

The `xint` packages source code

JEAN-FRANÇOIS BURNOL

jfbu (at) free (dot) fr

Package version: 1.4i (2021/06/11); documentation date: 2021/06/11.

From source file xint.dtx. Time-stamp: <11-06-2021 at 22:26:15 CEST>.

Contents

1	Timeline (in brief)	2
2	Package <code>xintkernel</code> implementation	4
3	Package <code>xinttools</code> implementation	23
4	Package <code>xintcore</code> implementation	65
5	Package <code>xint</code> implementation	122
6	Package <code>xintbinhex</code> implementation	164
7	Package <code>xintgcd</code> implementation	176
8	Package <code>xintfrac</code> implementation	187
9	Package <code>xintseries</code> implementation	279
10	Package <code>xintcfrac</code> implementation	289
11	Package <code>xintexpr</code> implementation	312
12	Package <code>xinttrig</code> implementation	434
13	Package <code>xintlog</code> implementation	457
14	Cumulative line count	494

1 Timeline (in brief)

This is 1.4i of 2021/06/11.

Please refer to [CHANGES.html](#) for a (very) detailed history.

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

- Release 1.4i of 2021/06/11: extension of the «simultaneous assignments» concept (backwards compatible).
- Release 1.4g of 2021/05/25: powers are now parsed in a right associative way. Removal of the single-character operators `&`, `|`, and `=` (deprecated at 1.1). Reformatted expandable error messages.
- Release 1.4e of 2021/05/05: logarithms and exponentials up to 62 digits, trigonometry still mainly done at high level but with guard digits so all digits up to the last one included can be trusted for faithful rounding and high probability of correct rounding.
- 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 1.3e 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>\XINTrestorecatcodes</code> , <code>\XINTsetcatcodes</code> , <code>\XINTrestorecatcodesendinginput</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	8	.16	<code>\xintLastOne</code>	13
.5	Token management utilities	8	.17	<code>\xintFirstOne</code>	13
.6	“gob til” macros and UD style fork	9	.18	<code>\xintLengthUpTo</code>	14
.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>	16
.9	<code>\xintdothis</code> , <code>\xintorthat</code>	10	.21	(WIP) <code>\xintUniformDeviate</code>	18
.10	<code>\xint_zapspace</code>	10	.22	<code>\xintMessage</code> , <code>\ifxintverbose</code>	19
.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 (2014/10/28). Separated package.

1.2i (2016/12/13). `\xintreplicate`, `\xintgobble`, `\xintLengthUpTo` and `\xintLastItem`, and faster `\xintLength`.

1.3b (2018/05/18). `\xintUniformDeviate`.

1.4 (2020/01/31) [~~commented~~ 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}%

```

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

```

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 \XINTrestorecatcodes, \XINTsetcatcodes, \XINTrestorecatcodesendinput

Renamed at 1.4e without underscores, in connexion with easing up reloading process for `xintlog.sty` and `xinttrig.sty`.

```

32   \def\PrepareCatcodes
33   {%
34   \endgroup
35   \def\XINTrestorecatcodes
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

```

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

```

69      \catcode32=\the\catcode32    %
70      \catcode61=\the\catcode61\relax    % =
71  }%
72  \edef\XINTrestorecatcodes\endinput
73  {%
74      \XINTrestorecatcodes\noexpand\endinput %
75  }%
76  \def\XINTsetcatcodes
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  \let\XINT_setcatcodes\XINTsetcatcodes
115  \let\XINT_restorecatcodes\XINTrestorecatcodes
116  \XINTsetcatcodes
117  }%
118 \PrepareCatcodes

Other modules could possibly be loaded under a different catcode regime.
119 \def\XINTsetupcatcodes {% for use by other modules

```

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

```
120 \edef\XINTrestorecatcodesendinginput
121 {%
122 \XINTrestorecatcodes\noexpand\endinput %
123 }%
124 \XINTsetcatcodes
125 }%
```

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`.

```
126 \ifdefined\ProvidesPackage
127 \let\XINT_providespackage\relax
128 \else
129 \def\XINT_providespackage #1#2[#3]%
130 {\immediate\write-1{Package: #2 #3}%
131 \expandafter\xdef\csname ver@#2.sty\endcsname{#3}}%
132 \fi
133 \XINT_providespackage
134 \ProvidesPackage {xintkernel}%
135 [2021/06/11 v1.4i Paraphernalia for the xint packages (JFB)]%
```

2.3 Constants

```
136 \chardef\xint_c_ 0
137 \chardef\xint_c_i 1
138 \chardef\xint_c_ii 2
139 \chardef\xint_c_iii 3
140 \chardef\xint_c_iv 4
141 \chardef\xint_c_v 5
142 \chardef\xint_c_vi 6
143 \chardef\xint_c_vii 7
144 \chardef\xint_c_viii 8
145 \chardef\xint_c_ix 9
146 \chardef\xint_c_x 10
147 \chardef\xint_c_xii 12
148 \chardef\xint_c_xiv 14
149 \chardef\xint_c_xvi 16
150 \chardef\xint_c_xvii 17
151 \chardef\xint_c_xviii 18
152 \chardef\xint_c_xx 20
153 \chardef\xint_c_xxii 22
154 \chardef\xint_c_ii^v 32
155 \chardef\xint_c_ii^vi 64
156 \chardef\xint_c_ii^vii 128
157 \mathchardef\xint_c_ii^viii 256
158 \mathchardef\xint_c_ii^xii 4096
159 \mathchardef\xint_c_x^iv 10000
```

2.4 (WIP) \xint_texuniformdeviate and needed counts

```

160 \ifdefined\pdfuniformdeviate \let\xint_texuniformdeviate\pdfuniformdeviate\fi
161 \ifdefined\uniformdeviate \let\xint_texuniformdeviate\uniformdeviate \fi
162 \ifx\xint_texuniformdeviate\relax\let\xint_texuniformdeviate\xint_undefined\fi
163 \ifdefined\xint_texuniformdeviate
164 \csname newcount\endcsname\xint_c_ii^xiv
165 \xint_c_ii^xiv 16384 % "4000, 2**14
166 \csname newcount\endcsname\xint_c_ii^xxi
167 \xint_c_ii^xxi 2097152 % "200000, 2**21
168 \fi

```

2.5 Token management utilities

1.3b (2018/05/18). `\xint_gobandstop...` macros because this is handy for `\xintRandomDigits.1.3b`. `g` 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 `\renewcommanded` them they will be long and this will trigger `xint` redefinitions and warnings.

```

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

```



```

204 \long\def\xint_gob_andstop_iv    #1#2#3#4{ }%
205 \long\def\xint_gob_andstop_v    #1#2#3#4#5{ }%
206 \long\def\xint_gob_andstop_vi    #1#2#3#4#5#6{ }%
207 \long\def\xint_gob_andstop_vii  #1#2#3#4#5#6#7{ }%
208 \long\def\xint_gob_andstop_viii #1#2#3#4#5#6#7#8{ }%
209 \long\def\xint_firstofone  #1{#1}%
210 \long\def\xint_firstoftwo  #1#2{#1}%
211 \long\def\xint_secondoftwo  #1#2{#2}%
212 \long\def\xint_thirdofthree#1#2#3{#3}% 1.4d
213 \let\xint_stop_aftergobble\xint_gob_andstop_i
214 \long\def\xint_stop_atfirstofone  #1{ #1}%
215 \long\def\xint_stop_atfirstoftwo  #1#2{ #1}%
216 \long\def\xint_stop_atsecondoftwo #1#2{ #2}%
217 \long\def\xint_exchangetwo_keepbraces  #1#2{{#2}{#1}}%

```

2.6 “gob til” macros and UD style fork

```

218 \long\def\xint_gob_til_R #1\R {}%
219 \long\def\xint_gob_til_W #1\W {}%
220 \long\def\xint_gob_til_Z #1\Z {}%
221 \long\def\xint_gob_til_zero #10{}%
222 \long\def\xint_gob_til_one  #11{}%
223 \long\def\xint_gob_til_zeros_iii  #1000{}%
224 \long\def\xint_gob_til_zeros_iv   #10000{}%
225 \long\def\xint_gob_til_eightzeroes #100000000{}%
226 \long\def\xint_gob_til_dot    #1.{}%
227 \long\def\xint_gob_til_G     #1G{}%
228 \long\def\xint_gob_til_minus #1-{}%
229 \long\def\xint_UDzerominusfork #10-#2#3\krof {#2}%
230 \long\def\xint_UDzerofork      #10#2#3\krof {#2}%
231 \long\def\xint_UDsignfork      #1-#2#3\krof {#2}%
232 \long\def\xint_UDwfork         #1\W#2#3\krof {#2}%
233 \long\def\xint_UDXINTWfork     #1\XINT_W#2#3\krof {#2}%
234 \long\def\xint_UDzerosfork     #100#2#3\krof {#2}%
235 \long\def\xint_UDonezerofork   #110#2#3\krof {#2}%
236 \long\def\xint_UDsignsfork     #1--#2#3\krof {#2}%
237 \let\xint:\char
238 \long\def\xint_gob_til_xint:#1\xint:{}%
239 \long\def\xint_gob_til_^#1^{}%
240 \def\xint_bracedstopper{\xint:%}
241 \long\def\xint_gob_til_exclam #1!{}% This ! has catcode 12
242 \long\def\xint_gob_til_sc #1;{}%

```

2.7 \xint_afterfi

```

243 \long\def\xint_afterfi #1#2\fi {\fi #1}%

```

2.8 \xint_bye, \xint_Bye

1.09 (2013/09/23). [\xint_bye](#)

1.2i (2016/12/13). [\xint_Bye](#) for [\xintDSRr](#) and [\xintRound](#). Also [\xint_stop_afterbye](#).

```

244 \long\def\xint_bye #1\xint_bye {}%

```

```
245 \long\def\xint_Bye #1\xint_bye {}%
246 \long\def\xint_stop_afterbye #1\xint_bye { }%
```

2.9 \xintdothis, \xintorthat

1.1 (2014/10/28).

1.2 (2015/10/10). 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.

```
247 \long\def\xint_dothis #1#2\xint_orthat #3{\fi #1}% 1.1
248 \let\xint_orthat \xint_firstofone
249 \long\def\xintdothis #1#2\xintorthat #3{\fi #1}%
250 \let\xintorthat \xint_firstofone
```

2.10 \xint_zapspaces

1.1 (2014/10/28).

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 ([\e2](#) [def](#), [\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 (2015/11/22). [\xint_zapspaces_o](#).

1.2i (2016/12/13). Made [\long](#).

ATTENTION THAT [xinttools](#) HAS AN [\xintzapspaces](#) WHICH SHOULD NOT GET CONFUSED WITH THIS ONE.

```
251 \long\def\xint_zapspaces #1 #2{#1#2\xint_zapspaces }% 1.1
252 \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.

```
253 \def\xintodef #1{\expandafter\def\expandafter#1\expandafter }%
254 \def\xintoodef #1{\expandafter\expandafter\expandafter\def
```

```

255             \expandafter\expandafter\expandafter#1%
256             \expandafter\expandafter\expandafter}%
257 \def\xintfdef #1#2%
258     {\expandafter\def\expandafter#1\expandafter{\romannumeral`&&@#2}}%
259 \ifdefined\odef\else\let\odef\xintodef\fi
260 \ifdefined\oodef\else\let\oodef\xintoodef\fi
261 \ifdefined\fdef\else\let\fdef\xintfdef\fi

```

2.12 \xintReverseOrder

1.0 (2013/03/28). 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`.

```

262 \def\xintReverseOrder {\romannumeral0\xintreverseorder}%
263 \long\def\xintreverseorder #1%
264 {%
265     \XINT_rord_main {}#1%
266     \xint:
267     \xint_bye\xint_bye\xint_bye\xint_bye
268     \xint_bye\xint_bye\xint_bye\xint_bye
269     \xint:
270}%
271 \long\def\XINT_rord_main #1#2#3#4#5#6#7#8#9%
272 {%
273     \xint_bye #9\XINT_rord_cleanup\xint_bye
274     \XINT_rord_main {#9#8#7#6#5#4#3#2#1}%
275}%
276 \def\XINT_rord_cleanup #1{%
277 \long\def\XINT_rord_cleanup\xint_bye\XINT_rord_main ##1##2\xint:
278 {%
279     \expandafter#1\xint_gob_til_xint: ##1%
280}}\XINT_rord_cleanup {}%

```

2.13 \xintLength

1.0 (2013/03/28). Does not expand its argument. See `\xintNthElt{0}` from `xinttools` which f-expands its argument.

1.2g (2016/03/19). Added `\xintCSVLength` to `xinttools`.

1.2i (2016/12/13). 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...)

```

281 \def\xintLength {\romannumeral0\xintlenth}%
282 \def\xintlenth #1{%
283 \long\def\xintlenth ##1%
284 {%
285     \expandafter#1\the\numexpr\XINT_length_loop
286     ##1\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
287     \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v

```

```

288      \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
289      \relax
290 }}\xintlength{ }%
291 \long\def\xINT_length_loop #1#2#3#4#5#6#7#8#9%
292 {%
293      \xint_gob_til_xint: #9\xINT_length_finish_a\xint:
294      \xint_c_ix+\xINT_length_loop
295 }%
296 \def\xINT_length_finish_a\xint:\xint_c_ix+\xINT_length_loop
297      #1#2#3#4#5#6#7#8#9%
298 {%
299      #9\xint_bye
300 }%

```

2.14 \xintLastItem

1.2i (2016/12/13) [commented 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.

```

301 \def\xintLastItem {\romannumeral0\xintlastitem }%
302 \long\def\xintlastitem #1%
303 {%
304      \XINT_last_loop {}.#1%
305      {\xint:\XINT_last_loop_enda}{\xint:\XINT_last_loop_endb}%
306      {\xint:\XINT_last_loop_endc}{\xint:\XINT_last_loop_endd}%
307      {\xint:\XINT_last_loop_ende}{\xint:\XINT_last_loop_endf}%
308      {\xint:\XINT_last_loop_endg}{\xint:\XINT_last_loop_endh}\xint_bye
309 }%
310 \long\def\XINT_last_loop #1.#2#3#4#5#6#7#8#9%
311 {%
312      \xint_gob_til_xint: #9%
313      {#8}{#7}{#6}{#5}{#4}{#3}{#2}{#1}\xint:
314      \XINT_last_loop {#9}.%
315 }%
316 \long\def\XINT_last_loop_enda #1#2\xint_bye{ #1}%
317 \long\def\XINT_last_loop_endb #1#2#3\xint_bye{ #2}%
318 \long\def\XINT_last_loop_endc #1#2#3#4\xint_bye{ #3}%
319 \long\def\XINT_last_loop_endd #1#2#3#4#5\xint_bye{ #4}%
320 \long\def\XINT_last_loop_ende #1#2#3#4#5#6\xint_bye{ #5}%
321 \long\def\XINT_last_loop_endf #1#2#3#4#5#6#7\xint_bye{ #6}%
322 \long\def\XINT_last_loop_endg #1#2#3#4#5#6#7#8\xint_bye{ #7}%
323 \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.

```

324 \def\xintFirstItem      {\romannumeral0\xintfirstitem }%
325 \long\def\xintfirstitem #1{\XINT_firstitem #1{\xint:\XINT_firstitem_end}\xint_bye}%
326 \long\def\XINT_firstitem #1#2\xint_bye{\xint_gob_til_xint: #1\xint:\space #1}%
327 \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 legacy `xint` user manual was not written in this spirit. And thus `\xintLastItem` did brace stripping, thus we 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.

```

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

```

351 \def\xintFirstOne {\romannumeral0\xintfirstone }%
352 \long\def\xintfirstone #1{\XINT_firstone #1{\xint:\XINT_firstone_empty}\xint:}%
353 \long\def\XINT_firstone #1#2\xint:{\xint_gob_til_xint: #1\xint:{#1}}%
354 \def\XINT_firstone_empty\xint:#1{ }%

```

2.18 \xintLengthUpTo

1.2i (2016/12/13). 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 (2016/12/22). Changed ending and interface to core loop.

```

355 \def\xintLengthUpTo {\romannumeral0\xintlenthupto}%
356 \long\def\xintlenthupto #1#2%
357 {%
358   \expandafter\xINT_lenthupto_loop
359   \the\numexpr#1.#2\xint:\xint:\xint:\xint:\xint:\xint:
360     \xint_c_vii\xint_c_vi\xint_c_v\xint_c_iv
361     \xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye.%
362 }%
363 \def\xINT_lenthupto_loop_a #1%
364 {%
365   \xint_UDsignfork
366   #1\xINT_lenthupto_gt
367   -\XINT_lenthupto_loop
368   \krof #1%
369 }%
370 \long\def\xINT_lenthupto_gt #1\xint_bye.{-0}%
371 \long\def\xINT_lenthupto_loop #1.#2#3#4#5#6#7#8#9%
372 {%
373   \xint_gob_til_xint: #9\xINT_lenthupto_finish_a\xint:%
374   \expandafter\xINT_lenthupto_loop_a\the\numexpr #1-\xint_c_viii.%
375 }%
376 \def\xINT_lenthupto_finish_a\xint:\expandafter\xINT_lenthupto_loop_a
377   \the\numexpr #1-\xint_c_viii.#2#3#4#5#6#7#8#9%
378 {%
379   \expandafter\xINT_lenthupto_finish_b\the\numexpr #1-#9\xint_bye
380 }%
381 \def\xINT_lenthupto_finish_b #1#2.%
382 {%
383   \xint_UDsignfork
384   #1{-0}%
385   -{ #1#2}%
386   \krof
387 }%
```

2.19 \xintreplicate, \xintReplicate

1.2i (2016/12/13).

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 (2020/01/31) [commented 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 `\roman` `numeral` which is the lowercased named macros. Thus adding the camelcase one creates a couple `\xintReplicate`/`\xintreplicate` not obeying the general mold.

```

388 \def\xintReplicate{\romannumerical\xintreplicate}%
389 \def\xintreplicate#1%
390   {\expandafter\XINT_replicate\the\numexpr#1\endcsname}%
391 \def\XINT_replicate #1{\xint_UDsignfork
392   #1\XINT_rep_neg
393   -\XINT_rep
394   \krof #1}%
395 \long\def\XINT_rep_neg #1\endcsname #2{\xint_c}%
396 \def\XINT_rep #1{\csname XINT_rep_f#1\XINT_rep_a}%
397 \def\XINT_rep_a #1{\csname XINT_rep_#1\XINT_rep_a}%
398 \def\XINT_rep_\XINT_rep_a{\endcsname}%
399 \long\expandafter\def\csname XINT_rep_0\endcsname #1%
400   {\endcsname{#1#1#1#1#1#1#1#1#1#1}}%
401 \long\expandafter\def\csname XINT_rep_1\endcsname #1%
402   {\endcsname{#1#1#1#1#1#1#1#1#1#1}}%
403 \long\expandafter\def\csname XINT_rep_2\endcsname #1%
404   {\endcsname{#1#1#1#1#1#1#1#1#1#1}}%
405 \long\expandafter\def\csname XINT_rep_3\endcsname #1%
406   {\endcsname{#1#1#1#1#1#1#1#1#1#1}}%
407 \long\expandafter\def\csname XINT_rep_4\endcsname #1%
408   {\endcsname{#1#1#1#1#1#1#1#1#1#1}}%
409 \long\expandafter\def\csname XINT_rep_5\endcsname #1%
410   {\endcsname{#1#1#1#1#1#1#1#1#1#1}}%
411 \long\expandafter\def\csname XINT_rep_6\endcsname #1%
412   {\endcsname{#1#1#1#1#1#1#1#1#1#1}}%
413 \long\expandafter\def\csname XINT_rep_7\endcsname #1%
414   {\endcsname{#1#1#1#1#1#1#1#1#1#1}}%
415 \long\expandafter\def\csname XINT_rep_8\endcsname #1%
416   {\endcsname{#1#1#1#1#1#1#1#1#1#1}}%
417 \long\expandafter\def\csname XINT_rep_9\endcsname #1%
418   {\endcsname{#1#1#1#1#1#1#1#1#1#1}}%
419 \long\expandafter\def\csname XINT_rep_f0\endcsname #1%
420   {\xint_c}%
421 \long\expandafter\def\csname XINT_rep_f1\endcsname #1%
422   {\xint_c_ #1}%
423 \long\expandafter\def\csname XINT_rep_f2\endcsname #1%
424   {\xint_c_ #1#1}%
425 \long\expandafter\def\csname XINT_rep_f3\endcsname #1%
426   {\xint_c_ #1#1#1}%
427 \long\expandafter\def\csname XINT_rep_f4\endcsname #1%
428   {\xint_c_ #1#1#1#1}%
429 \long\expandafter\def\csname XINT_rep_f5\endcsname #1%
430   {\xint_c_ #1#1#1#1#1}%
431 \long\expandafter\def\csname XINT_rep_f6\endcsname #1%
432   {\xint_c_ #1#1#1#1#1#1}%

```



```

433 \long\expandafter\def\csname XINT_rep_f7\endcsname #1%
434   {\xint_c_ #1#1#1#1#1#1#1}%
435 \long\expandafter\def\csname XINT_rep_f8\endcsname #1%
436   {\xint_c_ #1#1#1#1#1#1#1}%
437 \long\expandafter\def\csname XINT_rep_f9\endcsname #1%
438   {\xint_c_ #1#1#1#1#1#1#1}%

```

2.20 \xintgobble, \xintGobble

1.2i (2016/12/13).

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

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

1.4 (2020/01/31) [commented 2020/01/11]. Added \xintGobble.

```

439 \def\xintGobble{\romannumeral\xintgobble}%
440 \def\xintgobble #1%
441   {\csname xint_c_\expandafter\XINT_gobble_a\the\numexpr#1.0}%
442 \def\XINT_gobble #1.{\csname xint_c_\XINT_gobble_a #1.0}%
443 \def\XINT_gobble_a #1{\xint_gob_til_zero#1\XINT_gobble_d0\XINT_gobble_b#1}%
444 \def\XINT_gobble_b #1.#2%
445   {\expandafter\XINT_gobble_c
446     \the\numexpr (#1+\xint_c_v)/\xint_c_ix-\xint_c_i\expandafter.%
447     \the\numexpr #2+\xint_c_i.#1.%}
448 \def\XINT_gobble_c #1.#2.#3%
449   {\csname XINT_g2\the\numexpr#3-\xint_c_ix*#1\relax\XINT_gobble_a #1.#2}%
450 \def\XINT_gobble_d0\XINT_gobble_b0.#1{\endcsname}%
451 \expandafter\let\csname XINT_g10\endcsname\endcsname
452 \long\expandafter\def\csname XINT_g11\endcsname#1{\endcsname}%
453 \long\expandafter\def\csname XINT_g12\endcsname#1#2{\endcsname}%
454 \long\expandafter\def\csname XINT_g13\endcsname#1#2#3{\endcsname}%
455 \long\expandafter\def\csname XINT_g14\endcsname#1#2#3#4{\endcsname}%
456 \long\expandafter\def\csname XINT_g15\endcsname#1#2#3#4#5{\endcsname}%
457 \long\expandafter\def\csname XINT_g16\endcsname#1#2#3#4#5#6{\endcsname}%
458 \long\expandafter\def\csname XINT_g17\endcsname#1#2#3#4#5#6#7{\endcsname}%
459 \long\expandafter\def\csname XINT_g18\endcsname#1#2#3#4#5#6#7#8{\endcsname}%
460 \expandafter\let\csname XINT_g20\endcsname\endcsname
461 \long\expandafter\def\csname XINT_g21\endcsname #1#2#3#4#5#6#7#8#9%
462   {\endcsname}%
463 \long\expandafter\edef\csname XINT_g22\endcsname #1#2#3#4#5#6#7#8#9%
464   {\expandafter\noexpand\csname XINT_g21\endcsname}%
465 \long\expandafter\edef\csname XINT_g23\endcsname #1#2#3#4#5#6#7#8#9%
466   {\expandafter\noexpand\csname XINT_g22\endcsname}%
467 \long\expandafter\edef\csname XINT_g24\endcsname #1#2#3#4#5#6#7#8#9%
468   {\expandafter\noexpand\csname XINT_g23\endcsname}%
469 \long\expandafter\edef\csname XINT_g25\endcsname #1#2#3#4#5#6#7#8#9%
470   {\expandafter\noexpand\csname XINT_g24\endcsname}%
471 \long\expandafter\edef\csname XINT_g26\endcsname #1#2#3#4#5#6#7#8#9%
472   {\expandafter\noexpand\csname XINT_g25\endcsname}%
473 \long\expandafter\edef\csname XINT_g27\endcsname #1#2#3#4#5#6#7#8#9%
474   {\expandafter\noexpand\csname XINT_g26\endcsname}%
475 \long\expandafter\edef\csname XINT_g28\endcsname #1#2#3#4#5#6#7#8#9%
476   {\expandafter\noexpand\csname XINT_g27\endcsname}%

```



```

477 \expandafter\let\csname XINT_g30\endcsname\endcsname
478 \long\expandafter\edef\csname XINT_g31\endcsname #1#2#3#4#5#6#7#8#9%
479 {\expandafter\noexpand\csname XINT_g28\endcsname}%
480 \long\expandafter\edef\csname XINT_g32\endcsname #1#2#3#4#5#6#7#8#9%
481 {\noexpand\csname XINT_g31\expandafter\noexpand\csname XINT_g28\endcsname}%
482 \long\expandafter\edef\csname XINT_g33\endcsname #1#2#3#4#5#6#7#8#9%
483 {\noexpand\csname XINT_g32\expandafter\noexpand\csname XINT_g28\endcsname}%
484 \long\expandafter\edef\csname XINT_g34\endcsname #1#2#3#4#5#6#7#8#9%
485 {\noexpand\csname XINT_g33\expandafter\noexpand\csname XINT_g28\endcsname}%
486 \long\expandafter\edef\csname XINT_g35\endcsname #1#2#3#4#5#6#7#8#9%
487 {\noexpand\csname XINT_g34\expandafter\noexpand\csname XINT_g28\endcsname}%
488 \long\expandafter\edef\csname XINT_g36\endcsname #1#2#3#4#5#6#7#8#9%
489 {\noexpand\csname XINT_g35\expandafter\noexpand\csname XINT_g28\endcsname}%
490 \long\expandafter\edef\csname XINT_g37\endcsname #1#2#3#4#5#6#7#8#9%
491 {\noexpand\csname XINT_g36\expandafter\noexpand\csname XINT_g28\endcsname}%
492 \long\expandafter\edef\csname XINT_g38\endcsname #1#2#3#4#5#6#7#8#9%
493 {\noexpand\csname XINT_g37\expandafter\noexpand\csname XINT_g28\endcsname}%
494 \expandafter\let\csname XINT_g40\endcsname\endcsname
495 \expandafter\edef\csname XINT_g41\endcsname
496 {\noexpand\csname XINT_g38\expandafter\noexpand\csname XINT_g31\endcsname}%
497 \expandafter\edef\csname XINT_g42\endcsname
498 {\noexpand\csname XINT_g41\expandafter\noexpand\csname XINT_g41\endcsname}%
499 \expandafter\edef\csname XINT_g43\endcsname
500 {\noexpand\csname XINT_g42\expandafter\noexpand\csname XINT_g41\endcsname}%
501 \expandafter\edef\csname XINT_g44\endcsname
502 {\noexpand\csname XINT_g43\expandafter\noexpand\csname XINT_g41\endcsname}%
503 \expandafter\edef\csname XINT_g45\endcsname
504 {\noexpand\csname XINT_g44\expandafter\noexpand\csname XINT_g41\endcsname}%
505 \expandafter\edef\csname XINT_g46\endcsname
506 {\noexpand\csname XINT_g45\expandafter\noexpand\csname XINT_g41\endcsname}%
507 \expandafter\edef\csname XINT_g47\endcsname
508 {\noexpand\csname XINT_g46\expandafter\noexpand\csname XINT_g41\endcsname}%
509 \expandafter\edef\csname XINT_g48\endcsname
510 {\noexpand\csname XINT_g47\expandafter\noexpand\csname XINT_g41\endcsname}%
511 \expandafter\let\csname XINT_g50\endcsname\endcsname
512 \expandafter\edef\csname XINT_g51\endcsname
513 {\noexpand\csname XINT_g48\expandafter\noexpand\csname XINT_g41\endcsname}%
514 \expandafter\edef\csname XINT_g52\endcsname
515 {\noexpand\csname XINT_g51\expandafter\noexpand\csname XINT_g51\endcsname}%
516 \expandafter\edef\csname XINT_g53\endcsname
517 {\noexpand\csname XINT_g52\expandafter\noexpand\csname XINT_g51\endcsname}%
518 \expandafter\edef\csname XINT_g54\endcsname
519 {\noexpand\csname XINT_g53\expandafter\noexpand\csname XINT_g51\endcsname}%
520 \expandafter\edef\csname XINT_g55\endcsname
521 {\noexpand\csname XINT_g54\expandafter\noexpand\csname XINT_g51\endcsname}%
522 \expandafter\edef\csname XINT_g56\endcsname
523 {\noexpand\csname XINT_g55\expandafter\noexpand\csname XINT_g51\endcsname}%
524 \expandafter\edef\csname XINT_g57\endcsname
525 {\noexpand\csname XINT_g56\expandafter\noexpand\csname XINT_g51\endcsname}%
526 \expandafter\edef\csname XINT_g58\endcsname
527 {\noexpand\csname XINT_g57\expandafter\noexpand\csname XINT_g51\endcsname}%
528 \expandafter\let\csname XINT_g60\endcsname\endcsname

```

```

529 \expandafter\edef\csname XINT_g61\endcsname
530 {\noexpand\csname XINT_g58\expandafter\noexpand\csname XINT_g51\endcsname}%
531 \expandafter\edef\csname XINT_g62\endcsname
532 {\noexpand\csname XINT_g61\expandafter\noexpand\csname XINT_g61\endcsname}%
533 \expandafter\edef\csname XINT_g63\endcsname
534 {\noexpand\csname XINT_g62\expandafter\noexpand\csname XINT_g61\endcsname}%
535 \expandafter\edef\csname XINT_g64\endcsname
536 {\noexpand\csname XINT_g63\expandafter\noexpand\csname XINT_g61\endcsname}%
537 \expandafter\edef\csname XINT_g65\endcsname
538 {\noexpand\csname XINT_g64\expandafter\noexpand\csname XINT_g61\endcsname}%
539 \expandafter\edef\csname XINT_g66\endcsname
540 {\noexpand\csname XINT_g65\expandafter\noexpand\csname XINT_g61\endcsname}%
541 \expandafter\edef\csname XINT_g67\endcsname
542 {\noexpand\csname XINT_g66\expandafter\noexpand\csname XINT_g61\endcsname}%
543 \expandafter\edef\csname XINT_g68\endcsname
544 {\noexpand\csname XINT_g67\expandafter\noexpand\csname XINT_g61\endcsname}%

```

2.21 (WIP) \xintUniformDeviate

1.3b (2018/05/18). See user manual for related information.

```

545 \ifdefined\xint_texuniformdeviate
546     \expandafter\xint_firstoftwo
547 \else\expandafter\xint_secondoftwo
548 \fi
549 {%
550     \def\xintUniformDeviate#1%
551         {\the\numexpr\expandafter\XINT_uniformdeviate_sgnfork\the\numexpr#1\xint:}%
552     \def\XINT_uniformdeviate_sgnfork#1%
553     {%
554         \if-#1\XINT_uniformdeviate_neg\fi \XINT_uniformdeviate{ }#1%
555     }%
556     \def\XINT_uniformdeviate_neg\fi\XINT_uniformdeviate#1-%
557     {%
558         \fi-\numexpr\XINT_uniformdeviate\relax
559     }%
560     \def\XINT_uniformdeviate#1#2\xint:
561     {%
562         \expandafter\XINT_uniformdeviate_a\the\numexpr%
563             -\xint_texuniformdeviate\xint_c_ii^vii%
564             -\xint_c_ii^vii*\xint_texuniformdeviate\xint_c_ii^vii%
565             -\xint_c_ii^xiv*\xint_texuniformdeviate\xint_c_ii^vii%
566             -\xint_c_ii^xxi*\xint_texuniformdeviate\xint_c_ii^vii%
567             +\xint_texuniformdeviate#2\xint:/#2)*#2\xint:+#2\fi\relax#1%
568     }%
569     \def\XINT_uniformdeviate_a #1\xint:
570     {%
571         \expandafter\XINT_uniformdeviate_b\the\numexpr#1-(#1%
572     }%
573     \def\XINT_uniformdeviate_b#1#2\xint:{#1#2\if-#1}%
574 }%
575 {%
576     \def\xintUniformDeviate#1%

```

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

```
577  {%
578      \the\numexpr
579      \XINT_expandableerror{(xintkernel) No uniformdeviate primitive!}%
580      0\relax
581  }%
582 }%
```

2.22 `\xintMessage`, `\ifxintverbose`

1.2c (2015/11/16). For use by `\xintdefvar` and `\xintdeffunc` of `xintexpr`.

1.2e (2015/11/22). Uses `\write128` rather than `\write16` for compatibility with future extended range of output streams, in LuaTeX in particular.

1.3e (2019/04/05). Set the `\newlinechar`.

```
583 \def\xintMessage #1#2#3{%
584     \edef\XINT_newlinechar{\the\newlinechar}%
585     \newlinechar10
586     \immediate\write128{Package #1 #2: (on line \the\inputlineno)}%
587     \immediate\write128{\space\space\space\space#3}%
588     \newlinechar\XINT_newlinechar\space
589 }%
590 \newif\ifxintverbose
```

2.23 `\ifxintglobaldefs`, `\XINT_global`

1.3c (2018/06/17).

```
591 \newif\ifxintglobaldefs
592 \def\XINT_global{\ifxintglobaldefs\global\fi}%
```

2.24 (WIP) Expandable error message

1.21 (2017/07/26) [**commented** 2017/07/26]. But really belongs to next major release beyond 1.3. Basically copied over from `l3kernel` code. Using `\ ! /` control sequence, which must be left undefined. `\xintError:` would be 6 letters more.

1.4 (2020/01/31) [**commented** 2020/01/25]. Finally rather than `\ ! /` I use `\xint/`.

1.4g (2021/05/25) [**commented** 2021/05/19]. Rewrote to use not an undefined control sequence but trigger "Use of `\xint/` doesn't match its definition." message.

1.4g (2021/05/25) [**commented** 2021/05/20]. Things evolve fast and I switch to a third method which will exploit "Paragraph ended before `\foo` was complete" style error. See

<https://github.com/latex3/latex3/issues/931#issuecomment-845367201>

However I can not fully exploit this because `xint` may be used with Plain etex which does not set `\newlinechar`. I can only use a poorman version with no usage of `^^J`. Also `xintsession` could use the `^^J`, maybe I will integrate it there.

I. Explanations on 2021/05/19 and 2021/05/20 before final change

First I tried out things with undefined control sequence such as

`\` an error was reported by xint ...

whose output produces a nice symmetrical display with no `\`, and with ... both on left and right but this reduces drastically the available space for the actual error context. No go. But see 2021/05/20 update below!

Having replaced `\xint/` by `"\xint "`, I next opted provisorily for `"\Hit RET at ?"` control sequence, despite it being quite longer. And then I thought about using `"\ xint error"`, possibly with an included `^^J` in the name, or in the context.

I experimented with `^^J` in the context. But the context size is much constrained, and when `\errorcontextlines` is at its default value of 5 for etex, not -1 as done by LaTeX, having the info shifted to the right makes it actually more visible. (however I have now updated [xintsession](#) to 0.2b which sets `\errorcontextlines` to 0)

So I was finally back here to square one, apart from having replaced `"\xint/"` by the more longish `"\ xint error"`, hesitating with `"\xintinterrupt"`...

Then I had the idea to replace the undefined control sequence method by a method with a macro `\foo` defined as `\def\foo.{}` but used as `\foo<space>` for example. This gives something like this (the first line will be otherwise if engine is run with `-file-line-error`):

! Use of `\xint/` doesn't match its definition.

<argument> `\xint/`

Ooops, looks like we are missing a] (hit RET)

`\xint/<space>` (where the space is the unexpected token, the definition expecting rather a full stop) makes for 7 characters to compare to `\ xint error` which had 12, so I gained back 5.

Back to `^^J`: I had overlooked that TeX in the first part of the error message will display `\macro` fully, so inserting `^^J` in its name allows arbitrarily long expandable error messages... as pointed out by BLF in latex3/issues#931 as I read on the morning of 2021/05/20. This is very nice but requires to predefine control sequences for each message, and also the actual arguments #1, #2, ... values can appear only in the context.

And the situation with `^^J` is somewhat complicated:

[xintsession](#) sets the `\newlinechar` to 10, but this is not the case with bare usage of `xintexpr` with etex. And this matters. To discuss `^^J` we have to separate two locations:

- it appears in the control sequence name,
- or in the context (which itself has two parts)

1) When in the context, what happens with `^^J` is independent of the setting of `\newlinechar`: and with TeXLive `pdflatex` the `^^J` will induce a linebreak, but with `xelatex` it must be used with option `-8bit`.

2) When in the control sequence name the behaviour in log/terminal of `^^J` is influenced by the setting of `\newlinechar`. Although with `pdflatex` it will always induce a linebreak, the actual count of characters where TeX will forcefully break is influenced by whether `^^J` is or not `\newlinechar`. And with `xelatex` if it is `\newlinechar`, it does not depend then if `-8bit` or not, but if not `\newlinechar` then it does and TeX forceful breaks also change as for `pdflatex`.

So, the control sequence name trick can be used to obtain arbitrarily long messages, but the `\newlinechar` must be set.

And in the context, we can try to insert some `^^J` but this would need with `xetex` the `-8bit` option, and anyhow the context size is limited, and there is apparently no trick to get it larger.

So, in view of all the above I decided not to use `^^J` (rather `&&J` here) at all, whether here in the control sequence or the context or inserted in `\XINT_signalcondition` in the context!

I also have a problem with usage from `bnumexpr` or `polexpr` for example, they would need their own to avoid perhaps displaying `\xint/` or analogous.

II. Finally I modified again the method (completely, and no more need for funny catcode 7 space as delimiter) as this allows a longer context message, starting at start of line, and which obeys `^^J` if `\newlinechar` is set to it. It also allows to incorporate non-limited generic explanations as a postfix, with linebreaks if `\newlinechar` is known.

But as [xintexpr](#) can be used with Plain+etex which does not set the `\newlinechar`, I can't use `^^J` out of the box. I can in [xintsession](#). What I decided finally is to make a conditional definition here.

In both cases I include the "hit RET" (how rather "hit <return>") in the control sequence name serving to both provide extra information and trigger the error from being defined short and finding a `\par`.

The maximal size was increased from 48 characters (method with `\xint/` being badly delimited), to now 55 characters (using `"! xint error:<^^J or space>"` as prefix to the message). Longer messages

are truncated at 56 characters with an appended "\ETC."

As it is late on this 2021/05/20, and in order to not have to change all usages, I keep `\XINT_signalcondition` (in [xintcore](#)) as a one argument macro for time being, so will not include a more specific module name.

The `\par` token has a special role here, and can't be (I)nserted without damage, but who would want to insert it in an expandable computation anyhow... and I don't need it in my custom error messages for sure.

On 2021/05/21 I add a test about `\newlinechar` at time of package loading, and make two distinct definitions: one using `^^J` in the control sequence, the other not using it.

The `-file-line-error` toggle makes it impossible to control if the line-break on first line will match next lines. In the `^^J` branch I insert `|` (no, finally " " with two spaces) at start of continuation lines. Also I preferred to ensure a good-looking first line break for the case it starts with a `!` Paragraph ended ... because a priori error messages will be read if `-file-line-error` was emitted only a fortiori (this toggle suggests some IDE launched TeX and probably `-interaction=nonstopmode`).

I will perhaps make another definition in [xintsession](#) (it currently loads [xintexpr](#) prior to having set the `\newlinechar`, so the no `^^J` definition will be used, if nothing else is modified there).

