\expandafter #2\expandafter{%
272     \romannumeral`&&\XINT:NEhook:f:one:from:one{\romannumeral`&&\xintCosd#3}}%
273 }%

```

ATTENTION ONLY FOR POSITIVE ARGUMENTS

```

274 \def\xintCosd#1{\romannumeral`&&\expandafter\xintcosd
275     \romannumeral0\XINTinfloatS[\XINTdigits]{#1}}%
276 \def\xintcosd #1[#2#3]%
277 {%
278     \xint_UDsignfork
279     #2\XINT_cosd
280     -\XINT_cosd_int

```

```

281 \krof#2#3.#1..%<< attention extra dot
282 }%
283 \def\XINT_cosd #1.#2.% NOT TO BE USED WITH VANISHING (OR NEGATIVE) #2.
284 {%
285 \expandafter\XINT_cosd_a
286 \romannumeral0\xinttrunc{\XINTdigits}{#2[#1]}%
287 }%
288 \def\XINT_cosd_a{\expandafter\XINT_cosd_i\the\numexpr\XINT_mod_ccclx_i0.}%
289 \def\XINT_cosd_int
290 {%
291 \expandafter\XINT_cosd_i\the\numexpr\expandafter\XINT_mod_ccclx_i
292 }%
293 \def\XINT_cosd_i #1.%
294 {%
295 \ifcase\numexpr#1/90\relax
296 \expandafter\XINT_cosd_A
297 \or\expandafter\XINT_cosd_B\the\numexpr-90+%
298 \or\expandafter\XINT_cosd_C\the\numexpr-180+%
299 \or\expandafter\XINT_cosd_D\the\numexpr-270+%
300 \else\expandafter\XINT_cosd_E\the\numexpr-360+%
301 \fi#1.%
302 }%

```

#2 will be empty in the "integer" branch, but attention in general branch to handling of negative integer part after the subtraction of 90, 180, 270, or 360, and avoid abusing A[N] notation which yes speeds up xintfrac parsing but has its pitfalls.

```

303 \def\XINT_cosd_A#1{%
304 \def\XINT_cosd_A##1.##2.%
305 {%
306 \XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_cos\expandafter
307 {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{##1.##2}{#1}}}%
308 }%
309 }\expandafter
310 \XINT_cosd_A\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%
311 \def\XINT_cosd_B#1{\xint_UDsignfork#1\XINT_cosd_B_n-\XINT_cosd_B_p\krof #1}%
312 \def\XINT_tmpa#1{%
313 \def\XINT_cosd_B_n-##1.##2.%
314 {%
315 \XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_sin\expandafter
316 {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{\xintSub{##1[0]}{.##2}}{#1}}}%
317 }%
318 \def\XINT_cosd_B_p##1.##2.%
319 {%
320 \xintiopp\XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_sin\expandafter
321 {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{##1.##2}{#1}}}%
322 }%
323 }\expandafter
324 \XINT_tmpa\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%
325 \def\XINT_cosd_C#1{\xint_UDsignfork#1\XINT_cosd_C_n-\XINT_cosd_C_p\krof #1}%
326 \def\XINT_tmpa#1{%
327 \def\XINT_cosd_C_n-##1.##2.%
328 {%

```

```

329 \xintiopp\XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_cos_\expandafter
330 {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{\xintSub{##1[0]}.{##2}}#1}}%
331 }%
332 \def\XINT_cosp_p##1.##2.%
333 {%
334 \xintiopp\XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_cos_\expandafter
335 {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{##1.##2}#1}}%
336 }%
337 }\expandafter
338 \XINT_tmpa\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%
339 \def\XINT_cosp_D#1{\xint_UDsignfork#1\XINT_cosp_D_n-\XINT_cosp_D_p\krof #1}%
340 \def\XINT_tmpa#1{%
341 \def\XINT_cosp_D_n-##1.##2.%
342 {%
343 \xintiopp\XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_sin_\expandafter
344 {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{\xintSub{##1[0]}.{##2}}#1}}%
345 }%
346 \def\XINT_cosp_D_p##1.##2.%
347 {%
348 \XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_sin_\expandafter
349 {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{##1.##2}#1}}%
350 }%
351 }\expandafter
352 \XINT_tmpa\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%
353 \def\XINT_cosp_E#1{%
354 \def\XINT_cosp_E-##1.##2.%
355 {%
356 \XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_cos_\expandafter
357 {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{\xintSub{##1[0]}.{##2}}#1}}%
358 }%
359 }\expandafter
360 \XINT_cosp_E\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%

```

12.8 sind(), cosd()