With some hesitation I do not insert a `^^J` after `! xint error:`, as Emacs/AucTeX will display only the first line prominently and then the rest (which is in `file:line:error` mode) in one block under `--- TeX said ---`. I use the `^^J` only in the generic helper message embedded in the control sequence. The cases with or without `\newlinechar` being 10 diverge a bit, as in the latter case I had to ensure acceptable linebreaks at 79 chars, and I did that first and then had spent enough time on the matter not to add more to backport the latest `^^J` style message.

```

593 \ifnum\newlinechar=10
594 \expandafter\def\csname
595 romannumeral (or \string\expanded,&&J \space
596 \string\numexpr, ...) expansion could produce its final output.&&J \space
597 See above the exception specifics.&&J \space
598 xint will try to recover (if in interactive mode, hit <return>&&J \space
599 at the ? prompt) and will go ahead hoping repair\endcsname
600 #1\xint:}%
601 \def\XINT_expandableerror#1{%
602 \def\XINT_expandableerror##1{%
603   \expandafter
604   \XINT_expandableerrorcontinue
605   #1! xint error: ##1\par
606 }}\expandafter\XINT_expandableerror\csname
607 romannumeral (or \string\expanded,&&J \space
608 \string\numexpr, ...) expansion could produce its final output.&&J \space
609 See above the exception specifics.&&J \space
610 xint will try to recover (if in interactive mode, hit <return>&&J \space
611 at the ? prompt) and will go ahead hoping repair\endcsname
612 \else
613 \expandafter\def\csname
614 numexpr or \string\expanded\space or \string\romannumeral\space expansion
615 could terminate because an exception was raised (see the above short explanation for
616 specifics). xint will now try to recover (hit <return> if in interactive
617 mode) and will go ahead hoping repair\endcsname
618 #1\xint:}%
619 \def\XINT_expandableerror#1{%
620 \def\XINT_expandableerror##1{%

```

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

```
621 \expandafter
622 \XINT_expandableerrorcontinue
623 #1! xint error: ##1\par
624 }}\expandafter\XINT_expandableerror\csname
625 numexpr or \string\expanded\space or \string\romannumeral\space expansion
626 could terminate because an exception was raised (see the above short explanation for
627 specifics). xint will now try to recover (hit <return> if in interactive
628 mode) and will go ahead hoping repair\endcsname
629 \fi
630 \def\XINT_expandableerrorcontinue#1\par{#1}%
631 \XINTrestorecatcodesendinginput%
```

3 Package **xinttools** implementation

.1	Catcodes, ε -TeX and reload detection . .	23		<code>\xintbracedouteriloopindex</code> , <code>\xintbreak-</code>	
.2	Package identification	24		<code>iloop</code> , <code>\xintbreakiloopanddo</code> , <code>\xintiloop-</code>	
.3	<code>\xintgodef</code> , <code>\xintgoodef</code> , <code>\xintgfdef</code> .	24		<code>skiptonext</code> , <code>\xintiloopskipandredo</code> . .	45
.4	<code>\xintRevWithBraces</code>	24	.25	<code>\XINT_xflet</code>	45
.5	<code>\xintZapFirstSpaces</code>	25	.26	<code>\xintApplyInline</code>	46
.6	<code>\xintZapLastSpaces</code>	26	.27	<code>\xintFor</code> , <code>\xintFor*</code> , <code>\xintBreakFor</code> ,	
.7	<code>\xintZapSpaces</code>	27		<code>\xintBreakForAndDo</code>	47
.8	<code>\xintZapSpacesB</code>	27	.28	<code>\XINT_forever</code> , <code>\xintintegers</code> , <code>\xintdi-</code>	
.9	<code>\xintCSVtoList</code> , <code>\xintCSVtoListNon-</code>			<code>mensions</code> , <code>\xinrationals</code>	49
	Stripped	27	.29	<code>\xintForpair</code> , <code>\xintForthree</code> , <code>\xintFor-</code>	
.10	<code>\xintListWithSep</code>	29		<code>four</code>	51
.11	<code>\xintNthElt</code>	30	.30	<code>\xintAssign</code> , <code>\xintAssignArray</code> , <code>\xint-</code>	
.12	<code>\xintNthOnePy</code>	31		<code>DigitsOf</code>	52
.13	<code>\xintKeep</code>	32	.31	CSV (non user documented) variants of	
.14	<code>\xintKeepUnbraced</code>	33		Length, Keep, Trim, NthElt, Reverse . .	55
.15	<code>\xintTrim</code>	34	.31.1	<code>\xintLength:f:csv</code>	56
.16	<code>\xintTrimUnbraced</code>	36	.31.2	<code>\xintLengthUpTo:f:csv</code>	57
.17	<code>\xintApply</code>	36	.31.3	<code>\xintKeep:f:csv</code>	57
.18	<code>\xintApply:x</code> (WIP, commented-out) . .	37	.31.4	<code>\xintTrim:f:csv</code>	59
.19	<code>\xintApplyUnbraced</code>	38	.31.5	<code>\xintNthEltPy:f:csv</code>	61
.20	<code>\xintApplyUnbraced:x</code> (WIP, commented-		.31.6	<code>\xintReverse:f:csv</code>	62
	out)	39	.31.7	<code>\xintFirstItem:f:csv</code>	62
.21	<code>\xintZip</code> (WIP, not public)	40	.31.8	<code>\xintLastItem:f:csv</code>	63
.22	<code>\xintSeq</code>	42	.31.9	<code>\xintKeep:x:csv</code>	63
.23	<code>\xintloop</code> , <code>\xintbreakloop</code> , <code>\xintbreak-</code>		.31.10	Public names for the undocumented csv macros:	
	<code>loopanddo</code> , <code>\xintloopskiptonext</code> . . .	44		<code>\xintCSVLength</code> , <code>\xintCSVKeep</code> , <code>\xintCSVKeepx</code> ,	
.24	<code>\xintiloop</code> , <code>\xintiloopindex</code> , <code>\xint-</code>			<code>\xintCSVTrim</code> , <code>\xintCSVNthEltPy</code> , <code>\xintCSVRe-</code>	
	<code>bracediloopindex</code> , <code>\xintouteriloopindex</code> ,			<code>verse</code> , <code>\xintCSVFirstItem</code> , <code>\xintCSVLastItem</code>	64

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 -TeX and reload detection

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

```

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

```



```

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

```

3.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xinttools}%
46 [2021/06/11 v1.4i Expandable and non-expandable utilities (JFB)]%

```

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

```

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

```

3.3 \xintgodef, \xintgoodef, \xintgfdef

1.09i. For use in \xintAssign.

```

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

```

3.4 \xintRevWithBraces

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


```

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

```

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

```

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

```

3.5 \xintZapFirstSpaces

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

```

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

```

If the original #1 started with a space, the grabbed #1 is empty. Thus _again? will see #1=\xint_bye, and hand over control to _again which will loop back into \XINT_zapbsp_a, with one

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

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

3.6 \xintZapLastSpaces

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

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

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

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

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

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

When we have been over all possible <sp><sp> things, we reach the ending space tokens, and #3 will be a bunch of spaces (possibly none) followed by \xint_bye. So the #1 in _end? will be \xint_bye. In all other cases #1 can not be \xint_bye (assuming naturally this token does nor arise in original input), hence control falls back to \XINT_zapbsp_e which will loop back to \XINT_zapbsp_a.

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

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

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

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

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

3.7 \xintZapSpaces

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

```

113 \def\xintZapSpaces {\romannumeral0\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_end_b`).

```

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

```

```

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

```

3.10 \xintListWithSep

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

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

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

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

```

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

```

```

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

```

3.11 \xintNthElt

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

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

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

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

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

```

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

```

```

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

```

3.12 \xintNthOnePy

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

```

285\def\xintNthOnePy          {\romannumeral0\xintnthonepy }%
286\def\xintNthOnePyNoExpand {\romannumeral0\xintnthonepynoeexpand}%
287\long\def\xintnthonepy #1#2{\expandafter\XINT_nthonepy_a\the\numexpr #1\expandafter.%
288   \expandafter{\romannumeral`&&@#2}}%
289\def\xintnthonepynoeexpand #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_nthonepy_nonneg_done #1{%
325 \long\def\XINT_nthonepy_nonneg_done ##1##2\xint_bye{%
326 \xint_gob_til_xint:##1\expandafter#1\xint_gobble_ii\xint:{{##1}}%
327 }\XINT_nthonepy_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_keepnone
337 0#1\XINT_keep_neg
338 0-{\XINT_keep_pos #1}%
339 \krof
340 }%
341 \long\def\XINT_keep_keepnone .#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_keepall
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|>= 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_keeppunbr_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{b}}` where each instance of `\macro` is f-expanded. The list itself is first f-expanded and may thus be a macro. Introduced with release 1.04.

```
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` (WIP, commented-out)

Done for 1.4 (2020/01/27). For usage in the NumPy-like slicing routines. Well, actually, in the end I stucked with old-fashioned (quadratic cost) `\xintApply` for 1.4 2020/01/31 release. See comments there.

(Comments mainly from 2020/01/27, but on 2020/02/24 I comment out the code and add an alternative)

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

This 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...

```
\long\def\xintApply:x #1#2%
{%
  \XINT_apply:x_loop {#1}#2%
  {\xint:\XINT_apply:x_loop_enda}{\xint:\XINT_apply:x_loop_endb}%
  {\xint:\XINT_apply:x_loop_endc}{\xint:\XINT_apply:x_loop_endd}%
  {\xint:\XINT_apply:x_loop_ende}{\xint:\XINT_apply:x_loop_endf}%
  {\xint:\XINT_apply:x_loop_endg}{\xint:\XINT_apply:x_loop_endh}\xint_bye
}%
\long\def\XINT_apply:x_loop #1#2#3#4#5#6#7#8#9%
{%
  \xint_gob_til_xint: #9\xint:
  {#1{#2}}{#1{#3}}{#1{#4}}{#1{#5}}{#1{#6}}{#1{#7}}{#1{#8}}{#1{#9}}%
  \XINT_apply:x_loop {#1}%
}%

```

```
\long\def\XINT_apply:x_loop_endh\xint: #1\xint_bye{}%
\long\def\XINT_apply:x_loop_endg\xint: #1#2\xint_bye{{#1}}%
\long\def\XINT_apply:x_loop_endf\xint: #1#2#3\xint_bye{{#1}{#2}}%
\long\def\XINT_apply:x_loop_ende\xint: #1#2#3#4\xint_bye{{#1}{#2}{#3}}%
\long\def\XINT_apply:x_loop_endd\xint: #1#2#3#4#5\xint_bye{{#1}{#2}{#3}{#4}}%
\long\def\XINT_apply:x_loop_endc\xint: #1#2#3#4#5#6\xint_bye{{#1}{#2}{#3}{#4}{#5}}%
\long\def\XINT_apply:x_loop_endb\xint: #1#2#3#4#5#6#7\xint_bye{{#1}{#2}{#3}{#4}{#5}{#6}}%
\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}}%
```

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

Picking one by one is possibly better for small number of items. Like this for example, the natural simple minded thing:

```
\long\def\xintApply:x #1#2%
{%
  \XINT_apply:x_loop {#1}#2\xint_bye\xint_bye
}%
\long\def\XINT_apply:x_loop #1#2%
{%
  \xint_bye #2\xint_bye {#1{#2}}%
  \XINT_apply:x_loop {#1}%
}%
}
```

Some variant on 2020/02/24

```
\long\def\xint_Bye#1\xint_Bye{}%
\long\def\xintApply:x #1#2%
{%
  \XINT_apply:x_loop {#1}#2%
  {\xint_bye}{\xint_bye}{\xint_bye}{\xint_bye}%
  {\xint_bye}{\xint_bye}{\xint_bye}{\xint_bye}\xint_bye
}%
\long\def\XINT_apply:x_loop #1#2#3#4#5#6#7#8#9%
{%
  \xint_Bye #2\xint_bye {#1{#2}}%
  \xint_Bye #3\xint_bye {#1{#3}}%
  \xint_Bye #4\xint_bye {#1{#4}}%
  \xint_Bye #5\xint_bye {#1{#5}}%
  \xint_Bye #6\xint_bye {#1{#6}}%
  \xint_Bye #7\xint_bye {#1{#7}}%
  \xint_Bye #8\xint_bye {#1{#8}}%
  \xint_Bye #9\xint_bye {#1{#9}}%
  \XINT_apply:x_loop {#1}%
}%
}
```

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.

```
554 \def\xintApplyUnbraced {\romannumeral0\xintapplyunbraced}%
555 \def\xintApplyUnbracedNoExpand {\romannumeral0\xintapplyunbracednoexpand}%
556 \long\def\xintapplyunbraced #1#2%
557 {%
558   \expandafter\XINT_applyunbr\expandafter {\romannumeral`&&@#2}%
}
```

```

559     {#1}%
560 }%
561 \long\def\XINT_applyunbr #1#2{\XINT_applyunbr_loop_a {}{#2}#1\xint_bye }%
562 \long\def\xintapplyunbracednoexpand #1#2%
563     {\XINT_applyunbr_loop_a {}{#1}#2\xint_bye }%
564 \long\def\XINT_applyunbr_loop_a #1#2#3%
565 {%
566     \xint_bye #3\XINT_applyunbr_end\xint_bye
567     \expandafter\XINT_applyunbr_loop_b
568     \expandafter {\romannumeral`&&@#2{#3}}{#1}{#2}%
569 }%
570 \long\def\XINT_applyunbr_loop_b #1#2{\XINT_applyunbr_loop_a {#2#1}}%
571 \long\def\XINT_applyunbr_end\xint_bye\expandafter\XINT_applyunbr_loop_b
572     \expandafter #1#2#3{ #2}%

```

3.20 \xintApplyUnbraced:x (WIP, commented-out)

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.

Finally, xintexpr.sty 1.4 code did not use this macro but the f-expandable one \xintApplyUnbraced.

For 1.4b I prefer leave the code commented out, and classify it as WIP.

```

\long\def\xintApplyUnbraced:x #1#2%
{%
    \XINT_applyunbraced:x_loop {#1}#2%
    {\xint:\XINT_applyunbraced:x_loop_enda}{\xint:\XINT_applyunbraced:x_loop_endb}%
    {\xint:\XINT_applyunbraced:x_loop_endc}{\xint:\XINT_applyunbraced:x_loop_endd}%
    {\xint:\XINT_applyunbraced:x_loop_ende}{\xint:\XINT_applyunbraced:x_loop_endf}%
    {\xint:\XINT_applyunbraced:x_loop_endg}{\xint:\XINT_applyunbraced:x_loop_endh}\xint_bye
}%
\long\def\XINT_applyunbraced:x_loop #1#2#3#4#5#6#7#8#9%
{%
    \xint_gob_til_xint: #9\xint:
        #1{#2}%
        \empty#1{#3}%
        \empty#1{#4}%
        \empty#1{#5}%
        \empty#1{#6}%
        \empty#1{#7}%
        \empty#1{#8}%
        \empty#1{#9}%
    \XINT_applyunbraced:x_loop {#1}%
}%
\long\def\XINT_applyunbraced:x_loop_endh\xint: #1\xint_bye{}%
\long\def\XINT_applyunbraced:x_loop_endg\xint: #1\empty#2\xint_bye{#1}%
\long\def\XINT_applyunbraced:x_loop_endf\xint: #1\empty
    #2\empty#3\xint_bye{#1#2}%
\long\def\XINT_applyunbraced:x_loop_ende\xint: #1\empty
    #2\empty
    #3\empty#4\xint_bye{#1#2#3}%
\long\def\XINT_applyunbraced:x_loop_endd\xint: #1\empty
    #2\empty

```

```

#3\empty
#4\empty#5\xint_bye{#1#2#3#4}%
\long\def\XINT_applyunbraced:x_loop_endc\xint: #1\empty
#2\empty
#3\empty
#4\empty
#5\empty#6\xint_bye{#1#2#3#4#5}%
\long\def\XINT_applyunbraced:x_loop_endb\xint: #1\empty
#2\empty
#3\empty
#4\empty
#5\empty
#6\empty#7\xint_bye{#1#2#3#4#5#6}%
\long\def\XINT_applyunbraced:x_loop_enda\xint: #1\empty
#2\empty
#3\empty
#4\empty
#5\empty
#6\empty
#7\empty#8\xint_bye{#1#2#3#4#5#6#7}%

```

3.21 \xintZip (WIP, not public)

1.4b. (2020/02/25)

Support for zip(). Requires \expanded.

The implementation here thus considers the argument is already completely expanded and is a sequence of nut-ples. I will come back at later date for more generic macros.

Consider even the name of the function zip() as WIP.

As per what this does, it imitates the zip() function. See xint-manual.pdf.

I use lame terminators. Will think again later on this. I have to be careful with the used terminators, in particular with the NE context in mind.

Generally speaking I will think another day about efficiency else I will never start this.

OK, done. More compact than I initially thought. Various things should be commented upon here. Well, actually not so compact in the end as I basically had to double the whole thing simply to avoid the overhead of having to grab the final result delimited by some \xint_bye\xint_bye\xint_bye\xint_bye\xint terminator. Now actually rather \xint_bye\xint_bye\xint_bye\xint_bye\xint:

```

573 \def\xintZip #1{\expanded\XINT_zip_A#1\xint_bye\xint_bye}%
574 \def\XINT_zip_A#1%
575 {%
576   \xint_bye#1{\expandafter}\xint_bye
577   \expanded{\unexpanded{\XINT_ziptwo_A
578     #1\xint_bye\xint_bye\xint_bye\xint_bye\xint:}\expandafter}%
579   \expanded\XINT_zip_a
580 }%
581 \def\XINT_zip_a#1%
582 {%
583   \xint_bye#1\XINT_zip_terminator\xint_bye
584   \expanded{\unexpanded{\XINT_ziptwo_a
585     #1\xint_bye\xint_bye\xint_bye\xint_bye\xint:}\expandafter}%
586   \expanded\XINT_zip_a
587 }%
588 \def\XINT_zip_terminator\xint_bye#1\xint_bye{}}\empty\empty\empty\empty\xint:}%

```



```

589 \def\XINT_ziptwo_a #1#2#3#4#5\xint:#6#7#8#9%
590 {%
591     \bgroup
592     \xint_bye #1\XINT_ziptwo_e \xint_bye
593     \xint_bye #6\XINT_ziptwo_e \xint_bye {{#1}#6}%
594     \xint_bye #2\XINT_ziptwo_e \xint_bye
595     \xint_bye #7\XINT_ziptwo_e \xint_bye {{#2}#7}%
596     \xint_bye #3\XINT_ziptwo_e \xint_bye
597     \xint_bye #8\XINT_ziptwo_e \xint_bye {{#3}#8}%
598     \xint_bye #4\XINT_ziptwo_e \xint_bye
599     \xint_bye #9\XINT_ziptwo_e \xint_bye {{#4}#9}%

```

Attention here that #6 can very well deliver no tokens at all. But the \ifx will then do the expected thing. Only mentioning!

By the way, the \xint_bye method means TeX needs to look into tokens but skipping braced groups. A conditional based method lets TeX look only at the start but then it has to find \else or \fi so here also it must look at tokens, and actually goes into braced groups. But (written 2020/02/26) I never did serious testing comparing the two, and in xint I have usually preferred \xint_bye/\xint_gob_til_foo types of methods (they proved superior than \ifnum to check for 0000 in numerical core context for example, at the early days when xint used blocks of 4 digits, not 8), or usage of \if/\ifx only on single tokens, combined with some \xint_dothis/\xint_orthat syntax.

```

600     \ifx \empty#6\expandafter\XINT_zipone_a\fi
601     \XINT_ziptwo_b #5\xint:
602 }%
603 \def\XINT_zipone_a\XINT_ziptwo_b{\XINT_zipone_b}%
604 \def\XINT_ziptwo_b #1#2#3#4#5\xint:#6#7#8#9%
605 {%
606     \xint_bye #1\XINT_ziptwo_e \xint_bye
607     \xint_bye #6\XINT_ziptwo_e \xint_bye {{#1}#6}%
608     \xint_bye #2\XINT_ziptwo_e \xint_bye
609     \xint_bye #7\XINT_ziptwo_e \xint_bye {{#2}#7}%
610     \xint_bye #3\XINT_ziptwo_e \xint_bye
611     \xint_bye #8\XINT_ziptwo_e \xint_bye {{#3}#8}%
612     \xint_bye #4\XINT_ziptwo_e \xint_bye
613     \xint_bye #9\XINT_ziptwo_e \xint_bye {{#4}#9}%
614     \XINT_ziptwo_b #5\xint:
615 }%
616 \def\XINT_ziptwo_e #1\XINT_ziptwo_b #2\xint:#3\xint:
617     {\iffalse{\fi}\xint_bye\xint_bye\xint_bye\xint_bye\xint:}%
618 \def\XINT_zipone_b #1#2#3#4%
619 {%
620     \xint_bye #1\XINT_zipone_e \xint_bye {{#1}}%
621     \xint_bye #2\XINT_zipone_e \xint_bye {{#2}}%
622     \xint_bye #3\XINT_zipone_e \xint_bye {{#3}}%
623     \xint_bye #4\XINT_zipone_e \xint_bye {{#4}}%
624     \XINT_zipone_b
625 }%
626 \def\XINT_zipone_e #1\XINT_zipone_b #2\xint:
627     {\iffalse{\fi}\xint_bye\xint_bye\xint_bye\xint_bye\xint:}%
628 \def\XINT_ziptwo_A #1#2#3#4#5\xint:#6#7#8#9%
629 {%
630     \bgroup
631     \xint_bye #1\XINT_ziptwo_end \xint_bye

```

```

632 \xint_bye #6\XINT_ziptwo_end \xint_bye {{#1}#6}%
633 \xint_bye #2\XINT_ziptwo_end \xint_bye
634 \xint_bye #7\XINT_ziptwo_end \xint_bye {{#2}#7}%
635 \xint_bye #3\XINT_ziptwo_end \xint_bye
636 \xint_bye #8\XINT_ziptwo_end \xint_bye {{#3}#8}%
637 \xint_bye #4\XINT_ziptwo_end \xint_bye
638 \xint_bye #9\XINT_ziptwo_end \xint_bye {{#4}#9}%
639 \ifx \empty#6\expandafter\XINT_zipone_A\fi
640 \XINT_ziptwo_B #5\xint:
641 }%
642 \def\XINT_zipone_A\XINT_ziptwo_B{\XINT_zipone_B}%
643 \def\XINT_ziptwo_B #1#2#3#4#5\xint:#6#7#8#9%
644 {%
645 \xint_bye #1\XINT_ziptwo_end \xint_bye
646 \xint_bye #6\XINT_ziptwo_end \xint_bye {{#1}#6}%
647 \xint_bye #2\XINT_ziptwo_end \xint_bye
648 \xint_bye #7\XINT_ziptwo_end \xint_bye {{#2}#7}%
649 \xint_bye #3\XINT_ziptwo_end \xint_bye
650 \xint_bye #8\XINT_ziptwo_end \xint_bye {{#3}#8}%
651 \xint_bye #4\XINT_ziptwo_end \xint_bye
652 \xint_bye #9\XINT_ziptwo_end \xint_bye {{#4}#9}%
653 \XINT_ziptwo_B #5\xint:
654 }%
655 \def\XINT_ziptwo_end #1\XINT_ziptwo_B #2\xint:#3\xint:{\iffalse{\fi}}%
656 \def\XINT_zipone_B #1#2#3#4%
657 {%
658 \xint_bye #1\XINT_zipone_end \xint_bye {{#1}}%
659 \xint_bye #2\XINT_zipone_end \xint_bye {{#2}}%
660 \xint_bye #3\XINT_zipone_end \xint_bye {{#3}}%
661 \xint_bye #4\XINT_zipone_end \xint_bye {{#4}}%
662 \XINT_zipone_B
663 }%
664 \def\XINT_zipone_end #1\XINT_zipone_B #2\xint:#3\xint:{\iffalse{\fi}}%

```

3.22 \xintSeq

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

```

665 \def\xintSeq {\romannumeral0\xintseq}%
666 \def\xintseq #1{\XINT_seq_chkopt #1\xint_bye}%
667 \def\XINT_seq_chkopt #1%
668 {%
669 \ifx [#1\expandafter\XINT_seq_opt
670 \else\expandafter\XINT_seq_noopt
671 \fi #1%
672 }%
673 \def\XINT_seq_noopt #1\xint_bye #2%
674 {%
675 \expandafter\XINT_seq\expandafter
676 {\the\numexpr#1\expandafter}\expandafter{\the\numexpr #2}%
677 }%
678 \def\XINT_seq #1#2%

```

```

679 {%
680   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
681   \expandafter\xint_stop_atfirstoftwo
682   \or
683   \expandafter\XINT_seq_p
684   \else
685   \expandafter\XINT_seq_n
686   \fi
687   {#2}{#1}%
688 }%
689 \def\XINT_seq_p #1#2%
690 {%
691   \ifnum #1>#2
692   \expandafter\expandafter\expandafter\XINT_seq_p
693   \else
694   \expandafter\XINT_seq_e
695   \fi
696   \expandafter{\the\numexpr #1-\xint_c_i}{#2}{#1}%
697 }%
698 \def\XINT_seq_n #1#2%
699 {%
700   \ifnum #1<#2
701   \expandafter\expandafter\expandafter\XINT_seq_n
702   \else
703   \expandafter\XINT_seq_e
704   \fi
705   \expandafter{\the\numexpr #1+\xint_c_i}{#2}{#1}%
706 }%
707 \def\XINT_seq_e #1#2#3{ }%
708 \def\XINT_seq_opt [\xint_bye #1]#2#3%
709 {%
710   \expandafter\XINT_seqo\expandafter
711   {\the\numexpr #2\expandafter}\expandafter
712   {\the\numexpr #3\expandafter}\expandafter
713   {\the\numexpr #1}%
714 }%
715 \def\XINT_seqo #1#2%
716 {%
717   \ifcase\ifnum #1=#2 0\else\ifnum #2>#1 1\else -1\fi\fi\space
718   \expandafter\XINT_seqo_a
719   \or
720   \expandafter\XINT_seqo_pa
721   \else
722   \expandafter\XINT_seqo_na
723   \fi
724   {#1}{#2}%
725 }%
726 \def\XINT_seqo_a #1#2#3{ {#1}}%
727 \def\XINT_seqo_o #1#2#3#4{ #4}%
728 \def\XINT_seqo_pa #1#2#3%
729 {%
730   \ifcase\ifnum #3=\xint_c_ 0\else\ifnum #3>\xint_c_ 1\else -1\fi\fi\space

```

```

731         \expandafter\XINT_seqo_o
732     \or
733         \expandafter\XINT_seqo_pb
734     \else
735         \xint_afterfi{\expandafter\space\xint_gobble_iv}%
736     \fi
737     {#1}{#2}{#3}{{#1}}%
738 }%
739 \def\XINT_seqo_pb #1#2#3%
740 {%
741     \expandafter\XINT_seqo_pc\expandafter{\the\numexpr #1+#3}{#2}{#3}%
742 }%
743 \def\XINT_seqo_pc #1#2%
744 {%
745     \ifnum #1>#2
746         \expandafter\XINT_seqo_o
747     \else
748         \expandafter\XINT_seqo_pd
749     \fi
750     {#1}{#2}%
751 }%
752 \def\XINT_seqo_pd #1#2#3#4{\XINT_seqo_pb {#1}{#2}{#3}{#4{#1}}}%
753 \def\XINT_seqo_na #1#2#3%
754 {%
755     \ifcase\ifnum #3=\xint_c_ 0\else\ifnum #3>\xint_c_ 1\else -1\fi\fi\space
756         \expandafter\XINT_seqo_o
757     \or
758         \xint_afterfi{\expandafter\space\xint_gobble_iv}%
759     \else
760         \expandafter\XINT_seqo_nb
761     \fi
762     {#1}{#2}{#3}{{#1}}%
763 }%
764 \def\XINT_seqo_nb #1#2#3%
765 {%
766     \expandafter\XINT_seqo_nc\expandafter{\the\numexpr #1+#3}{#2}{#3}%
767 }%
768 \def\XINT_seqo_nc #1#2%
769 {%
770     \ifnum #1<#2
771         \expandafter\XINT_seqo_o
772     \else
773         \expandafter\XINT_seqo_nd
774     \fi
775     {#1}{#2}%
776 }%
777 \def\XINT_seqo_nd #1#2#3#4{\XINT_seqo_nb {#1}{#2}{#3}{#4{#1}}}%

```

3.23 \xintloop, \xintbreakloop, \xintbreakloopaddo, \xintloopskiptonext

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

```

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

```

```

779 \long\def\xintloop_again\fi\xint_gobble_i #1{\fi
780           #1\xintloop_again\fi\xint_gobble_i {#1}}%
781 \long\def\xintbreakloop #1\xintloop_again\fi\xint_gobble_i #2{%}
782 \long\def\xintbreakloopanddo #1#2\xintloop_again\fi\xint_gobble_i #3{#1}%
783 \long\def\xintloopskiptonext #1\xintloop_again\fi\xint_gobble_i #2{%
784           #2\xintloop_again\fi\xint_gobble_i {#2}}%

```

3.24 \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.

```

785 \def\xintilooop [#1+#2]{%
786     \expandafter\xintilooop_a\the\numexpr #1\expandafter.\the\numexpr #2.%}
787 \long\def\xintilooop_a #1.#2.#3#4\repeat{%
788     #3#4\xintilooop_again\fi\xint_gobble_iii {#1}{#2}{#3#4}}%
789 \def\xintilooop_again\fi\xint_gobble_iii #1#2{%
790     \fi\expandafter\xintilooop_again_b\the\numexpr#1+#2.#2.%}
791 \long\def\xintilooop_again_b #1.#2.#3{%
792     #3\xintilooop_again\fi\xint_gobble_iii {#1}{#2}{#3}}%
793 \long\def\xintbreakilooop #1\xintilooop_again\fi\xint_gobble_iii #2#3#4{%}
794 \long\def\xintbreakilooopanddo
795     #1.#2\xintilooop_again\fi\xint_gobble_iii #3#4#5{#1}%
796 \long\def\xintilooopindex #1\xintilooop_again\fi\xint_gobble_iii #2%
797     {#2#1\xintilooop_again\fi\xint_gobble_iii {#2}}%
798 \long\def\xintbracedilooopindex #1\xintilooop_again\fi\xint_gobble_iii #2%
799     {{#2}#1\xintilooop_again\fi\xint_gobble_iii {#2}}%
800 \long\def\xintouterilooopindex #1\xintilooop_again
801     #2\xintilooop_again\fi\xint_gobble_iii #3%
802     {#3#1\xintilooop_again #2\xintilooop_again\fi\xint_gobble_iii {#3}}%
803 \long\def\xintbracedouterilooopindex #1\xintilooop_again
804     #2\xintilooop_again\fi\xint_gobble_iii #3%
805     {{#3}#1\xintilooop_again #2\xintilooop_again\fi\xint_gobble_iii {#3}}%
806 \long\def\xintilooopskiptonext #1\xintilooop_again\fi\xint_gobble_iii #2#3{%
807     \expandafter\xintilooop_again_b \the\numexpr#2+#3.#3.%}
808 \long\def\xintilooopskipandredo #1\xintilooop_again\fi\xint_gobble_iii #2#3#4{%
809     #4\xintilooop_again\fi\xint_gobble_iii {#2}{#3}{#4}}%

```

3.25 \XINT_xflet

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

```

810 \def\XINT_xflet #1%
811 {%
812     \def\XINT_xflet_macro {#1}\XINT_xflet_zapsp
813 }%
814 \def\XINT_xflet_zapsp
815 {%
816     \expandafter\futurelet\expandafter\XINT_token
817     \expandafter\XINT_xflet_sp?\romannumeral`&&@%
818 }%

```

```

819 \def\XINT_xflet_sp?
820 {%
821   \ifx\XINT_token\XINT_sptoken
822     \expandafter\XINT_xflet_zapsp
823   \else\expandafter\XINT_xflet_zapspB
824   \fi
825 }%
826 \def\XINT_xflet_zapspB
827 {%
828   \expandafter\futurelet\expandafter\XINT_tokenB
829   \expandafter\XINT_xflet_spB?\romannumeral`&&@%
830 }%
831 \def\XINT_xflet_spB?
832 {%
833   \ifx\XINT_tokenB\XINT_sptoken
834     \expandafter\XINT_xflet_zapspB
835   \else\expandafter\XINT_xflet_eq?
836   \fi
837 }%
838 \def\XINT_xflet_eq?
839 {%
840   \ifx\XINT_token\XINT_tokenB
841     \expandafter\XINT_xflet_macro
842   \else\expandafter\XINT_xflet_zapsp
843   \fi
844 }%

```

3.26 \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.

```

845 \catcode`Z 3
846 \long\def\xintApplyInline #1#2%
847 {%
848   \long\expandafter\def\expandafter\XINT_inline_macro
849   \expandafter ##\expandafter 1\expandafter {#1{##1}}%
850   \XINT_xflet\XINT_inline_b #2Z% this Z has catcode 3
851 }%
852 \def\XINT_inline_b
853 {%
854   \ifx\XINT_token Z\expandafter\xint_gobble_i
855   \else\expandafter\XINT_inline_d\fi
856 }%
857 \long\def\XINT_inline_d #1%
858 {%
859   \long\def\XINT_item{#1}\XINT_xflet\XINT_inline_e
860 }%
861 \def\XINT_inline_e
862 {%

```

```

863 \ifx\XINT_token Z\expandafter\XINT_inline_w
864 \else\expandafter\XINT_inline_f\fi
865 }%
866 \def\XINT_inline_f
867 {%
868 \expandafter\XINT_inline_g\expandafter{\XINT_inline_macro {##1}}%
869 }%
870 \long\def\XINT_inline_g #1%
871 {%
872 \expandafter\XINT_inline_macro\XINT_item
873 \long\def\XINT_inline_macro ##1{#1}\XINT_inline_d
874 }%
875 \def\XINT_inline_w #1%
876 {%
877 \expandafter\XINT_inline_macro\XINT_item
878 }%

```

3.27 \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.

```

879 \def\XINT_tmpa #1#2{\ifnum #2<#1 \xint_afterfi {{#####2}}\fi}%
880 \def\XINT_tmpb #1#2{\ifnum #1<#2 \xint_afterfi {{#####2}}\fi}%
881 \def\XINT_tmpc #1%
882 {%
883 \expandafter\edef \csname XINT_for_left#1\endcsname
884 {\xintApplyUnbraced {\XINT_tmpa #1}{123456789}}%
885 \expandafter\edef \csname XINT_for_right#1\endcsname
886 {\xintApplyUnbraced {\XINT_tmpb #1}{123456789}}%
887 }%
888 \xintApplyInline \XINT_tmpc {123456789}%
889 \long\def\xintBreakFor #1Z{}%
890 \long\def\xintBreakForAndDo #1#2Z{#1}%
891 \def\xintFor {\let\xintifForFirst\xint_firstoftwo
892 \let\xintifForLast\xint_secondoftwo
893 \futurelet\XINT_token\XINT_for_ifstar }%
894 \def\XINT_for_ifstar {\ifx\XINT_token*\expandafter\XINT_forx
895 \else\expandafter\XINT_for \fi }%

```

```

896 \catcode`U 3 % with numexpr
897 \catcode`V 3 % with xintfrac.sty (xint.sty not enough)
898 \catcode`D 3 % with dimexpr
899 \def\XINT_flet_zapsp
900 {%
901   \futurelet\XINT_token\XINT_flet_sp?
902 }%
903 \def\XINT_flet_sp?
904 {%
905   \ifx\XINT_token\XINT_sptoken
906     \xint_afterfi{\expandafter\XINT_flet_zapsp\romannumeral0}%
907   \else\expandafter\XINT_flet_macro
908   \fi
909 }%
910 \long\def\XINT_for #1#2in#3#4#5%
911 {%
912   \expandafter\XINT_toks\expandafter
913   {\expandafter\XINT_for_d\the\numexpr #2\relax {#5}}%
914   \def\XINT_flet_macro {\expandafter\XINT_for_forever?\space}%
915   \expandafter\XINT_flet_zapsp #3Z%
916 }%
917 \def\XINT_for_forever? #1Z%
918 {%
919   \ifx\XINT_token U\XINT_to_forever\fi
920   \ifx\XINT_token V\XINT_to_forever\fi
921   \ifx\XINT_token D\XINT_to_forever\fi
922   \expandafter\the\expandafter\XINT_toks\romannumeral0\xintcsvtolist {#1}Z%
923 }%
924 \def\XINT_to_forever\fi #1\xintcsvtolist #2{\fi \XINT_forever #2}%
925 \long\def\XINT_forx *#1#2in#3#4#5%
926 {%
927   \expandafter\XINT_toks\expandafter
928   {\expandafter\XINT_forx_d\the\numexpr #2\relax {#5}}%
929   \XINT_xflet\XINT_forx_forever? #3Z%
930 }%
931 \def\XINT_forx_forever?
932 {%
933   \ifx\XINT_token U\XINT_to_forxever\fi
934   \ifx\XINT_token V\XINT_to_forxever\fi
935   \ifx\XINT_token D\XINT_to_forxever\fi
936   \XINT_forx_empty?
937 }%
938 \def\XINT_to_forxever\fi #1\XINT_forx_empty? {\fi \XINT_forever }%
939 \catcode`U 11
940 \catcode`D 11
941 \catcode`V 11
942 \def\XINT_forx_empty?
943 {%
944   \ifx\XINT_token Z\expandafter\xintBreakFor\fi
945   \the\XINT_toks
946 }%
947 \long\def\XINT_for_d #1#2#3%

```



```

948 {%
949 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
950 \XINT_toks {{#3}}%
951 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
952 \the\XINT_toks \csname XINT_for_right#1\endcsname}%
953 \XINT_toks {\XINT_x\let\xintifForFirst\xint_secondoftwo
954 \let\xintifForLast\xint_secondoftwo\XINT_for_d #1{#2}}%
955 \futurelet\XINT_token\XINT_for_last?
956}%
957\long\def\XINT_forx_d #1#2#3%
958{%
959 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
960 \XINT_toks {{#3}}%
961 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
962 \the\XINT_toks \csname XINT_for_right#1\endcsname}%
963 \XINT_toks {\XINT_x\let\xintifForFirst\xint_secondoftwo
964 \let\xintifForLast\xint_secondoftwo\XINT_forx_d #1{#2}}%
965 \XINT_x\let\XINT_for_last?
966}%
967\def\XINT_for_last?
968{%
969 \ifx\XINT_token Z\expandafter\XINT_for_last?yes\fi
970 \the\XINT_toks
971}%
972\def\XINT_for_last?yes
973{%
974 \let\xintifForLast\xint_firstoftwo
975 \xintBreakForAndDo{\XINT_x\xint_gobble_i Z}%
976}%

```

3.28 \XINT_forever, \xintintegers, \xintdimensions, \xintrationals

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

```

977\catcode`U 3
978\catcode`D 3
979\catcode`V 3
980\let\xintegers U%
981\let\xintintegers U%
982\let\xintdimensions D%
983\let\xinrationals V%
984\def\XINT_forever #1%
985{%
986 \expandafter\XINT_forever_a
987 \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi a\expandafter\endcsname
988 \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi i\expandafter\endcsname
989 \csname XINT_?expr_\ifx#1UU\else\ifx#1DD\else V\fi\fi \endcsname
990}%