```

361 \xintdeffloatfunc sind(x) := (x)??
362 {\x>=-45)?
363 {\sin_(x*oneDegree)}
364 {-sind_(-x)}
365 }
366 {0}
367 {\x<=45)?
368 {\sin_(x*oneDegree)}
369 {\sind_(x)}
370 }
371 ;%
372 \xintdeffloatfunc cosd(x) := (x)??
373 {\x>=-45)?
374 {\cos_(x*oneDegree)}
375 {\cosd_(-x)}
376 }
377 {1}
378 {\x<=45)?

```



```

379             {cos_(x*oneDegree)}
380             {cosd_(x)}
381         }
382     ;%

```

12.9 sin(), cos()

For some reason I did not define sin() and cos() in January 2019 ??

```

383 \xintdeffloatfunc sin(x):= (abs(x)<0.79)?
384     {sin_(x)}
385     {(x)}??
386     {-sind_(-x*oneRadian)}
387     {0}
388     {sind_(x*oneRadian)}
389 }
390 ;%
391 \xintdeffloatfunc cos(x):= (abs(x)<0.79)?
392     {cos_(x)}
393     {cosd_(abs(x*oneRadian))}
394 ;%

```

12.10 sinc()

Should I also consider adding $(1-\cos(x))/(x^2/2)$? it is $\text{sinc}^2(x/2)$ but avoids a square.

```

395 \xintdeffloatfunc sinc(x):= (abs(x)<0.79) ?
396     {sin_aux(sqr(x))}
397     {sind_(abs(x)*oneRadian)/abs(x)}
398 ;%

```

12.11 tan(), tand(), cot(), cotd()

The 0 in cot(x) is a dummy place holder, 1/0 would raise an error at time of definition...

```

399 \xintdeffloatfunc tand(x):= sind(x)/cosd(x);%
400 \xintdeffloatfunc cotd(x):= cosd(x)/sind(x);%
401 \xintdeffloatfunc tan(x) := (x)??
402     {(x>-0.79)?
403         {sin(x)/cos(x)}
404         {-cotd(90+x*oneRadian)}
405     }
406 }
407 {0}
408 {(x<0.79)?
409     {sin(x)/cos(x)}
410     {cotd(90-x*oneRadian)}
411 }
412 ;%
413 \xintdeffloatfunc cot(x) := (abs(x)<0.79)?
414     {cos(x)/sin(x)}
415     {(x)}??
416     {-tand(90+x*oneRadian)}

```

```
{0}
{tand(90-x*oneRadian)}
};%
```

12.12 sec(), secd(), csc(), cscd()