```

```

991 \catcode`U 11
992 \catcode`D 11
993 \catcode`V 11
994 \def\XINT_?expr_Ua #1#2%
995   {\expandafter{\expandafter\numexpr\the\numexpr #1\expandafter\relax
996     \expandafter\relax\expandafter}%
997   \expandafter{\the\numexpr #2}}%
998 \def\XINT_?expr_Da #1#2%
999   {\expandafter{\expandafter\dimexpr\number\dimexpr #1\expandafter\relax
1000     \expandafter s\expandafter p\expandafter\relax\expandafter}%
1001   \expandafter{\number\dimexpr #2}}%
1002 \catcode`Z 11
1003 \def\XINT_?expr_Va #1#2%
1004 {%
1005   \expandafter\XINT_?expr_Vb\expandafter
1006     {\romannumeral`&&\xintrowwithzeros{\xintZapSpacesB{#2}}}%
1007     {\romannumeral`&&\xintrowwithzeros{\xintZapSpacesB{#1}}}%
1008 }%
1009 \catcode`Z 3
1010 \def\XINT_?expr_Vb #1#2{\expandafter\XINT_?expr_Vc #2.#1}%
1011 \def\XINT_?expr_Vc #1/#2.#3/#4.%
1012 {%
1013   \xintifEq {#2}{#4}%
1014     {\XINT_?expr_Vf {#3}{#1}{#2}}%
1015     {\expandafter\XINT_?expr_Vd\expandafter
1016       {\romannumeral0\xintiimul {#2}{#4}}%
1017       {\romannumeral0\xintiimul {#1}{#4}}%
1018       {\romannumeral0\xintiimul {#2}{#3}}%
1019     }%
1020 }%
1021 \def\XINT_?expr_Vd #1#2#3{\expandafter\XINT_?expr_Ve\expandafter {#2}{#3}{#1}}%
1022 \def\XINT_?expr_Ve #1#2{\expandafter\XINT_?expr_Vf\expandafter {#2}{#1}}%
1023 \def\XINT_?expr_Vf #1#2#3{{#2/#3}{0}{#1}{#2}{#3}}%
1024 \def\XINT_?expr_Ui {{\numexpr 1\relax}{1}}%
1025 \def\XINT_?expr_Di {{\dimexpr 0pt\relax}{65536}}%
1026 \def\XINT_?expr_Vi {{1/1}{0111}}%
1027 \def\XINT_?expr_U #1#2%
1028   {\expandafter{\expandafter\numexpr\the\numexpr #1+#2\relax\relax}{#2}}%
1029 \def\XINT_?expr_D #1#2%
1030   {\expandafter{\expandafter\dimexpr\the\numexpr #1+#2\relax sp\relax}{#2}}%
1031 \def\XINT_?expr_V #1#2{\XINT_?expr_Vx #2}%
1032 \def\XINT_?expr_Vx #1#2%
1033 {%
1034   \expandafter\XINT_?expr_Vy\expandafter
1035     {\romannumeral0\xintiiadd {#1}{#2}}{#2}%
1036 }%
1037 \def\XINT_?expr_Vy #1#2#3#4%
1038 {%
1039   \expandafter{\romannumeral0\xintiiadd {#3}{#1}/#4}{#1}{#2}{#3}{#4}}%
1040 }%
1041 \def\XINT_forever_a #1#2#3#4%
1042 {%

```

```

1043 \ifx #4[\expandafter\XINT_forever_opt_a
1044 \else\expandafter\XINT_forever_b
1045 \fi #1#2#3#4%
1046 }%
1047 \def\XINT_forever_b #1#2#3Z{\expandafter\XINT_forever_c\the\XINT_toks #2#3}%
1048 \long\def\XINT_forever_c #1#2#3#4#5%
1049 {\expandafter\XINT_forever_d\expandafter #2#4#5{#3}Z}%
1050 \def\XINT_forever_opt_a #1#2#3[#4+#5]#6Z%
1051 {%
1052 \expandafter\expandafter\expandafter
1053 \XINT_forever_opt_c\expandafter\the\expandafter\XINT_toks
1054 \romannumeral`&&@#1{#4}{#5}#3%
1055 }%
1056 \long\def\XINT_forever_opt_c #1#2#3#4#5#6{\XINT_forever_d #2{#4}{#5}#6{#3}Z}%
1057 \long\def\XINT_forever_d #1#2#3#4#5%
1058 {%
1059 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#5}%
1060 \XINT_toks {#2}}%
1061 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
1062 \the\XINT_toks \csname XINT_for_right#1\endcsname }%
1063 \XINT_x
1064 \let\xintifForFirst\xint_seconddoftwo
1065 \let\xintifForLast\xint_seconddoftwo
1066 \expandafter\XINT_forever_d\expandafter #1\romannumeral`&&@#4{#2}{#3}#4{#5}%
1067 }%

```

3.29 \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.

```

1068 \catcode`j 3
1069 \long\def\xintForpair #1#2#3in#4#5#6%
1070 {%
1071 \let\xintifForFirst\xint_firstoftwo
1072 \let\xintifForLast\xint_seconddoftwo
1073 \XINT_toks {\XINT_forpair_d #2{#6}}%
1074 \expandafter\the\expandafter\XINT_toks #4jZ%
1075 }%
1076 \long\def\XINT_forpair_d #1#2#3(#4)#5%
1077 {%
1078 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
1079 \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
1080 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
1081 \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_i\endcsname}%
1082 \ifx #5j\expandafter\XINT_for_last?yes\fi
1083 \XINT_x
1084 \let\xintifForFirst\xint_seconddoftwo
1085 \let\xintifForLast\xint_seconddoftwo

```

```

1086 \XINT_forpair_d #1{#2}%
1087 }%
1088 \long\def\xintForthree #1#2#3in#4#5#6%
1089 {%
1090 \let\xintifForFirst\xint_firstoftwo
1091 \let\xintifForLast\xint_secondoftwo
1092 \XINT_toks {\XINT_forthree_d #2{#6}}%
1093 \expandafter\the\expandafter\XINT_toks #4jZ%
1094 }%
1095 \long\def\XINT_forthree_d #1#2#3(#4)#5%
1096 {%
1097 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
1098 \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
1099 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
1100 \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_ii\endcsname}%
1101 \ifx #5j\expandafter\XINT_for_last?yes\fi
1102 \XINT_x
1103 \let\xintifForFirst\xint_secondoftwo
1104 \let\xintifForLast\xint_secondoftwo
1105 \XINT_forthree_d #1{#2}%
1106 }%
1107 \long\def\xintForfour #1#2#3in#4#5#6%
1108 {%
1109 \let\xintifForFirst\xint_firstoftwo
1110 \let\xintifForLast\xint_secondoftwo
1111 \XINT_toks {\XINT_forfour_d #2{#6}}%
1112 \expandafter\the\expandafter\XINT_toks #4jZ%
1113 }%
1114 \long\def\XINT_forfour_d #1#2#3(#4)#5%
1115 {%
1116 \long\def\XINT_y ##1##2##3##4##5##6##7##8##9{#2}%
1117 \XINT_toks \expandafter{\romannumeral0\xintcsvtolist{ #4}}%
1118 \long\edef\XINT_x {\noexpand\XINT_y \csname XINT_for_left#1\endcsname
1119 \the\XINT_toks \csname XINT_for_right\the\numexpr#1+\xint_c_iii\endcsname}%
1120 \ifx #5j\expandafter\XINT_for_last?yes\fi
1121 \XINT_x
1122 \let\xintifForFirst\xint_secondoftwo
1123 \let\xintifForLast\xint_secondoftwo
1124 \XINT_forfour_d #1{#2}%
1125 }%
1126 \catcode`Z 11
1127 \catcode`j 11

```

3.30 \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.

```

1128 \def\xintAssign{\def\XINT_flet_macro {\XINT_assign_fork}\XINT_flet_zapsp }%
1129 \def\XINT_assign_fork

```

```

1130 {%
1131     \let\XINT_assign_def\def
1132     \ifx\XINT_token[\expandafter\XINT_assign_opt
1133         \else\expandafter\XINT_assign_a
1134     \fi
1135 }%
1136 \def\XINT_assign_opt [#1]%
1137 {%
1138     \ifcsname #1def\endcsname
1139         \expandafter\let\expandafter\XINT_assign_def \csname #1def\endcsname
1140     \else
1141         \expandafter\let\expandafter\XINT_assign_def \csname xint#1def\endcsname
1142     \fi
1143     \XINT_assign_a
1144 }%
1145 \long\def\XINT_assign_a #1\to
1146 {%
1147     \def\XINT_flet_macro{\XINT_assign_b}%
1148     \expandafter\XINT_flet_zapsp\romannumeral`&&@#1\xint:\to
1149 }%
1150 \long\def\XINT_assign_b
1151 {%
1152     \ifx\XINT_token\bgroup
1153         \expandafter\XINT_assign_c
1154     \else\expandafter\XINT_assign_f
1155     \fi
1156 }%
1157 \long\def\XINT_assign_f #1\xint:\to #2%
1158 {%
1159     \XINT_assign_def #2{#1}%
1160 }%
1161 \long\def\XINT_assign_c #1%
1162 {%
1163     \def\xint_temp {#1}%
1164     \ifx\xint_temp\xint_bracedstopper
1165         \expandafter\XINT_assign_e
1166     \else
1167         \expandafter\XINT_assign_d
1168     \fi
1169 }%
1170 \long\def\XINT_assign_d #1\to #2%
1171 {%
1172     \expandafter\XINT_assign_def\expandafter #2\expandafter{\xint_temp}%
1173     \XINT_assign_c #1\to
1174 }%
1175 \def\XINT_assign_e #1\to {}%
1176 \def\xintRelaxArray #1%
1177 {%
1178     \edef\XINT_restoreescapechar {\escapechar\the\escapechar\relax}%
1179     \escapechar -1
1180     \expandafter\def\expandafter\xint_arrayname\expandafter {\string #1}%
1181     \XINT_restoreescapechar

```

```

1182 \xintilop [\csname\xint_arrayname 0\endcsname+-1]
1183 \global
1184 \expandafter\let\csname\xint_arrayname\xintilopindex\endcsname\relax
1185 \ifnum \xintilopindex > \xint_c_
1186 \repeat
1187 \global\expandafter\let\csname\xint_arrayname 00\endcsname\relax
1188 \global\let #1\relax
1189 }%
1190 \def\xintAssignArray{\def\XINT_flet_macro {\XINT_assignarray_fork}%
1191 \XINT_flet_zapsp }%
1192 \def\XINT_assignarray_fork
1193 {%
1194 \let\XINT_assignarray_def\def
1195 \ifx\XINT_token[\expandafter\XINT_assignarray_opt
1196 \else\expandafter\XINT_assignarray
1197 \fi
1198 }%
1199 \def\XINT_assignarray_opt [#1]%
1200 {%
1201 \ifcsname #1def\endcsname
1202 \expandafter\let\expandafter\XINT_assignarray_def \csname #1def\endcsname
1203 \else
1204 \expandafter\let\expandafter\XINT_assignarray_def
1205 \csname xint#1def\endcsname
1206 \fi
1207 \XINT_assignarray
1208 }%
1209 \long\def\XINT_assignarray #1\to #2%
1210 {%
1211 \edef\XINT_restoreescapechar {\escapechar\the\escapechar\relax }%
1212 \escapechar -1
1213 \expandafter\def\expandafter\xint_arrayname\expandafter {\string #2}%
1214 \XINT_restoreescapechar
1215 \def\xint_itemcount {0}%
1216 \expandafter\XINT_assignarray_loop \romannumeral`&&@#1\xint:
1217 \csname\xint_arrayname 00\expandafter\endcsname
1218 \csname\xint_arrayname 0\expandafter\endcsname
1219 \expandafter {\xint_arrayname}#2%
1220 }%
1221 \long\def\XINT_assignarray_loop #1%
1222 {%
1223 \def\xint_temp {#1}%
1224 \ifx\xint_temp\xint_bracedstopper
1225 \expandafter\def\csname\xint_arrayname 0\expandafter\endcsname
1226 \expandafter{\the\numexpr\xint_itemcount}%
1227 \expandafter\expandafter\expandafter\XINT_assignarray_end
1228 \else
1229 \expandafter\def\expandafter\xint_itemcount\expandafter
1230 {\the\numexpr\xint_itemcount+\xint_c_i}%
1231 \expandafter\XINT_assignarray_def
1232 \csname\xint_arrayname\xint_itemcount\expandafter\endcsname
1233 \expandafter{\xint_temp }%

```

```

1234     \expandafter\XINT_assignarray_loop
1235   \fi
1236 }%
1237 \def\XINT_assignarray_end #1#2#3#4%
1238 {%
1239   \def #4##1%
1240   {%
1241     \romannumeral0\expandafter #1\expandafter{\the\numexpr ##1}%
1242   }%
1243   \def #1##1%
1244   {%
1245     \ifnum ##1<\xint_c_
1246       \xint_afterfi{\XINT_expandableerror{Array index is negative: ##1.} }%
1247     \else
1248       \xint_afterfi {%
1249         \ifnum ##1>#2
1250           \xint_afterfi
1251             {\XINT_expandableerror{Array index is beyond range: ##1 > #2.} }%
1252           \else\xint_afterfi
1253             {\expandafter\expandafter\expandafter\space\csname #3##1\endcsname}%
1254           \fi}%
1255     \fi
1256   }%
1257 }%
1258 \let\xintDigitsOf\xintAssignArray

```

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

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

These macros will remain undocumented in the user manual:

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

-- crucial design elements may change in future:

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

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

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

space token) and produces empty output for empty list. This means that the status of the «nil» object for the `xintexpr` parsers is not completely clarified (currently it is represented by a space token).

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

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

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

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

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

3.31.1 `\xintLength:f:csv`

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

```
1259 \def\xintLength:f:csv {\romannumeral0\xintlength:f:csv}%
1260 \def\xintlength:f:csv #1%
1261 {\long\def\xintlength:f:csv ##1{%
1262   \expandafter#1\the\numexpr\expandafter\XINT_length:f:csv_a
1263   \romannumeral`&&@##1\xint:,\xint:,\xint:,\xint:,%
1264   \xint:,\xint:,\xint:,\xint:,\xint:,%
1265   \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
1266   \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
1267   \relax
1268 }}\xintlength:f:csv { }%
```

Must first check if empty list.

```
1269 \long\def\XINT_length:f:csv_a #1%
1270 {%
1271   \xint_gob_til_xint: #1\xint_c_\xint_bye\xint:%
1272   \XINT_length:f:csv_loop #1%
1273 }%
1274 \long\def\XINT_length:f:csv_loop #1,#2,#3,#4,#5,#6,#7,#8,#9,%
1275 {%
1276   \xint_gob_til_xint: #9\XINT_length:f:csv_finish\xint:%
1277   \xint_c_ix+\XINT_length:f:csv_loop
1278 }%
1279 \def\XINT_length:f:csv_finish\xint:\xint_c_ix+\XINT_length:f:csv_loop
1280   #1,#2,#3,#4,#5,#6,#7,#8,#9,{#9\xint_bye}%
```


3.31.2 \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.

```

1281 \def\xintLengthUpTo:f:csv {\romannumeral0\xintlenthupto:f:csv}%
1282 \long\def\xintlenthupto:f:csv #1#2%
1283 {%
1284   \expandafter\xINT_lenthupto:f:csv_a
1285   \the\numexpr#1\expandafter.%
1286   \romannumeral'&&@#2\xint:,\xint:,\xint:,\xint:,%
1287   \xint:,\xint:,\xint:,\xint:,%
1288   \xint_c_viii,\xint_c_vii,\xint_c_vi,\xint_c_v,%
1289   \xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye.%
1290 }%

1291 \long\def\xINT_lenthupto:f:csv_a #1.#2%
1292 {%
1293   \xint_gob_til_xint: #2\xINT_lenthupto:f:csv_empty\xint:%
1294   \XINT_lenthupto:f:csv_loop_b #1.#2%
1295 }%
1296 \def\xINT_lenthupto:f:csv_empty\xint:%
1297   \XINT_lenthupto:f:csv_loop_b #1.#2\xint_bye.{ #1}%
1298 \def\xINT_lenthupto:f:csv_loop_a #1%
1299 {%
1300   \xint_UDsignfork
1301   #1\xINT_lenthupto:f:csv_gt
1302   -\XINT_lenthupto:f:csv_loop_b
1303   \krof #1%
1304 }%
1305 \long\def\xINT_lenthupto:f:csv_gt #1\xint_bye.{-0}%
1306 \long\def\xINT_lenthupto:f:csv_loop_b #1.#2,#3,#4,#5,#6,#7,#8,#9,%
1307 {%
1308   \xint_gob_til_xint: #9\xINT_lenthupto:f:csv_finish_a\xint:%
1309   \expandafter\xINT_lenthupto:f:csv_loop_a\the\numexpr #1-\xint_c_viii.%
1310 }%
1311 \def\xINT_lenthupto:f:csv_finish_a\xint:
1312   \expandafter\xINT_lenthupto:f:csv_loop_a
1313   \the\numexpr #1-\xint_c_viii.#2,#3,#4,#5,#6,#7,#8,#9,%
1314 {%
1315   \expandafter\xINT_lenthupto:f:csv_finish_b\the\numexpr #1-#9\xint_bye
1316 }%
1317 \def\xINT_lenthupto:f:csv_finish_b #1#2.%
1318 {%
1319   \xint_UDsignfork
1320   #1{-0}%
1321   -{ #1#2}%
1322   \krof
1323 }%

```

3.31.3 \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.

```

1324 \def\xintKeep:f:csv {\romannumeral0\xintkeep:f:csv}%
1325 \long\def\xintkeep:f:csv #1#2%
1326 {%
1327   \expandafter\xint_stop_aftergobble
1328   \romannumeral0\expandafter\XINT_keep:f:csv_a
1329   \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
1330}%
1331 \def\XINT_keep:f:csv_a #1%
1332 {%
1333   \xint_UDzerominusfork
1334   #1-\XINT_keep:f:csv_keeptime
1335   0#1\XINT_keep:f:csv_neg
1336   0-{\XINT_keep:f:csv_pos #1}%
1337   \krof
1338}%
1339 \long\def\XINT_keep:f:csv_keeptime .#1{,%
1340 \long\def\XINT_keep:f:csv_neg #1.#2%
1341 {%
1342   \expandafter\XINT_keep:f:csv_neg_done\expandafter,%
1343   \romannumeral0%
1344   \expandafter\XINT_keep:f:csv_neg_a\the\numexpr
1345   #1-\numexpr\XINT_length:f:csv_a
1346   #2\xint:,\xint:,\xint:,\xint:,%
1347   \xint:,\xint:,\xint:,\xint:,\xint:,%
1348   \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
1349   \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
1350   .#2\xint_bye
1351}%
1352 \def\XINT_keep:f:csv_neg_a #1%
1353 {%
1354   \xint_UDsignfork
1355   #1{\expandafter\XINT_keep:f:csv_trimloop\the\numexpr-\xint_c_ix+}%
1356   -\XINT_keep:f:csv_keeptime
1357   \krof
1358}%
1359 \def\XINT_keep:f:csv_keeptime #1.{}%
1360 \long\def\XINT_keep:f:csv_neg_done #1\xint_bye{#1}%
1361 \def\XINT_keep:f:csv_trimloop #1#2.%
1362 {%
1363   \xint_gob_til_minus#1\XINT_keep:f:csv_trimloop_finish-%
1364   \expandafter\XINT_keep:f:csv_trimloop
1365   \the\numexpr#1#2-\xint_c_ix\expandafter.\XINT_keep:f:csv_trimloop_trimnine
1366}%
1367 \long\def\XINT_keep:f:csv_trimloop_trimnine #1,#2,#3,#4,#5,#6,#7,#8,#9,{}%
1368 \def\XINT_keep:f:csv_trimloop_finish-%
1369   \expandafter\XINT_keep:f:csv_trimloop
1370   \the\numexpr-#1-\xint_c_ix\expandafter.\XINT_keep:f:csv_trimloop_trimnine

```

```

1371     {\csname XINT_trim:f:csv_finish#1\endcsname}%
1372 \long\def\XINT_keep:f:csv_pos #1.#2%
1373 {%
1374     \expandafter\XINT_keep:f:csv_pos_fork
1375     \romannumeral0\XINT_lengthupto:f:csv_a
1376     #1.#2\xint:,\xint:,\xint:,\xint:,%
1377     \xint:,\xint:,\xint:,\xint:,%
1378     \xint_c_viii,\xint_c_vii,\xint_c_vi,\xint_c_v,%
1379     \xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye.%
1380     .#1.{ }#2\xint_bye%
1381 }%
1382 \def\XINT_keep:f:csv_pos_fork #1#2.%
1383 {%
1384     \xint_UDsignfork
1385     #1{\expandafter\XINT_keep:f:csv_loop\the\numexpr-\xint_c_viii+}%
1386     -\XINT_keep:f:csv_pos_keepall
1387     \krof
1388 }%
1389 \long\def\XINT_keep:f:csv_pos_keepall #1.#2#3\xint_bye{,#3}%
1390 \def\XINT_keep:f:csv_loop #1#2.%
1391 {%
1392     \xint_gob_til_minus#1\XINT_keep:f:csv_loop_end-%
1393     \expandafter\XINT_keep:f:csv_loop
1394     \the\numexpr#1#2-\xint_c_viii\expandafter.\XINT_keep:f:csv_loop_pickeight
1395 }%
1396 \long\def\XINT_keep:f:csv_loop_pickeight
1397     #1#2,#3,#4,#5,#6,#7,#8,#9,{#{#1,#2,#3,#4,#5,#6,#7,#8,#9}}%
1398 \def\XINT_keep:f:csv_loop_end-\expandafter\XINT_keep:f:csv_loop
1399     \the\numexpr-#1-\xint_c_viii\expandafter.\XINT_keep:f:csv_loop_pickeight
1400     {\csname XINT_keep:f:csv_end#1\endcsname}%
1401 \long\expandafter\def\csname XINT_keep:f:csv_end1\endcsname
1402     #1#2,#3,#4,#5,#6,#7,#8,#9\xint_bye {#1,#2,#3,#4,#5,#6,#7,#8}%
1403 \long\expandafter\def\csname XINT_keep:f:csv_end2\endcsname
1404     #1#2,#3,#4,#5,#6,#7,#8\xint_bye {#1,#2,#3,#4,#5,#6,#7}%
1405 \long\expandafter\def\csname XINT_keep:f:csv_end3\endcsname
1406     #1#2,#3,#4,#5,#6,#7\xint_bye {#1,#2,#3,#4,#5,#6}%
1407 \long\expandafter\def\csname XINT_keep:f:csv_end4\endcsname
1408     #1#2,#3,#4,#5,#6\xint_bye {#1,#2,#3,#4,#5}%
1409 \long\expandafter\def\csname XINT_keep:f:csv_end5\endcsname
1410     #1#2,#3,#4,#5\xint_bye {#1,#2,#3,#4}%
1411 \long\expandafter\def\csname XINT_keep:f:csv_end6\endcsname
1412     #1#2,#3,#4\xint_bye {#1,#2,#3}%
1413 \long\expandafter\def\csname XINT_keep:f:csv_end7\endcsname
1414     #1#2,#3\xint_bye {#1,#2}%
1415 \long\expandafter\def\csname XINT_keep:f:csv_end8\endcsname
1416     #1#2\xint_bye {#1}%

```

3.31.4 \xintTrim:f:csv

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

```

1417 \def\xintTrim:f:csv {\romannumeral0\xinttrim:f:csv }%
1418 \long\def\xinttrim:f:csv #1#2%

```

```

1419 {%
1420   \expandafter\xint_stop_aftergobble
1421   \romannumeral0\expandafter\XINT_trim:f:csv_a
1422   \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
1423 }%
1424 \def\XINT_trim:f:csv_a #1%
1425 {%
1426   \xint_UDzerominusfork
1427   #1-\XINT_trim:f:csv_trimnone
1428   0#1\XINT_trim:f:csv_neg
1429   0-{\XINT_trim:f:csv_pos #1}%
1430   \krof
1431 }%
1432 \long\def\XINT_trim:f:csv_trimnone .#1{,#1}%
1433 \long\def\XINT_trim:f:csv_neg #1.#2%
1434 {%
1435   \expandafter\XINT_trim:f:csv_neg_a\the\numexpr
1436   #1-\numexpr\XINT_length:f:csv_a
1437   #2\xint:,\xint:,\xint:,\xint:,%
1438   \xint:,\xint:,\xint:,\xint:,\xint:,%
1439   \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
1440   \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
1441   .{ }#2\xint_bye
1442 }%
1443 \def\XINT_trim:f:csv_neg_a #1%
1444 {%
1445   \xint_UDsignfork
1446   #1{\expandafter\XINT_keep:f:csv_loop\the\numexpr-\xint_c_viii+}%
1447   -\XINT_trim:f:csv_trimall
1448   \krof
1449 }%
1450 \def\XINT_trim:f:csv_trimall {\expandafter,\xint_bye}%
1451 \long\def\XINT_trim:f:csv_pos #1.#2%
1452 {%
1453   \expandafter\XINT_trim:f:csv_pos_done\expandafter,%
1454   \romannumeral0%
1455   \expandafter\XINT_trim:f:csv_loop\the\numexpr#1-\xint_c_ix.%
1456   #2\xint:,\xint:,\xint:,\xint:,\xint:,%
1457   \xint:,\xint:,\xint:,\xint:,\xint:\xint_bye
1458 }%
1459 \def\XINT_trim:f:csv_loop #1#2.%
1460 {%
1461   \xint_gob_til_minus#1\XINT_trim:f:csv_finish-%
1462   \expandafter\XINT_trim:f:csv_loop\the\numexpr#1#2\XINT_trim:f:csv_loop_trimnine
1463 }%
1464 \long\def\XINT_trim:f:csv_loop_trimnine #1,#2,#3,#4,#5,#6,#7,#8,#9,%
1465 {%
1466   \xint_gob_til_xint: #9\XINT_trim:f:csv_toofew\xint:-\xint_c_ix.%
1467 }%
1468 \def\XINT_trim:f:csv_toofew\xint:{*\xint_c_}%
1469 \def\XINT_trim:f:csv_finish-%
1470   \expandafter\XINT_trim:f:csv_loop\the\numexpr-#1\XINT_trim:f:csv_loop_trimnine

```

```

1471 {%
1472   \csname XINT_trim:f:csv_finish#1\endcsname
1473 }%
1474 \long\expandafter\def\csname XINT_trim:f:csv_finish1\endcsname
1475   #1,#2,#3,#4,#5,#6,#7,#8,{ }%
1476 \long\expandafter\def\csname XINT_trim:f:csv_finish2\endcsname
1477   #1,#2,#3,#4,#5,#6,#7,{ }%
1478 \long\expandafter\def\csname XINT_trim:f:csv_finish3\endcsname
1479   #1,#2,#3,#4,#5,#6,{ }%
1480 \long\expandafter\def\csname XINT_trim:f:csv_finish4\endcsname
1481   #1,#2,#3,#4,#5,{ }%
1482 \long\expandafter\def\csname XINT_trim:f:csv_finish5\endcsname
1483   #1,#2,#3,#4,{ }%
1484 \long\expandafter\def\csname XINT_trim:f:csv_finish6\endcsname
1485   #1,#2,#3,{ }%
1486 \long\expandafter\def\csname XINT_trim:f:csv_finish7\endcsname
1487   #1,#2,{ }%
1488 \long\expandafter\def\csname XINT_trim:f:csv_finish8\endcsname
1489   #1,{ }%
1490 \expandafter\let\csname XINT_trim:f:csv_finish9\endcsname\space
1491 \long\def\XINT_trim:f:csv_pos_done #1\xint:#2\xint_bye{#1}%

```

3.31.5 \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.

```

1492 \def\xintNthEltPy:f:csv {\romannumeral0\xintntheltpy:f:csv }%
1493 \long\def\xintntheltpy:f:csv #1#2%
1494 {%
1495   \expandafter\XINT_nthelt:f:csv_a
1496   \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
1497 }%
1498 \def\XINT_nthelt:f:csv_a #1%
1499 {%
1500   \xint_UDsignfork
1501     #1\XINT_nthelt:f:csv_neg
1502     -\XINT_nthelt:f:csv_pos
1503   \krof #1%
1504 }%
1505 \long\def\XINT_nthelt:f:csv_neg -#1.#2%
1506 {%
1507   \expandafter\XINT_nthelt:f:csv_neg_fork
1508   \the\numexpr\XINT_length:f:csv_a
1509   #2\xint:,\xint:,\xint:,\xint:,%
1510   \xint:,\xint:,\xint:,\xint:,\xint:,%
1511   \xint_c_ix,\xint_c_viii,\xint_c_vii,\xint_c_vi,%
1512   \xint_c_v,\xint_c_iv,\xint_c_iii,\xint_c_ii,\xint_c_i,\xint_bye
1513   -#1.#2,\xint_bye
1514 }%
1515 \def\XINT_nthelt:f:csv_neg_fork #1%
1516 {%
1517   \if#1-\expandafter\xint_stop_afterbye\fi

```

```

1518 \expandafter\XINT_nthelt:f:csv_neg_done
1519 \romannumeral0%
1520 \expandafter\XINT_keep:f:csv_trimloop\the\numexpr-\xint_c_ix+#1%
1521 }%
1522 \long\def\XINT_nthelt:f:csv_neg_done#1,#2\xint_bye{ #1}%
1523 \long\def\XINT_nthelt:f:csv_pos #1.#2%
1524 {%
1525 \expandafter\XINT_nthelt:f:csv_pos_done
1526 \romannumeral0%
1527 \expandafter\XINT_trim:f:csv_loop\the\numexpr#1-\xint_c_ix.%
1528 #2\xint:,\xint:,\xint:,\xint:,\xint:,%
1529 \xint:,\xint:,\xint:,\xint:,\xint:,\xint_bye
1530 }%
1531 \def\XINT_nthelt:f:csv_pos_done #1{%
1532 \long\def\XINT_nthelt:f:csv_pos_done ##1,##2\xint_bye{%
1533 \xint_gob_til_xint:##1\XINT_nthelt:f:csv_pos_cleanup\xint:#1##1}%
1534 }\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...

```

1535 \xint_firstofone{\long\def\XINT_nthelt:f:csv_pos_cleanup\xint:} %
1536 #1\xint:{ #1}%

```

3.31.6 `\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.

```

1537 \def\xintReverse:f:csv {\romannumeral0\xintreverse:f:csv}%
1538 \long\def\xintreverse:f:csv #1%
1539 {%
1540 \expandafter\XINT_reverse:f:csv_loop
1541 \expandafter{\expandafter}\romannumeral`&&@#1,%
1542 \xint:,%
1543 \xint_bye,\xint_bye,\xint_bye,\xint_bye,%
1544 \xint_bye,\xint_bye,\xint_bye,\xint_bye,%
1545 \xint:
1546 }%
1547 \long\def\XINT_reverse:f:csv_loop #1#2,#3,#4,#5,#6,#7,#8,#9,%
1548 {%
1549 \xint_bye #9\XINT_reverse:f:csv_cleanup\xint_bye
1550 \XINT_reverse:f:csv_loop {,#9,#8,#7,#6,#5,#4,#3,#2#1}%
1551 }%
1552 \long\def\XINT_reverse:f:csv_cleanup\xint_bye\XINT_reverse:f:csv_loop #1#2\xint:
1553 {%
1554 \XINT_reverse:f:csv_finish #1%
1555 }%
1556 \long\def\XINT_reverse:f:csv_finish #1\xint:,{ }%

```

3.31.7 `\xintFirstItem:f:csv`

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

```
1557 \def\xintFirstItem:f:csv {\romannumeral0\xintfirstitem:f:csv}%
1558 \long\def\xintfirstitem:f:csv #1%
1559 {%
1560   \expandafter\XINT_first:f:csv_a\romannumeral`&&@#1,\xint_bye
1561 }%
1562 \long\def\XINT_first:f:csv_a #1,#2\xint_bye{ #1}%
```

3.31.8 \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.

```
1563 \def\xintLastItem:f:csv {\romannumeral0\xintlasteritem:f:csv}%
1564 \long\def\xintlasteritem:f:csv #1%
1565 {%
1566   \expandafter\XINT_last:f:csv_loop\expandafter{\expandafter}\expandafter.%
1567   \romannumeral`&&@#1,%
1568   \xint:\XINT_last_loop_enda,\xint:\XINT_last_loop_endb,%
1569   \xint:\XINT_last_loop_endc,\xint:\XINT_last_loop_endd,%
1570   \xint:\XINT_last_loop_ende,\xint:\XINT_last_loop_endf,%
1571   \xint:\XINT_last_loop_endg,\xint:\XINT_last_loop_endh,\xint_bye
1572 }%
1573 \long\def\XINT_last:f:csv_loop #1.#2,#3,#4,#5,#6,#7,#8,#9,%
1574 {%
1575   \xint_gob_til_xint: #9%
1576   {#8}{#7}{#6}{#5}{#4}{#3}{#2}{#1}\xint:
1577   \XINT_last:f:csv_loop {#9}%.%
1578 }%
```

3.31.9 \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.

```
1579 \def\xintKeep:x:csv #1#2%
1580 {%
1581   \expandafter\xint_gobble_i
1582   \romannumeral0\expandafter\XINT_keep:x:csv_pos
1583   \the\numexpr #1\expandafter.\expandafter{\romannumeral`&&@#2}%
1584 }%
1585 \def\XINT_keep:x:csv_pos #1.#2%
1586 {%
1587   \expandafter\XINT_keep:x:csv_loop\the\numexpr#1-\xint_c_viii.%
1588   #2\xint_Bye,\xint_Bye,\xint_Bye,\xint_Bye,%
1589   \xint_Bye,\xint_Bye,\xint_Bye,\xint_Bye,\xint_bye
1590 }%
1591 \def\XINT_keep:x:csv_loop #1%
1592 {%
1593   \xint_gob_til_minus#1\XINT_keep:x:csv_finish-%
```

```

1594 \XINT_keep:x:csv_loop_pickeight #1%
1595 }%
1596 \def\XINT_keep:x:csv_loop_pickeight #1.#2,#3,#4,#5,#6,#7,#8,#9,%
1597 {%
1598     ,#2,#3,#4,#5,#6,#7,#8,#9%
1599 \expandafter\XINT_keep:x:csv_loop\the\numexpr#1-\xint_c_viii.%
1600 }%
1601 \def\XINT_keep:x:csv_finish-\XINT_keep:x:csv_loop_pickeight -#1.%
1602 {%
1603     \csname XINT_keep:x:csv_finish#1\endcsname
1604 }%
1605 \expandafter\def\csname XINT_keep:x:csv_finish1\endcsname
1606     #1,#2,#3,#4,#5,#6,#7,{, #1,#2,#3,#4,#5,#6,#7\xint_Bye}%
1607 \expandafter\def\csname XINT_keep:x:csv_finish2\endcsname
1608     #1,#2,#3,#4,#5,#6,{, #1,#2,#3,#4,#5,#6\xint_Bye}%
1609 \expandafter\def\csname XINT_keep:x:csv_finish3\endcsname
1610     #1,#2,#3,#4,#5,{, #1,#2,#3,#4,#5\xint_Bye}%
1611 \expandafter\def\csname XINT_keep:x:csv_finish4\endcsname
1612     #1,#2,#3,#4,{, #1,#2,#3,#4\xint_Bye}%
1613 \expandafter\def\csname XINT_keep:x:csv_finish5\endcsname
1614     #1,#2,#3,{, #1,#2,#3\xint_Bye}%
1615 \expandafter\def\csname XINT_keep:x:csv_finish6\endcsname
1616     #1,#2,{, #1,#2\xint_Bye}%
1617 \expandafter\def\csname XINT_keep:x:csv_finish7\endcsname
1618     #1,{, #1\xint_Bye}%
1619 \expandafter\let\csname XINT_keep:x:csv_finish8\endcsname\xint_Bye

```

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

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

```

1620 \let\xintCSVLength \xintLength:f:csv
1621 \let\xintCSVKeep \xintKeep:f:csv
1622 \let\xintCSVKeepx \xintKeep:x:csv
1623 \let\xintCSVTrim \xintTrim:f:csv
1624 \let\xintCSVNthEltPy \xintNthEltPy:f:csv
1625 \let\xintCSVReverse \xintReverse:f:csv
1626 \let\xintCSVFirstItem\xintFirstItem:f:csv
1627 \let\xintCSVLastItem \xintLastItem:f:csv
1628 \let\XINT_tmpa\relax \let\XINT_tmpb\relax \let\XINT_tmpc\relax
1629 \XINTrestorecatcodesendinginput%

```


4 Package [xintcore](#) implementation

.1	Catcodes, ε -T _E X and reload detection . . .	65	.25	\XINT_zeroes_forviii	78
.2	Package identification	66	.26	\XINT_sepbyviii_Z	79
.3	(WIP!) Error conditions and exceptions . . .	66	.27	\XINT_sepbyviii_andcount	79
.4	Counts for holding needed constants . . .	69	.28	\XINT_rsepbyviii	79
Routines handling integers as lists of token digits . . .			69	.29	\XINT_sepandrev
.5	\XINT_cuz_small	69	.30	\XINT_sepandrev_andcount	80
.6	\xintNum, \xintiNum	70	.31	\XINT_rev_nounsep	81
.7	\xintiiSgn	70	.32	\XINT_unrevbyviii	81
.8	\xintiiOpp	71	Core arithmetic		82
.9	\xintiiAbs	71	.33	\xintiiAdd	82
.10	\xintFDg	72	.34	\xintiiCmp	85
.11	\xintLDg	72	.35	\xintiiSub	87
.12	\xintDouble	73	.36	\xintiiMul	92
.13	\xintHalf	73	.37	\xintiiDivision	96
.14	\xintInc	74	Derived arithmetic		111
.15	\xintDec	74	.38	\xintiiQuo, \xintiiRem	111
.16	\xintDSL	75	.39	\xintiiDivRound	111
.17	\xintDSR	75	.40	\xintiiDivTrunc	112
.18	\xintDSRr	75	.41	\xintiiModTrunc	112
Blocks of eight digits			76	.42	\xintiiDivMod
.19	\XINT_cuz	76	.43	\xintiiDivFloor	114
.20	\XINT_cuz_byviii	76	.44	\xintiiMod	114
.21	\XINT_unsep_loop	77	.45	\xintiiSqr	114
.22	\XINT_unsep_cuzsmall	77	.46	\xintiiPow	115
.23	\XINT_div_unsepQ	78	.47	\xintiiFac	118
.24	\XINT_div_unsepR	78	.48	\XINT_useiimessage	121

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 (2021/06/11) 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 [2021/06/11 v1.4i 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
77 % #2

```

On 2021/05/19, 1.4g, I re-examined `\XINT_expandableerror` experimenting at first with an added `^^J` to shift to next line the actual message.

Previously I was calling it thrice (condition #2, user context #3, next tokens #5) here but it seems more reasonable to use it only once. As total size is so limited, I decided to only display #3 (information for user) and drop the #2 (condition, first argument of `\XINT_signalcondition`) and the display of the #5 (next tokens, fourth argument of `\XINT_signalcondition`).

Besides, why was I doing here `\xint_stop_atfirstofone{#5}`, which adds limitations to usage? Now inserting #5 directly so callers will have to insert a `\romannumeral0` stopping space token if needed. I thus have to update all usages across (mainly, I think) `xintfrac`. Done, but using here `\xint_firstofone{#5}`. This looks silly, but allows some hypothetical future usage by user of `I\xintUse{stuff}` usage where `\xintUse` would be `\xint_firstofthree`.

The problem is that this would have to be explained to user in the error context but space there is so extremely limited...

After having reviewed existing usage of `\XINT_signalcondition`, I noticed there was free space in most cases and added here " (hit RET)" after #3.

I experimented with `^^J` here too (its effect in the "context" is independent of the `\newlinechar` setting, but it depends on the engine: works with TeXLive pdftex, requires -8bit with xetex)

However, due to `\errorcontextlines` being 5 by default in etex (but `xintsession 0.2b` sets it to 0), I finally decided to not insert a `^^J` (&&J) at all to separate the " (hit RET)" hint.

On 2021/05/20 evening I found another completely different method for `\XINT_expandableerror`, which has some advantages. In particular it allows me to not use here "#3 (hit RET)" but simply "#3" as such information can be integrated in a non size limited generic message.

The maximal size of #3 here was increased from 48 characters (method with `\xint/` being badly delimited), to now 55 characters, longer messages being truncated at 56 characters with an appended "\ETC."

```

78 \XINT_expandableerror{#3}%
79 % not for X3.274
80 %\XINT_expandableerror{<RET>, or I\xintUse{...}<RET>, or I\xintCTRLC<RET>}%
81 \xint_firstofone{#5}%
82 }%
83 }%
84 %% \def\xintUse{\xint_firstofthree} % defined in xint.sty
85 \def\XINT_ifFlagRaised #1{%
86 \ifcsname XINT_#1Flag_ON\endcsname
87 \expandafter\xint_firstoftwo
88 \else
89 \expandafter\xint_secondoftwo
90 \fi}%
91 \def\XINT_resetFlag #1%
92 {\expandafter\let\csname XINT_#1Flag_ON\endcsname\XINT_undefined}%
93 \def\XINT_resetFlags {% WIP
94 \XINT_resetFlag{InvalidOperation}% also from DivisionUndefined
95 \XINT_resetFlag{DivisionByZero}%
96 \XINT_resetFlag{Underflow}% (\xintiiPow with negative exponent)
97 \XINT_resetFlag{Overflow}% not encountered so far in xint code 1.21
98 % .. others ..

```

```

99 }%
100 \def\XINT_RaiseFlag #1{\expandafter\xint_gobble_i\csname XINT_#1Flag_ON\endcsname}%
    NOT IMPLEMENTED! WORK IN PROGRESS! (ALL SIGNALS TRAPPED, NO HANDLERS USED)
101 \catcode`. 11
102 \let\XINT_Clamped.handler\xint_firstofone % WIP
103 \def\XINT_InvalidOperation.handler#1{NaN}% WIP
104 \def\XINT_ConversionSyntax.handler#1{NaN}% WIP
105 \def\XINT_DivisionByZero.handler#1{SignedInfinity(#1)}% WIP
106 \def\XINT_DivisionImpossible.handler#1{NaN}% WIP
107 \def\XINT_DivisionUndefined.handler#1{NaN}% WIP
108 \let\XINT_Inexact.handler\xint_firstofone % WIP
109 \def\XINT_InvalidContext.handler#1{NaN}% WIP
110 \let\XINT_Rounded.handler\xint_firstofone % WIP
111 \let\XINT_Subnormal.handler\xint_firstofone% WIP
112 \def\XINT_Overflow.handler#1{NaN}% WIP
113 \def\XINT_Underflow.handler#1{NaN}% WIP
114 \catcode`. 12

```

4.4 Counts for holding needed constants

```

115 \ifdefined\m@ne\let\xint_c_mone\m@ne
116      \else\csname newcount\endcsname\xint_c_mone \xint_c_mone -1 \fi
117 \ifdefined\xint_c_x^viii\else
118 \csname newcount\endcsname\xint_c_x^viii \xint_c_x^viii 1000000000
119 \fi
120 \ifdefined\xint_c_x^ix\else
121 \csname newcount\endcsname\xint_c_x^ix \xint_c_x^ix 10000000000
122 \fi
123 \newcount\xint_c_x^viii_mone \xint_c_x^viii_mone 99999999
124 \newcount\xint_c_xii_e_viii \xint_c_xii_e_viii 12000000000
125 \newcount\xint_c_xi_e_viii_mone \xint_c_xi_e_viii_mone 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.

```

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

```

4.6 \xintNum, \xintiNum

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

Very old routine got completely rewritten at 1.21.

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

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

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

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

\xintNum gets redefined by xintfrac.

```

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

```

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

```

149   \expandafter\XINT_num_loop
150   \else

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

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

```

4.7 \xintiiSgn

1.21 made \xintiiSgn robust against non terminated input.

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

```

156 \def\xintiiSgn {\romannumeral0\xintiisgn }%
157 \def\xintiisgn #1%
158 {%
159   \expandafter\XINT_sgn \romannumeral`&&@#1\xint:
160 }%

```

```

161 \def\XINT_sgn #1#2\xint:
162 {%
163   \xint_UDzerominusfork
164   #1-{ 0}%
165   0#1{-1}%
166   0-{ 1}%
167   \krof
168 }%
169 \def\XINT_Sgn #1#2\xint:
170 {%
171   \xint_UDzerominusfork
172   #1-{0}%
173   0#1{-1}%
174   0-{1}%
175   \krof
176 }%
177 \def\XINT_cntSgn #1#2\xint:
178 {%
179   \xint_UDzerominusfork
180   #1-\xint_c_
181   0#1\xint_c_mone
182   0-\xint_c_i
183   \krof
184 }%

```

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".

```

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

```

4.9 \xintiiAbs

Attention \xintiiAbs non robust against non terminated input.

```

199 \def\xintiiAbs {\romannumeral0\xintiiabs }%
200 \def\xintiiabs #1%
201 {%
202   \expandafter\XINT_abs \romannumeral`&&@#1%
203 }%

```

```

204 \def\XINT_abs #1%
205 {%
206     \xint_UDsignfork
207     #1{ }%
208     -{ #1}%
209     \krof
210 }%
211 \def\XINT_Abs #1%
212 {%
213     \xint_UDsignfork
214     #1{ }%
215     -{#1}%
216     \krof
217 }%

```

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`.

```

218 \def\xintFDg {\romannumeral0\xintfdg }%
219 \def\xintfdg #1{\expandafter\XINT_fdg \romannumeral`&&@#1\xint:\Z}%
220 \def\XINT_FDg #1%
221     {\romannumeral0\expandafter\XINT_fdg\romannumeral`&&@\xintnum{#1}\xint:\Z }%
222 \def\XINT_fdg #1#2#3\Z
223 {%
224     \xint_UDzerominusfork
225     #1-{ 0}%    zero
226     0#1{ #2}%   negative
227     0-{ #1}%    positive
228     \krof
229 }%

```

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 `\xintiiLDg`, gives to `\xintLDg` former meaning of `\xintiiLDg`.

Attention \xintLDg non robust against non terminated input.

```

230 \def\xintLDg {\romannumeral0\xintldg }%
231 \def\xintldg #1{\expandafter\XINT_ldg_fork\romannumeral`&&@#1%
232   \XINT_ldg_c{}{}{}{}{}{}{}{}{}\xint_bye\relax}%
233 \def\XINT_ldg_fork #1%
234 {%
235     \xint_UDsignfork
236       #1\XINT_ldg
237       -{\XINT_ldg#1}%
238     \krof
239 }%
240 \def\XINT_ldg #1{%
241 \def\XINT_ldq ##1##2###3####4###5###6###7###8###9%
```



```

242   {\expandafter#1%
243     \the\numexpr##9##8##7##6##5##4##3##2##1*\xint_c_+\XINT_ldg_a##9}%
244 }\XINT_ldg{ }%
245 \def\XINT_ldg_a#1#2{\XINT_ldg_cbye#2\XINT_ldg_d#1\XINT_ldg_c\XINT_ldg_b#2}%
246 \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}%
247 \def\XINT_ldg_c    #1#2\xint_bye{#1}%
248 \def\XINT_ldg_cbye #1\XINT_ldg_c{}}%
249 \def\XINT_ldg_d#1#2\xint_bye{#1}%

```

4.12 \xintDouble

Attention \xintDouble non robust against non terminated input.

```

250 \def\xintDouble {\romannumeral0\xintdouble}%
251 \def\xintdouble #1{\expandafter\XINT_dbl_fork\romannumeral`&&@#1%
252   \xint_bye2345678\xint_bye*\xint_c_ii\relax}%
253 \def\XINT_dbl_fork #1%
254 {%
255   \xint_UDsignfork
256   #1\XINT_dbl_neg
257   -\XINT_dbl
258   \krof #1%
259 }%
260 \def\XINT_dbl_neg-{\expandafter-\romannumeral0\XINT_dbl}%
261 \def\XINT_dbl #1{%
262 \def\XINT_dbl ##1##2##3##4##5##6##7##8%
263   {\expandafter#1\the\numexpr##1##2##3##4##5##6##7##8\XINT_dbl_a}%
264 }\XINT_dbl{ }%
265 \def\XINT_dbl_a #1#2#3#4#5#6#7#8%
266   {\expandafter\XINT_dbl_e\the\numexpr 1#1#2#3#4#5#6#7#8\XINT_dbl_a}%
267 \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.

```

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

```

```
286 {\expandafter\XINT_half_e\the\numexpr(1#1#2#3#4#5#6#7#8\XINT_half_a}%
287 \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.

```
288 \def\xintInc {\romannumeral0\xintinc}%
289 \def\xintinc #1{\expandafter\XINT_inc_fork\romannumeral`&&@#1%
290      \xint_bye23456789\xint_bye+ \xint_c_i\relax}%
291 \def\XINT_inc_fork #1%
292 {%
293     \xint_UDsignfork
294     #1\XINT_inc_neg
295     -\XINT_inc
296     \krof #1%
297 }%
298 \def\XINT_inc_neg-#1\xint_bye#2\relax
299     {\xintiopp\XINT_dec #1\XINT_dec_bye234567890\xint_bye}%
300 \def\XINT_inc #1{%
301 \def\XINT_inc ##1##2##3##4##5##6##7##8##9%
302     {\expandafter#1\the\numexpr##1##2##3##4##5##6##7##8##9\XINT_inc_a}%
303 }\XINT_inc{ }%
304 \def\XINT_inc_a #1#2#3#4#5#6#7#8#9%
305     {\expandafter\XINT_inc_e\the\numexpr 1#1#2#3#4#5#6#7#8#9\XINT_inc_a}%
306 \def\XINT_inc_e#1{\if12+ \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.

```
307 \def\xintDec {\romannumeral0\xintdec}%
308 \def\xintdec #1{\expandafter\XINT_dec_fork\romannumeral`&&@#1%
309     \XINT_dec_bye234567890\xint_bye}%
310 \def\XINT_dec_fork #1%
311 {%
312     \xint_UDsignfork
313     #1\XINT_dec_neg
314     -\XINT_dec
315     \krof #1%
316 }%
317 \def\XINT_dec_neg-#1\XINT_dec_bye#2\xint_bye
318     {\expandafter-%
319     \romannumeral0\XINT_inc #1\xint_bye23456789\xint_bye+ \xint_c_i\relax}%
320 \def\XINT_dec #1{%
321 \def\XINT_dec ##1##2##3##4##5##6##7##8##9%
322     {\expandafter#1\the\numexpr##1##2##3##4##5##6##7##8##9\XINT_dec_a}%
323 }\XINT_dec{ }%
324 \def\XINT_dec_a #1#2#3#4#5#6#7#8#9%
```

```

325 {\expandafter\XINT_dec_e\the\numexpr 1#1#2#3#4#5#6#7#8#9\XINT_dec_a}%
326 \def\XINT_dec_bye #1\XINT_dec_a#2#3\xint_bye
327 {\if#20-\xint_c_ii\relax+\else-\fi\xint_c_i\relax}%
328 \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.

```

329 \def\xintDSL {\romannumeral0\xintdsl}%
330 \def\xintdsl #1{\expandafter\XINT_dsl\romannumeral`&&@#10}%
331 \def\XINT_dsl#1{%
332 \def\XINT_dsl ##1{\xint_gob_til_zero ##1\xint_dsl_zero 0#1##1}%
333 }\XINT_dsl{ }%
334 \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.

```

335 \def\xintDSR{\romannumeral0\xintdsr}%
336 \def\xintdsr #1{\expandafter\XINT_dsr_fork\romannumeral`&&@#1%
337 \xint_bye\xint_Bye3456789\xint_bye+\xint_c_v)/\xint_c_x-\xint_c_i\relax}%
338 \def\XINT_dsr_fork #1%
339 {%
340 \xint_UDsignfork
341 #1\XINT_dsr_neg
342 -\XINT_dsr
343 \krof #1%
344 }%
345 \def\XINT_dsr_neg-{\xintiio\XINT_dsr}%
346 \def\XINT_dsr #1{%
347 \def\XINT_dsr ##1##2##3##4##5##6##7##8##9%
348 {\expandafter#1\the\numexpr(##1##2##3##4##5##6##7##8##9\XINT_dsr_a}%
349 }\XINT_dsr{ }%
350 \def\XINT_dsr_a#1{\xint_Bye#1\xint_bye\XINT_dsr_b#1}%
351 \def\XINT_dsr_b #1#2#3#4#5#6#7#8#9%
352 {\expandafter\XINT_dsr_e\the\numexpr(1#1#2#3#4#5#6#7#8#9\XINT_dsr_a}%
353 \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.

```

354 \def\xintDSRr{\romannumeral0\xintdsrr}%
355 \def\xintdsrr #1{\expandafter\XINT_dsrr_fork\romannumeral`&&@#1%
356 \xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax}%
357 \def\XINT_dsrr_fork #1%

```

```

358 {%
359     \xint_UDsignfork
360     #1\XINT_dsrr_neg
361     -\XINT_dsrr
362     \krof #1%
363 }%
364 \def\XINT_dsrr_neg-{\xintiiopp\XINT_dsrr}%
365 \def\XINT_dsrr #1{%
366 \def\XINT_dsrr ##1##2##3##4##5##6##7##8##9%
367     {\expandafter#1\the\numexpr##1##2##3##4##5##6##7##8##9\XINT_dsrr_a}%
368 }\XINT_dsrr{ }%
369 \def\XINT_dsrr_a#1{\xint_Bye#1\xint_bye\XINT_dsrr_b#1}%
370 \def\XINT_dsrr_b #1#2#3#4#5#6#7#8#9%
371     {\expandafter\XINT_dsrr_e\the\numexpr1#1#2#3#4#5#6#7#8#9\XINT_dsrr_a}%
372 \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.

```

373 \def\XINT_cuz #1{%
374 \def\XINT_cuz {\expandafter#1\the\numexpr\XINT_cuz_loop}%
375 }\XINT_cuz{ }%
376 \def\XINT_cuz_loop #1#2#3#4#5#6#7#8#9%
377 {%
378     #1#2#3#4#5#6#7#8%
379     \xint_gob_til_R #9\XINT_cuz_hitend\R
380     \ifnum #1#2#3#4#5#6#7#8>\xint_c_
381         \expandafter\XINT_cuz_cleantoend
382     \else\expandafter\XINT_cuz_loop
383     \fi #9%
384 }%
385 \def\XINT_cuz_hitend\R #1\R{\relax}%
386 \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

```

387 \def\XINT_cuz_byviii #1#2#3#4#5#6#7#8#9%
388 {%
389     \xint_gob_til_R #9\XINT_cuz_byviii_e \R
390     \xint_gob_til_eightzeroes #1#2#3#4#5#6#7#8\XINT_cuz_byviii_z 00000000%

```

```

391 \XINT_cuz_byviii_done #1#2#3#4#5#6#7#8#9%
392 }%
393 \def\XINT_cuz_byviii_z 00000000\XINT_cuz_byviii_done 00000000{\XINT_cuz_byviii}%
394 \def\XINT_cuz_byviii_done #1\R { #1}%
395 \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>!)
\mint_bye!2!3!4!5!6!7!8!9!\mint_bye\mint_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 \mintInc etc... and is compatible with some 1! block without digits among the treated blocks, they will disappear.

```

396 \def\XINT_unsep_loop #1!#2!#3!#4!#5!#6!#7!#8!#9!%
397 {%
398 \expandafter\XINT_unsep_clean
399 \the\numexpr #1\expandafter\XINT_unsep_clean
400 \the\numexpr #2\expandafter\XINT_unsep_clean
401 \the\numexpr #3\expandafter\XINT_unsep_clean
402 \the\numexpr #4\expandafter\XINT_unsep_clean
403 \the\numexpr #5\expandafter\XINT_unsep_clean
404 \the\numexpr #6\expandafter\XINT_unsep_clean
405 \the\numexpr #7\expandafter\XINT_unsep_clean
406 \the\numexpr #8\expandafter\XINT_unsep_clean
407 \the\numexpr #9\XINT_unsep_loop
408 }%
409 \def\XINT_unsep_clean 1{\relax}%

```

4.22 \XINT_unsep_cuzsmall

This is used as

```

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

```

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

Redone for 1.21: the 1.2 variant was strangely in $O(N^2)$ style.

```

410 \def\XINT_unsep_cuzsmall
411 {%
412 \expandafter\XINT_unsep_cuzsmall_x\the\numexpr0\XINT_unsep_loop
413 }%
414 \def\XINT_unsep_cuzsmall_x #1{%
415 \def\XINT_unsep_cuzsmall_x 0##1##2##3##4##5##6##7##8%
416 {%
417 \expandafter#1\the\numexpr ##1##2##3##4##5##6##7##8\relax
418 }}\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.

```

419 \def\XINT_div_unsepQ_delim {\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax\Z}%
420 \def\XINT_div_unsepQ
421 {%
422   \expandafter\XINT_div_unsepQ_x\the\numexpr0\XINT_unsep_loop
423 }%
424 \def\XINT_div_unsepQ_x #1{%
425 \def\XINT_div_unsepQ_x 0##1##2##3##4##5##6##7##8##9%
426 {%
427   \xint_gob_til_Z ##9\XINT_div_unsepQ_one\Z
428   \xint_gob_til_eightzeroes ##1##2##3##4##5##6##7##8\XINT_div_unsepQ_y 00000000%
429   \expandafter#1\the\numexpr ##1##2##3##4##5##6##7##8\relax ##9%
430 }}\XINT_div_unsepQ_x{ }%
431 \def\XINT_div_unsepQ_y #1{%
432 \def\XINT_div_unsepQ_y ##1\relax ##2##3##4##5##6##7##8##9%
433 {%
434   \expandafter#1\the\numexpr ##2##3##4##5##6##7##8##9\relax
435 }}\XINT_div_unsepQ_y{ }%
436 \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.

```

437 \def\XINT_div_unsepR
438 {%
439   \expandafter\XINT_div_unsepR_x\the\numexpr0\XINT_unsep_loop
440 }%
441 \def\XINT_div_unsepR_x#1{%
442 \def\XINT_div_unsepR_x 0{\expandafter#1\the\numexpr\XINT_cuz_loop}%
443 }\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.

```

444 \def\XINT_zeroes_forviii #1#2#3#4#5#6#7#8%
445 {%
446   \xint_gob_til_R #8\XINT_zeroes_forviii_end\R\XINT_zeroes_forviii
447 }%
448 \def\XINT_zeroes_forviii_end#1{%
449 \def\XINT_zeroes_forviii_end\R\XINT_zeroes_forviii ##1##2##3##4##5##6##7##8##9\W

```

```
450 {%
451     \expandafter#1\xint_gob_til_one ##2##3##4##5##6##7##8%
452 }}\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.

```
453 \def\XINT_sepbyviii_Z #1#2#3#4#5#6#7#8%
454 {%
455     1#1#2#3#4#5#6#7#8\expandafter!\the\numexpr\XINT_sepbyviii_Z
456 }%
457 \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.

```
458 \def\XINT_sepbyviii_andcount
459 {%
460     \expandafter\XINT_sepbyviii_andcount_a\the\numexpr\XINT_sepbyviii
461 }%
462 \def\XINT_sepbyviii #1#2#3#4#5#6#7#8%
463 {%
464     1#1#2#3#4#5#6#7#8\expandafter!\the\numexpr\XINT_sepbyviii
465 }%
466 \def\XINT_sepbyviii_end #1\relax {\relax\XINT_sepbyviii_andcount_end!}%
467 \def\XINT_sepbyviii_andcount_a {\XINT_sepbyviii_andcount_b \xint_c_\xint:}%
468 \def\XINT_sepbyviii_andcount_b #1\xint:#2!#3!#4!#5!#6!#7!#8!#9!%
469 {%
470     #2\expandafter!\the\numexpr#3\expandafter!\the\numexpr#4\expandafter
471     !\the\numexpr#5\expandafter!\the\numexpr#6\expandafter!\the\numexpr
472     #7\expandafter!\the\numexpr#8\expandafter!\the\numexpr#9\expandafter!\the\numexpr
473     \expandafter\XINT_sepbyviii_andcount_b\the\numexpr #1+\xint_c_viii\xint:%
474 }%
475 \def\XINT_sepbyviii_andcount_end #1\XINT_sepbyviii_andcount_b\the\numexpr
476     #2+\xint_c_viii\xint:#3#4\W {\expandafter\xint:\the\numexpr #2+#3\xint:}%

```

4.28 \XINT_rsepbyviii

This is used as

```
\the\numexpr1\XINT_rsepbyviii <8Ndigits>%
```

```

\XINT_rsepbyviii_end_A 2345678%
\XINT_rsepbyviii_end_B 2345678\relax UV%
and will produce
1<8digits>!1<8digits>\xint:1<8digits>!...
where the original digits are organized by eight, and the order inside successive pairs of blocks
separated by \xint: has been reversed. Output ends either in 1<8d>!1<8d>\xint:1U\xint: (even) or
1<8d>!1<8d>\xint:1V!1<8d>\xint: (odd)
The U an V should be \numexpr1 stoppers (or will expand and be ended by !). This macro is currently
(1.2..1.21) exclusively used in combination with \XINT_sepandrev_andcount or \XINT_sepandrev.
477 \def\XINT_rsepbyviii #1#2#3#4#5#6#7#8%
478 {%
479   \XINT_rsepbyviii_b {#1#2#3#4#5#6#7#8}%
480 }%
481 \def\XINT_rsepbyviii_b #1#2#3#4#5#6#7#8#9%
482 {%
483   #2#3#4#5#6#7#8#9\expandafter!\the\numexpr
484   1#1\expandafter\xint:\the\numexpr 1\XINT_rsepbyviii
485 }%
486 \def\XINT_rsepbyviii_end_B #1\relax #2#3{#2\xint:}%
487 \def\XINT_rsepbyviii_end_A #1#2\expandafter #3\relax #4#5{#5!1#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:\R
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 \nu-
mexpr1 stoppers) to share same syntax as \XINT_sepandrev_andcount, they are gobbled (#2 in
\XINT_sepandrev_done).
488 \def\XINT_sepandrev
489 {%
490   \expandafter\XINT_sepandrev_a\the\numexpr 1\XINT_rsepbyviii
491 }%
492 \def\XINT_sepandrev_a {\XINT_sepandrev_b {}}%
493 \def\XINT_sepandrev_b #1#2\xint:#3\xint:#4\xint:#5\xint:#6\xint:#7\xint:#8\xint:#9\xint:%
494 {%
495   \xint_gob_til_R #9\XINT_sepandrev_end\R
496   \XINT_sepandrev_b {#9!#8!#7!#6!#5!#4!#3!#2!#1}%
497 }%
498 \def\XINT_sepandrev_end\R\XINT_sepandrev_b #1#2\W {\XINT_sepandrev_done #1}%
499 \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.
500 \def\xINT_sepandrev_andcount
501 {%
502   \expandafter\xINT_sepandrev_andcount_a\the\numexpr 1\xINT_rsepybyviii
503 }%
504 \def\xINT_sepandrev_andcount_a {\xINT_sepandrev_andcount_b 0!{}}%
505 \def\xINT_sepandrev_andcount_b #1!#2#3\xint:#4\xint:#5\xint:#6\xint:#7\xint:#8\xint:#9\xint:%
506 {%
507   \xint_gob_til_R #9\xINT_sepandrev_andcount_end\R
508   \expandafter\xINT_sepandrev_andcount_b \the\numexpr #1+\xint_c_i!%
509   {#9!#8!#7!#6!#5!#4!#3!#2}%
510 }%
511 \def\xINT_sepandrev_andcount_end\R
512   \expandafter\xINT_sepandrev_andcount_b\the\numexpr #1+\xint_c_i!#2#3#4\W
513 {\expandafter\xINT_sepandrev_andcount_done\the\numexpr #3+\xint_c_xiv*#1!#2}%
514 \def\xINT_sepandrev_andcount_done#1{%
515 \def\xINT_sepandrev_andcount_done##1!##2!##3!{\expandafter#1\the\numexpr##1-##3\xint:}%
516 }\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.

```

517 \def\xINT_rev_nounsep #1#2!#3!#4!#5!#6!#7!#8!#9!%
518 {%
519   \xint_gob_til_R #9\xINT_rev_nounsep_end\R
520   \XINT_rev_nounsep {#9!#8!#7!#6!#5!#4!#3!#2!#1}%
521 }%
522 \def\xINT_rev_nounsep_end\R\xINT_rev_nounsep #1#2\W {\XINT_rev_nounsep_done #1}%
523 \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\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).

```

524 \def\xINT_unrevbyviii #11#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
525 {%
526   \xint_gob_til_R #9\xINT_unrevbyviii_a\R
527   \XINT_unrevbyviii {#9#8#7#6#5#4#3#2#1}%
528 }%
529 \def\xINT_unrevbyviii_a#1{%
530 \def\xINT_unrevbyviii_a\R\xINT_unrevbyviii ##1##2\W
531   {\expandafter#1\romannumeral`&&@\xint_gob_til_sc ##1}%

```

```
532 }\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.

```
533 \def\XINT_smallunrevbyviii #1!1#2!1#3!1#4!1#5!1#6!1#7!1#8!#9\W%
534 {%
535     \expandafter\XINT_cuz_small\xint_gob_til_sc #8#7#6#5#4#3#2#1%
536 }%
```

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.

```
537 \def\xintiiAdd {\romannumeral0\xintiiadd }%
538 \def\xintiiadd #1{\expandafter\XINT_iiadd\romannumeral`&&@#1\xint:}%
539 \def\XINT_iiadd #1#2\xint:#3%
540 {%
541     \expandafter\XINT_add_nfork\expandafter#1\romannumeral`&&@#3\xint:#2\xint:
542 }%
543 \def\XINT_add_fork #1#2\xint:#3\xint:{\XINT_add_nfork #1#3\xint:#2\xint:}%
544 \def\XINT_add_nfork #1#2%
545 {%
546     \xint_UDzerofork
547     #1\XINT_add_firstiszero
548     #2\XINT_add_secondiszero
549     0{}}%
550 \krof
551 \xint_UDsignsfork
552     #1#2\XINT_add_minusminus
553     #1-\XINT_add_minusplus
554     #2-\XINT_add_plusminus
555     --\XINT_add_plusplus
556 \krof #1#2%
557 }%
558 \def\XINT_add_firstiszero #1\krof 0#2#3\xint:#4\xint:{ #2#3}%
559 \def\XINT_add_secondiszero #1\krof #20#3\xint:#4\xint:{ #2#4}%
560 \def\XINT_add_minusminus #1#2%
561     {\expandafter\romannumeral0\XINT_add_pp_a {}}}%
562 \def\XINT_add_minusplus #1#2{\XINT_sub_mm_a {}}#2}%
563 \def\XINT_add_plusminus #1#2%
```

```

564   {\expandafter\XINT_opp\romannumeral0\XINT_sub_mm_a #1{}}%
565 \def\XINT_add_pp_a #1#2#3\xint:
566 {%
567   \expandafter\XINT_add_pp_b
568     \romannumeral0\expandafter\XINT_sepandrev_andcount
569     \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
570     #2#3\XINT_rsepbyviii_end_A 2345678%
571     \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
572     \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
573     \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
574   \X #1%
575 }%
576 \let\XINT_add_plusplus \XINT_add_pp_a
577 \def\XINT_add_pp_b #1\xint:#2\X #3\xint:
578 {%
579   \expandafter\XINT_add_checklengths
580   \the\numexpr #1\expandafter\xint:%
581   \romannumeral0\expandafter\XINT_sepandrev_andcount
582   \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
583   #3\XINT_rsepbyviii_end_A 2345678%
584   \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
585   \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
586   \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
587   1;!1;!1;!1;!1\W #21;!1;!1;!1;!1\W
588   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\W
589 }%

```

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 ?

```

590 \def\XINT_add_checklengths #1\xint:#2\xint:%
591 {%
592   \ifnum #2>#1
593     \expandafter\XINT_add_exchange
594   \else
595     \expandafter\XINT_add_A
596   \fi
597   #1\xint:#2\xint:%
598 }%
599 \def\XINT_add_exchange #1\xint:#2\xint:#3\W #4\W
600 {%
601   \XINT_add_A #2\xint:#1\xint:#4\W #3\W
602 }%
603 \def\XINT_add_A #1\xint:#2\xint:%
604 {%
605   \ifnum #1>\xint_c_vi
606     \expandafter\XINT_add_aa
607   \else \expandafter\XINT_add_aa_small
608   \fi
609 }%
610 \def\XINT_add_aa {\expandafter\XINT_add_out\the\numexpr\XINT_add_a \xint_c_ii}%
611 \def\XINT_add_out{\expandafter\XINT_cuz_small\romannumeral0\XINT_unrevbyviii {}}%
612 \def\XINT_add_aa_small
613   {\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;!!;!!;!!\W #21;!!;!!;!!\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).

```

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

```

```

658 }%
659 \def\XINT_add_g #1#2\xint:%
660 {%
661     1#2\expandafter!\the\numexpr\XINT_add_h #1%
662 }%
663 \def\XINT_add_h #1#2#3!#4!%
664 {%
665     \xint_gob_til_sc #2\XINT_add_hi ;%
666     \expandafter\XINT_add_i\the\numexpr#1+1#2#3+#4-\xint_c_ii\xint:%
667 }%
668 \def\XINT_add_hi;%
669     \expandafter\XINT_add_i\the\numexpr#1+#2+#3-\xint_c_ii\xint:#4W
670 {%
671     \XINT_add_k #1#3!%
672 }%
673 \def\XINT_add_i #1#2\xint:%
674 {%
675     1#2\expandafter!\the\numexpr\XINT_add_a #1%
676 }%
677 \def\XINT_add_k #1{\if #12\expandafter\XINT_add_ke\else\expandafter\XINT_add_l \fi}%
678 \def\XINT_add_ke #11;#2W {\XINT_add_kf #11;!}%
679 \def\XINT_add_kf 1{1\relax }%
680 \def\XINT_add_l 1#1#2{\xint_gob_til_sc #1\XINT_add_lf ;\XINT_add_m 1#1#2}%
681 \def\XINT_add_lf #1W {1\relax 00000001!1;!}%
682 \def\XINT_add_m #1!\{\expandafter\XINT_add_n\the\numexpr\xint_c_i+#1\xint:%}
683 \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>.)

```

684 \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.

```

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

```

```

702      --\XINT_cmp_plusplus
703      \krof #1#2%
704 }%
705 \def\XINT_cmp_firstiszero #1\krof 0#2#3\xint:#4\xint:
706 {%
707      \xint_UDzerominusfork
708      #2-{ 0}%
709      0#2{ 1}%
710      0-{ -1}%
711      \krof
712 }%
713 \def\XINT_cmp_secondiszero #1\krof #20#3\xint:#4\xint:
714 {%
715      \xint_UDzerominusfork
716      #2-{ 0}%
717      0#2{ -1}%
718      0-{ 1}%
719      \krof
720 }%
721 \def\XINT_cmp_plusminus #1\xint:#2\xint:{ 1}%
722 \def\XINT_cmp_minusplus #1\xint:#2\xint:{ -1}%
723 \def\XINT_cmp_minusminus
724      --{\expandafter\XINT_opp\romannumeral0\XINT_cmp_plusplus {}{}}%
725 \def\XINT_cmp_plusplus #1#2#3\xint:
726 {%
727      \expandafter\XINT_cmp_pp
728      \the\numexpr\expandafter\XINT_sepbyviii_andcount
729      \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
730      #2#3\XINT_sepbyviii_end 2345678\relax
731      \xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!%
732      \xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_\W
733      #1%
734 }%
735 \def\XINT_cmp_pp #1\xint:#2\xint:#3\xint:
736 {%
737      \expandafter\XINT_cmp_checklengths
738      \the\numexpr #2\expandafter\xint:%
739      \the\numexpr\expandafter\XINT_sepbyviii_andcount
740      \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
741      #3\XINT_sepbyviii_end 2345678\relax
742      \xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!%
743      \xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_\W
744      #1;!1;!1;!1;!1;\W
745 }%
746 \def\XINT_cmp_checklengths #1\xint:#2\xint:#3\xint:
747 {%
748      \ifnum #1=#3
749          \expandafter\xint_firstoftwo
750      \else
751          \expandafter\xint_secondoftwo
752      \fi
753      \XINT_cmp_a {\XINT_cmp_distinctlengths {#1}{#3}}#2;!1;!1;!1;\W

```

```

754 }%
755 \def\XINT_cmp_distinctlengths #1#2#3\W #4\W
756 {%
757     \ifnum #1>#2
758         \expandafter\xint_firstoftwo
759     \else
760         \expandafter\xint_secondoftwo
761     \fi
762     { -1}{ 1}%
763 }%
764 \def\XINT_cmp_a 1#1!1#2!1#3!1#4!#5\W 1#6!1#7!1#8!1#9!%
765 {%
766     \xint_gob_til_sc #1\XINT_cmp_equal ;%
767     \ifnum #1>#6 \XINT_cmp_gt\fi
768     \ifnum #1<#6 \XINT_cmp_lt\fi
769     \xint_gob_til_sc #2\XINT_cmp_equal ;%
770     \ifnum #2>#7 \XINT_cmp_gt\fi
771     \ifnum #2<#7 \XINT_cmp_lt\fi
772     \xint_gob_til_sc #3\XINT_cmp_equal ;%
773     \ifnum #3>#8 \XINT_cmp_gt\fi
774     \ifnum #3<#8 \XINT_cmp_lt\fi
775     \xint_gob_til_sc #4\XINT_cmp_equal ;%
776     \ifnum #4>#9 \XINT_cmp_gt\fi
777     \ifnum #4<#9 \XINT_cmp_lt\fi
778     \XINT_cmp_a #5\W
779 }%
780 \def\XINT_cmp_lt#1{\def\XINT_cmp_lt\fi ##1\W ##2\W {\fi#1-1}}\XINT_cmp_lt{ }%
781 \def\XINT_cmp_gt#1{\def\XINT_cmp_gt\fi ##1\W ##2\W {\fi#11}}\XINT_cmp_gt{ }%
782 \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_l_Ida. This bug shows only in rare circumstances which escaped out test suite :(Fixed at 1.2q.

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

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

```

783 \def\xintiiSub {\romannumeral0\xintiisub }%
784 \def\xintiisub #1{\expandafter\XINT_iisub\romannumeral`&&@#1\xint:}%

```

```

785 \def\XINT_iisub #1#2\xint:#3%
786 {%
787     \expandafter\XINT_sub_nfork\expandafter
788     #1\romannumeral`&&@#3\xint:#2\xint:
789 }%
790 \def\XINT_sub_nfork #1#2%
791 {%
792     \xint_UDzerofork
793     #1\XINT_sub_firstiszero
794     #2\XINT_sub_secondiszero
795     0{}}%
796     \krof
797     \xint_UDsignsfork
798     #1#2\XINT_sub_minusminus
799     #1-\XINT_sub_minusplus
800     #2-\XINT_sub_plusminus
801     --\XINT_sub_plusplus
802     \krof #1#2%
803 }%
804 \def\XINT_sub_firstiszero #1\krof 0#2#3\xint:#4\xint:{\XINT_opp #2#3}%
805 \def\XINT_sub_secondiszero #1\krof #20#3\xint:#4\xint:{ #2#4}%
806 \def\XINT_sub_plusminus #1#2{\XINT_add_pp_a #1{}}%
807 \def\XINT_sub_plusplus #1#2%
808     {\expandafter\XINT_opp\romannumeral0\XINT_sub_mm_a #1#2}%
809 \def\XINT_sub_minusplus #1#2%
810     {\expandafter-\romannumeral0\XINT_add_pp_a {}}#2}%
811 \def\XINT_sub_minusminus #1#2{\XINT_sub_mm_a {}}}%
812 \def\XINT_sub_mm_a #1#2#3\xint:
813 {%
814     \expandafter\XINT_sub_mm_b
815     \romannumeral0\expandafter\XINT_sepandrev_andcount
816     \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
817     #2#3\XINT_rsepbyviii_end_A 2345678%
818     \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
819     \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
820     \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
821     \X #1%
822 }%
823 \def\XINT_sub_mm_b #1\xint:#2\X #3\xint:
824 {%
825     \expandafter\XINT_sub_checklengths
826     \the\numexpr #1\expandafter\xint:%
827     \romannumeral0\expandafter\XINT_sepandrev_andcount
828     \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
829     #3\XINT_rsepbyviii_end_A 2345678%
830     \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
831     \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
832     \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
833     1;!1;!1;!1;!1;\W
834     #21;!1;!1;!1;!1;\W
835     1;!1\R!1\R!1\R!1\R!%
836     1\R!1\R!1\R!1\R!1\R!\W

```



```

837 }%
838 \def\XINT_sub_checklengths #1\xint:#2\xint:%
839 {%
840     \ifnum #2>#1
841         \expandafter\XINT_sub_exchange
842     \else
843         \expandafter\XINT_sub_aa
844     \fi
845 }%
846 \def\XINT_sub_exchange #1\W #2\W
847 {%
848     \expandafter\XINT_opp\romannumeral0\XINT_sub_aa #2\W #1\W
849 }%
850 \def\XINT_sub_aa
851 {%
852     \expandafter\XINT_sub_out\the\numexpr\XINT_sub_a\xint_c_i
853 }%

```

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.

```

854 \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 $1 < 8d < 10^8$, #1 (=B) *must* be at most as long as #2 (=A), (in radix 10^8) and the routine wants to compute $\#2 - \#1 = A - B$

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

\numexpr governed expansion stops with various possibilities:

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

Various type of post actions are then needed:

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

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

```

855 \def\XINT_sub_a #1!#2!#3!#4!#5\W #6!#7!#8!#9!%

```

```

856 {%
857     \XINT_sub_b
858     #1!#6!#2!#7!#3!#8!#4!#9!%
859     #5\W
860 }%

```

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.

```

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

```

```

905      #4!#5!#6!#7!#8\W
906 {%
907      \XINT_sub_k #1#2!#5!#7!%
908 }%
909 \def\XINT_sub_fi;%
910      \expandafter\XINT_sub_g\the\numexpr#1+1#2-#3\xint:
911      #4!#5!#6\W
912 {%
913      \XINT_sub_k #1#2!#5!%
914 }%
915 \def\XINT_sub_hi;%
916      \expandafter\XINT_sub_i\the\numexpr#1+1#2-#3\xint:
917      #4\W
918 {%
919      \XINT_sub_k #1#2!%
920 }%

```

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.

```

921 \def\XINT_sub_k #1#2#3%
922 {%
923      \xint_gob_til_sc #3\XINT_sub_p;\XINT_sub_l #1#2#3%
924 }%
925 \def\XINT_sub_l #1%
926      {\xint_UDzerofork #1\XINT_sub_l_carry 0\XINT_sub_l_Ia\krof}%
927 \def\XINT_sub_l_Ia 1#1!#2\W{1\relax#1;!1\XINT_sub_fix_none!}%

928 \def\XINT_sub_l_carry 1#1!{\ifcase #1
929      \expandafter \XINT_sub_l_Id
930      \or \expandafter \XINT_sub_l_Ic
931      \else\expandafter \XINT_sub_l_Ib\fi 1#1!}%
932 \def\XINT_sub_l_Ib #1#2\W {-\xint_c_i+#1;!1\XINT_sub_fix_none!}%
933 \def\XINT_sub_l_Ic 1#1!1#2#3!#4!#5\W
934 {%
935      \xint_gob_til_sc #2\XINT_sub_l_Ica;%
936      1\relax 00000000!1#2#3!#4!;!1\XINT_sub_fix_none!%
937 }%

```

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

```

938 \def\XINT_sub_l_Ica#1\W
939 {%
940      1;!1\XINT_sub_fix_cuz!%
941      1;!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
942      \xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z
943 }%
944 \def\XINT_sub_l_Id 1#1!%
945      {199999999\expandafter!\the\numexpr \XINT_sub_l_Id_a}%
946 \def\XINT_sub_l_Id_a 1#1!{\ifcase #1

```


1.21: \xintiiMul made robust against non terminated input.