```

420 \xintdeffloatfunc sec(x) := inv(cos(x));%
421 \xintdeffloatfunc csc(x) := inv(sin(x));%
422 \xintdeffloatfunc secd(x) := inv(cosd(x));%
423 \xintdeffloatfunc cscd(x) := inv(sind(x));%

```

12.13 Core routine for inverse trigonometry

Compute $\arcsin(x)$

The approach I shall first describe (which is only a first step towards our final approach) converges quintically but requires an initial square root computation. For `atan(x)`, we do not have to do any such square root extraction. See code next.

The algorithm (for this first approach): we have $0 \leq t < 0.72$, let $t_1 = t \cdot (1 + t^2/6)$. We also have $u = \sqrt{1 - t^2}$. We seek $a = \arcsin t$ with $t = \sin(a)$.

Then $t_1 < \arcsin t$ and the difference (we don't know it!) δ_{-1} is < 0.02 . We compute $D = t \cdot \cos(t_1) - u \cdot \sin(t_1)$. This computation is done "exactly" via the `\xintexpr` encapsulation. In other terms we use doubled precision. Anyhow, currently (1.3e) the Float macros of `xintfrac.sty` for multiplication do go via such exact multiplication when the mantissas have the expected sizes. So we can't gain but only lose due to catastrophic subtraction in using float operations here.

Thus D is $\sin(a-t_1) = \sin(\delta_1)$. And $\delta_1 = \text{Arcsin } D$, but D is small! We then use again two terms of the Arcsin series and define $t_2 = t_1 + D * (1 + D^2/6)$. Let $\delta_2 = a - t_2$. Then δ_2 is of the order of the neglected term $3 * (\delta_1)^5/40$.

©copyright J.F. Burnol, March 30, 2019. This surely has a name.

The algorithm is quintically convergent. One can do the same to go from exp to log. Basically the idea is that we can improve the Newton Method for any function f for which knowing target value of f implies one also knows target value of its derivative. In fact I obtained the quintic algorithm by combining the Newton formula with the one from using $f(x)/f'(a)$ and not $f(x)/f'(x)$ in the update to cancel the two quadratic errors.

One iteration (t2) gives about 9 digits, two iterations (t3) 49 digits ! And if we want hepta-convergence we only need to use one more term of the Arcsin series in the update of the t_n... really this is very nice.

And actually (t2) already gives 30 digits of floating point precision for input $t < 0.1$. Let's confirm this:

[illegible]

Each iteration costs a computation of one cos and one sine done at the full final precision. This is stupid because we should compute at an evolving precision, but anyhow this is not our problem

anymore as our final algorithm is not a loop but it does exactly one iteration for all inputs. As exemplified above it remains true that we could improve its speed for small inputs by using shorter auxiliary series (see below).

In January I used a loop via an `iter()` construct, with some `subs()` to avoid repeating computations. This can only be done in an `\xintNewFunction`. Here is how it looked after some optimization for the stopping criteria, after replacing generic Newton algorithm by a specific quintic one for arcsine:

```
\begingroup
\edef\x{\endgroup
\noexpand\xintNewFunction{asin_1}[2]{
  iter(##1*(1+sqr(##1)/6);
% FIXME : réfléchir au critère d'arrêt.
%
% Je n'utilise pas abs(D) pour un micro-gain est-ce que le risque en vaut la
% chandelle ? (avec abs(D) on pourrait utiliser la fonction avec un #1 négatif)
%
% Am I sure rounding errors could not cause neverending loop?
% Such things should be done with increased precision and rounded at end.
  subs((D<\ifcase\numexpr2+\XINTdigits-5*(\XINTdigits/5)\relax
    3.68\or2.32\or1.47\or0.923\or0.582\fi
    e-\the\numexpr\XINTdigits/5\relax)
    ?{break(@+D*(1+sqr(D)/6))}{@+D*(1+sqr(D)/6)},
    D=\noexpand\xintexpr
      subs(##1*cos_aux(X) - ##2*@*sin_aux(X), X=sqr(@))
    \relax
  ),
  i=1++)dummy iteration index, not used but needed by iter()
}}\x
```

I don't have time to explain the final algorithm below and how the transition values were chosen or why (the series below is enough up to 59 digits of precision). It does only one iteration, in all cases. Using it for arcsine requires a preliminary square root extraction, but for arctangent one arranges things to avoid having to compute a square root.

©copyright J.F. Burnol, April 5, 2019. This surely has a name.

Certainly I can do similar things to compute logarithms.

```
424 \xintdeffloatfunc asin_aux(X) := 1
425 \ifnum\XINTdigits>3 % actually 4 would achieve 1ulp in place of <0.5ulp
426       + X(1/6
427 \ifnum\XINTdigits>9
428       + X(3/40
429 \ifnum\XINTdigits>16
430       + X(5/112
431 \ifnum\XINTdigits>25
432       + X(35/1152
433 \ifnum\XINTdigits>35
434       + X(63/2816
435 \ifnum\XINTdigits>46
436       + X(231/13312
437       )\fi)\fi)\fi)\fi)\fi)\fi;%
438 \xintdeffloatfunc asin_o(D, T) := T + D*asin_aux(sqr(D));%
439 \xintdeffloatfunc asin_n(V, T, t, u) :=% V is square of T
440       asin_o (\xintexpr t*cos_aux(V) - u*T*sin_aux(V)\relax, T);%
441 \xintdeffloatfunc asin_m(T, t, u) := asin_n(sqr(T), T, t, u);%
```