```

981 \def\xintiiMul {\romannumeral0\xintiimul }%
982 \def\xintiimul #1%
983 {%
984   \expandafter\XINT_iimul\romannumeral`&&#1\xint:
985 }%
986 \def\XINT_iimul #1#2\xint:#3%
987 {%
988   \expandafter\XINT_mul_nfork\expandafter #1\romannumeral`&&#3\xint:#2\xint:
989 }%

```

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

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

```

990 \def\XINT_mul_fork #1#2\xint:#3\xint:{\XINT_mul_nfork #1#3\xint:#2\xint:}%
991 \def\XINT_mul_nfork #1#2%
992 {%
993   \xint_UDzerofork
994   #1\XINT_mul_zero
995   #2\XINT_mul_zero
996   0{}}%
997 \krof
998 \xint_UDsignsfork
999   #1#2\XINT_mul_minusminus
1000   #1-\XINT_mul_minusplus
1001   #2-\XINT_mul_plusminus
1002   --\XINT_mul_plusplus
1003 \krof #1#2%
1004 }%
1005 \def\XINT_mul_zero #1\krof #2#3\xint:#4\xint:{ 0}%
1006 \def\XINT_mul_minusminus #1#2{\XINT_mul_plusplus {}}}%
1007 \def\XINT_mul_minusplus #1#2%
1008   {\expandafter-\romannumeral0\XINT_mul_plusplus {}#2}%
1009 \def\XINT_mul_plusminus #1#2%
1010   {\expandafter-\romannumeral0\XINT_mul_plusplus #1{}}}%
1011 \def\XINT_mul_plusplus #1#2#3\xint:
1012 {%
1013   \expandafter\XINT_mul_pre_b
1014   \romannumeral0\expandafter\XINT_sepandrev_andcount
1015   \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
1016   #2#3\XINT_rsepybviii_end_A 2345678%
1017   \XINT_rsepybviii_end_B 2345678\relax\xint_c_ii\xint_c_i
1018   \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
1019   \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
1020   \W #1%
1021 }%
1022 \def\XINT_mul_pre_b #1\xint:#2\W #3\xint:
1023 {%
1024   \expandafter\XINT_mul_checklengths
1025   \the\numexpr #1\expandafter\xint:%
1026   \romannumeral0\expandafter\XINT_sepandrev_andcount
1027   \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
1028   #3\XINT_rsepybviii_end_A 2345678%
1029   \XINT_rsepybviii_end_B 2345678\relax\xint_c_ii\xint_c_i

```

```

1030          \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
1031          \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
1032      1;!\W #21;!\%
1033      1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1034 }%

```

Cooking recipe, 2015/10/05.

```

1035 \def\xint_mul_checklengths #1\xint:#2\xint:%
1036 {%
1037     \ifnum #2=\xint_c_i\expandafter\xint_mul_smallbyfirst\fi
1038     \ifnum #1=\xint_c_i\expandafter\xint_mul_smallbysecond\fi
1039     \ifnum #2<#1
1040         \ifnum \numexpr (#2-\xint_c_i)*(#1-#2)<383
1041             \XINT_mul_exchange
1042         \fi
1043     \else
1044         \ifnum \numexpr (#1-\xint_c_i)*(#2-#1)>383
1045             \XINT_mul_exchange
1046         \fi
1047     \fi
1048     \XINT_mul_start
1049 }%
1050 \def\xint_mul_smallbyfirst #1\xint_mul_start 1#2!1;!\W
1051 {%
1052     \ifnum#2=\xint_c_i\expandafter\xint_mul_oneisone\fi
1053     \ifnum#2<\xint_c_xxii\expandafter\xint_mul_verysmall\fi
1054     \expandafter\xint_mul_out\the\numexpr\xint_smallmul 1#2!%
1055 }%
1056 \def\xint_mul_smallbysecond #1\xint_mul_start #2\W 1#3!1;!\%
1057 {%
1058     \ifnum#3=\xint_c_i\expandafter\xint_mul_oneisone\fi
1059     \ifnum#3<\xint_c_xxii\expandafter\xint_mul_verysmall\fi
1060     \expandafter\xint_mul_out\the\numexpr\xint_smallmul 1#3!#2%
1061 }%
1062 \def\xint_mul_oneisone #1!\{\xint_mul_out }%
1063 \def\xint_mul_verysmall\expandafter\xint_mul_out
1064         \the\numexpr\xint_smallmul 1#1!%
1065     {\expandafter\xint_mul_out\the\numexpr\xint_verysmallmul 0\xint:#1!}%
1066 \def\xint_mul_exchange #1\xint_mul_start #2\W #31;!\%
1067     {\fi\fi\xint_mul_start #31;!\W #2}%

1068 \def\xint_mul_start
1069     {\expandafter\xint_mul_out\the\numexpr\xint_mul_loop 100000000!1;!\W}%
1070 \def\xint_mul_out
1071     {\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. 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.

```
1072 \def\XINT_mul_loop #1\W #2\W 1#3!%
1073 {%
1074   \xint_gob_til_sc #3\XINT_mul_e ;%
1075   \expandafter\XINT_mul_a\the\numexpr \XINT_smallmul 1#3!#2\W
1076   #1\W #2\W
1077 }%
```

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

```
1078 \def\XINT_mul_a #1\W #2\W
1079 {%
1080   \expandafter\XINT_mul_b\the\numexpr
1081   \XINT_add_a \xint_c_ii #21;!1;!1;!W #11;!1;!1;!W\W
1082 }%
1083 \def\XINT_mul_b 1#1!{1#1\expandafter!\the\numexpr\XINT_mul_loop }%
1084 \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.

```
1085 \def\XINT_minimulwc_a 1#1\xint:#2\xint:#3!#4#5#6#7#8\xint:%
1086 {%
1087   \expandafter\XINT_minimulwc_b
1088   \the\numexpr \xint_c_x^ix+#1+#3*#8\xint:
1089               #3*#4#5#6#7+#2*#8\xint:
1090               #2*#4#5#6#7\xint:%
1091 }%
1092 \def\XINT_minimulwc_b 1#1#2#3#4#5#6\xint:#7\xint:%
1093 {%
1094   \expandafter\XINT_minimulwc_c
1095   \the\numexpr \xint_c_x^ix+#1#2#3#4#5+#7\xint:#6\xint:%
1096 }%
1097 \def\XINT_minimulwc_c 1#1#2#3#4#5#6\xint:#7\xint:#8\xint:%
1098 {%
1099   1#6#7\expandafter!%
1100   \the\numexpr\expandafter\XINT_smallmul_a
1101   \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8\xint:%
1102 }%
1103 \def\XINT_smallmul 1#1#2#3#4#5!{\XINT_smallmul_a 100000000\xint:#1#2#3#4\xint:#5!}%
1104 \def\XINT_smallmul_a #1\xint:#2\xint:#3!1#4!%
1105 {%
1106   \xint_gob_til_sc #4\XINT_smallmul_e;%
1107   \XINT_minimulwc_a #1\xint:#2\xint:#3!#4\xint:#2\xint:#3!%
1108 }%
1109 \def\XINT_smallmul_e;\XINT_minimulwc_a 1#1\xint:#2;#3!%
1110   {\xint_gob_til_eightzeroes #1\XINT_smallmul_f 000000001\relax #1!1;!}%
```

```

1111 \def\XINT_smallmul_f 000000001\relax 00000000!1{1\relax}%
1112 \def\XINT_verysmallmul #1\xint:#2!1#3!%
1113 {%
1114     \xint_gob_til_sc #3\XINT_verysmallmul_e;%
1115     \expandafter\XINT_verysmallmul_a
1116     \the\numexpr #2*#3+#1\xint:#2!%
1117 }%
1118 \def\XINT_verysmallmul_e;\expandafter\XINT_verysmallmul_a\the\numexpr
1119     #1+#2#3\xint:#4!%
1120 {\xint_gob_til_zero #2\XINT_verysmallmul_f 0\xint_c_x^viii+#2#3!1;!}%
1121 \def\XINT_verysmallmul_f #1!1{1\relax}%
1122 \def\XINT_verysmallmul_a #1#2\xint:%
1123 {%
1124     \unless\ifnum #1#2<\xint_c_x^ix
1125     \expandafter\XINT_verysmallmul_bi\else
1126     \expandafter\XINT_verysmallmul_bj\fi
1127     \the\numexpr \xint_c_x^ix+#1#2\xint:%
1128 }%
1129 \def\XINT_verysmallmul_bj{\expandafter\XINT_verysmallmul_cj }%
1130 \def\XINT_verysmallmul_cj 1#1#2\xint:%
1131     {1#2\expandafter!\the\numexpr\XINT_verysmallmul #1\xint:}%
1132 \def\XINT_verysmallmul_bi\the\numexpr\xint_c_x^ix+#1#2#3\xint:%
1133     {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.

```

1134 \def\XINT_minimul_a #1\xint:#2!#3#4#5#6#7!%
1135 {%
1136     \expandafter\XINT_minimul_b
1137     \the\numexpr \xint_c_x^viii+#2*#7\xint:#2*#3#4#5#6+#1*#7\xint:#1*#3#4#5#6\xint:%
1138 }%
1139 \def\XINT_minimul_b 1#1#2#3#4#5\xint:#6\xint:%
1140 {%
1141     \expandafter\XINT_minimul_c
1142     \the\numexpr \xint_c_x^ix+#1#2#3#4+#6\xint:#5\xint:%
1143 }%
1144 \def\XINT_minimul_c 1#1#2#3#4#5#6\xint:#7\xint:#8\xint:%
1145 {%
1146     1#6#7\expandafter!\the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8!%
1147 }%

```

4.37 \xintiiDivision

Completely rewritten for 1.2.

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

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

```
1148 \def\xintiDivision    {\romannumeral0\xintiDivision}%
1149 \def\xintiDivision    #1{\expandafter\XINT_iidivision \romannumeral`&&@#1\xint:}%
1150 \def\XINT_iidivision    #1#2\xint:#3{\expandafter\XINT_iidivision_a\expandafter #1%
1151                                \romannumeral`&&@#3\xint:#2\xint:}%
```

On regarde les signes de A et de B.

```
1152 \def\XINT_iidivision_a #1#2% #1 de A, #2 de B.
1153 {%
1154     \if0#2\xint_dothis{\XINT_iidivision_divbyzero #1#2}\fi
1155     \if0#1\xint_dothis\XINT_iidivision_aiszero\fi
1156     \if-#2\xint_dothis{\expandafter\XINT_iidivision_bneg
1157                         \romannumeral0\XINT_iidivision_bpos #1}\fi
1158     \xint_orthat{\XINT_iidivision_bpos #1#2}%
1159}%
1160 \def\XINT_iidivision_divbyzero#1#2#3\xint:#4\xint:
1161     {\if0#1\xint_dothis{\XINT_signalcondition{DivisionUndefined}}\fi
1162     \xint_orthat{\XINT_signalcondition{DivisionByZero}}%
1163     {Division by zero: #1#4/#2#3.}}{\{0\}{0}}}%
1164 \def\XINT_iidivision_aiszero #1\xint:#2\xint:{\{0\}{0}}}%
1165 \def\XINT_iidivision_bneg #1% q->-q, r unchanged
1166     {\expandafter{\romannumeral0\XINT_opp #1}}%
1167 \def\XINT_iidivision_bpos #1%
1168 {%
1169     \xint_UDsignfork
1170     #1\XINT_iidivision_aneg
1171     -{\XINT_iidivision_apos #1}%
1172     \krof
1173}%
```

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.

```
1174 \def\XINT_iidivision_apos #1#2\xint:#3\xint:{\XINT_div_prepare {#2}{#1#3}}%
1175 \def\XINT_iidivision_aneg #1\xint:#2\xint:
1176     {\expandafter
1177     \XINT_iidivision_aneg_b\romannumeral0\XINT_div_prepare {#1}{#2}{#1}}%
1178 \def\XINT_iidivision_aneg_b #1#2{\if0\XINT_Sgn #2\xint:
1179     \expandafter\XINT_iidivision_aneg_rzero
1180     \else
1181     \expandafter\XINT_iidivision_aneg_rpos
1182     \fi {#1}{#2}}%
1183 \def\XINT_iidivision_aneg_rzero #1#2#3{\{-#1\}{0}}% necessarily q was >0
1184 \def\XINT_iidivision_aneg_rpos #1%
```

```

1185 {%
1186     \expandafter\XINT_iidivision_aneg_end\expandafter
1187         {\expandafter-\romannumeral0\xintinc {#1}}% q-> -(1+q)
1188 }%
1189 \def\XINT_iidivision_aneg_end #1#2#3%
1190 {%
1191     \expandafter\xint_exchangetwo_keepbraces
1192     \expandafter{\romannumeral0\XINT_sub_mm_a {}}{#3\xint:#2\xint:}{#1}% r-> b-r
1193 }%

```

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.

```

1194 \def\XINT_div_prepare #1%
1195 {%
1196     \XINT_div_prepare_a #1\R\R\R\R\R\R\R {10}0000001\W !{#1}%
1197 }%
1198 \def\XINT_div_prepare_a #1#2#3#4#5#6#7#8#9%
1199 {%
1200     \xint_gob_til_R #9\XINT_div_prepare_small\R
1201     \XINT_div_prepare_b #9%
1202 }%

```

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).

```

1203 \def\XINT_div_prepare_small\R #1!#2%
1204 {%
1205     \ifcase #2
1206     \or\expandafter\XINT_div_BisOne
1207     \or\expandafter\XINT_div_BisTwo
1208     \else\expandafter\XINT_div_small_a
1209     \fi {#2}%
1210 }%
1211 \def\XINT_div_BisOne #1#2{{#2}{0}}%
1212 \def\XINT_div_BisTwo #1#2%
1213 {%
1214     \expandafter\expandafter\expandafter\XINT_div_BisTwo_a
1215     \ifodd\xintLDg{#2} \expandafter1\else \expandafter0\fi {#2}%
1216 }%
1217 \def\XINT_div_BisTwo_a #1#2%
1218 {%
1219     \expandafter{\romannumeral0\XINT_half
1220     #2\xint_bye\xint_Bye345678\xint_bye
1221     *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax}{#1}%
1222 }%

```

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.

```

1223 \def\XINT_div_small_a #1#2%
1224 {%
1225     \expandafter\XINT_div_small_b
1226     \the\numexpr #1/\xint_c_ii\expandafter
1227     \xint:\the\numexpr \xint_c_x^viii+#1\expandafter!%
1228     \romannumeral0%
1229     \XINT_div_small_ba #2\R\R\R\R\R\R\R{10}0000001\W

```

```
1230      #2\XINT_sepbyviii_Z_end 2345678\relax
1231 }%
```

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

```
1232 \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 $A<10^8$.

```
1233 \def\XINT_div_small_ba #1#2#3#4#5#6#7#8#9%
1234 {%
1235   \xint_gob_til_R #9\XINT_div_smallsmall\R
1236   \expandafter\XINT_div_dosmalldiv
1237   \the\numexpr\expandafter\XINT_sepbyviii_Z
1238   \romannumeral0\XINT_zeroes_forviii
1239   #1#2#3#4#5#6#7#8#9%
1240 }%
```

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

```
1241 \def\XINT_div_smallsmall\R
1242   \expandafter\XINT_div_dosmalldiv
1243   \the\numexpr\expandafter\XINT_sepbyviii_Z
1244   \romannumeral0\XINT_zeroes_forviii #1\R #2\relax
1245   {\XINT_div_dosmallsmall}\{#1}\}%
1246 \def\XINT_div_dosmallsmall #1\xint:1#2!#3%
1247 {%
1248   \expandafter\XINT_div_smallsmallend
1249   \the\numexpr (#3+#1)/#2-\xint_c_i\xint:#2\xint:#3\xint:%
1250 }%
1251 \def\XINT_div_smallsmallend #1\xint:#2\xint:#3\xint:{\expandafter
1252   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #3-#1*#2}\}%
```

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

```
1253 \def\XINT_div_dosmalldiv
1254   {\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.

```
1255 \def\XINT_div_prepare_b
1256   {\expandafter\XINT_div_prepare_c\romannumeral0\XINT_zeroes_forviii }%
1257 \def\XINT_div_prepare_c #1!%
1258 {%
1259   \XINT_div_prepare_d #1.00000000!\{#1}\}%
1260 }%
1261 \def\XINT_div_prepare_d #1#2#3#4#5#6#7#8#9%
1262 {%
1263   \expandafter\XINT_div_prepare_e\xint_gob_til_dot #1#2#3#4#5#6#7#8#9!%
1264 }%
1265 \def\XINT_div_prepare_e #1!#2!#3#4%
1266 {%
```

```
1267 \XINT_div_prepare_f #4#3\X {#1}{#3}%
1268 }%
```

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.

```
1269 \def\XINT_div_prepare_f #1#2#3#4#5#6#7#8#9\X
1270 {%
1271 \expandafter\XINT_div_prepare_g
1272 \the\numexpr #1#2#3#4#5#6#7#8+\xint_c_i\expandafter
1273 \xint:\the\numexpr (#1#2#3#4#5#6#7#8+\xint_c_i)/\xint_c_ii\expandafter
1274 \xint:\the\numexpr #1#2#3#4#5#6#7#8\expandafter
1275 \xint:\romannumeral0\XINT_sepandrev_andcount
1276 #1#2#3#4#5#6#7#8#9\XINT_rsepbyviii_end_A 2345678%
1277 \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
1278 \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
1279 \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
1280 \X
1281 }%
1282 \def\XINT_div_prepare_g #1\xint:#2\xint:#3\xint:#4\xint:#5\X #6#7#8%
1283 {%
1284 \expandafter\XINT_div_prepare_h
1285 \the\numexpr\expandafter\XINT_sepbyviii_andcount
1286 \romannumeral0\XINT_zeroes_forviii #8#7\R\R\R\R\R\R\R\{10}0000001\W
1287 #8#7\XINT_sepbyviii_end 2345678\relax
1288 \xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!%
1289 \xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_\W
1290 {#1}{#2}{#3}{#4}{#5}{#6}%
1291 }%
1292 \def\XINT_div_prepare_h #11\xint:#2\xint:#3#4#5#6#7#8%
1293 {%
1294 \XINT_div_start_a {#2}{#6}{#1}{#3}{#4}{#5}{#7}{#8}%
1295 }%
```

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`.

```
1296 \def\XINT_div_start_a #1#2%
1297 {%
1298 \ifnum #1 < #2
1299 \expandafter\XINT_div_zeroQ
1300 \else
1301 \expandafter\XINT_div_start_b
1302 \fi
1303 {#1}{#2}%
1304 }%
1305 \def\XINT_div_zeroQ #1#2#3#4#5#6#7%
1306 {%
1307 \expandafter\XINT_div_zeroQ_end
1308 \romannumeral0\XINT_unsep_cuzsmall
1309 #3\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax\xint:
```

```

1310 }%
1311 \def\XINT_div_zeroQ_end #1\xint:#2%
1312   {\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»
1313 \def\XINT_div_start_b #1#2#3#4#5#6%
1314 {%
1315   \expandafter\XINT_div_finish\the\numexpr
1316   \XINT_div_start_c {#2}\xint:#3\xint:{#6}{#{1}{#2}{#{4}{#5}}{#6}}%
1317 }%
1318 \def\XINT_div_finish
1319 {%
1320   \expandafter\XINT_div_finish_a \romannumeral`&&\XINT_div_unsepQ
1321 }%
1322 \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».
1323 \def\XINT_div_finish_b #1%
1324 {%
1325   \if0#1%
1326     \expandafter\XINT_div_finish_bRzero
1327   \else
1328     \expandafter\XINT_div_finish_bRpos
1329   \fi
1330   #1%
1331 }%
1332 \def\XINT_div_finish_bRzero 0\xint:#1#2{{#1}{0}}%
1333 \def\XINT_div_finish_bRpos #1\xint:#2#3%
1334 {%
1335   \expandafter\xint_exchangetwo_keepbraces\XINT_div_cleanR #1#3\xint:{#2}%
1336 }%
1337 \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.
1338 \def\XINT_div_start_c #1%
1339 {%
1340   \ifnum #1>\xint_c_vi
1341     \expandafter\XINT_div_start_ca
1342   \else
1343     \expandafter\XINT_div_start_cb
1344   \fi {#1}%
1345 }%
1346 \def\XINT_div_start_ca #1#2\xint:#3!#4!#5!#6!#7!#8!#9!%
1347 {%
1348   \expandafter\XINT_div_start_c\expandafter
1349   {\the\numexpr #1-\xint_c_vii}#2#3!#4!#5!#6!#7!#8!#9!\xint:%
1350 }%
1351 \def\XINT_div_start_cb #1%
1352   {\csname XINT_div_start_c\romannumeral\numexpr#1\endcsname}%
1353 \def\XINT_div_start_c_i #1\xint:#2!%
1354   {\XINT_div_start_c_ #1#2!\xint:}%
1355 \def\XINT_div_start_c_ii #1\xint:#2!#3!%
1356   {\XINT_div_start_c_ #1#2!#3!\xint:}%

```

```

1357 \def\XINT_div_start_c_iii #1\xint:#2!#3!#4!%
1358     {\XINT_div_start_c_   #1#2!#3!#4!\xint:}%
1359 \def\XINT_div_start_c_iv  #1\xint:#2!#3!#4!#5!%
1360     {\XINT_div_start_c_   #1#2!#3!#4!#5!\xint:}%
1361 \def\XINT_div_start_c_v   #1\xint:#2!#3!#4!#5!#6!%
1362     {\XINT_div_start_c_   #1#2!#3!#4!#5!#6!\xint:}%
1363 \def\XINT_div_start_c_vi  #1\xint:#2!#3!#4!#5!#6!#7!%
1364     {\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».
1365 \def\XINT_div_start_c_ 1#1!#2\xint:#3\xint:#4#5#6%
1366 {%
1367     \XINT_div_I_a {#1}{#4}{1#1!#2}{#6}{0000000000}#5{#3}{#6}%
1368 }%

    Ceci est le point de retour de la boucle principale. a, x, alpha, B, q0, L, K, {x'y}, x, alpha',
    B«c»
1369 \def\XINT_div_I_a #1#2%
1370 {%
1371     \expandafter\XINT_div_I_b\the\numexpr #1/#2\xint:{#1}{#2}%
1372 }%
1373 \def\XINT_div_I_b #1%
1374 {%
1375     \xint_gob_til_zero #1\XINT_div_I_czero 0\XINT_div_I_c #1%
1376 }%

    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».
1377 \def\XINT_div_I_czero 0\XINT_div_I_c 0\xint:#1#2#3#4#5{1#5\XINT_div_I_g {#3}}%
1378 \def\XINT_div_I_c #1\xint:#2#3%
1379 {%
1380     \expandafter\XINT_div_I_da\the\numexpr #2-#1*#3\xint:#1\xint:{#2}{#3}%
1381 }%

    r.q.alpha, B, q0, L, K, {x'y}, x, alpha', B«c»
1382 \def\XINT_div_I_da #1\xint:%
1383 {%
1384     \ifnum #1>\xint_c_ix
1385         \expandafter\XINT_div_I_dP
1386     \else
1387         \ifnum #1<\xint_c_
1388             \expandafter\expandafter\expandafter\XINT_div_I_dN
1389         \else
1390             \expandafter\expandafter\expandafter\XINT_div_I_db
1391         \fi
1392     \fi
1393 }%

    attention très mauvaises notations avec _b et _db.
1394 \def\XINT_div_I_dN #1\xint:%
1395 {%
1396     \expandafter\XINT_div_I_b\the\numexpr #1-\xint_c_i\xint:%
1397 }%
```

```

1398 \def\XINT_div_I_db #1\xint:#2#3#4#5%
1399 {%
1400     \expandafter\XINT_div_I_dc\expandafter #1%
1401     \romannumeral0\expandafter\XINT_div_sub\expandafter
1402         {\romannumeral0\XINT_rev_nounsep {#4\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1403         {\the\numexpr\XINT_div_verysmallmul #1!#51;!}%
1404     \Z {#4}{#5}%
1405 }%

```

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

```

1406 \def\XINT_div_I_dc #1#2%
1407 {%
1408     \if-#2\expandafter\XINT_div_I_dd\else\expandafter\XINT_div_I_de\fi
1409     #1#2%
1410 }%
1411 \def\XINT_div_I_dd #1-\Z
1412 {%
1413     \if #11\expandafter\XINT_div_I_dz\fi
1414     \expandafter\XINT_div_I_dp\the\numexpr #1-\xint_c_i\xint: XX%
1415 }%
1416 \def\XINT_div_I_dz #1XX#2#3#4%
1417 {%
1418     1#4\XINT_div_I_g {#2}%
1419 }%
1420 \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»

```

1421 \def\XINT_div_I_dp #1\xint:#2#3#4#5#6%
1422 {%
1423     1#6+#1\expandafter\XINT_div_I_g\expandafter
1424     {\romannumeral0\expandafter\XINT_div_sub\expandafter
1425         {\romannumeral0\XINT_rev_nounsep {#4\R!\R!\R!\R!\R!\R!\R!\R!\W}%
1426         {\the\numexpr\XINT_div_verysmallmul #1!#51;!}%
1427     }%
1428 }%

```

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»

```

1429 \def\XINT_div_I_g #1#2#3#4#5#6#7%
1430 {%
1431     \expandafter !\the\numexpr
1432     \ifnum#2=#3
1433         \expandafter\XINT_div_exittofinish
1434     \else
1435         \expandafter\XINT_div_I_h
1436     \fi
1437     {#4}#1\xint:#6\xint:{{#4}{#5}{#3}{#2}}{#7}%
1438 }%

```

{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>

```
1439 \def\XINT_div_exittofinish #1#2\xint:#3\xint:#4#5%
```

```
1440 {%
```

```
1441     1\expandafter\expandafter\expandafter!\expandafter\XINT_div_unsepQ_delim
```

```
1442     \romannumeral0\XINT_div_unsepR #2#3%
```

```
1443     \xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relaxR\xint:
```

```
1444 }%
```

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

```
1445 \def\XINT_div_I_h #1\xint:#2!\xint:#4#5%
```

```
1446 {%
```

```
1447     \XINT_div_II_b #1#2!\xint:{#5}{#4}{#3}{#5}%
```

```
1448 }%
```

{x'y}alpha.B, {{x'y},x,K,L}, nouveau alpha',B,<c>

```
1449 \def\XINT_div_II_b #1#2!#3!%
```

```
1450 {%
```

```
1451     \xint_gob_til_eightzeroes #2\XINT_div_II_skipc 00000000%
```

```
1452     \XINT_div_II_c #1{1#2}{#3}%
```

```
1453 }%
```

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

```
1454 \def\XINT_div_II_skipc 00000000\XINT_div_II_c #1#2#3#4#5\xint:#6#7%
```

```
1455 {%
```

```
1456     \XINT_div_II_k #7{#4!#5}{#6}{00000000}%
```

```
1457 }%
```

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

```
1458 \def\XINT_div_II_c #1#2#3#4%
```

```
1459 {%
```

```
1460     \expandafter\XINT_div_II_d\the\numexpr\XINT_div_xmini
```

```
1461     #1\xint:#2!#3!#4!{#1}{#2}{#3!#4!%
```

```
1462 }%
```

```
1463 \def\XINT_div_xmini #1%
```

```
1464 {%
```

```
1465     \xint_gob_til_one #1\XINT_div_xmini_a 1\XINT_div_mini #1%
```

```
1466 }%
```

```
1467 \def\XINT_div_xmini_a 1\XINT_div_mini 1#1%
```

```
1468 {%
```

```
1469     \xint_gob_til_zero #1\XINT_div_xmini_b 0\XINT_div_mini 1#1%
```

```
1470 }%
```

```
1471 \def\XINT_div_xmini_b 0\XINT_div_mini 10#1#2#3#4#5#6#7%
```

```
1472 {%
```

```
1473     \xint_gob_til_zero #7\XINT_div_xmini_c 0\XINT_div_mini 10#1#2#3#4#5#6#7%
```

```
1474 }%
```

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

```
1475 \def\XINT_div_xmini_c 0\XINT_div_mini 100000000\xint:50000000!#1!#2!{#1!}%
```


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

1476 \def\XINT_div_II_d 1#1#2#3#4#5!#6#7#8\xint:#9%

1477 {%

1478 \expandafter\XINT_div_II_e

1479 \romannumeral0\expandafter\XINT_div_sub\expandafter

1480 {\romannumeral0\XINT_rev_nounsep {}#8\R!\R!\R!\R!\R!\R!\R!\W}%

1481 {\the\numexpr\XINT_div_smallmul_a 1000000000\xint:#1#2#3#4\xint:#5!#91;!}%

1482 \xint:{#6}{#7}{#9}{#1#2#3#4#5}%

1483 }%

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

1484 \def\XINT_div_II_e 1#1!%

1485 {%

1486 \xint_gob_til_eightzeroes #1\XINT_div_II_skipf 00000000%

1487 \XINT_div_II_f 1#1!%

1488 }%

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»

1489 \def\XINT_div_II_skipf 00000000\XINT_div_II_f 100000000!#1\xint:#2#3#4#5#6%

1490 {%

1491 \XINT_div_II_k #6{#1}{#4}{#5}%

1492 }%

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.

1493 \def\XINT_div_II_f #1!#2!#3\xint:%

1494 {%

1495 \XINT_div_II_fa {#1!#2!}{#1!#2!#3}%

1496 }%

1497 \def\XINT_div_II_fa #1#2#3#4%

1498 {%

1499 \expandafter\XINT_div_II_g \the\numexpr\XINT_div_xmini #3\xint:#4!#1{#2}%

1500 }%

#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»

1501 \def\XINT_div_II_g 1#1#2#3#4#5!#6#7#8%

1502 {%

1503 \expandafter \XINT_div_II_h

1504 \the\numexpr 1#1#2#3#4#5+#8\expandafter\expandafter\expandafter

1505 \xint:\expandafter\expandafter\expandafter

1506 {\expandafter\xint_gob_til_exclam

1507 \romannumeral0\expandafter\XINT_div_sub\expandafter

1508 {\romannumeral0\XINT_rev_nounsep {}#6\R!\R!\R!\R!\R!\R!\R!\W}%

1509 {\the\numexpr\XINT_div_smallmul_a 1000000000\xint:#1#2#3#4\xint:#5!#71;!}%

1510 {#7}%

1511 }%

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»

```

1512 \def\XINT_div_II_h 1#1\xint:#2#3#4%
1513 {%
1514     \XINT_div_II_k #4{#2}{#3}{#1}%
1515 }%

    {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»

1516 \def\XINT_div_II_k #1#2#3#4#5%
1517 {%
1518     \expandafter\XINT_div_II_l \the\numexpr #4-\xint_c_i\xint:{#3}#1{#2}#5\xint:%
1519 }%
1520 \def\XINT_div_II_l #1\xint:#2#3#4#51#6!%
1521 {%
1522     \XINT_div_II_m {{#1}{#2}}{{#3}{#4}}{{#5}}{{#5}}{{#6}1#6!%
1523 }%

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

1524 \def\XINT_div_II_m #1#2#3#4\xint:#5#6%
1525 {%
1526     \XINT_div_I_a {#3}{#2}{#4}{#5}{#6}#1%
1527 }%

```

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

```

1528 \def\XINT_div_minimulwc_a 1#1\xint:#2\xint:#3!#4#5#6#7#8\xint:%
1529 {%
1530     \expandafter\XINT_div_minimulwc_b
1531     \the\numexpr \xint_c_x^ix+#1+#3*#8\xint:#3*#4#5#6#7+#2*#8\xint:#2*#4#5#6#7\xint:%
1532 }%
1533 \def\XINT_div_minimulwc_b 1#1#2#3#4#5#6\xint:#7\xint:%
1534 {%
1535     \expandafter\XINT_div_minimulwc_c
1536     \the\numexpr \xint_c_x^ix+#1#2#3#4#5+#7\xint:#6\xint:%
1537 }%
1538 \def\XINT_div_minimulwc_c 1#1#2#3#4#5#6\xint:#7\xint:#8\xint:%
1539 {%
1540     1#6#7\expandafter!%
1541     \the\numexpr\expandafter\XINT_div_smallmul_a
1542     \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8\xint:%
1543 }%
1544 \def\XINT_div_smallmul_a #1\xint:#2\xint:#3!1#4!%
1545 {%
1546     \xint_gob_til_sc #4\XINT_div_smallmul_e;%
1547     \XINT_div_minimulwc_a #1\xint:#2\xint:#3!#4\xint:#2\xint:#3!%
1548 }%
1549 \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.

```

1550 \def\XINT_div_verysmallmul #1%
1551     {\xint_gob_til_one #1\XINT_div_verysmallisone 1\XINT_div_verysmallmul_a 0\xint:#1}%
1552 \def\XINT_div_verysmallisone 1\XINT_div_verysmallmul_a 0\xint:1!1#11;!--%

```

```

1553 {1\relax #1100000000!}%
1554 \def\XINT_div_verysmallmul_a #1\xint:#2!1#3!%
1555 {%
1556   \xint_gob_til_sc #3\XINT_div_verysmallmul_e;%
1557   \expandafter\XINT_div_verysmallmul_b
1558   \the\numexpr \xint_c_x^ix+#2*#3+#1\xint:#2!%
1559 }%
1560 \def\XINT_div_verysmallmul_b 1#1#2\xint:%
1561   {1#2\expandafter!\the\numexpr\XINT_div_verysmallmul_a #1\xint:}%
1562 \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.

```

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

```

```

1603 {%
1604     1#2\expandafter!\the\numexpr\XINT_div_sub_h #1%
1605 }%
1606 \def\XINT_div_sub_h #1#2#3!#4!%
1607 {%
1608     \xint_gob_til_sc #4\XINT_div_sub_hi ;%
1609     \expandafter\XINT_div_sub_i\the\numexpr#1-#3+1#4-\xint_c_i\xint:%
1610 }%
1611 \def\XINT_div_sub_i 1#1#2\xint:%
1612 {%
1613     1#2\expandafter!\the\numexpr\XINT_div_sub_a #1%
1614 }%
1615 \def\XINT_div_sub_bi;%
1616     \expandafter\XINT_div_sub_c\the\numexpr#1-#2+#3\xint:#4!#5!#6!#7!#8!#9!;! \W
1617 {%
1618     \XINT_div_sub_l #1#2!#5!#7!#9!%
1619 }%
1620 \def\XINT_div_sub_di;%
1621     \expandafter\XINT_div_sub_e\the\numexpr#1-#2+#3\xint:#4!#5!#6!#7!#8\W
1622 {%
1623     \XINT_div_sub_l #1#2!#5!#7!%
1624 }%
1625 \def\XINT_div_sub_fi;%
1626     \expandafter\XINT_div_sub_g\the\numexpr#1-#2+#3\xint:#4!#5!#6\W
1627 {%
1628     \XINT_div_sub_l #1#2!#5!%
1629 }%
1630 \def\XINT_div_sub_hi;%
1631     \expandafter\XINT_div_sub_i\the\numexpr#1-#2+#3\xint:#4\W
1632 {%
1633     \XINT_div_sub_l #1#2!%
1634 }%
1635 \def\XINT_div_sub_l #1%
1636 {%
1637     \xint_UDzerofork
1638     #1{-2\relax}%
1639     0\XINT_div_sub_r
1640     \krof
1641 }%
1642 \def\XINT_div_sub_r #1!%
1643 {%
1644     -\ifnum 0#1=\xint_c_ 1\else2\fi\relax
1645 }%

```

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

$\backslash\expandafter\XINT_sdiv_out\the\numexpr\XINT_smalldivx_a \ x.1B!1<8d>!\dots1<8d>!1;!$
avec $x = \text{round}(B/2)$, $1B = 10^8 + B$, et A déjà en blocs $1<8d>!$ (non renversés). Le $\backslash\the\numexpr\XINT_smalldivx_a$
va produire $Q \setminus Z \ R \setminus W$ 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.

```

1646 \def\XINT_sdiv_out #1;!#2!%
1647     {\expandafter
1648         {\romannumeral0\XINT_unsep_cuzsmall

```

```
1649      #1\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax}%
1650      {#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.

```
1651 \def\xint_smallldivx_a #1\xint:1#2!1#3!%
1652 {%
1653     \expandafter\xint_smallldivx_b
1654     \the\numexpr (#3+#1)/#2-\xint_c_i!#1\xint:#2!#3!%
1655 }%
1656 \def\xint_smallldivx_b #1#2!%
1657 {%
1658     \if0#1\else
1659         \xint_c_x^viii+#1#2\xint_afterfi{\expandafter!\the\numexpr}\fi
1660     \xint_smallldiv_c #1#2!%
1661 }%
1662 \def\xint_smallldiv_c #1!#2\xint:#3!#4!%
1663 {%
1664     \expandafter\xint_smallldiv_d\the\numexpr #4-#1*#3!#2\xint:#3!%
1665 }%
```

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.

```
1666 \def\xint_smallldiv_d #1!#2!1#3#4!%
1667 {%
1668     \xint_gob_til_sc #3\xint_smallldiv_end ;%
1669     \xint_smallldiv_e #1!#2!1#3#4!%
1670 }%
1671 \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.

```
1672 \def\xint_smallldiv_e #1!#2\xint:#3!%
1673 {%
1674     \expandafter\xint_smallldiv_f\the\numexpr
1675     \xint_c_xi_e_viii mone+#1*\xint_c_x^viii/#3!#2\xint:#3!#1!%
1676 }%
1677 \def\xint_smallldiv_f 1#1#2#3#4#5#6!#7\xint:#8!%
1678 {%
1679     \xint_gob_til_zero #1\xint_smallldiv_fz 0%
1680     \expandafter\xint_smallldiv_g
1681     \the\numexpr\xint_minimul_a #2#3#4#5\xint:#6!#8!#2#3#4#5#6!#7\xint:#8!%
1682 }%
1683 \def\xint_smallldiv_fz 0%
1684     \expandafter\xint_smallldiv_g\the\numexpr\xint_minimul_a
1685     9999\xint:9999!#1!99999999!#2!0!1#3!%
1686 {%
1687     \xint_smallldiv_i \xint:#3!\xint_c_!#2!%
1688 }%
1689 \def\xint_smallldiv_g 1#1!1#2!#3!#4!#5!#6!%
1690 {%
1691     \expandafter\xint_smallldiv_h\the\numexpr 1#6-#1\xint:#2!#5!#3!#4!%
```

```

1692 }%
1693 \def\XINT_smallldiv_h #1#2\xint:#3!#4!%
1694 {%
1695     \expandafter\XINT_smallldiv_i\the\numexpr #4-#3+#1-\xint_c_i\xint:#2!%
1696 }%
1697 \def\XINT_smallldiv_i #1\xint:#2!#3!#4\xint:#5!%
1698 {%
1699     \expandafter\XINT_smallldiv_j\the\numexpr (#1#2+#4)/#5-\xint_c_i!#3!#1#2!#4\xint:#5!%
1700 }%
1701 \def\XINT_smallldiv_j #1!#2!%
1702 {%
1703     \xint_c_x^viii+#1+#2\expandafter!\the\numexpr\XINT_smallldiv_k
1704     #1!%
1705 }%

```

On boucle vers \XINT_smallldiv_d.

```

1706 \def\XINT_smallldiv_k #1!#2!#3\xint:#4!%
1707 {%
1708     \expandafter\XINT_smallldiv_d\the\numexpr #2-#1*#4!#3\xint:#4!%
1709 }%

```

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

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

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

```

1710 \def\XINT_div_mini #1\xint:#2!#3!%
1711 {%
1712     \expandafter\XINT_div_mini_a\the\numexpr
1713     \xint_c_xi_e_viii_mone+#3*\xint_c_x^viii/#1!#1\xint:#2!#3!%
1714 }%

```

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.

```

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

```

```

1730 \the\numexpr 1#6-#1\xint:#2!#5!#3!#4!%
1731 }%
1732 \def\XINT_div_mini_c 1#1#2\xint:#3!#4!%
1733 {%
1734 \expandafter\XINT_div_mini_d
1735 \the\numexpr #4-#3+#1-\xint_c_i\xint:#2!%
1736 }%
1737 \def\XINT_div_mini_d #1\xint:#2!#3!#4\xint:#5!%
1738 {%
1739 \xint_c_x^viii_mone+#3+(#1#2+#5)/#4!%
1740 }%

```

Derived arithmetic

4.38 \xintiiQuo, \xintiiRem

```

1741 \def\xintiiQuo {\romannumeral0\xintiiquo }%
1742 \def\xintiiRem {\romannumeral0\xintiirem }%
1743 \def\xintiiquo
1744 {\expandafter\xint_stop_atfirstoftwo\romannumeral0\xintiidivision }%
1745 \def\xintiirem
1746 {\expandafter\xint_stop_atsecondoftwo\romannumeral0\xintiidivision }%

```

4.39 \xintiiDivRound

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

1.2.1: \xintiiDivRound made robust against non terminated input.

```

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

```

```

1772      -{\xintiioopp\XINT_iidivround_pos #1}%
1773      \krof
1774 }%
1775 \def\XINT_iidivround_pos #1#2\xint:#3\xint:
1776 {%
1777     \expandafter\expandafter\expandafter\XINT_dsrr
1778     \expandafter\xint_firstoftwo
1779     \romannumeral0\XINT_div_prepare {#2}{#1#30}%
1780     \xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax
1781 }%

```

4.40 \xintiiDivTrunc

1.21: \xintiiDivTrunc made robust against non terminated input.

```

1782 \def\xintiiDivTrunc {\romannumeral0\xintiiidivtrunc }%
1783 \def\xintiiidivtrunc #1{\expandafter\XINT_iidivtrunc\romannumeral`&&@#1\xint:}%
1784 \def\XINT_iidivtrunc #1#2\xint:#3{\expandafter\XINT_iidivtrunc_a\expandafter #1%
1785     \romannumeral`&&@#3\xint:#2\xint:}%
1786 \def\XINT_iidivtrunc_a #1#2% #1 de A, #2 de B.
1787 {%
1788     \if0#2\xint_dothis{\XINT_iidivtrunc_divbyzero#1#2}\fi
1789     \if0#1\xint_dothis\XINT_iidivtrunc_aiszero\fi
1790     \if-#2\xint_dothis{\XINT_iidivtrunc_bneg #1}\fi
1791     \xint_orthat{\XINT_iidivtrunc_bpos #1#2}%
1792 }%

```

Attention to not move DivRound code beyond that point.

```

1793 \let\XINT_iidivtrunc_divbyzero\XINT_iidivround_divbyzero
1794 \let\XINT_iidivtrunc_aiszero \XINT_iidivround_aiszero
1795 \def\XINT_iidivtrunc_bpos #1%
1796 {%
1797     \xint_UDsignfork
1798     #1{\xintiioopp\XINT_iidivtrunc_pos {}}%
1799     -{\XINT_iidivtrunc_pos #1}%
1800     \krof
1801 }%
1802 \def\XINT_iidivtrunc_bneg #1%
1803 {%
1804     \xint_UDsignfork
1805     #1{\XINT_iidivtrunc_pos {}}%
1806     -{\xintiioopp\XINT_iidivtrunc_pos #1}%
1807     \krof
1808 }%
1809 \def\XINT_iidivtrunc_pos #1#2\xint:#3\xint:
1810     {\expandafter\xint_stop_atfirstoftwo
1811     \romannumeral0\XINT_div_prepare {#2}{#1#3}}%

```

4.41 \xintiiModTrunc

Renamed from \xintiiMod to \xintiiModTrunc at 1.2p.

```

1812 \def\xintiiModTrunc {\romannumeral0\xintiiidmodtrunc }%
1813 \def\xintiiidmodtrunc #1{\expandafter\XINT_iidmodtrunc\romannumeral`&&@#1\xint:}%

```



```

1814 \def\XINT_iimodtrunc #1#2\xint:#3{\expandafter\XINT_iimodtrunc_a\expandafter #1%
1815         \romannumeral`&&#3\xint:#2\xint:}%
1816 \def\XINT_iimodtrunc_a #1#2% #1 de A, #2 de B.
1817 {%
1818     \if0#2\xint_dothis{\XINT_iimodtrunc_divbyzero#1#2}\fi
1819     \if0#1\xint_dothis\XINT_iimodtrunc_aiszero\fi
1820     \if-#2\xint_dothis{\XINT_iimodtrunc_bneg #1}\fi
1821     \xint_orthat{\XINT_iimodtrunc_bpos #1#2}%
1822 }%

```

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

```

1823 \let\XINT_iimodtrunc_divbyzero\XINT_iidivround_divbyzero
1824 \let\XINT_iimodtrunc_aiszero \XINT_iidivround_aiszero
1825 \def\XINT_iimodtrunc_bpos #1%
1826 {%
1827     \xint_UDsignfork
1828     #1{\xintiioopp\XINT_iimodtrunc_pos {}}%
1829     -{\XINT_iimodtrunc_pos #1}%
1830     \krof
1831 }%
1832 \def\XINT_iimodtrunc_bneg #1%
1833 {%
1834     \xint_UDsignfork
1835     #1{\xintiioopp\XINT_iimodtrunc_pos {}}%
1836     -{\XINT_iimodtrunc_pos #1}%
1837     \krof
1838 }%
1839 \def\XINT_iimodtrunc_pos #1#2\xint:#3\xint:
1840     {\expandafter\xint_stop_atsecondoftwo\romannumeral0\XINT_div_prepare
1841     {#2}{#1#3}}%

```

4.42 \xintiDivMod

1.2p (2017/12/05). It is associated with floored division (like Python divmod function), and with the // operator in [\xintiexpr](#).

```

1842 \def\xintiDivMod {\romannumeral0\xintiDivMod }%
1843 \def\xintiDivMod #1{\expandafter\XINT_iidivmod\romannumeral`&&#1\xint:}%
1844 \def\XINT_iidivmod #1#2\xint:#3{\expandafter\XINT_iidivmod_a\expandafter #1%
1845         \romannumeral`&&#3\xint:#2\xint:}%
1846 \def\XINT_iidivmod_a #1#2% #1 de A, #2 de B.
1847 {%
1848     \if0#2\xint_dothis{\XINT_iidivmod_divbyzero#1#2}\fi
1849     \if0#1\xint_dothis\XINT_iidivmod_aiszero\fi
1850     \if-#2\xint_dothis{\XINT_iidivmod_bneg #1}\fi
1851     \xint_orthat{\XINT_iidivmod_bpos #1#2}%
1852 }%
1853 \def\XINT_iidivmod_divbyzero #1#2\xint:#3\xint:
1854 {%
1855     \XINT_signalcondition{DivisionByZero}{Division by zero: #1#3/#2.}{}%
1856     {{0}{0}}% à revoir...
1857 }%
1858 \def\XINT_iidivmod_aiszero #1\xint:#2\xint:{{0}{0}}%

```

```

1859 \def\XINT_iidivmod_bneg #1%
1860 {%
1861     \expandafter\XINT_iidivmod_bneg_finish
1862     \romannumeral0\xint_UDsignfork
1863         #1{\XINT_iidivmod_bpos {}}%
1864         -{\XINT_iidivmod_bpos {-#1}}%
1865     \krof
1866 }%
1867 \def\XINT_iidivmod_bneg_finish#1#2%
1868 {%
1869     \expandafter\xint_exchangetwo_keepbraces\expandafter
1870     {\romannumeral0\xintiopp#2}{#1}%
1871 }%
1872 \def\XINT_iidivmod_bpos #1#2\xint:#3\xint:{\xintiidivision{#1#3}{#2}}%

```

4.43 \xintiiDivFloor

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

```

1873 \def\xintiiDivFloor {\romannumeral0\xintiidivfloor}%
1874 \def\xintiidivfloor {\expandafter\xint_stop_atfirstoftwo
1875     \romannumeral0\xintiidivmod}%

```

4.44 \xintiiMod

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

```

1876 \def\xintiiMod {\romannumeral0\xintiimod}%
1877 \def\xintiimod {\expandafter\xint_stop_atsecondoftwo
1878     \romannumeral0\xintiidivmod}%

```

4.45 \xintiiSqr

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

```

1879 \def\xintiiSqr {\romannumeral0\xintiisqr }%
1880 \def\xintiisqr #1%
1881 {%
1882     \expandafter\XINT_sqr\romannumeral0\xintiiaabs{#1}\xint:
1883 }%
1884 \def\XINT_sqr #1\xint:
1885 {%
1886     \expandafter\XINT_sqr_a
1887     \romannumeral0\expandafter\XINT_sepandrev_andcount
1888     \romannumeral0\XINT_zeroes_forviii #1\R\R\R\R\R\R\R\R{10}0000001\W
1889     #1\XINT_rsepbyviii_end_A 2345678%
1890     \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
1891     \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
1892     \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_W
1893     \xint:
1894 }%

```

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

```

1895 \def\XINT_sqr_a #1\xint:
1896 {%
1897     \ifnum #1=\xint_c_i \expandafter\XINT_sqr_small
1898         \else\expandafter\XINT_sqr_start\fi
1899 }%
1900 \def\XINT_sqr_small #1#2#3#4#5!\xint:
1901 {%
1902     \ifnum #1#2#3#4#5<46341 \expandafter\XINT_sqr_verysmall\fi
1903     \expandafter\XINT_sqr_small_out
1904     \the\numexpr\XINT_minimul_a #1#2#3#4\xint:#5!#1#2#3#4#5!%
1905 }%
1906 \def\XINT_sqr_verysmall#1{%
1907 \def\XINT_sqr_verysmall
1908     \expandafter\XINT_sqr_small_out\the\numexpr\XINT_minimul_a ##1!##2!%
1909     {\expandafter#1\the\numexpr ##2*##2\relax}%
1910 }\XINT_sqr_verysmall{ }%
1911 \def\XINT_sqr_small_out #1!1#2!%
1912 {%
1913     \XINT_cuz #2#1\R
1914 }%

```

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

```

1915 \def\XINT_sqr_start #1\xint:
1916 {%
1917     \expandafter\XINT_mul_out
1918     \the\numexpr\XINT_mul_loop
1919         1000000000!1;!W #11;!W #11;%
1920     1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1921 }%

```

4.46 \xintiiPow

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

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

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

```

1922 \def\xintiiPow {\romannumeral0\xintiipow }%
1923 \def\xintiipow #1#2%
1924 {%
1925     \expandafter\xint_pow\the\numexpr #2\expandafter
1926     .\romannumeral`&&@#1\xint:
1927 }%
1928 \def\xint_pow #1.#2%#3\xint:
1929 {%
1930     \xint_UDzerominusfork
1931     #2-\XINT_pow_AisZero
1932     0#2\XINT_pow_Aneg
1933     0-{\XINT_pow_Apos #2}%
1934     \krof {#1}%

```

```

1935 }%
1936 \def\XINT_pow_AisZero #1#2\xint:
1937 {%
1938     \ifcase\XINT_cntSgn #1\xint:
1939         \xint_afterfi { 1}%
1940     \or
1941         \xint_afterfi { 0}%
1942     \else
1943         \xint_afterfi
1944         {\XINT_signalcondition{DivisionByZero}{0 raised to power #1.}}{ 0}%
1945     \fi
1946 }%
1947 \def\XINT_pow_Aneg #1%
1948 {%
1949     \ifodd #1
1950         \expandafter\XINT_opp\romannumeral0%
1951     \fi
1952     \XINT_pow_Apos {}{#1}%
1953 }%
1954 \def\XINT_pow_Apos #1#2{\XINT_pow_Apos_a {#2}#1}%
1955 \def\XINT_pow_Apos_a #1#2#3%
1956 {%
1957     \xint_gob_til_xint: #3\XINT_pow_Apos_short\xint:
1958     \XINT_pow_AatleastTwo {#1}#2#3%
1959 }%
1960 \def\XINT_pow_Apos_short\xint:\XINT_pow_AatleastTwo #1#2\xint:
1961 {%
1962     \ifcase #2
1963         \xintError:thiscannothappen
1964     \or \expandafter\XINT_pow_AisOne
1965     \else\expandafter\XINT_pow_AatleastTwo
1966     \fi {#1}#2\xint:
1967 }%
1968 \def\XINT_pow_AisOne #1\xint:{ 1}%
1969 \def\XINT_pow_AatleastTwo #1%
1970 {%
1971     \ifcase\XINT_cntSgn #1\xint:
1972         \expandafter\XINT_pow_BisZero
1973     \or
1974         \expandafter\XINT_pow_I_in
1975     \else
1976         \expandafter\XINT_pow_BisNegative
1977     \fi
1978     {#1}%
1979 }%
1980 \def\XINT_pow_BisNegative #1\xint:{\XINT_signalcondition{Underflow}%
1981     {Inverse power is not an integer.}}{ 0}%
1982 \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).

1983 \def\XINT_pow_I_in #1#2\xint:
1984 {%

```

```

1985 \expandafter\XINT_pow_I_loop
1986 \the\numexpr #1\expandafter\xint:%
1987 \romannumeral0\expandafter\XINT_sepandrev
1988 \romannumeral0\XINT_zeroes_forviii #2\R\R\R\R\R\R\R\{10}0000001\W
1989 #2\XINT_rsepbyviii_end_A 2345678%
1990 \XINT_rsepbyviii_end_B 2345678\relax XX%
1991 \R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\R\xint:\W
1992 1;\!W
1993 1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1994 }%
1995 \def\XINT_pow_I_loop #1\xint:%
1996 {%
1997 \ifnum #1 = \xint_c_i\expandafter\XINT_pow_I_exit\fi
1998 \ifodd #1
1999 \expandafter\XINT_pow_II_in
2000 \else
2001 \expandafter\XINT_pow_I_squareit
2002 \fi #1\xint:%
2003 }%
2004 \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.

2005 \def\XINT_pow_I_squareit #1\xint:#2\W%
2006 {%
2007 \expandafter\XINT_pow_I_loop
2008 \the\numexpr #1/\xint_c_ii\expandafter\xint:%
2009 \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W
2010 }%
2011 \def\XINT_pow_mulbutcheckifsmall #1!1#2%
2012 {%
2013 \xint_gob_til_sc #2\XINT_pow_mul_small;%
2014 \XINT_mul_loop 100000000!1;\!W #1!1#2%
2015 }%
2016 \def\XINT_pow_mul_small;\XINT_mul_loop
2017 100000000!1;\!W 1#1!1;\!W
2018 {%
2019 \XINT_smallmul 1#1!%
2020 }%
2021 \def\XINT_pow_II_in #1\xint:#2\W
2022 {%
2023 \expandafter\XINT_pow_II_loop
2024 \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter\xint:%
2025 \the\numexpr\XINT_pow_mulbutcheckifsmall #2\W #2\W #2\W
2026 }%
2027 \def\XINT_pow_II_loop #1\xint:%
2028 {%
2029 \ifnum #1 = \xint_c_i\expandafter\XINT_pow_II_exit\fi
2030 \ifodd #1
2031 \expandafter\XINT_pow_II_odda
2032 \else
2033 \expandafter\XINT_pow_II_even

```



```

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

```

and I was quite surprised that it was only about 1.6x--2x slower in the range N=200 to 2000 than the `\xintiFac` 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 `\xintiBinomial`, 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`.

```

2059 \def\xintiFac {\romannumeral0\xintiifac }%
2060 \def\xintiifac #1{\expandafter\XINT_fac_fork\the\numexpr#1.}%
2061 \def\XINT_fac_fork #1#2.%
2062 {%
2063   \xint_UDzerominusfork
2064   #1-\XINT_fac_zero
2065   0#1\XINT_fac_neg
2066   0-\XINT_fac_checksize
2067   \krof #1#2.%
2068 }%
2069 \def\XINT_fac_zero #1.{ 1}%
2070 \def\XINT_fac_neg #1.{\XINT_signalcondition{InvalidOperation}{Factorial of
2071   negative argument: #1.}}{ 0}%
2072 \def\XINT_fac_checksize #1.%
2073 {%
2074   \ifnum #1>\xint_c_x^iv \xint_dothis{\XINT_fac_toobig #1.}\fi
2075   \ifnum #1>465 \xint_dothis{\XINT_fac_bigloop_a #1.}\fi
2076   \ifnum #1>101 \xint_dothis{\XINT_fac_medloop_a #1.\XINT_mul_out}\fi
2077   \xint_orthat{\XINT_fac_smallloop_a #1.\XINT_mul_out}%
2078   1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
2079 }%
2080 \def\XINT_fac_toobig
2081 #1.#2\W{\XINT_signalcondition{InvalidOperation}{Factorial
2082   argument is too large: #1 > 10^4.}}{ 0}%
2083 \def\XINT_fac_bigloop_a #1.%
2084 {%
2085   \expandafter\XINT_fac_bigloop_b \the\numexpr
2086   #1+\xint_c_i-\xint_c_ii*((#1-464)/\xint_c_ii).#1.%

```

```

2087 }%
2088 \def\XINT_fac_bigloop_b #1.#2.%
2089 {%
2090     \expandafter\XINT_fac_medloop_a
2091     \the\numexpr #1-\xint_c_i.{\XINT_fac_bigloop_loop #1.#2.}%
2092 }%
2093 \def\XINT_fac_bigloop_loop #1.#2.%
2094 {%
2095     \ifnum #1>#2 \expandafter\XINT_fac_bigloop_exit\fi
2096     \expandafter\XINT_fac_bigloop_loop
2097     \the\numexpr #1+\xint_c_ii\expandafter.%
2098     \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_bigloop_mul #1!%
2099 }%
2100 \def\XINT_fac_bigloop_exit #1!{\XINT_mul_out}%
2101 \def\XINT_fac_bigloop_mul #1!%
2102 {%
2103     \expandafter\XINT_smallmul
2104     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2105 }%
2106 \def\XINT_fac_medloop_a #1.%
2107 {%
2108     \expandafter\XINT_fac_medloop_b
2109     \the\numexpr #1+\xint_c_i-\xint_c_iii*((#1-100)/\xint_c_iii).#1.%
2110 }%
2111 \def\XINT_fac_medloop_b #1.#2.%
2112 {%
2113     \expandafter\XINT_fac_smallloop_a
2114     \the\numexpr #1-\xint_c_i.{\XINT_fac_medloop_loop #1.#2.}%
2115 }%
2116 \def\XINT_fac_medloop_loop #1.#2.%
2117 {%
2118     \ifnum #1>#2 \expandafter\XINT_fac_loop_exit\fi
2119     \expandafter\XINT_fac_medloop_loop
2120     \the\numexpr #1+\xint_c_iii\expandafter.%
2121     \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_medloop_mul #1!%
2122 }%
2123 \def\XINT_fac_medloop_mul #1!%
2124 {%
2125     \expandafter\XINT_smallmul
2126     \the\numexpr
2127     \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2128 }%
2129 \def\XINT_fac_smallloop_a #1.%
2130 {%
2131     \csname
2132     XINT_fac_smallloop_\the\numexpr #1-\xint_c_iv*(#1/\xint_c_iv)\relax
2133     \endcsname #1.%
2134 }%
2135 \expandafter\def\csname XINT_fac_smallloop_1\endcsname #1.%
2136 {%
2137     \XINT_fac_smallloop_loop 2.#1.1000000011; !%
2138 }%

```



```

2139 \expandafter\def\csname XINT_fac_smallloop_-2\endcsname #1.%
2140 {%
2141     \XINT_fac_smallloop_loop 3.#1.1000000002!1;!%
2142 }%
2143 \expandafter\def\csname XINT_fac_smallloop_-1\endcsname #1.%
2144 {%
2145     \XINT_fac_smallloop_loop 4.#1.1000000006!1;!%
2146 }%
2147 \expandafter\def\csname XINT_fac_smallloop_0\endcsname #1.%
2148 {%
2149     \XINT_fac_smallloop_loop 5.#1.1000000024!1;!%
2150 }%
2151 \def\XINT_fac_smallloop_loop #1.#2.%
2152 {%
2153     \ifnum #1>#2 \expandafter\XINT_fac_loop_exit\fi
2154     \expandafter\XINT_fac_smallloop_loop
2155     \the\numexpr #1+\xint_c_iv\expandafter.%
2156     \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_smallloop_mul #1!%
2157 }%
2158 \def\XINT_fac_smallloop_mul #1!%
2159 {%
2160     \expandafter\XINT_smallmul
2161     \the\numexpr
2162         \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2163 }%
2164 \def\XINT_fac_loop_exit #1!#2;!#3{#3#2;!}%

```

4.48 \XINT_useiimessage

1.2o

```

2165 \def\XINT_useiimessage #1% used in LaTeX only
2166 {%
2167     \XINT_ifFlagRaised {#1}%
2168     {\@backslashchar#1
2169         (load xintfrac or use \@backslashchar xintii\xint_gobble_iv#1!)\MessageBreak}%
2170     }%
2171 }%
2172 \XINTrestorecatcodesendinginput%

```

5 Package *xint* implementation

.1	Package identification	123	.35	\xintiifNotZero	135
.2	More token management	123	.36	\xintiifOne	135
.3	(WIP) A constant needed by \xintRan- domDigits et al.	123	.37	\xintiifOdd	136
.4	\xintLen, \xintiLen	124	.38	\xintifTrueAelseB, \xintifFalseAelseB	136
.5	\xintiiLogTen	124	.39	\xintIsTrue, \xintIsFalse	136
.6	\xintReverseDigits	124	.40	\xintNOT	136
.7	\xintiiE	125	.41	\xintAND, \xintOR, \xintXOR	136
.8	\xintDecSplit	126	.42	\xintANDof	137
.9	\xintDecSplitL	127	.43	\xintORof	137
.10	\xintDecSplitR	128	.44	\xintXORof	137
.11	\xintDSHr	128	.45	\xintiiMax	138
.12	\xintDSH	128	.46	\xintiiMin	139
.13	\xintDSx	129	.47	\xintiiMaxof	140
.14	\xintiiEq	130	.48	\xintiiMinof	140
.15	\xintiiNotEq	131	.49	\xintiiSum	141
.16	\xintiiGeq	131	.50	\xintiiPrd	141
.17	\xintiiGt	131	.51	\xintiiSquareRoot	142
.18	\xintiiLt	132	.52	\xintiiSqrt, \xintiiSqrtR	149
.19	\xintiiGtorEq	132	.53	\xintiiBinomial	149
.20	\xintiiLtorEq	132	.54	\xintiiPFactorial	154
.21	\xintiiIsZero	132	.55	\xintBool, \xintToggle	157
.22	\xintiiIsNotZero	132	.56	\xintiiGCD	158
.23	\xintiiIsOne	132	.57	\xintiiGCDof	158
.24	\xintiiOdd	132	.58	\xintiiLCM	159
.25	\xintiiEven	133	.59	\xintiiLCMof	159
.26	\xintiiMON	133	.60	(WIP) \xintRandomDigits	160
.27	\xintiiMMON	133	.61	(WIP) \XINT_eightrandomdigits, \xin- tEightRandomDigits	160
.28	\xintSgnFork	133	.62	(WIP) \xintRandBit	161
.29	\xintiiifSgn	134	.63	(WIP) \xintXRandomDigits	161
.30	\xintiiifCmp	134	.64	(WIP) \xintiiRandRangeAtoB	161
.31	\xintiiifEq	134	.65	(WIP) \xintiiRandRange	161
.32	\xintiiifGt	134	.66	(WIP) Adjustments for engines without uniformdeviate primitive	163
.33	\xintiiifLt	135			
.34	\xintiiifZero	135			

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 [2021/06/11 v1.4i 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 900000000
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:\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:\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 #11#2!1#3!1#4!1#5!1#6!1#7!1#8!1#9!%
113 {%
114     \xint_gob_til_R #9\XINT_revdigits_end\R
115     \XINT_revdigits_b {#9#8#7#6#5#4#3#2#1}%
116 }%
117 \def\XINT_revdigits_end#1{%
118 \def\XINT_revdigits_end\R\XINT_revdigits_b ##1##2\W
119     {\expandafter#1\xint_gob_til_Z ##1}%
120 }\XINT_revdigits_end{ }%
121 \let\xintRev\xintReverseDigits

```

5.7 \xintiiE

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

```

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

```

```

130 \krof #1%
131 }%

le #2 a le bon pattern terminé par ; #1=0 est OK pour \XINT_rep.
132 \def\XINT_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\xintiifeq{#1}{#2}{1}{0}}%

```

5.15 \xintiiNotEq

Pour xintexpr. Pas de version en lowercase.

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

5.16 \xintiiGeq

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

1.2l made \xintiiGeq robust against non terminated items.

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

This macro should have been called \xintGEq for example.

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

335 \def\XINT_geq #1#2\xint:#3%
336 {%
337     \expandafter\XINT_geq_fork\expandafter #1\romannumeral0\xintnum{#3}\xint:#2\xint:
338 }%
339 \def\XINT_geq_fork #1#2%
340 {%
341     \xint_UDzerofork
342     #1\XINT_geq_firstiszero
343     #2\XINT_geq_secondiszero
344     0{}}%
345     \krof
346     \xint_UDsignsfork
347     #1#2\XINT_geq_minusminus
348     #1-\XINT_geq_minusplus
349     #2-\XINT_geq_plusminus
350     --\XINT_geq_plusplus
351     \krof #1#2%
352 }%
353 \def\XINT_geq_firstiszero #1\krof 0#2#3\xint:#4\xint:
354     {\xint_UDzerofork #2{ 1}0{ 0}\krof }%
355 \def\XINT_geq_secondiszero #1\krof #20#3\xint:#4\xint:{ 1}%
356 \def\XINT_geq_plusminus #1-{\XINT_geq_plusplus #1{}}%
357 \def\XINT_geq_minusplus -#1{\XINT_geq_plusplus { }#1}%
358 \def\XINT_geq_minusminus --{\XINT_geq_plusplus { }{ }}%
359 \def\XINT_geq_plusplus
360     {\expandafter\XINT_geq_finish\romannumeral0\XINT_cmp_plusplus}%
361 \def\XINT_geq_finish #1{\if-#1\expandafter\XINT_geq_no
362     \else\expandafter\XINT_geq_yes\fi}%
363 \def\XINT_geq_no 1{ 0}%
364 \def\XINT_geq_yes { 1}%

```

5.17 \xintiiGt

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

5.18 \xintiiLt

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

5.19 \xintiiGtorEq

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

5.20 \xintiiLtorEq

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

5.21 \xintiiIsZero

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

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

5.22 \xintiiIsNotZero

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

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

5.23 \xintiiIsOne

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

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

```
374 \def\xintiiIsOne {\romannumeral0\xintiiisone }%
375 \def\xintiiisone #1{\expandafter\XINT_isone\romannumeral`&&@#1XY}%
376 \def\XINT_isone #1#2#3Y%
377 {%
378     \unless\if#2X\xint_dothis{ 0}\fi
379     \unless\if#11\xint_dothis{ 0}\fi
380     \xint_orthat{ 1}%
381 }%
382 \def\XINT_isOne #1{\XINT_is_One#1XY}%
383 \def\XINT_is_One #1#2#3Y%
384 {%
385     \unless\if#2X\xint_dothis0\fi
386     \unless\if#11\xint_dothis0\fi
387     \xint_orthat1%
388 }%
```

5.24 \xintiiOdd

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

```
389 \def\xintiiOdd {\romannumeral0\xintiiodd }%
390 \def\xintiiodd #1%
391 {%
392     \ifodd\xintLDg{#1} %<- intentional space
```

```

393      \xint_afterfi{ 1}%
394  \else
395      \xint_afterfi{ 0}%
396  \fi
397 }%

```

5.25 \xintiEven

```

398 \def\xintiEven {\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 \xintiMON

MINUS ONE TO THE POWER N

```

407 \def\xintiMON {\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 \xintiMMON

MINUS ONE TO THE POWER N-1

```

416 \def\xintiMMON {\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\xintsgnfork }%
426 \def\xintsgnfork #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\xintiiiszero}%

```

5.41 \xintAND, \xintOR, \xintXOR

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

```

516 \def\xintAND {\romannumeral0\xintand }%

```



```

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

```

5.42 \xintANDof

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

1.2l made \xintANDof robust against non terminated items.

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

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_zeroplus #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_zeroplus % 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_zeroplus #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 \xintiMaxof

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

NOT compatible with empty list.

1.2l made \xintiMaxof robust against non terminated items.

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

```

642 \def\xintiMaxof {\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 \xintiMinof

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

1.4 refactoring for \xintiexpr matters.

```

657 \def\xintiMinof {\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:
735   {\XINT_signalcondition{InvalidOperation}%
736     {Square root of negative: #1.}\{\{0\}\{0\}}}%
737 \def\XINT_sqrt #1\xint:
738 {%
739   \expandafter\XINT_sqrt_start\romannumeral0\xintlenth {#1}.#1.%
740 }%
741 \def\XINT_sqrt_start #1.%
742 {%
743   \ifnum #1<\xint_c_x\xint_dothis\XINT_sqrt_small_a\fi
744   \xint_orthat\XINT_sqrt_big_a #1.%
745 }%
746 \def\XINT_sqrt_small_a #1.{\XINT_sqrt_a #1.\XINT_sqrt_small_d }%
747 \def\XINT_sqrt_big_a #1.{\XINT_sqrt_a #1.\XINT_sqrt_big_d }%
748 \def\XINT_sqrt_a #1.%
749 {%
750   \ifodd #1
751     \expandafter\XINT_sqrt_b0
752   \else
753     \expandafter\XINT_sqrt_bE
754   \fi
755   #1.%
756 }%
757 \def\XINT_sqrt_bE #1.#2#3#4%
758 {%
759   \XINT_sqrt_c {#3#4}#2{#1}#3#4%
760 }%
761 \def\XINT_sqrt_b0 #1.#2#3%
762 {%
763   \XINT_sqrt_c #3#2{#1}#3%
764 }%
765 \def\XINT_sqrt_c #1#2%
766 {%
767   \expandafter #2%
768   \the\numexpr \ifnum #1>\xint_c_ii

```

```

769         \ifnum #1>\xint_c_vi
770         \ifnum #1>12 \ifnum #1>20 \ifnum #1>30
771         \ifnum #1>42 \ifnum #1>56 \ifnum #1>72
772         \ifnum #1>90
773     10\else 9\fi \else 8\fi \else 7\fi \else 6\fi \else 5\fi
774     \else 4\fi \else 3\fi \else 2\fi \else 1\fi .%
775 }%

776 \def\xint_sqrt_small_d #1.#2%
777 {%
778     \expandafter\xint_sqrt_small_e
779     \the\numexpr #1\ifcase \numexpr #2/\xint_c_ii-\xint_c_i\relax
780         \or 0\or 00\or 000\or 0000\fi .%
781 }%

782 \def\xint_sqrt_small_e #1.#2.%
783 {%
784     \expandafter\xint_sqrt_small_ea\the\numexpr #1*#1-#2.#1.%
785 }%

786 \def\xint_sqrt_small_ea #1%
787 {%
788     \if0#1\xint_dothis\xint_sqrt_small_ez\fi
789     \if-#1\xint_dothis\xint_sqrt_small_eb\fi
790     \xint_orthat\xint_sqrt_small_f #1%
791 }%
792 \def\xint_sqrt_small_ez 0.#1.{\expandafter{\the\numexpr#1+\xint_c_i
793     \expandafter}\expandafter{\the\numexpr #1*\xint_c_ii+\xint_c_i}}%

794 \def\xint_sqrt_small_eb -#1.#2.%
795 {%
796     \expandafter\xint_sqrt_small_ec \the\numexpr
797     (#1-\xint_c_i+#2)/(\xint_c_ii*#2).#1.#2.%
798 }%

799 \def\xint_sqrt_small_ec #1.#2.#3.%
800 {%
801     \expandafter\xint_sqrt_small_f \the\numexpr
802     -#2+\xint_c_ii*#3*#1+#1*#1\expandafter.\the\numexpr #3+#1.%
803 }%

804 \def\xint_sqrt_small_f #1.#2.%
805 {%
806     \expandafter\xint_sqrt_small_g
807     \the\numexpr (#1+#2)/(\xint_c_ii*#2)-\xint_c_i.#1.#2.%
808 }%

809 \def\xint_sqrt_small_g #1#2.%
810 {%
811     \if 0#1%
812         \expandafter\xint_sqrt_small_end
813     \else
814         \expandafter\xint_sqrt_small_h
815     \fi
816     #1#2.%
817 }%

```



```

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

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

829     \the\numexpr (#2-\xint_c_i)/\xint_c_ii.#1;%
830 }%

831 \def\XINT_sqrt_big_eE #1;#2#3#4#5#6#7#8#9%
832 {%
833     \XINT_sqrt_big_eE_a #1;{#2#3#4#5#6#7#8#9}%
834 }%

835 \def\XINT_sqrt_big_eE_a #1.#2;#3%
836 {%
837     \expandafter\XINT_sqrt_bigormed_f
838     \romannumeral0\XINT_sqrt_small_e #2000.#3.#1;%
839 }%

840 \def\XINT_sqrt_big_e0 #1;#2#3#4#5#6#7#8#9%
841 {%
842     \XINT_sqrt_big_e0_a #1;{#2#3#4#5#6#7#8#9}%
843 }%
844 \def\XINT_sqrt_big_e0_a #1.#2;#3#4%
845 {%
846     \expandafter\XINT_sqrt_bigormed_f
847     \romannumeral0\XINT_sqrt_small_e #20000.#3#4.#1;%
848 }%

849 \def\XINT_sqrt_bigormed_f #1#2#3;%
850 {%
851     \ifnum#3<\xint_c_ix
852         \xint_dothis {\csname XINT_sqrt_med_f\romannumeral#3\endcsname}%
853     \fi
854     \xint_orthat\XINT_sqrt_big_f #1.#2.#3;%
855 }%
856 \def\XINT_sqrt_med_fv {\XINT_sqrt_med_fa .}%
857 \def\XINT_sqrt_med_fvi {\XINT_sqrt_med_fa 0.}%
858 \def\XINT_sqrt_med_fvii {\XINT_sqrt_med_fa 00.}%
859 \def\XINT_sqrt_med_fviii{\XINT_sqrt_med_fa 000.}%

860 \def\XINT_sqrt_med_fa #1.#2.#3.#4;%
861 {%
862     \expandafter\XINT_sqrt_med_fb
863     \the\numexpr (#30#1-5#1)/(\xint_c_ii*#2).#1.#2.#3.%
864 }%

```

```

865 \def\XINT_sqrt_med_fb #1.#2.#3.#4.#5.%
866 {%
867     \expandafter\XINT_sqrt_small_ea
868     \the\numexpr (#40#2-\xint_c_ii*#3*#1)*10#2+(#1*#1-#5)\expandafter.%
869     \the\numexpr #30#2-#1.%
870 }%

871 \def\XINT_sqrt_big_f #1;#2#3#4#5#6#7#8#9%
872 {%
873     \XINT_sqrt_big_fa #1;{#2#3#4#5#6#7#8#9}%
874 }%

875 \def\XINT_sqrt_big_fa #1.#2.#3;#4%
876 {%
877     \expandafter\XINT_sqrt_big_ga
878     \the\numexpr #3-\xint_c_viii\expandafter.%
879     \romannumeral0\XINT_sqrt_med_fa 000.#1.#2.;#4.%
880 }%

881 \def\XINT_sqrt_big_ga #1.#2#3%
882 {%
883     \ifnum #1>\xint_c_viii
884         \expandafter\XINT_sqrt_big_gb\else
885         \expandafter\XINT_sqrt_big_ka
886     \fi #1.#3.#2.%
887 }%

888 \def\XINT_sqrt_big_gb #1.#2.#3.%
889 {%
890     \expandafter\XINT_sqrt_big_gc
891     \the\numexpr (\xint_c_ii*#2-\xint_c_i)*\xint_c_x^viii/(\xint_c_iv*#3).%
892     #3.#2.#1;%
893 }%

894 \def\XINT_sqrt_big_gc #1.#2.#3.%
895 {%
896     \expandafter\XINT_sqrt_big_gd
897     \romannumeral0\xintiiadd
898         {\xintiiSub {#3000000000}\xintDouble{\xintiiMul{#2}{#1}}00000000}%
899         {\xintiiSqr {#1}}.%
900     \romannumeral0\xintiisub{#2000000000}{#1}.%
901 }%

902 \def\XINT_sqrt_big_gd #1.#2.%
903 {%
904     \expandafter\XINT_sqrt_big_ge #2.#1.%
905 }%

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

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

```

```

912 \expandafter\XINT_sqrt_big_gloop
913 \expandafter\xint_c_xvi\expandafter.%
914 \the\numexpr #3-\xint_c_viii\expandafter.%
915 \romannumeral0\xintiisub {#2}{\xintiNum{#4}}.#1.%
916 }%

917 \def\XINT_sqrt_big_gloop #1.#2.%
918 {%
919 \unless\ifnum #1<#2 \xint_dothis\XINT_sqrt_big_ka \fi
920 \xint_orthat{\XINT_sqrt_big_gi #1.}#2.%
921 }%

922 \def\XINT_sqrt_big_gi #1.%
923 {%
924 \expandafter\XINT_sqrt_big_gj\romannumeral\xintreplicate{#1}0.#1.%
925 }%

926 \def\XINT_sqrt_big_gj #1.#2.#3.#4.#5.%
927 {%
928 \expandafter\XINT_sqrt_big_gk
929 \romannumeral0\xintiidivision {#4#1}%
930 {\XINT_dbl #5\xint_bye2345678\xint_bye*\xint_c_ii\relax}.%
931 #1.#5.#2.#3.%
932 }%

933 \def\XINT_sqrt_big_gk #1#2.#3.#4.%
934 {%
935 \expandafter\XINT_sqrt_big_gl
936 \romannumeral0\xintiiaadd {#2#3}{\xintiiSqr{#1}}.%
937 \romannumeral0\xintiisub {#4#3}{#1}.%
938 }%

939 \def\XINT_sqrt_big_gl #1.#2.%
940 {%
941 \expandafter\XINT_sqrt_big_gm #2.#1.%
942 }%

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

946 \romannumeral0\XINT_split_fromleft\xint_c_ii*#3.#5\xint_bye2345678\xint_bye..%
947 #1.#2.#3.#4.%
948 }%

949 \def\XINT_sqrt_big_gn #1.#2.#3.#4.#5.#6.%
950 {%
951 \expandafter\XINT_sqrt_big_gloop
952 \the\numexpr \xint_c_ii*#5\expandafter.%
953 \the\numexpr #6-#5\expandafter.%
954 \romannumeral0\xintiisub{#4}{\xintiNum{#1}}.#3.#2.%
955 }%

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

```

```

959 \romannumeral0\XINT_dsx_addzeros {#1}#3;.%
960 \romannumeral0\xintiisub
961 {\XINT_dsx_addzerosnofuss {\xint_c_ii*#1}#2;}%
962 {\xintiNum{#4}}.%
963 }%
964 \def\XINT_sqrt_big_kb #1.#2.%
965 {%
966 \expandafter\XINT_sqrt_big_kc #2.#1.%
967 }%

968 \def\XINT_sqrt_big_kc #1%
969 {%
970 \if0#1\xint_dothis\XINT_sqrt_big_kz\fi
971 \xint_orthat\XINT_sqrt_big_kloop #1%
972 }%
973 \def\XINT_sqrt_big_kz 0.#1.%
974 {%
975 \expandafter\XINT_sqrt_big_kend
976 \romannumeral0%
977 \xintinc{\XINT_dbl#1\xint_bye2345678\xint_bye*\xint_c_ii\relax}.#1.%
978 }%
979 \def\XINT_sqrt_big_kend #1.#2.%
980 {%
981 \expandafter{\romannumeral0\xintinc{#2}}{#1}%
982 }%

983 \def\XINT_sqrt_big_kloop #1.#2.%
984 {%
985 \expandafter\XINT_sqrt_big_ke
986 \romannumeral0\xintiidivision{#1}%
987 {\romannumeral0\XINT_dbl #2\xint_bye2345678\xint_bye*\xint_c_ii\relax}{#2}%
988 }%

989 \def\XINT_sqrt_big_ke #1%
990 {%
991 \if0\XINT_Sgn #1\xint:
992 \expandafter \XINT_sqrt_big_end
993 \else \expandafter \XINT_sqrt_big_kf
994 \fi {#1}%
995 }%

996 \def\XINT_sqrt_big_kf #1#2#3%
997 {%
998 \expandafter\XINT_sqrt_big_kg
999 \romannumeral0\xintiisub {#3}{#1}.%
1000 \romannumeral0\xintiiadd {#2}{\xintiiSqr {#1}}.%
1001 }%
1002 \def\XINT_sqrt_big_kg #1.#2.%
1003 {%
1004 \expandafter\XINT_sqrt_big_kloop #2.#1.%
1005 }%

1006 \def\XINT_sqrt_big_end #1#2#3{{#3}{#2}}%

```

5.52 \xintiisqrt, \xintiisqrtr

```
1007 \def\xintiisqrt {\romannumeral0\xintiisqrt }%
1008 \def\xintiisqrtr {\expandafter\XINT_sqrt_post\romannumeral0\xintiisquareroot }%
1009 \def\XINT_sqrt_post #1#2{\XINT_dec #1\XINT_dec_bye234567890\xint_bye}%
1010 \def\xintiisqrtr {\romannumeral0\xintiisqrtr }%
1011 \def\xintiisqrtr {\expandafter\XINT_sqrtr_post\romannumeral0\xintiisquareroot }%
```

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

```
1012 \def\XINT_sqrtr_post #1#2%
1013 {\xintiiflt {\#2}{\#1}{\#1}{\XINT_dec #1\XINT_dec_bye234567890\xint_bye}}%
```

5.53 \xintiibinomial

2015/11/28-29 for 1.2f.