```
442 \xintdeffloatfunc asin_l(t, u)      := asin_m(t*asin_aux(sqr(t)), t, u);%
```

12.14 asin(), asind()

Only non-negative arguments t and u for $\text{asin}_a(t,u)$, and $\text{asind}_a(t,u)$.

At 1.4 usage of $\text{sqrt}_()$ which has only one argument, whereas currently $\text{sqrt}()$ admits a second optional argument hence sub-optimality here if we use $\text{sqrt}()$, especially since 1.4 handles more fully such functions with optional argument in \xintdeffunc .

Actually thinking of making $\text{sqrt}()$ a one argument only function and $\text{sqrt}_()$ will be the one with two arguments. But I worked hard on the \xintdeffunc hooks, thus some reticence, because why then not do that for all others?

```
443 \xintdeffloatfunc asin_a(t, u) := (t<u)?
444                               {asin_l(t, u)}
445                               {Piover2 - asin_l(u, t)}
446                               ;%
447 \xintdeffloatfunc asind_a(t, u) := (t<u)?
448                               {asin_l(t, u) * oneRadian}
449                               {90 - asin_l(u, t) * oneRadian}
450                               ;%
451 \xintdeffloatfunc asin(t) := (t)??
452                               {-asin_a(-t, sqrt_(1-sqr(t)))}
453                               {0}
454                               {asin_a(t, sqrt_(1-sqr(t)))}
455                               ;%
456 \xintdeffloatfunc asind(t) := (t)??
457                               {-asind_a(-t, sqrt_(1-sqr(t)))}
458                               {0}
459                               {asind_a(t, sqrt_(1-sqr(t)))}
460                               ;%
```

12.15 acos(), acosd()

```
461 \xintdeffloatfunc acos(t) := Piover2 - asin(t);%
462 \xintdeffloatfunc acosd(t) := 90 - asind(t);%
```

12.16 atan(), atand()

This involves no square root!

TeX hackers note 1:

The $\text{subs}(, x = ..)$ mechanism has no utility in a function definition, there is no parallel mechanism at the underlying macros, so in fact the substituted things will remain unevaluated if they involve indeterminates, so this is exactly like not trying to make things more efficient at all.

Currently, the only way is thus to employ auxiliary functions like is done next. Contrarily to TeX macros, we must define the functions one after the other in the correct order, so the auxiliaries come first.

TeX hackers note 2:

At 1.4, the way to inject lazy conditionals in function definitions has changed. Prior one used $\text{if}(,,)$ and $\text{ifsgn}(,,)$ which was counter-intuitive because in pure numeric context they evaluate all branches. Now one must use $?$ and $??$ which are the lazy conditionals from the numeric context.

radians

```

463 \xintdeffloatfunc atan_b(t, w, z) := 0.5 * (w < 0)?
464                                     {Pi - asin_a(2z * t, -w*z)}
465                                     {asin_a(2z * t, w*z)}
466                                     ;%
467 \xintdeffloatfunc atan_a(t, T) := atan_b(t, 1-T, inv(1+T));%
468 \xintdeffloatfunc atan(t) := (t)??
469                                     {-atan_a(-t, sqr(t))}
470                                     {0}
471                                     {atan_a(t, sqr(t))}
472                                     ;%

```

degrees

```

473 \xintdeffloatfunc atand_b(t, w, z) := 0.5 * (w < 0)?
474                                     {180 - asind_a(2z * t, -w*z)}
475                                     {asind_a(2z * t, w*z)}
476                                     ;%
477 \xintdeffloatfunc atand_a(t, T) := atand_b(t, 1-T, inv(1+T));%
478 \xintdeffloatfunc atand(t) := (t)??
479                                     {-atand_a(-t, sqr(t))}
480                                     {0}
481                                     {atand_a(t, sqr(t))}
482                                     ;%

```

12.17 Arg(), atan2(), Argd(), atan2d(), pArg(), pArgd()

Arg(x,y) function from $-\pi$ (excluded) to $+\pi$ (included)