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

I really lack some kind of infinity or NaN value.

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

```
1014 \def\xintiibinomial {\romannumeral0\xintiibinomial }%
1015 \def\xintiibinomial #1#2%
1016 {%
1017 \expandafter\XINT_binom_pre\the\numexpr #1\expandafter.\the\numexpr #2.%
1018 }%
1019 \def\XINT_binom_pre #1.#2.%
1020 {%
1021 \expandafter\XINT_binom_fork \the\numexpr#1-#2.#2.#1.%
1022 }%
```

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.

```
1023 \def\XINT_binom_fork #1#2.#3#4.#5#6.%
1024 {%
1025 \if-#5\xint_dothis{\XINT_signalcondition{InvalidOperation}}%
1026 {Binomial with negative first argument: #5#6.}{\fi}
1027 \if-#1\xint_dothis{ 0}\fi
1028 \if-#3\xint_dothis{ 0}\fi
1029 \if0#1\xint_dothis{ 1}\fi
1030 \if0#3\xint_dothis{ 1}\fi
1031 \ifnum #5#6>\xint_c_x^viii_mone\xint_dothis
1032 {\XINT_signalcondition{InvalidOperation}}%
1033 {Binomial with too large argument: #5#6 >= 10^8.}{\fi}
1034 \ifnum #1#2>#3#4 \xint_dothis{\XINT_binom_a #1#2.#3#4.}\fi
1035 \xint_orthat{\XINT_binom_a #3#4.#1#2.}%
1036 }%
```

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

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

```
1037 \def\xint_binom_a #1.#2.%
1038 {%
1039     \expandafter\xint_binom_b\the\numexpr \xint_c_i+#1.1.#2.1000000001!1!;!0!%
1040 }%
```

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

```
1041 \def\xint_binom_b #1.%
1042 {%
1043     \ifnum #1>9999 \xint_dothis\xint_binom_vbigloop \fi
1044     \ifnum #1>463 \xint_dothis\xint_binom_bigloop \fi
1045     \ifnum #1>98 \xint_dothis\xint_binom_medloop \fi
1046     \ifnum #1>29 \xint_dothis\xint_binom_smallloop \fi
1047     \xint_orthat\xint_binom_vsmallloop #1.%
1048 }%
```

$y.j.k$. 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.

```
1049 \def\xint_binom_smallloop #1.#2.#3.%
1050 {%
1051     \ifcase\numexpr #3-#2\relax
1052         \expandafter\xint_binom_end_
1053     \or \expandafter\xint_binom_end_i
1054     \or \expandafter\xint_binom_end_ii
1055     \or \expandafter\xint_binom_end_iii
1056     \else\expandafter\xint_binom_smallloop_a
1057     \fi #1.#2.#3.%
1058 }%
```

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

```
1059 \def\xint_binom_smallloop_a #1.#2.#3.%
1060 {%
1061     \expandafter\xint_binom_smallloop_b
1062     \the\numexpr #1+\xint_c_iv\expandafter.%
1063     \the\numexpr #2+\xint_c_iv\expandafter.%
1064     \the\numexpr #3\expandafter.%
1065     \the\numexpr\expandafter\xint_binom_div
1066     \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1067     !\romannumeral0\expandafter\xint_binom_mul
1068     \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1069 }%
1070 \def\xint_binom_smallloop_b #1.%
1071 {%
1072     \ifnum #1>98 \expandafter\xint_binom_medloop \else
1073         \expandafter\xint_binom_smallloop \fi #1.%
1074 }%
```

Ici on prend trois par trois.

```

1075 \def\XINT_binom_medloop #1.#2.#3.%
1076 {%
1077     \ifcase\numexpr #3-#2\relax
1078         \expandafter\XINT_binom_end_
1079     \or \expandafter\XINT_binom_end_i
1080     \or \expandafter\XINT_binom_end_ii
1081     \else\expandafter\XINT_binom_medloop_a
1082     \fi #1.#2.#3.%
1083 }%
1084 \def\XINT_binom_medloop_a #1.#2.#3.%
1085 {%
1086     \expandafter\XINT_binom_medloop_b
1087     \the\numexpr #1+\xint_c_iii\expandafter.%
1088     \the\numexpr #2+\xint_c_iii\expandafter.%
1089     \the\numexpr #3\expandafter.%
1090     \the\numexpr\expandafter\XINT_binom_div
1091         \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
1092     !\romannumeral0\expandafter\XINT_binom_mul
1093     \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1094 }%
1095 \def\XINT_binom_medloop_b #1.%
1096 {%
1097     \ifnum #1>463 \expandafter\XINT_binom_bigloop \else
1098         \expandafter\XINT_binom_medloop \fi #1.%
1099 }%

```

Ici on prend deux par deux.

```

1100 \def\XINT_binom_bigloop #1.#2.#3.%
1101 {%
1102     \ifcase\numexpr #3-#2\relax
1103         \expandafter\XINT_binom_end_
1104     \or \expandafter\XINT_binom_end_i
1105     \else\expandafter\XINT_binom_bigloop_a
1106     \fi #1.#2.#3.%
1107 }%
1108 \def\XINT_binom_bigloop_a #1.#2.#3.%
1109 {%
1110     \expandafter\XINT_binom_bigloop_b
1111     \the\numexpr #1+\xint_c_ii\expandafter.%
1112     \the\numexpr #2+\xint_c_ii\expandafter.%
1113     \the\numexpr #3\expandafter.%
1114     \the\numexpr\expandafter\XINT_binom_div
1115         \the\numexpr #2*(#2+\xint_c_i)\expandafter
1116     !\romannumeral0\expandafter\XINT_binom_mul
1117     \the\numexpr #1*(#1+\xint_c_i)!%
1118 }%
1119 \def\XINT_binom_bigloop_b #1.%
1120 {%
1121     \ifnum #1>9999 \expandafter\XINT_binom_vbigloop \else
1122         \expandafter\XINT_binom_bigloop \fi #1.%
1123 }%

```

Et finalement un par un.

```

1124 \def\XINT_binom_vbigloop #1.#2.#3.%
1125 {%
1126     \ifnum #3=#2
1127         \expandafter\XINT_binom_end_
1128     \else\expandafter\XINT_binom_vbigloop_a
1129     \fi #1.#2.#3.%
1130 }%
1131 \def\XINT_binom_vbigloop_a #1.#2.#3.%
1132 {%
1133     \expandafter\XINT_binom_vbigloop
1134     \the\numexpr #1+\xint_c_i\expandafter.%
1135     \the\numexpr #2+\xint_c_i\expandafter.%
1136     \the\numexpr #3\expandafter.%
1137     \the\numexpr\expandafter\XINT_binom_div\the\numexpr #2\expandafter
1138     !\romannumeral0\XINT_binom_mul #1!%
1139 }%

```

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.

```

1140 \def\XINT_binom_vsmallloop #1.#2.#3.%
1141 {%
1142     \ifcase\numexpr #3-#2\relax
1143         \expandafter\XINT_binom_vsmallend_
1144     \or \expandafter\XINT_binom_vsmallend_i
1145     \or \expandafter\XINT_binom_vsmallend_ii
1146     \or \expandafter\XINT_binom_vsmallend_iii
1147     \else\expandafter\XINT_binom_vsmallloop_a
1148     \fi #1.#2.#3.%
1149 }%
1150 \def\XINT_binom_vsmallloop_a #1.%
1151 {%
1152     \ifnum #1>26 \expandafter\XINT_binom_smallloop_a \else
1153         \expandafter\XINT_binom_vsmallloop_b \fi #1.%
1154 }%
1155 \def\XINT_binom_vsmallloop_b #1.#2.#3.%
1156 {%
1157     \expandafter\XINT_binom_vsmallloop
1158     \the\numexpr #1+\xint_c_iv\expandafter.%
1159     \the\numexpr #2+\xint_c_iv\expandafter.%
1160     \the\numexpr #3\expandafter.%
1161     \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1162     \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1163     !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1164 }%
1165 \def\XINT_binom_mul #1!#2!;!0!%
1166 {%
1167     \expandafter\XINT_rev_nounsep\expandafter{\expandafter}%
1168     \the\numexpr\expandafter\XINT_smallmul
1169     \the\numexpr\xint_c_x^viii+#1\expandafter

```



```

1170    !\romannumeral0\XINT_rev_nounsep {}1;!#2%
1171    \R!\R!\R!\R!\R!\R!\R!\R!\W
1172    \R!\R!\R!\R!\R!\R!\R!\R!\W
1173    1;!%
1174 }%
1175 \def\XINT_binom_div #1!1;!%
1176 {%
1177     \expandafter\XINT_smallldivx_a
1178     \the\numexpr #1/\xint_c_ii\expandafter\xint:
1179     \the\numexpr \xint_c_x^viii+#1!%
1180 }%

```

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

```

1181 \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.

```

1182 \def\XINT_binom_end_iii #1.#2.#3.%
1183 {%
1184     \expandafter\XINT_binom_finish
1185     \the\numexpr\expandafter\XINT_binom_div
1186         \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1187         !\romannumeral0\expandafter\XINT_binom_mul
1188         \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1189 }%
1190 \def\XINT_binom_end_ii #1.#2.#3.%
1191 {%
1192     \expandafter\XINT_binom_finish
1193     \the\numexpr\expandafter\XINT_binom_div
1194         \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
1195         !\romannumeral0\expandafter\XINT_binom_mul
1196         \the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1197 }%
1198 \def\XINT_binom_end_i #1.#2.#3.%
1199 {%
1200     \expandafter\XINT_binom_finish
1201     \the\numexpr\expandafter\XINT_binom_div
1202         \the\numexpr #2*(#2+\xint_c_i)\expandafter
1203         !\romannumeral0\expandafter\XINT_binom_mul
1204         \the\numexpr #1*(#1+\xint_c_i)!%
1205 }%
1206 \def\XINT_binom_end_ #1.#2.#3.%
1207 {%
1208     \expandafter\XINT_binom_finish
1209     \the\numexpr\expandafter\XINT_binom_div\the\numexpr #2\expandafter
1210     !\romannumeral0\XINT_binom_mul #1!%
1211 }%
1212 \def\XINT_binom_finish #1;!0!%
1213 {\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).

```

1214 \def\XINT_binom_vsmallend_iii #1.%
1215 {%
1216     \ifnum #1>26 \expandafter\XINT_binom_end_iii \else
1217         \expandafter\XINT_binom_vsmallend_iiib \fi #1.%
1218 }%
1219 \def\XINT_binom_vsmallend_iiib #1.#2.#3.%
1220 {%
1221     \expandafter\XINT_binom_vsmallfinish
1222     \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1223     \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)*(#2+\xint_c_iii)\expandafter
1224     !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1225 }%
1226 \def\XINT_binom_vsmallend_ii #1.%
1227 {%
1228     \ifnum #1>27 \expandafter\XINT_binom_end_ii \else
1229         \expandafter\XINT_binom_vsmallend_iib \fi #1.%
1230 }%
1231 \def\XINT_binom_vsmallend_iib #1.#2.#3.%
1232 {%
1233     \expandafter\XINT_binom_vsmallfinish
1234     \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1235     \the\numexpr #2*(#2+\xint_c_i)*(#2+\xint_c_ii)\expandafter
1236     !\the\numexpr #1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1237 }%
1238 \def\XINT_binom_vsmallend_i #1.%
1239 {%
1240     \ifnum #1>28 \expandafter\XINT_binom_end_i \else
1241         \expandafter\XINT_binom_vsmallend_ib \fi #1.%
1242 }%
1243 \def\XINT_binom_vsmallend_ib #1.#2.#3.%
1244 {%
1245     \expandafter\XINT_binom_vsmallfinish
1246     \the\numexpr \expandafter\XINT_binom_vsmallmuldiv
1247     \the\numexpr #2*(#2+\xint_c_i)\expandafter
1248     !\the\numexpr #1*(#1+\xint_c_i)!%
1249 }%
1250 \def\XINT_binom_vsmallend_ #1.%
1251 {%
1252     \ifnum #1>29 \expandafter\XINT_binom_end_ \else
1253         \expandafter\XINT_binom_vsmallend_b \fi #1.%
1254 }%
1255 \def\XINT_binom_vsmallend_b #1.#2.#3.%
1256 {%
1257     \expandafter\XINT_binom_vsmallfinish
1258     \the\numexpr\XINT_binom_vsmallmuldiv #2!#1!%
1259 }%
1260 \def\XINT_binom_vsmallfinish#1{%
1261 \def\XINT_binom_vsmallfinish1##1!1!;!0!{\expandafter#1\the\numexpr##1\relax}%
1262 }\XINT_binom_vsmallfinish{ }%

```

5.54 \xintiiPFactorial

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

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

```
1263 \def\xintiiPFactorial {\romannumeral0\xintiipfactorial }%
1264 \def\xintiipfactorial #1#2%
1265 {%
1266     \expandafter\XINT_pfac_fork\the\numexpr#1\expandafter.\the\numexpr #2.%
1267 }%
1268 \def\xintPFactorial{\romannumeral0\xintpfactorial}%
1269 \let\xintpfactorial\xintiipfactorial
```

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

```
1270 \def\XINT_pfac_fork #1#2.#3#4.%
1271 {%
1272     \unless\ifnum #1#2<#3#4 \xint_dothis\XINT_pfac_one\fi
1273     \if-#3\xint_dothis\XINT_pfac_neg\fi
1274     \if-#1\xint_dothis\XINT_pfac_zero\fi
1275     \ifnum #3#4>\xint_c_x^viii_mone\xint_dothis\XINT_pfac_outofrange\fi
1276     \xint_orthat \XINT_pfac_a #1#2.#3#4.%
1277 }%
1278 \def\XINT_pfac_outofrange #1.#2.%
1279     {\XINT_signalcondition{InvalidOperation}%
1280     {pFactorial with too large argument: #2 >= 10^8.}}{0}%
1281 \def\XINT_pfac_one #1.#2.{ 1}%
1282 \def\XINT_pfac_zero #1.#2.{ 0}%
1283 \def\XINT_pfac_neg -#1.-#2.%
1284 {%
1285     \ifnum #1>\xint_c_x^viii\xint_dothis\XINT_pfac_outofrange\fi
1286     \xint_orthat
1287     {\ifodd\numexpr#2-#1\relax\xint_afterfi{\expandafter-\romannumeral`&&@\fi
1288     \expandafter\XINT_pfac_a }%
1289     \the\numexpr #2-\xint_c_i\expandafter.\the\numexpr#1-\xint_c_i.%
1290 }%
1291 \def\XINT_pfac_a #1.#2.%
1292 {%
1293     \expandafter\XINT_pfac_b\the\numexpr \xint_c_i+#1.#2.1000000001!1;!%
1294     1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1295 }%
1296 \def\XINT_pfac_b #1.%
1297 {%
1298     \ifnum #1>9999 \xint_dothis\XINT_pfac_vbigloop \fi
1299     \ifnum #1>463 \xint_dothis\XINT_pfac_bigloop \fi
1300     \ifnum #1>98 \xint_dothis\XINT_pfac_medloop \fi
1301     \xint_orthat\XINT_pfac_smallloop #1.%
1302 }%
1303 \def\XINT_pfac_smallloop #1.#2.%
1304 {%
1305     \ifcase\numexpr #2-#1\relax
```

```

1306      \expandafter\XINT_pfac_end_
1307  \or \expandafter\XINT_pfac_end_i
1308  \or \expandafter\XINT_pfac_end_ii
1309  \or \expandafter\XINT_pfac_end_iii
1310  \else\expandafter\XINT_pfac_smallloop_a
1311  \fi #1.#2.%
1312 }%
1313 \def\XINT_pfac_smallloop_a #1.#2.%
1314 {%
1315   \expandafter\XINT_pfac_smallloop_b
1316   \the\numexpr #1+\xint_c_iv\expandafter.%
1317   \the\numexpr #2\expandafter.%
1318   \the\numexpr\expandafter\XINT_smallmul
1319   \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1320 }%
1321 \def\XINT_pfac_smallloop_b #1.%
1322 {%
1323   \ifnum #1>98 \expandafter\XINT_pfac_medloop \else
1324               \expandafter\XINT_pfac_smallloop \fi #1.%
1325 }%
1326 \def\XINT_pfac_medloop #1.#2.%
1327 {%
1328   \ifcase\numexpr #2-#1\relax
1329     \expandafter\XINT_pfac_end_
1330   \or \expandafter\XINT_pfac_end_i
1331   \or \expandafter\XINT_pfac_end_ii
1332   \else\expandafter\XINT_pfac_medloop_a
1333   \fi #1.#2.%
1334 }%
1335 \def\XINT_pfac_medloop_a #1.#2.%
1336 {%
1337   \expandafter\XINT_pfac_medloop_b
1338   \the\numexpr #1+\xint_c_iii\expandafter.%
1339   \the\numexpr #2\expandafter.%
1340   \the\numexpr\expandafter\XINT_smallmul
1341   \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1342 }%
1343 \def\XINT_pfac_medloop_b #1.%
1344 {%
1345   \ifnum #1>463 \expandafter\XINT_pfac_bigloop \else
1346               \expandafter\XINT_pfac_medloop \fi #1.%
1347 }%
1348 \def\XINT_pfac_bigloop #1.#2.%
1349 {%
1350   \ifcase\numexpr #2-#1\relax
1351     \expandafter\XINT_pfac_end_
1352   \or \expandafter\XINT_pfac_end_i
1353   \else\expandafter\XINT_pfac_bigloop_a
1354   \fi #1.#2.%
1355 }%
1356 \def\XINT_pfac_bigloop_a #1.#2.%
1357 {%

```

```

1358 \expandafter\XINT_pfac_bigloop_b
1359 \the\numexpr #1+\xint_c_ii\expandafter.%
1360 \the\numexpr #2\expandafter.%
1361 \the\numexpr\expandafter
1362 \XINT_smallmul\the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
1363 }%
1364 \def\XINT_pfac_bigloop_b #1.%
1365 {%
1366 \ifnum #1>9999 \expandafter\XINT_pfac_vbigloop \else
1367 \expandafter\XINT_pfac_bigloop \fi #1.%
1368 }%
1369 \def\XINT_pfac_vbigloop #1.#2.%
1370 {%
1371 \ifnum #2=#1
1372 \expandafter\XINT_pfac_end_
1373 \else\expandafter\XINT_pfac_vbigloop_a
1374 \fi #1.#2.%
1375 }%
1376 \def\XINT_pfac_vbigloop_a #1.#2.%
1377 {%
1378 \expandafter\XINT_pfac_vbigloop
1379 \the\numexpr #1+\xint_c_i\expandafter.%
1380 \the\numexpr #2\expandafter.%
1381 \the\numexpr\expandafter\XINT_smallmul\the\numexpr\xint_c_x^viii+#1!%
1382 }%
1383 \def\XINT_pfac_end_iii #1.#2.%
1384 {%
1385 \expandafter\XINT_mul_out
1386 \the\numexpr\expandafter\XINT_smallmul
1387 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1388 }%
1389 \def\XINT_pfac_end_ii #1.#2.%
1390 {%
1391 \expandafter\XINT_mul_out
1392 \the\numexpr\expandafter\XINT_smallmul
1393 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
1394 }%
1395 \def\XINT_pfac_end_i #1.#2.%
1396 {%
1397 \expandafter\XINT_mul_out
1398 \the\numexpr\expandafter\XINT_smallmul
1399 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
1400 }%
1401 \def\XINT_pfac_end_ #1.#2.%
1402 {%
1403 \expandafter\XINT_mul_out
1404 \the\numexpr\expandafter\XINT_smallmul\the\numexpr \xint_c_x^viii+#1!%
1405 }%

```

5.55 \xintBool, \xintToggle

1.09c

```
1406 \def\xintBool #1{\romannumeral`&&@%
1407         \csname if#1\endcsname\expandafter1\else\expandafter0\fi }%
1408 \def\xintToggle #1{\romannumeral`&&@iftoggle{#1}{1}{0}}%
```

5.56 \xintiigcd

1.3d: \xintiigcd code from xintgcd is copied here to support gcd() function in \xintiexpr.

1.4: removed from xintgcd the original caode as now xintgcd loads xint.

1.4d (2021/03/29) [commented 2021/03/22]. Damn'ed! Since 1.3d (2019/01/06) the code was broken if one of the arguments vanished due to a typo in macro names: "AisZero" at one location and "Aiszero" at next, and same for B...

How could this not be detected by my tests !?!

This caused \xintiigcdof hence the gcd() function in \xintiexpr to break as soon as one argument was zero.

```
1409 \def\xintiigcd {\romannumeral0\xintiigcd }%
1410 \def\xintiigcd #1{\expandafter\XINT_iigcd\romannumeral0\xintiabs#1\xint:}%
1411 \def\XINT_iigcd #1#2\xint:#3%
1412 {%
1413     \expandafter\XINT_gcd_fork\expandafter#1%
1414     \romannumeral0\xintiabs#3\xint:#1#2\xint:
1415 }%
1416 \def\XINT_gcd_fork #1#2%
1417 {%
1418     \xint_UDzerofork
1419     #1\XINT_gcd_Aiszero
1420     #2\XINT_gcd_Biszero
1421     0\XINT_gcd_loop
1422     \krof
1423     #2%
1424 }%
1425 \def\XINT_gcd_Aiszero #1\xint:#2\xint:{ #1}%
1426 \def\XINT_gcd_Biszero #1\xint:#2\xint:{ #2}%
1427 \def\XINT_gcd_loop #1\xint:#2\xint:
1428 {%
1429     \expandafter\expandafter\expandafter\XINT_gcd_CheckRem
1430     \expandafter\xint_secondoftwo
1431     \romannumeral0\XINT_div_prepare {#1}{#2}\xint:#1\xint:
1432 }%
1433 \def\XINT_gcd_CheckRem #1%
1434 {%
1435     \xint_gob_til_zero #1\XINT_gcd_end0\XINT_gcd_loop #1%
1436 }%
1437 \def\XINT_gcd_end0\XINT_gcd_loop #1\xint:#2\xint:{ #2}%

```

5.57 \xintiigcdof

New with 1.09a (was located in 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.

1.4d fixes breakage inherited since 1.3d rom \xintiigcd, in case any argument vanished. Currently does not support empty list of arguments.

```

1438 \def\xintiigCDof    {\romannumeral0\xintiigcdof }%
1439 \def\xintiigcdof    #1{\expandafter\XINT_iigcdof_a\romannumeral`&&@#1^}%
1440 \def\XINT_iigCDof    {\romannumeral0\XINT_iigcdof_a}%
1441 \def\XINT_iigcdof_a  #1{\expandafter\XINT_iigcdof_b\romannumeral`&&@#1!}%
1442 \def\XINT_iigcdof_b  #1!#2{\expandafter\XINT_iigcdof_c\romannumeral`&&@#2!{#1}!}%
1443 \def\XINT_iigcdof_c  #1{\xint_gob_til_^ #1\XINT_iigcdof_e ^\XINT_iigcdof_d #1}%
1444 \def\XINT_iigcdof_d  #1!{\expandafter\XINT_iigcdof_b\romannumeral0\xintiigcd {#1}}%
1445 \def\XINT_iigcdof_e  #1!#2!{ #2}%

```

5.58 \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`.

```

1446 \def\xintiilCM {\romannumeral0\xintiilcm}%
1447 \def\xintiilcm #1{\expandafter\XINT_iilcm\romannumeral0\xintiabs#1\xint:}%
1448 \def\XINT_iilcm #1#2\xint:#3%
1449 {%
1450     \expandafter\XINT_lcm_fork\expandafter#1%
1451         \romannumeral0\xintiabs#3\xint:#1#2\xint:
1452 }%
1453 \def\XINT_lcm_fork #1#2%
1454 {%
1455     \xint_UDzerofork
1456     #1\XINT_lcm_iszero
1457     #2\XINT_lcm_iszero
1458     0\XINT_lcm_notzero
1459     \krof
1460     #2%
1461 }%
1462 \def\XINT_lcm_iszero #1\xint:#2\xint:{ 0}%
1463 \def\XINT_lcm_notzero #1\xint:#2\xint:
1464 {%
1465     \expandafter\XINT_lcm_end\romannumeral0%
1466     \expandafter\expandafter\expandafter\XINT_gcd_CheckRem
1467     \expandafter\xint_secondoftwo
1468     \romannumeral0\XINT_div_prepare {#1}{#2}\xint:#1\xint:
1469     \xint:#1\xint:#2\xint:
1470 }%
1471 \def\XINT_lcm_end #1\xint:#2\xint:#3\xint:{\xintiimul {#2}{\xintiQuo{#3}{#1}}}%

```

5.59 \xintiilCMof

See comments of `\xintiigCDof`

```

1472 \def\xintiilCMof    {\romannumeral0\xintiilcmof }%
1473 \def\xintiilcmof    #1{\expandafter\XINT_iilcmof_a\romannumeral`&&@#1^}%
1474 \def\XINT_iilCMof    {\romannumeral0\XINT_iilcmof_a}%
1475 \def\XINT_iilcmof_a  #1{\expandafter\XINT_iilcmof_b\romannumeral`&&@#1!}%
1476 \def\XINT_iilcmof_b  #1!#2{\expandafter\XINT_iilcmof_c\romannumeral`&&@#2!{#1}!}%
1477 \def\XINT_iilcmof_c  #1{\xint_gob_til_^ #1\XINT_iilcmof_e ^\XINT_iilcmof_d #1}%
1478 \def\XINT_iilcmof_d  #1!{\expandafter\XINT_iilcmof_b\romannumeral0\xintiilcm {#1}}%
1479 \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.

```

1480 \def\xintRandomDigits{\romannumeral0\xintrandomdigits}%
1481 \def\xintrandomdigits#1%
1482 {%
1483     \csname xint_gob_andstop_\expandafter\XINT_randomdigits\the\numexpr#1\xint:
1484 }%
1485 \def\XINT_randomdigits#1\xint:
1486 {%
1487     \expandafter\XINT_randomdigits_a
1488     \the\numexpr(#1+\xint_c_iii)/\xint_c_viii\xint:#1\xint:
1489 }%
1490 \def\XINT_randomdigits_a#1\xint:#2\xint:
1491 {%
1492     \romannumeral\numexpr\xint_c_viii*#1-#2\csname XINT_%
1493     \romannumeral\XINT_replicate #1\endcsname \csname
1494     XINT_rdg\endcsname
1495 }%
1496 \def\XINT_rdg
1497 {%
1498     \expandafter\XINT_rdg_aux\the\numexpr%
1499         \xint_c_nine_x^viii%
1500         -\xint_texuniformdeviate\xint_c_ii^vii%
1501         -\xint_c_ii^vii*\xint_texuniformdeviate\xint_c_ii^vii%
1502         -\xint_c_ii^xiv*\xint_texuniformdeviate\xint_c_ii^vii%
1503         -\xint_c_ii^xxi*\xint_texuniformdeviate\xint_c_ii^vii%
1504         +\xint_texuniformdeviate\xint_c_x^viii%
1505         \relax%
1506 }%
1507 \def\XINT_rdg_aux#1{XINT_rdg\endcsname}%
1508 \let\XINT_XINT_rdg\endcsname

```

5.61 (WIP) \XINT_eightrandomdigits, \xintEightRandomDigits

1.3b. 1.4 adds some public alias...

```

1509 \def\XINT_eightrandomdigits
1510 {%
1511     \expandafter\xint_gobble_i\the\numexpr%
1512         \xint_c_nine_x^viii%
1513         -\xint_texuniformdeviate\xint_c_ii^vii%
1514         -\xint_c_ii^vii*\xint_texuniformdeviate\xint_c_ii^vii%
1515         -\xint_c_ii^xiv*\xint_texuniformdeviate\xint_c_ii^vii%
1516         -\xint_c_ii^xxi*\xint_texuniformdeviate\xint_c_ii^vii%
1517         +\xint_texuniformdeviate\xint_c_x^viii%
1518         \relax%
1519 }%
1520 \let\xintEightRandomDigits\XINT_eightrandomdigits
1521 \def\xintRandBit{\xint_texuniformdeviate\xint_c_ii}%

```


5.62 (WIP) \xintRandBit

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

```
1522 \def\xintRandBit{\xint_texuniformdeviate\xint_c_ii}%
```

5.63 (WIP) \xintXRandomDigits

1.3b.

```
1523 \def\xintXRandomDigits#1%
1524 {%
1525   \csname xint_gobble_\expandafter\xint_xrandomdigits\the\numexpr#1\xint:
1526 }%
1527 \def\xint_xrandomdigits#1\xint:
1528 {%
1529   \expandafter\xint_xrandomdigits_a
1530   \the\numexpr(#1+\xint_c_iii)/\xint_c_viii\xint:#1\xint:
1531 }%
1532 \def\xint_xrandomdigits_a#1\xint:#2\xint:
1533 {%
1534   \romannumeral\numexpr\xint_c_viii*#1-#2\expandafter\endcsname
1535   \romannumeral`&&\romannumeral
1536   \XINT_replicate #1\endcsname\xint_eightrandomdigits
1537 }%
```

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 \xintiexpr will call as is.

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

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

```
1538 \def\xintiirandrangeAtoB{\romannumeral`&&\xintiirandrangeAtoB}%
1539 \def\xintiirandrangeAtoB#1%
1540 {%
1541   \expandafter\xint_randrangeAtoB_a\romannumeral`&&#1\xint:
1542 }%
1543 \def\xint_randrangeAtoB_a#1\xint:#2%
1544 {%
1545   \xintiirandrangeAtoB_a#1\xint:#2%
1546   \romannumeral0\xintiirandrangeAtoB_a#1\xint:#2%
1547   {#1}%
1548 }%
```

5.65 (WIP) \xintiirandrange

1.3b. Support for randrange().

```
1549 \def\xintiirandrange{\romannumeral`&&\xintiirandrange}%
1550 \def\xintiirandrange#1%
1551 {%
1552   \expandafter\xint_randrange\romannumeral`&&#1\xint:
```

```

1553 }%
1554 \def\XINT_randrange #1%
1555 {%
1556     \xint_UDzerominusfork
1557     #1-\XINT_randrange_err:empty
1558     0#1\XINT_randrange_err:empty
1559     0-\XINT_randrange_a
1560     \krof #1%
1561 }%
1562 \def\XINT_randrange_err:empty#1\xint:
1563 {%
1564     \XINT_expandableerror{Empty range for randrange.} 0%
1565 }%
1566 \def\XINT_randrange_a #1\xint:
1567 {%
1568     \expandafter\XINT_randrange_b\romannumeral0\xintlength{#1}.#1\xint:
1569 }%
1570 \def\XINT_randrange_b #1.%
1571 {%
1572     \ifnum#1<\xint_c_x\xint_dothis{\the\numexpr\XINT_uniformdeviate{}}\fi
1573     \xint_orthat{\XINT_randrange_c #1.}%
1574 }%
1575 \def\XINT_randrange_c #1.#2#3#4#5#6#7#8#9%
1576 {%
1577     \expandafter\XINT_randrange_d
1578     \the\numexpr\expandafter\XINT_uniformdeviate\expandafter
1579     {\expandafter}\the\numexpr\xint_c_i+#2#3#4#5#6#7#8#9\xint:\xint:
1580     #2#3#4#5#6#7#8#9\xint:#1\xint:
1581 }%

```

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.

```

1582 \def\XINT_randrange_d #1\xint:#2\xint:
1583 {%
1584     \ifnum#1=\xint_c_\xint_dothis\XINT_randrange_Z\fi
1585     \ifnum#1=#2 \xint_dothis\XINT_randrange_A\fi
1586     \xint_orthat\XINT_randrange_e #1\xint:
1587 }%
1588 \def\XINT_randrange_e #1\xint:#2\xint:#3\xint:
1589 {%
1590     \the\numexpr#1\expandafter\relax
1591     \romannumeral0\xintrandomdigits{#2-\xint_c_viii}%
1592 }%

```

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...

```

1593 \def\XINT_randrange_Z 0\xint:#1\xint:#2\xint:
1594 {%
1595     \xintinum{\xintRandomDigits{#1-\xint_c_viii}}%
1596 }%

```

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.

```

1597 \def\XINT_randrange_A #1\xint:#2\xint:#3\xint:
1598 {%
1599   \expandafter\XINT_randrange_B
1600   \romannumeral0\xintrandomdigits{#2-\xint_c_viii}\xint:
1601   #3\xint:#2.#1\xint:
1602 }%
1603 \def\XINT_randrange_B #1\xint:#2\xint:#3.#4\xint:
1604 {%
1605   \xintiiflt{#1}{#2}{\XINT_randrange_E}{\XINT_randrange_again}%
1606   #4#1\xint:#3.#4#2\xint:
1607 }%
1608 \def\XINT_randrange_E #1\xint:#2\xint:{ #1}%
1609 \def\XINT_randrange_again #1\xint:{\XINT_randrange_c}%

```

5.66 (WIP) Adjustments for engines without uniformdeviate primitive

1.3b.

```

1610 \ifdefined\xint_texuniformdeviate
1611 \else
1612   \def\xintrandomdigits#1%
1613   {%
1614     \XINT_expandableerror
1615     {No uniformdeviate at engine level.} 0%
1616   }%
1617   \let\xintXRandomDigits\xintRandomDigits
1618   \def\XINT_randrange#1\xint:
1619   {%
1620     \XINT_expandableerror
1621     {No uniformdeviate at engine level.} 0%
1622   }%
1623 \fi
1624 \XINTrestorecatcodesendinginput%

```

6 Package *xintbinhex* implementation

.1	Catcodes, ε -TeX and reload detection . . .	164	.6	<code>\xintDecToBin</code>	169
.2	Package identification	165	.7	<code>\xintHexToDec</code>	170
.3	Constants, etc...	165	.8	<code>\xintBinToDec</code>	172
.4	Helper macros	166	.9	<code>\xintBinToHex</code>	173
.4.1	<code>\XINT_zeroes_foriv</code>	166	.10	<code>\xintHexToBin</code>	174
.5	<code>\xintDecToHex</code>	166	.11	<code>\xintCHexToBin</code>	174

The commenting is currently (2021/06/11) 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\x\empty % LaTeX, first loading,
33 % variable is initialized, but \ProvidesPackage not yet seen
34 \ifx\w\relax % xintkernel.sty not yet loaded.
35 \def\z{\endgroup\RequirePackage{xintkernel}}%
36 \fi
37 \else
38 \aftergroup\endinput % xintbinhex already loaded.
39 \fi
40 \fi
41 \fi
42 \z%
43 \XINTsetupcatcodes% defined in xintkernel.sty

```

6.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintbinhex}%
46 [2021/06/11 v1.4i 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_csdth_#1\endcsname
57 {\endcsname\ifcase #1
58 0000\or 0001\or 0010\or 0011\or 0100\or 0101\or 0110\or 0111\or
59 1000\or 1001\or 1010\or 1011\or 1100\or 1101\or 1110\or 1111\fi}%
60 \expandafter\XINT_tmpa\fi }%
61 \XINT_tmpa {0}{1}{2}{3}{4}{5}{6}{7}{8}{9}{10}{11}{12}{13}{14}{15}\relax
62 \let\XINT_tmpa\relax
63 \expandafter\def\csname XINT_csbth_0000\endcsname {\endcsname0}%
64 \expandafter\def\csname XINT_csbth_0001\endcsname {\endcsname1}%
65 \expandafter\def\csname XINT_csbth_0010\endcsname {\endcsname2}%
66 \expandafter\def\csname XINT_csbth_0011\endcsname {\endcsname3}%
67 \expandafter\def\csname XINT_csbth_0100\endcsname {\endcsname4}%
68 \expandafter\def\csname XINT_csbth_0101\endcsname {\endcsname5}%
69 \expandafter\def\csname XINT_csbth_0110\endcsname {\endcsname6}%
70 \expandafter\def\csname XINT_csbth_0111\endcsname {\endcsname7}%
71 \expandafter\def\csname XINT_csbth_1000\endcsname {\endcsname8}%
72 \expandafter\def\csname XINT_csbth_1001\endcsname {\endcsname9}%
73 \expandafter\def\csname XINT_csbth_1010\endcsname {\endcsname A}%
74 \expandafter\def\csname XINT_csbth_1011\endcsname {\endcsname B}%
75 \expandafter\def\csname XINT_csbth_1100\endcsname {\endcsname C}%

```

```

76 \expandafter\def\csname XINT_csbth_1101\endcsname {\endcsname D}%
77 \expandafter\def\csname XINT_csbth_1110\endcsname {\endcsname E}%
78 \expandafter\def\csname XINT_csbth_1111\endcsname {\endcsname F}%
79 \let\XINT_csbth_none \endcsname
80 \expandafter\def\csname XINT_cshtb_0\endcsname {\endcsname0000}%
81 \expandafter\def\csname XINT_cshtb_1\endcsname {\endcsname0001}%
82 \expandafter\def\csname XINT_cshtb_2\endcsname {\endcsname0010}%
83 \expandafter\def\csname XINT_cshtb_3\endcsname {\endcsname0011}%
84 \expandafter\def\csname XINT_cshtb_4\endcsname {\endcsname0100}%
85 \expandafter\def\csname XINT_cshtb_5\endcsname {\endcsname0101}%
86 \expandafter\def\csname XINT_cshtb_6\endcsname {\endcsname0110}%
87 \expandafter\def\csname XINT_cshtb_7\endcsname {\endcsname0111}%
88 \expandafter\def\csname XINT_cshtb_8\endcsname {\endcsname1000}%
89 \expandafter\def\csname XINT_cshtb_9\endcsname {\endcsname1001}%
90 \def\XINT_cshtb_A {\endcsname1010}%
91 \def\XINT_cshtb_B {\endcsname1011}%
92 \def\XINT_cshtb_C {\endcsname1100}%
93 \def\XINT_cshtb_D {\endcsname1101}%
94 \def\XINT_cshtb_E {\endcsname1110}%
95 \def\XINT_cshtb_F {\endcsname1111}%
96 \let\XINT_cshtb_none \endcsname

```

6.4 Helper macros

6.4.1 \XINT_zeroes_foriv

```

\romannumeral0\XINT_zeroes_foriv #1\R{0\R}{00\R}{000\R}%
\endcsname\R{0\R}{00\R}{000\R}\R\W
expands to the <empty> or 0 or 00 or 000 needed which when adjoined to #1 extend it to length 4N.
97 \def\XINT_zeroes_foriv #1#2#3#4#5#6#7#8%
98 {%
99     \xint_gob_til_R #8\XINT_zeroes_foriv_end\R\XINT_zeroes_foriv
100}%
101 \def\XINT_zeroes_foriv_end\R\XINT_zeroes_foriv #1#2\W
102     {\XINT_zeroes_foriv_done #1}%
103 \def\XINT_zeroes_foriv_done #1\R{ #1}%

```

6.5 \xintDecToHex

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

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

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

```

104 \def\xintDecToHex {\romannumeral0\xintdectohex}%
105 \def\xintdectohex #1%
106 {%
107     \expandafter\XINT_dth_checkin\romannumeral`&&@#1\xint:
108}%
109 \def\XINT_dth_checkin #1%
110 {%
111     \xint_UDsignfork
112     #1\XINT_dth_neg

```

```

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

```

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

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

```

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

```

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

```

149 \def\XINT_dthb_nextfour #1#2#3#4#5%
150 {%
151     \xint_bye#5\XINT_dthb_lastpass\xint_bye
152     #1#2#3#4!\XINT_dthb_exclam\relax\XINT_dthb_nextfour#5%
153 }%
154 \def\XINT_dthb_lastpass\xint_bye #1!#2\xint_bye#3{#1!#3!}%
155 \def\XINT_dth_tohex
156 {%

```

```

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

```

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

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

```

204 \def\XINT_dth_finish !\XINT_dth_tohex!#1#2#3%

```



```

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

```

6.6 \xintDecToBin

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

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

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

Most of the code canvas is shared with \xintDecToHex.