```

483 \xintdeffloatfunc Arg(x, y) := (y > x)?
484                                     {(y > -x)?
485                                     {Piover2 - atan(x/y)}
486                                     {(y < 0)?
487                                     {-Pi + atan(y/x)}
488                                     {Pi + atan(y/x)}
489                                     }
490                                     }
491                                     {(y > -x)?
492                                     {atan(y/x)}
493                                     {-Piover2 + atan(x/-y)}
494                                     }
495                                     ;%

```

atan2(y,x) = Arg(x,y) ... (some people have atan2 with arguments reversed but the convention here seems the most often encountered)

```

496 \xintdeffloatfunc atan2(y,x) := Arg(x, y);%

```

Argd(x,y) function from -180 (excluded) to +180 (included)

```

497 \xintdeffloatfunc Argd(x, y) := (y > x)?
498                                     {(y > -x)?
499                                     {90 - atand(x/y)}
500                                     {(y < 0)?
501                                     {-180 + atand(y/x)}

```

```

502             {180 + atand(y/x)}
503         }
504     }
505     {(y>-x)?
506         {atand(y/x)}
507         {-90 + atand(x/-y)}
508     }
509     ;%

```

`atan2d(y,x) = Argd(x,y)`

```

510 \xintdeffloatfunc atan2d(y,x) := Argd(x, y);%

```

`pArg(x,y)` function from 0 (included) to 2π (excluded) I hesitated between `pArg`, `Argpos`, and `Argplus`. Opting for `pArg` in the end.

```

511 \xintdeffloatfunc pArg(x, y):= (y>x)?
512     {(y>-x)?
513         {Piover2 - atan(x/y)}
514         {Pi + atan(y/x)}
515     }
516     {(y>-x)?
517         {(y<0)?
518             {twoPi + atan(y/x)}
519             {atan(y/x)}
520         }
521         {threePiover2 + atan(x/-y)}
522     }
523     ;%

```

`pArgd(x,y)` function from 0 (included) to 360 (excluded)

```

524 \xintdeffloatfunc pArgd(x, y):=(y>x)?
525     {(y>-x)?
526         {90 - atan(x/y)*oneRadian}
527         {180 + atan(y/x)*oneRadian}
528     }
529     {(y>-x)?
530         {(y<0)?
531             {360 + atan(y/x)*oneRadian}
532             {atan(y/x)*oneRadian}
533         }
534         {270 + atan(x/-y)*oneRadian}
535     }
536     ;%

```

12.18 Synonyms: `tg()`, `cotg()`

These are my childhood notations and I am attached to them. In radians only. We skip some overhead here by using a `\let` at core level.

```

537 \expandafter\let\csname XINT_flexpr_func_tg\expandafter\endcsname
538     \csname XINT_flexpr_func_tan\endcsname
539 \expandafter\let\csname XINT_flexpr_func_cotg\expandafter\endcsname
540     \csname XINT_flexpr_func_cot\endcsname

```

12.19 Let the functions be known to the `\xintexpr` parser

See [xint.pdf](#) for some explanations (as well as code comments in `xintexpr.sty`). In fact it is this context which led to my addition at 1.3e of `\xintdefefunc` to the `\xintexpr` syntax.

```

541 \xintFor #1 in {sin, cos, tan, sec, csc, cot,
542             asin, acos, atan}\do
543 {%
544     \xintdefefunc #1(x) := \xintfloatexpr #1(sfloat(x))\relax;%
545     \xintdefefunc #1d(x):= \xintfloatexpr #1d(sfloat(x))\relax;%
546 }%
547 \xintFor #1 in {Arg, pArg, atan2}\do
548 {%
549     \xintdefefunc #1(x, y) := \xintfloatexpr #1(sfloat(x), sfloat(y))\relax;%
550     \xintdefefunc #1d(x, y):= \xintfloatexpr #1d(sfloat(x), sfloat(y))\relax;%
551 }%
552 \xintdefefunc tg(x) := \xintfloatexpr tg(sfloat(x))\relax;%
553 \xintdefefunc cotg(x):= \xintfloatexpr cotg(sfloat(x))\relax;%
554 \xintdefefunc sinc(x):= \xintfloatexpr sinc(sfloat(x))\relax;%

```

Restore used dummy variables to their status prior to the package reloading. On first loading this is not needed naturally, because this is done immediately at end of `xintexpr.sty`.

```

555 \xintFor* #1 in {iDTVtuwxyzX}\do{\xintrestorevariable{#1}}%

```

13 Package **xintlog** implementation

Contents

13.1	Catcodes, ε -T _E X and reload detection	432
13.2	Library identification	433
13.3	Loading of poormanlog package	433
13.4	The log10() and pow10() functions	433
13.5	The log(), exp(), and pow() functions	434
13.6	\poormanloghack	435

I almost included extended precision implementation for 1.3e but was a bit short on time; besides I hesitated between using poormanlog at starting point or not. For up to 50 digits, it would help reduce considerably the needed series for the logarithm. For more digits I should rather apply my copyrighted method of the arcsine (it must be in literature).

13.1 Catcodes, ε -T_EX and reload detection

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2 \catcode13=5 % ^^M
3 \endlinechar=13 %
4 \catcode123=1 % {
5 \catcode125=2 % }
6 \catcode64=11 % @
7 \catcode35=6 % #
8 \catcode44=12 % ,
9 \catcode45=12 % -
10 \catcode46=12 % .
11 \catcode58=12 % :
12 \catcode94=7 % ^
13 \def\z{\endgroup}%
14 \def\empty{}\def\space{ }\newlinechar10
15 \expandafter\let\expandafter\w\csname ver@xintexpr.sty\endcsname
16 \expandafter\let\expandafter\x\csname ver@xintlog.sty\endcsname
17 \expandafter
18 \ifx\csname PackageInfo\endcsname\relax
19 \def\y#1#2{\immediate\write-1{Package #1 Info:^^J%
20 \space\space\space\space#2.}}%
21 \else
22 \def\y#1#2{\PackageInfo{#1}{#2}}%
23 \fi
24 \expandafter
25 \ifx\csname numexpr\endcsname\relax
26 \y{xintlog}{\numexpr not available, aborting input}%
27 \aftergroup\endinput
28 \else
29 \ifx\w\relax % xintexpr.sty not yet loaded.
30 \y{xintlog}%
31 {Loading should be via \ifx\x\empty\string\usepackage{xintexpr.sty}
32 \else\string\input\space xintexpr.sty \fi
33 rather, aborting}%
34 \aftergroup\endinput
35 \else

```



```

36     \ifx\x\relax % first loading (initiated from xintexpr.sty)
37     \else
38     \ifx\x\empty % LaTeX first loading, \ProvidesPackage not yet seen
39     \else
40     \y{xintlog}{Already loaded, aborting}%
41     \aftergroup\endinput
42     \fi
43     \fi
44     \fi
45     \fi
46 \z%
```

Attention to catcode regime when loading below poormanlog. It (v0.04) uses ^ with its normal catcode but `\XINT_setcatcodes` would set it to letter.

This file can only be loaded from `xintexpr.sty` and it restores catcodes near its end. To play it safe and be hopefully immune to whatever is done in `poormanlog` or in `xinttrig.sty` which is loaded before, we will switch to standard catcode regime here.

As I learned the hard way (I never use my user macros), at the worst moment when wrapping up the final things for 1.3e release, `\xintexprSafeCatcodes` MUST be followed by some `\xintexprRestoreCatcodes` quickly, else next time it is used (for example by `\xintdefvar`) the `\xintexprRestoreCatcodes` will restore an obsolete catcode regime...

13.2 Library identification

```

47 \xintexprSafeCatcodes\catcode`_ 11
48 \XINT_providespackage
49 \ProvidesPackage{xintlog}%
50 [2020/02/19 v1.4a Logarithms and exponentials for xintexpr (JFB)]%
```

13.3 Loading of poormanlog package

Attention to catcode regime when loading `poormanlog`. It matters less now for 1.3f as those chunks of code from `poormanlog.tex` v0.04 which needed specific `xintexpr` like catcodes got transferred here anyway.

```

51 \ifdefined\RequirePackage
52 \RequirePackage{poormanlog}%
53 \else
54 \input poormanlog.tex
55 \fi
```

`\XINT_setcatcodes` switches to the standard catcode regime of `xint*.sty` files. And we need the `xintexpr` catcode for ! too (cf `\XINT_expr_func_pow`)

See the remark above about importance of doing `\xintexprRestoreCatcodes` if `\xintexprSafeCatcodes` has been used...

```

56 \xintexprRestoreCatcodes\csname XINT_setcatcodes\endcsname\catcode`\! 11
```

13.4 The `log10()` and `pow10()` functions

The support macros from `poormanlog` v0.04 `\PoorManLogBaseTen`, `\PoorManLogPowerOfTen`, `\PoorManPower` got transferred into `xintfrac.sty` at 1.3f.

```

57 \expandafter\def\csname XINT_expr_func_log10\endcsname#1#2#3%
58 {%
```

```

59 \expandafter #1\expandafter #2\expandafter{%
60 \romannumeral`&&\XINT:NEhook:f:one:from:one
61 {\romannumeral`&&\PoorManLogBaseTen#3}}%
62 }%
63 \expandafter\let\csname XINT_flexpr_func_log10\expandafter\endcsname
64 \csname XINT_expr_func_log10\endcsname
65 \expandafter\def\csname XINT_expr_func_pow10\endcsname#1#2#3%
66 {%
67 \expandafter #1\expandafter #2\expandafter{%
68 \romannumeral`&&\XINT:NEhook:f:one:from:one
69 {\romannumeral`&&\PoorManPowerOfTen#3}}%
70 }%
71 \expandafter\let\csname XINT_flexpr_func_pow10\expandafter\endcsname
72 \csname XINT_expr_func_pow10\endcsname

```

13.5 The log(), exp(), and pow() functions

The log10() and pow10() were defined by poormanlog v0.04 but have been moved here at xint 1.3f. The support macros are defined in xintfrac.sty.

```

73 \def\XINT_expr_func_log #1#2#3%
74 {%
75 \expandafter #1\expandafter #2\expandafter{%
76 \romannumeral`&&\XINT:NEhook:f:one:from:one
77 {\romannumeral`&&\xintLog#3}}%
78 }%
79 \def\XINT_flexpr_func_log #1#2#3%
80 {%
81 \expandafter #1\expandafter #2\expandafter{%
82 \romannumeral`&&\XINT:NEhook:f:one:from:one
83 {\romannumeral`&&\XINTinFloatLog#3}}%
84 }%
85 \def\XINT_expr_func_exp #1#2#3%
86 {%
87 \expandafter #1\expandafter #2\expandafter{%
88 \romannumeral`&&\XINT:NEhook:f:one:from:one
89 {\romannumeral`&&\xintExp#3}}%
90 }%
91 \def\XINT_flexpr_func_exp #1#2#3%
92 {%
93 \expandafter #1\expandafter #2\expandafter{%
94 \romannumeral`&&\XINT:NEhook:f:one:from:one
95 {\romannumeral`&&\XINTinFloatExp#3}}%
96 }%

```

Attention that the ! is of catcode 11 here.

```

97 \def\XINT_expr_func_pow #1#2#3%
98 {%
99 \expandafter #1\expandafter #2\expandafter{%
100 \romannumeral`&&\XINT:NEhook:f:one:from:two
101 {\romannumeral`&&\PoorManPower#3}}%
102 }%
103 \let\XINT_flexpr_func_pow\XINT_expr_func_pow

```

13.6 \poormanloghack

With `\poormanloghack{**}`, the `**` operator will use `pow10(y*log10(x))`. Same for `^`. Sync'd with `xintexpr 1.4`.

```

104 \catcode\* 11
105 \def\poormanloghack**
106 {%
107   \def\XINT_tmpa ##1##2##3##4##5##6%
108   {%
109     \def ##3####1% \XINT_expr_op_<op>
110     {%
111       \expanded{\unexpanded{##4{####1}}\expandafter}%
112       \romannumeral`&&\expandafter##2\romannumeral`&&\XINT_expr_getnext
113     }%
114     \def##2####1% \XINT_expr_check_<op>
115     {%
116       \xint_UDsignfork
117       #####1{\expandafter##2\romannumeral`&&@##1}%
118       -{##5####1}%
119       \krof
120     }%
121     \def##5####1####2% \XINT_expr_checkp_<op>
122     {%
123       \ifnum ####1>\XINT_expr_precedence_**
124         \expandafter##5%
125         \romannumeral`&&\csname XINT_##6_op_####2\expandafter\endcsname
126       \else
127         \expandafter ####1\expandafter ####2%
128       \fi
129     }%
130   }%
131   \expandafter\XINT_tmpa
132   \csname XINT_expr_op_-ix\expandafter\endcsname
133   \csname XINT_expr_check_**\endcsname
134   \XINT_expr_op_**
135   \XINT_expr_exec_**
136   \XINT_expr_checkp_** {expr}%
137   \expandafter\XINT_tmpa
138   \csname XINT_flexpr_op_-ix\expandafter\endcsname
139   \csname XINT_flexpr_check_**\endcsname
140   \XINT_flexpr_op_**
141   \XINT_flexpr_exec_**
142   \XINT_flexpr_checkp_** {flexpr}%
143   \def\XINT_expr_exec_** ##1##2##3##4% \XINT_expr_exec_<op>
144   {%
145     \expandafter##2\expandafter##3\expandafter{%
146       \romannumeral`&&\XINT:NEhook:f:one:from:two
147       {\romannumeral`&&\PoorManPower##1##4}}%
148   }%
149   \let\XINT_flexpr_exec_**\XINT_expr_exec_**
150 }%
151 \def\poormanloghack^

```

```

152 {%
153 \def\XINT_tmpa ##1##2##3##4##5##6%
154 {%
155 \def ##3####1% \XINT_expr_op_<op>
156 {%
157 \expanded{\unexpanded{##4{####1}}\expandafter}%
158 \romannumeral`&&\expandafter##2\romannumeral`&&\XINT_expr_getnext
159 }%
160 \def##2####1% \XINT_expr_check_-_<op>
161 {%
162 \xint_UDsignfork
163 ####1{\expandafter##2\romannumeral`&&@##1}%
164 -{##5####1}%
165 \krof
166 }%
167 \def##5####1####2% \XINT_expr_checkp_<op>
168 {%
169 \ifnum ####1>\XINT_expr_precedence_^
170 \expandafter##5%
171 \romannumeral`&&\csname XINT_##6_op_####2\expandafter\endcsname
172 \else
173 \expandafter ##41\expandafter ##42%
174 \fi
175 }%
176 }%
177 \expandafter\XINT_tmpa
178 \csname XINT_expr_op_-ix\expandafter\endcsname
179 \csname XINT_expr_check_-_^\endcsname
180 \XINT_expr_op_^
181 \XINT_expr_exec_^
182 \XINT_expr_checkp_^ {expr}%
183 \expandafter\XINT_tmpa
184 \csname XINT_flexpr_op_-ix\expandafter\endcsname
185 \csname XINT_flexpr_check_-_^\endcsname
186 \XINT_flexpr_op_^
187 \XINT_flexpr_exec_^
188 \XINT_flexpr_checkp_^ {flexpr}%
189 \def\XINT_expr_exec_^ ##1##2##3##4% \XINT_expr_exec_<op>
190 {%
191 \expandafter##2\expandafter##3\expandafter{%
192 \romannumeral`&&\XINT:NEhook:f:one:from:two
193 {\romannumeral`&&\PoorManPower##1##4}}%
194 }%
195 \let\XINT_flexpr_exec_^ \XINT_expr_exec_^
196 }%
197 \def\poormanloghack#1{\csname poormanloghack#1\endcsname}%

```

IMPORTANT: We don't worry about resetting catcodes now as this file is theoretically only loadable from xintexpr.sty itself which will take care of the needed restore.

14 Cumulative line count

`xintkernel`: 597. Total number of code lines: 16652. (but 3827 lines among them
`xinttools`:1609. start either with `{%` or with `%}`.)
`xintcore`:2165. Each package starts with circa 50 lines dealing with cat-
`xint`:1623. codes, package identification and reloading management,
`xintbinhex`: 472. also for Plain TeX. Version 1.4a of 2020/02/19.
`xintgcd`: 368.
`xintfrac`:3478.
`xintseries`: 386.
`xintcfrac`:1029.
`xintexpr`:4173.
`xinttrig`: 555.
`xintlog`: 197.