```

211 \def\xintDecToBin {\romannumeral0\xintdectobin }%
212 \def\xintdectobin #1%
213 {%
214   \expandafter\XINT_dtb_checkin\romannumeral`&&@#1\xint:
215 }%
216 \def\XINT_dtb_checkin #1%
217 {%
218   \xint_UDsignfork
219     #1\XINT_dtb_neg
220     -{\XINT_dtb_main #1}%
221   \krof
222 }%
223 \def\XINT_dtb_neg {\expandafter-\romannumeral0\XINT_dtb_main}%
224 \def\XINT_dtb_main #1\xint:
225 {%
226   \expandafter\XINT_dtb_finish
227   \romannumeral`&&\expandafter\XINT_dtb_start
228   \romannumeral0\XINT_zeroes_foriv
229     #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
230   #1\xint_bye\XINT_dtb_tobin
231 }%
232 \def\XINT_dtb_tobin
233 {%
234   \expandafter\expandafter\expandafter\XINT_dtb_tobin_a\cename\XINT_tosixteenbits
235 }%
236 \def\XINT_dtb_tobin_a\endcsname{!\XINT_dtb_tobin!}%
237 \def\XINT_tosixteenbits #1!%
238 {%
239   \expandafter\XINT_tosixteenbits_a
240   \the\numexpr (#1+\xint_c_ii^vii)/\xint_c_ii^viii-\xint_c_i\xint:
241   #1\xint:
242 }%
243 \def\XINT_tosixteenbits_a #1\xint:#2\xint:
244 {%
245   \expandafter\XINT_tosixteenbits_c
246   \the\numexpr (#1+\xint_c_viii)/\xint_c_xvi-\xint_c_i\xint:
247   #1\xint:
248   \the\numexpr #2-\xint_c_ii^viii*#1!%
249 }%

```

```

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

```

6.7 \xintHexToDec

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

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

Increased maximal size at 1.2n.

1.2m version robust against non terminated inputs.

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

```

276 \def\xintHexToDec {\romannumeral0\xinthextodec }%
277 \def\xinthextodec #1%
278 {%
279     \expandafter\XINT_htd_checkin\romannumeral`&&@#1\xint:
280 }%
281 \def\XINT_htd_checkin #1%
282 {%
283     \xint_UDsignfork
284     #1\XINT_htd_neg
285     -{\XINT_htd_main #1}%
286     \krof
287 }%
288 \def\XINT_htd_neg {\expandafter-\romannumeral0\XINT_htd_main}%
289 \def\XINT_htd_main #1\xint:
290 {%

```

```

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

```

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

```

308 \def\XINT_htd_again #1\XINT_htd_nextfour #2%
309 {%
310 \xint_bye #2\XINT_htd_finish\xint_bye
311 \expandafter\XINT_htd_A\the\numexpr
312 \XINT_htd_a #1\XINT_htd_nextfour #2%
313 }%
314 \def\XINT_htd_a #1!#2!#3!#4!#5!#6!#7!#8!#9!%
315 {%
316 #1\expandafter\XINT_htd_update
317 \the\numexpr #2\expandafter\XINT_htd_update
318 \the\numexpr #3\expandafter\XINT_htd_update
319 \the\numexpr #4\expandafter\XINT_htd_update
320 \the\numexpr #5\expandafter\XINT_htd_update
321 \the\numexpr #6\expandafter\XINT_htd_update
322 \the\numexpr #7\expandafter\XINT_htd_update
323 \the\numexpr #8\expandafter\XINT_htd_update
324 \the\numexpr #9\expandafter\XINT_htd_update
325 \the\numexpr \XINT_htd_a
326 }%
327 \def\XINT_htd_nextfour #1#2#3#4%
328 {%
329 *\xint_c_ii^xvi+"#1#2#3#4+1000000000\relax\xint_bye!%
330 2!3!4!5!6!7!8!9!\xint_bye\XINT_htd_nextfour
331 }%

```

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

```

332 \def\XINT_htd_update 1#1#2#3#4#5#6!%
333 {%
334 *\xint_c_ii^xvi+10000#1#2#3#4#5!%#6!%
335 }%
336 \def\XINT_htd_A 1#1%

```

```

337 {%
338   \if#10\expandafter\XINT_htd_Aa\else
339     \expandafter\XINT_htd_Ab
340   \fi 1#1%
341 }%
342 \def\XINT_htd_Aa 10#1#2#3#4{\XINT_htd_again #1#2#3#4!}%
343 \def\XINT_htd_Ab 1#1#2#3#4#5{\XINT_htd_again #1!#2#3#4#5!}%
344 \def\XINT_htd_finish\xint_bye
345   \expandafter\XINT_htd_A\the\numexpr \XINT_htd_a #1\XINT_htd_nextfour
346 {%
347   \expandafter\XINT_htd_finish_cuz\the\numexpr0\XINT_htd_unsep_loop #1%
348 }%
349 \def\XINT_htd_unsep_loop #1!#2!#3!#4!#5!#6!#7!#8!#9!%
350 {%
351   \expandafter\XINT_unsep_clean
352   \the\numexpr 1#1#2\expandafter\XINT_unsep_clean
353   \the\numexpr 1#3#4\expandafter\XINT_unsep_clean
354   \the\numexpr 1#5#6\expandafter\XINT_unsep_clean
355   \the\numexpr 1#7#8\expandafter\XINT_unsep_clean
356   \the\numexpr 1#9\XINT_htd_unsep_loop_a
357 }%
358 \def\XINT_htd_unsep_loop_a #1!#2!#3!#4!#5!#6!#7!#8!#9!%
359 {%
360   #1\expandafter\XINT_unsep_clean
361   \the\numexpr 1#2#3\expandafter\XINT_unsep_clean
362   \the\numexpr 1#4#5\expandafter\XINT_unsep_clean
363   \the\numexpr 1#6#7\expandafter\XINT_unsep_clean
364   \the\numexpr 1#8#9\XINT_htd_unsep_loop
365 }%
366 \def\XINT_unsep_clean 1{\relax}% also in xintcore
367 \def\XINT_htd_finish_cuz #1{%
368 \def\XINT_htd_finish_cuz ##1##2##3##4##5%
369   {\expandafter#1\the\numexpr ##1##2##3##4##5\relax}%
370 }\XINT_htd_finish_cuz{ }%

```

6.8 \xintBinToDec

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

Increased maximal size at 1.2n.

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

Robust against non-terminated input.

```

371 \def\xintBinToDec {\romannumeral0\xintbintodec }%
372 \def\xintbintodec #1%
373 {%
374   \expandafter\XINT_btd_checkin\romannumeral`&&@#1\xint:
375 }%
376 \def\XINT_btd_checkin #1%
377 {%
378   \xint_UDsignfork
379     #1\XINT_btd_N
380     -{\XINT_btd_main #1}%
381   \krof

```

```

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

```

6.9 \xintBinToHex

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

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

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

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

Robust against non-terminated input.

```

399 \def\xintBinToHex {\romannumeral0\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_hnb_loop}%
472 \XINTrestorecatcodesendinginput%
```

7 Package [xintgcd](#) implementation

.1	Catcodes, ε -TeX and reload detection . .	176	.5	<code>\xintBezoutAlgorithm</code>	182
.2	Package identification	177	.6	<code>\xintTypesetEuclideanAlgorithm</code>	184
.3	<code>\xintBezout</code>	177	.7	<code>\xintTypesetBezoutAlgorithm</code>	185
.4	<code>\xintEuclideanAlgorithm</code>	181			

The commenting is currently (2021/06/11) very sparse.

Release [1.09h](#) has modified a bit the `\xintTypesetEuclideanAlgorithm` and `\xintTypesetBezoutAlgorithm` layout with respect to line indentation in particular. And they use the [xinttools](#) `\xintloop` rather than the Plain TeX or $\mathbb{E}\mathbb{T}\mathbb{X}$'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 [2021/06/11 v1.4i 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 1.21.
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\xintiiopp{#2}}{\romannumeral0\xintiiopp{#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\xintiopp {#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\xintiopp{#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\xintiiadd{\XINT_mul_plusplus{}}{#1\xint:#4\xint:}{#6}}%

```

```

183      {\romannumeral0\xintiiadd{\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}}\dots\{A\}\{B\}}\Z
a(n) = r(n-1). Pour n=0 on a juste {0}\{B\}\{A\}\{A\}\{B\}}\Z
\XINT_div_prepare {u}{v} divise v par u

```

```

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 #3\Z {#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#2\Z
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#4\Z%
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

$\{N\}{A}\{0\}\{1\}{D=r(n)}\{B\}\{1\}\{0\}\{q1\}{r1}\{\alpha1=q1\}\{\beta1=1\}$
 $\{q2\}{r2}\{\alpha2\}\{\beta2\}...\{qN\}{rN=0}\{\alphaN=A/D\}\{\betaN=B/D\}$
 $\alpha0=1, \beta0=0, \alpha(-1)=0, \beta(-1)=1$

```

241 \def\xintBezoutAlgorithm {\romannumeral0\xintbezoutalgorithm}%
242 \def\xintbezoutalgorithm #1%
243 {%
244     \expandafter \XINT_bezalg
245     \expandafter{\romannumeral0\xinttiabs{\xintNum{#1}}}%
246 }%
247 \def\XINT_bezalg #1#2%
248 {%
249     \expandafter\XINT_bezalg_fork\romannumeral0\xinttiabs{\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}{1}{#2}{#2}{1}{0}{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}{D}{B}{q1}{r1}{q2}{r2}{q3}{r3}...{qN}{rN=0}
{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:

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

$\backslash U_1 = N = \text{nombre d'étapes}$, $\backslash U_3 = \text{PGCD}$, $\backslash U_2 = A$, $\backslash U_4 = B$ $q_1 = \backslash U_5$, $q_2 = \backslash U_7 \rightarrow q_n = \backslash U_{\langle 2n+3 \rangle}$, $r_n = \backslash U_{\langle 2n+4 \rangle}$ $bn = rn$. $B = r_0$. $A = r(-1)$

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

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

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

```

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   \begingroup
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     \begingroup
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             $\U\times\A - \V\times\B = \D$%
```

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

```
364    \fi
365    \par
366  \endgroup
367 }%
368 \XINTrestorecatcodesendinginput%
```

8 Package **xintfrac** implementation

.1	Catcodes, ε -TeX and reload detection . . .	188	.46	\xintBinomial	223
.2	Package identification	189	.47	\xintPFactorial	223
.3	\XINT_cntSgnFork	189	.48	\xintPrd	224
.4	\xintLen	189	.49	\xintDiv	224
.5	\XINT_outfrac	189	.50	\xintDivFloor	225
.6	\XINT_inFrac	190	.51	\xintDivTrunc	225
.7	\XINT_frac_gen	192	.52	\xintDivRound	225
.8	\XINT_factortens	194	.53	\xintModTrunc	225
.9	\xintEq, \xintNotEq, \xintGt, \xintLt, \xintGtorEq, \xintLtorEq, \xintIsZero, \xintIsNotZero, \xintOdd, \xintEven, \xintifSgn, \xintifCmp, \xintifEq, \xin- tifGt, \xintifLt, \xintifZero, \xin- tifNotZero, \xintifOne, \xintifOdd . . .	195	.54	\xintDivMod	226
.10	\xintRaw	197	.55	\xintMod	227
.11	\xintiLogTen	197	.56	\xintIsOne	228
.12	\xintPRaw	198	.57	\xintGeq	228
.13	\xintSPRaw	198	.58	\xintMax	229
.14	\xintFracToSci	198	.59	\xintMaxof	230
.15	\xintRawWithZeros	200	.60	\xintMin	231
.16	\xintDecToString	200	.61	\xintMinof	231
.17	\xintDecToStringREZ	201	.62	\xintCmp	232
.18	\xintFloor, \xintiFloor	201	.63	\xintAbs	233
.19	\xintCeil, \xintiCeil	201	.64	\xintOpp	233
.20	\xintNumerator	201	.65	\xintInv	233
.21	\xintDenominator	202	.66	\xintSgn	234
.22	\xintTeXFrac	202	.67	\xintGCD	234
.23	\xintTeXsignedFrac	203	.68	\xintGCDof	235
.24	\xintTeXfromSci	203	.69	\xintLCM	236
.25	\xintTeXOver	204	.70	\xintLCMof	237
.26	\xintTeXsignedOver	204	.71	Floating point macros	238
.27	\xintREZ	205	.72	\xintDigits, \xintSetDigits	240
.28	\xintE	206	.73	\xintFloat	240
.29	\xintIrr, \xintPIrr	206	.74	\XINTinFloat, \XINTinFloatS, \XIN- TiLogTen	241
.30	\xintifInt	208	.75	\xintPFloat, \xintPFloatE	248
.31	\xintIsInt	208	.76	\XINTinFloatFrac	250
.32	\xintJrr	208	.77	\xintFloatAdd, \XINTinFloatAdd	250
.33	\xintTFrac	209	.78	\xintFloatSub, \XINTinFloatSub	251
.34	\xintTrunc, \xintiTrunc	210	.79	\xintFloatMul, \XINTinFloatMul	252
.35	\xintTTrunc	213	.80	\xintFloatSqr, \XINTinFloatSqr	253
.36	\xintNum	213	.81	\XINTinFloatInv	253
.37	\xintRound, \xintiRound	213	.82	\xintFloatDiv, \XINTinFloatDiv	254
.38	\xintXTrunc	213	.83	\xintFloatPow, \XINTinFloatPow	255
.39	\xintAdd	219	.84	\xintFloatPower, \XINTinFloatPower	258
.40	\xintSub	221	.85	\xintFloatFac, \XINTFloatFac	261
.41	\xintSum	221	.86	\xintFloatPFactorial, \XINTinFloatP- Factorial	266
.42	\xintMul	221	.87	\xintFloatBinomial, \XINTinFloatBino- mial	270
.43	\xintSqr	222	.88	\xintFloatSqrt, \XINTinFloatSqrt	271
.44	\xintPow	222	.89	\xintFloatE, \XINTinFloatE	274
.45	\xintFac	223	.90	\XINTinFloatMod	275
			.91	\XINTinFloatDivFloor	275
			.92	\XINTinFloatDivMod	275

.93	\xintifFloatInt	276	.97	(WIP) \XINTinRandomFloatS, \XINTinRandomFloatSdigits	277
.94	\xintFloatIsInt	276	.98	(WIP) \XINTinRandomFloatSixteen	277
.95	\xintFloatIntType	276			
.96	\XINTinFloatdigits, \XINTinFloatSdigits	276			

The commenting is currently (2021/06/11) 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 [2021/06/11 v1.4i 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[#3]%
80 {%
81     \XINT_signalcondition{DivisionByZero}{Division by zero: #1/#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 lift-ing 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:
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:\xint:\Z~#3~%
179 }%
180 \def\XINT_frac_gen_Cb #1.#2e%
181 {%
182     \expandafter\XINT_frac_gen_Cc\romannumeral`&&@#2.#1e%
183 }%
184 \def\XINT_frac_gen_Cc #1.#2~#3!#4e#5e#6\XINT_T
185 {%
186     \expandafter\XINT_frac_gen_F\the\numexpr #5-#2-%
187     \numexpr\XINT_length_loop
188     #1\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
189     \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
190     \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
191     \relax\expandafter~%
192     \romannumeral0\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
193     #3\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z
194     ~#4#1~%
195 }%
196 \def\XINT_frac_gen_F #1~#2%
197 {%
198     \xint_UDzerominusfork
199     #2-\XINT_frac_gen_Gdivbyzero
200     0#2{\XINT_frac_gen_G -{}}%
201     0-{\XINT_frac_gen_G }{#2}%
202     \krof #1~%
203 }%
204 \def\XINT_frac_gen_Gdivbyzero #1~#2~%
205 {%
206     \expandafter\XINT_frac_gen_Gdivbyzero_a
207     \romannumeral0\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
208     #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z~#1~%
209 }%
210 \def\XINT_frac_gen_Gdivbyzero_a #1~#2~%
211 {%
212     \XINT_signalcondition{DivisionByZero}{Division by zero: #1/0.}{}{{#2}{#1}{0}}%
213 }%
214 \def\XINT_frac_gen_G #1#2#3~#4~#5~%
215 {%
216     \expandafter\XINT_frac_gen_Ga

```

```

217 \romannumeral0\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
218 #1#5\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z~#3~{#2#4}%
219 }%
220 \def\XINT_frac_gen_Ga #1#2~#3~%
221 {%
222     \xint_gob_til_zero #1\XINT_frac_gen_zero 0%
223     {#3}{#1#2}%
224 }%
225 \def\XINT_frac_gen_zero 0#1#2#3{{0}{0}{1}}%

```

8.8 \XINT_factortens

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

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

Testing shows significant gain at 100 digits or more.

```

226 \def\XINT_factortens #1{\expandafter\XINT_factortens_z
227 \romannumeral0\XINT_factortens_a#1%
228 \XINT_factortens_b123456789.}%
229 \def\XINT_factortens_z.\XINT_factortens_y{ }%
230 \def\XINT_factortens_a #1#2#3#4#5#6#7#8#9%
231 {\expandafter\XINT_factortens_x
232 \the\numexpr 1#1#2#3#4#5#6#7#8#9\XINT_factortens_a}%
233 \def\XINT_factortens_b#1\XINT_factortens_a#2#3.%
234 {\XINT_factortens_cc 000000000-#2.}%
235 \def\XINT_factortens_x1#1.#2{#2#1}%
236 \def\XINT_factortens_y{\XINT_factortens_y}%
237 \def\XINT_factortens_cc #1#2#3#4#5#6#7#8#9%
238 {\if90\xint_dothis
239 {\expandafter\XINT_factortens_d\the\numexpr #8#7#6#5#4#3#2#1\relax
240 \xint_c_i 2345678.}\fi
241 \xint_orthat{\XINT_factortens_yy{#1#2#3#4#5#6#7#8#9}}}%
242 \def\XINT_factortens_yy #1#2.{.\XINT_factortens_y#1.0.}%
243 \def\XINT_factortens_c #1#2#3#4#5#6#7#8#9%
244 {\if90\xint_dothis
245 {\expandafter\XINT_factortens_d\the\numexpr #8#7#6#5#4#3#2#1\relax
246 \xint_c_i 2345678.}\fi
247 \xint_orthat{.\XINT_factortens_y #1#2#3#4#5#6#7#8#9.}}%
248 \def\XINT_factortens_d #1#2#3#4#5#6#7#8#9%
249 {\if10\expandafter\XINT_factortens_e\fi
250 \XINT_factortens_f #9#9#8#7#6#5#4#3#2#1.}%
251 \def\XINT_factortens_f #1#2\xint_c_i#3.#4.#5.%
252 {\expandafter\XINT_factortens_g\the\numexpr#1+#5.#3.}%
253 \def\XINT_factortens_g #1.#2.{.\XINT_factortens_y#2.#1.}%
254 \def\XINT_factortens_e #1..#2.%
255 {\expandafter.\expandafter\XINT_factortens_c
256 \the\numexpr\xint_c_ix+#2.}%

```

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

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

```

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

```

```

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

```

```

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

```

8.10 \xintRaw

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

```

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

```

8.11 \xintiLogTen

New at 1.3e. The exponent a, such that $10^a \leq \text{abs}(x) < 10^{(a+1)}$.

```

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

This private macro was for internal usage by \xinttheexpr. It got moved here at 1.4 and is not used anymore by the package.

It checks if input has a [N] part, if yes uses \xintPRaw, else simply lets the input pass through as is.

```

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}}%

```

8.14 \xintFracToSci

1.4, refactored and much simplified at 1.4e.

It only needs to be x-expandable, and indeed the implementation here is only x-expandable.

(2021/04/13) the user documentation was really deplorable, I have tried to improve it and in the process tried to remember what this macro was supposed to do, and improved comments here, also lamentable.

At 1.4e-dev this became provisorily basically like defunct `\xintSPRaw`, but doing less parsing at it does not go to `\xintPRaw` with its `\XINT_infrac` induced overhead. Previous 1.4b `\xintFracToSci` was much complicated from having to allow fixed point notation on input and scientific notation with a catcode 12 "e". Refactoring of `\xintiexpr` has removed these constraints. Now:

Input: A, A/B, A[N], A/B[N]

Output: AeN/B with special cases:

0 if input gives a zero value

/B is skipped in output if B=1 in input

eN is skipped in output if N=0 in input

0[N] when N not zero is possible as input, but 0/B currently not I think, and -0 for example never arises as one is guaranteed that A is in strict integer format.

(2021/05/05) Finally for 1.4e release I modify. This is breaking change for all `\xinteval` output in case of scientific notation: it will not be with an integer mantissa, but in regular scientific notation, using the same rules as `\xintPFloat`.

Of course there will be no float rounding applied! Also, as [0] will always or almost always be present from an `\xinteval`, we want then to use integer not scientific notation. But expression contained decimal fixed point input, or uses scientific functions, then probably the N will not be zero and this will trigger usage of scientific notation in output.

Implementing these changes sort of ruin our previous efforts to minimize grabbing the argument, but well. So the rules now are

Input: A, A/B, A[N], A/B[N]

Output: A, A/B, A if N=0, A/B if N=0

If N is not zero, scientific notation like `\xintPFloat`, i.e. behaviour like `\xint-floateval` apart from the rounding to Digits. In particular trailing zeros are trimmed.

The zero gives 0, except in A[N] and A/B[N] cases, it may give 0.0

As a result of these last minute 1.4e changes, the `\xintFracToSciE` is removed.

```

434 \def\xintFracToSci #1{\expandafter\XINT_FracToSci\romannumeral`&&@#1/\W[\R}%
435 \def\XINT_FracToSci #1/#2#3[#4%
436 {%
437   \xint_gob_til_W #2\XINT_FracToSci_noslash\W
438   \xint_gob_til_R #4\XINT_FracToSci_slash_noN\R
439   \XINT_FracToSci_slash_N #1/#2#3[#4%
440 }%
441 \def\XINT_FracToSci_noslash#1\XINT_FracToSci_slash_N #2[#3%
442 {%
443   \xint_gob_til_R #3\XINT_FracToSci_noslash_noN\R
444   \XINT_FracToSci_noslash_N #2[#3%
445 }%
446 \def\XINT_FracToSci_noslash_noN\R\XINT_FracToSci_noslash_N #1/\W[\R{#1}%
447 \def\XINT_FracToSci_noslash_N #1[#2]/\W[\R%
448 {%
449   \ifnum#2=\xint_c_ #1\else
450     \romannumeral0\expandafter\XINT_pfloat_fork\romannumeral0\xintrez{#1[#2]]%
451   \fi
452 }%
453 \def\XINT_FracToSci_slash_noN\R\XINT_FracToSci_slash_N #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_slash_N #1#2/#3[#4]/\W[\R%
459 {%
460   \ifnum#4=\xint_c_ #1#2\else
461     \romannumeral0\expandafter\XINT_pfloat_fork\romannumeral0\xintrez{#1#2[#4]}%
462   \fi
463   \if\XINT_isOne{#3}1\else\if#10\else/#3\fi\fi
464 }%

```

8.15 \xintRawWithZeros

This was called \xintRaw in versions earlier than 1.07

```

465 \def\xintRawWithZeros {\romannumeral0\xintrawwithzeros }%
466 \def\xintrawwithzeros
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}%

```

8.16 \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\xintraw{#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 }%
```

8.17 \xintDecToStringREZ

1.4e. And I took this opportunity to improve documentation in manual.

```
501 \def\xintDecToStringREZ{\romannumeral0\xintdectosttringrez}%
502 \def\xintdectosttringrez#1{\expandafter\XINT_dectostr\romannumeral0\xintrez{#1}}%
```

8.18 \xintFloor, \xintiFloor

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

```
503 \def\xintFloor {\romannumeral0\xintfloor }%
504 \def\xintfloor #1% devrais-je faire \xintREZ?
505     {\expandafter\XINT_ifloor \romannumeral0\xintraawithzeros {#1}. /1[0]}%
506 \def\xintiFloor {\romannumeral0\xintifloor }%
507 \def\xintifloor #1%
508     {\expandafter\XINT_ifloor \romannumeral0\xintraawithzeros {#1}.}%
509 \def\XINT_ifloor #1/#2.{\xintiiquo {#1}{#2}}%
```

8.19 \xintCeil, \xintiCeil

1.09a

```
510 \def\xintCeil {\romannumeral0\xintceil }%
511 \def\xintceil #1{\xintiiopp {\xintFloor {\xintOpp{#1}}}}%
512 \def\xintiCeil {\romannumeral0\xinticeil }%
513 \def\xinticeil #1{\xintiiopp {\xintiFloor {\xintOpp{#1}}}}%
```

8.20 \xintNumerator

```
514 \def\xintNumerator {\romannumeral0\xintnumerator }%
515 \def\xintnumerator
516 {%
517     \expandafter\XINT_numer\romannumeral0\XINT_infrac
518 }%
519 \def\XINT_numer #1%
520 {%
521     \ifcase\XINT_cntSgn #1\xint:
522         \expandafter\XINT_numer_B
523     \or
524         \expandafter\XINT_numer_A
525     \else
526         \expandafter\XINT_numer_B
527     \fi
528     {#1}%
529 }%
530 \def\XINT_numer_A #1#2#3{\XINT_dsx_addzeros{#1}#2;%
531 \def\XINT_numer_B #1#2#3{ #2}%

```

8.21 \xintDenominator

```

532 \def\xintDenominator {\romannumeral0\xintdenominator }%
533 \def\xintdenominator
534 {%
535   \expandafter\XINT_denom_fork\romannumeral0\XINT_infrac
536 }%
537 \def\XINT_denom_fork #1%
538 {%
539   \ifnum#1<\xint_c_
540     \expandafter\XINT_denom_B
541   \else
542     \expandafter\XINT_denom_A
543   \fi
544   #1.%
545 }%
546 \def\XINT_denom_A #1.#2#3{ #3}%
547 \def\XINT_denom_B -#1.#2#3{\XINT_dsx_addzeros{#1}#3;}%

```

8.22 \xintTeXFrac

1.03 (2013/04/14). Useless typesetting macro.

Renamed (2021/05/24) from \xintFrac at 1.4g. Old name deprecated but still usable.

```

548 \ifdefined\documentclass
549 \def\xintfracTeXDeprecation#1#2{%
550 \PackageWarning{xintfrac}{\string#1 is deprecated. Use \string#2\MessageBreak
551   to suppress this warning}#2%
552 }%
553 \else
554 \edef\xintfracTeXDeprecation#1#2{{\newlinechar10
555 \immediate\noexpand\write128{&&JPackage xintfrac Warning: \noexpand\string#1 is
556 deprecated. Use \noexpand\string#2&&J%
557 (xintfrac)\xintReplicate{16}{ }to suppress this warning
558 on input line \noexpand\the\inputlineno.&&J}}#2%
559 }%
560 \fi
561 \def\xintFrac {\xintfracTeXDeprecation\xintFrac\xintTeXFrac}%
562 \def\xintTeXFrac{\romannumeral0\xintfrac }%
563 \def\xintfrac #1%
564 {%
565   \expandafter\XINT_fracfrac_A\romannumeral0\XINT_infrac {#1}%
566 }%
567 \def\XINT_fracfrac_A #1{\XINT_fracfrac_B #1\Z }%
568 \catcode\^=7
569 \def\XINT_fracfrac_B #1#2\Z
570 {%
571   \xint_gob_til_zero #1\XINT_fracfrac_C 0\XINT_fracfrac_D {10^{#1#2}}%
572 }%
573 \def\XINT_fracfrac_C 0\XINT_fracfrac_D #1#2#3%
574 {%
575   \if1\XINT_isOne {#3}%
576   \xint_afterfi {\expandafter\xint_stop_atfirstoftwo\xint_gobble_ii }%

```

```

577 \fi
578 \space
579 \frac {#2}{#3}%
580 }%
581 \def\XINT_fracfrac_D #1#2#3%
582 {%
583 \if1\XINT_isOne {#3}\XINT_fracfrac_E\fi
584 \space
585 \frac {#2}{#3}#1%
586 }%
587 \def\XINT_fracfrac_E \fi\space\frac #1#2{\fi \space #1\cdot }%

```

8.23 \xintTeXsignedFrac

Renamed (2021/05/24) from \xintSignedFrac at 1.4g. Old name deprecated but still usable.

```

588 \def\xintSignedFrac {\xintfracTeXDeprecation\xintSignedFrac\xintTeXsignedFrac}%
589 \def\xintTeXsignedFrac{\romannumeral0\xintsignedfrac }%
590 \def\xintsignedfrac #1%
591 {%
592 \expandafter\XINT_sgnfrac_a\romannumeral0\XINT_infrac {#1}%
593 }%
594 \def\XINT_sgnfrac_a #1#2%
595 {%
596 \XINT_sgnfrac_b #2\Z {#1}%
597 }%
598 \def\XINT_sgnfrac_b #1%
599 {%
600 \xint_UDsignfork
601 #1\XINT_sgnfrac_N
602 -{\XINT_sgnfrac_P #1}%
603 \krof
604 }%
605 \def\XINT_sgnfrac_P #1\Z #2%
606 {%
607 \XINT_fracfrac_A {#2}{#1}%
608 }%
609 \def\XINT_sgnfrac_N
610 {%
611 \expandafter-\romannumeral0\XINT_sgnfrac_P
612 }%

```

8.24 \xintTeXfromSci

1.4g. The main problem is how to name this and related macros.

I use \expanded here, as \xintFracToSci is not f-expandable. But why do I bother with the external \expanded?

Some complications as I want this to be usable on output of \xintFracToSci hence need to handle the case of a /B. After some hesitations I ended with the following which looks reasonable:

- if no scientific part, use \frac (or \over) for A/B
- if scientific part, postfix /B as \cdot B^{-1}

```

613 \def\xintTeXfromSci#1%
614 {%

```

```

615 \expanded{\expandafter\XINT_texpfromsci\expanded{#1}/\relax/\xint:}%
616 }%
617 \def\XINT_texpfromsci #1/#2#3/#4\xint:
618 {%
619 \XINT_texpfromsci_a #1e\relax e\xint:
620 {\ifx\relax#2\xint_dothis\xint_firstofone\fi
621 \xint_orthat{\XINT_texpfromsci_frac{#2#3}}}%
622 {\unless\ifx\relax#2\cdot{#2#3}^{-1}\fi}%
623 }%
624 \def\XINT_texpfromsci_a #1e#2#3e#4\xint:#5#6%
625 {%
626 \ifx\relax#2#5{#1}\else#1\cdot10^{#2#3}#6\fi
627 }%
628 \ifdefined\frac
629 \def\XINT_texpfromsci_frac#1#2{\noexpand\frac{#2}{#1}}%
630 \else
631 \def\XINT_texpfromsci_frac#1#2{{#2\over#1}}%
632 \fi

```

8.25 \xintTeXOver

Renamed (2021/05/24) from \xintFwOver at 1.4g. Old name deprecated but still usable.

```

633 \def\xintFwOver {\xintfracTeXDeprecation\xintFwOver\xintTeXOver}%
634 \def\xintTeXOver{\romannumeral0\xintfwover}%
635 \def\xintfwover #1%
636 {%
637 \expandafter\XINT_fwover_A\romannumeral0\XINT_infrac {#1}%
638 }%
639 \def\XINT_fwover_A #1{\XINT_fwover_B #1\Z}%
640 \def\XINT_fwover_B #1#2\Z
641 {%
642 \xint_gob_til_zero #1\XINT_fwover_C 0\XINT_fwover_D {10^{#1#2}}}%
643 }%
644 \catcode\^ =11
645 \def\XINT_fwover_C #1#2#3#4#5%
646 {%
647 \if0\XINT_isOne {#5}\xint_afterfi { {#4\over #5}}}%
648 \else\xint_afterfi { #4}%
649 \fi
650 }%
651 \def\XINT_fwover_D #1#2#3%
652 {%
653 \if0\XINT_isOne {#3}\xint_afterfi { {#2\over #3}}}%
654 \else\xint_afterfi { #2\cdot}%
655 \fi
656 #1%
657 }%

```

8.26 \xintTeXsignedOver

Renamed (2021/05/24) from \xintSignedFwOver at 1.4g. Old name deprecated but still usable.

```

658 \def\xintSignedFwOver {\xintfracTeXDeprecation\xintSignedFwOver\xintTeXsignedOver}%

```

```

659 \def\xintTeXsignedOver{\romannumeral0\xintsignedfwover }%
660 \def\xintsignedfwover #1%
661 {%
662   \expandafter\XINT_sgnfwover_a\romannumeral0\XINT_infrac {#1}%
663 }%
664 \def\XINT_sgnfwover_a #1#2%
665 {%
666   \XINT_sgnfwover_b #2\Z {#1}%
667 }%
668 \def\XINT_sgnfwover_b #1%
669 {%
670   \xint_UDsignfork
671     #1\XINT_sgnfwover_N
672     -{\XINT_sgnfwover_P #1}%
673   \krof
674 }%
675 \def\XINT_sgnfwover_P #1\Z #2%
676 {%
677   \XINT_fwover_A {#2}{#1}%
678 }%
679 \def\XINT_sgnfwover_N
680 {%
681   \expandafter-\romannumeral0\XINT_sgnfwover_P
682 }%

```

8.27 \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.

```

683 \def\xintREZ {\romannumeral0\xintrez }%
684 \def\xintrez
685 {%
686   \expandafter\XINT_rez_A\romannumeral0\XINT_infrac
687 }%
688 \def\XINT_rez_A #1#2%
689 {%
690   \XINT_rez_AB #2\Z {#1}%
691 }%
692 \def\XINT_rez_AB #1%
693 {%
694   \xint_UDzerominusfork
695     #1-\XINT_rez_zero
696     0#1\XINT_rez_neg
697     0-{\XINT_rez_B #1}%
698   \krof
699 }%
700 \def\XINT_rez_zero #1\Z #2#3{ 0/1[0]}%
701 \def\XINT_rez_neg {\expandafter-\romannumeral0\XINT_rez_B }%
702 \def\XINT_rez_B #1\Z
703 {%
704   \expandafter\XINT_rez_C\romannumeral0\XINT_factortens {#1}%

```

```

705 }%
706 \def\XINT_rez_C #1.#2.#3#4%
707 {%
708     \expandafter\XINT_rez_D\romannumeral0\XINT_factortens {#4}#3+#2.#1.%
709 }%
710 \def\XINT_rez_D #1.#2.#3.%
711 {%
712     \expandafter\XINT_rez_E\the\numexpr #3-#2.#1.%
713 }%
714 \def\XINT_rez_E #1.#2.#3.{ #3/#2[#1]}%

```

8.28 \xintE

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

```

715 \def\xintE {\romannumeral0\xinte }%
716 \def\xinte #1%
717 {%
718     \expandafter\XINT_e \romannumeral0\XINT_infrac {#1}%
719 }%
720 \def\XINT_e #1#2#3#4%
721 {%
722     \expandafter\XINT_e_end\the\numexpr #1+#4.{#2}{#3}%
723 }%
724 \def\XINT_e_end #1.#2#3{ #2/#3[#1]}%

```

8.29 \xintIrr, \xintPIrr

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

```

725 \def\xintIrr {\romannumeral0\xintirr }%
726 \def\xintPIrr{\romannumeral0\xintpirr }%
727 \def\xintirr #1%
728 {%
729     \expandafter\XINT_irr_start\romannumeral0\xintraewithzeros {#1}\Z
730 }%
731 \def\xintpirr #1%
732 {%
733     \expandafter\XINT_pirr_start\romannumeral0\xintra{#1}%
734 }%
735 \def\XINT_irr_start #1#2/#3\Z
736 {%
737     \if0\XINT_isOne {#3}%
738     \xint_afterfi
739     {\xint_UDsignfork
740         #1\XINT_irr_negative
741         -{\XINT_irr_nonneg #1}%
742     \krof}%
743 \else
744     \xint_afterfi{\XINT_irr_denomiseone #1}%
745 \fi
746 #2\Z {#3}%
747 }%

```

```

748 \def\XINT_pirr_start #1#2/#3[%
749 {%
750     \if0\XINT_isOne {#3}%
751     \xint_afterfi
752         {\xint_UDsignfork
753             #1\XINT_irr_negative
754             -{\XINT_irr_nonneg #1}%
755         \krof}%
756     \else
757     \xint_afterfi{\XINT_irr_denomisine #1}%
758     \fi
759     #2\Z {#3}[%
760 }%
761 \def\XINT_irr_denomisine #1\Z #2{ #1/1}% changed in 1.08
762 \def\XINT_irr_negative #1\Z #2{\XINT_irr_D #1\Z #2\Z -}%
763 \def\XINT_irr_nonneg #1\Z #2{\XINT_irr_D #1\Z #2\Z \space}%
764 \def\XINT_irr_D #1#2\Z #3#4\Z
765 {%
766     \xint_UDzerosfork
767         #3#1\XINT_irr_indeterminate
768         #30\XINT_irr_divisionbyzero
769         #10\XINT_irr_zero
770         00\XINT_irr_loop_a
771     \krof
772     {#3#4}{#1#2}{#3#4}{#1#2}%
773 }%
774 \def\XINT_irr_indeterminate #1#2#3#4#5%
775 {%
776     \XINT_signalcondition{DivisionUndefined}{0/0 indeterminate fraction.}{0/1}%
777 }%
778 \def\XINT_irr_divisionbyzero #1#2#3#4#5%
779 {%
780     \XINT_signalcondition{DivisionByZero}{Division by zero: #5#2/0.}{0/1}%
781 }%
782 \def\XINT_irr_zero #1#2#3#4#5{ 0/1}% changed in 1.08
783 \def\XINT_irr_loop_a #1#2%
784 {%
785     \expandafter\XINT_irr_loop_d
786     \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
787 }%
788 \def\XINT_irr_loop_d #1#2%
789 {%
790     \XINT_irr_loop_e #2\Z
791 }%
792 \def\XINT_irr_loop_e #1#2\Z
793 {%
794     \xint_gob_til_zero #1\XINT_irr_loop_exit0\XINT_irr_loop_a {#1#2}%
795 }%
796 \def\XINT_irr_loop_exit0\XINT_irr_loop_a #1#2#3#4%
797 {%
798     \expandafter\XINT_irr_loop_exitb\expandafter
799     {\romannumeral0\xintiiquo {#3}{#2}}%

```

```

800     {\romannumeral0\xintiipro {#4}{#2}}%
801 }%
802 \def\XINT_irr_loop_exitb #1#2%
803 {%
804     \expandafter\XINT_irr_finish\expandafter {#2}{#1}%
805 }%
806 \def\XINT_irr_finish #1#2#3{#3#1/#2}% changed in 1.08

```

8.30 \xintifInt

```

807 \def\xintifInt {\romannumeral0\xintifint }%
808 \def\xintifint #1{\expandafter\XINT_ifint\romannumeral0\xintraawwithzeros {#1}.}%
809 \def\XINT_ifint #1/#2.%
810 {%
811     \if 0\xintiiRem {#1}{#2}%
812     \expandafter\xint_stop_atfirstoftwo
813     \else
814     \expandafter\xint_stop_atsecondoftwo
815     \fi
816 }%

```

8.31 \xintIsInt

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

```

817 \def\xintIsInt {\romannumeral0\xintisint }%
818 \def\xintisint #1%
819     {\expandafter\XINT_ifint\romannumeral0\xintraawwithzeros {#1}.10}%

```

8.32 \xintJrr

```

820 \def\xintJrr {\romannumeral0\xintjrr }%
821 \def\xintjrr #1%
822 {%
823     \expandafter\XINT_jrr_start\romannumeral0\xintraawwithzeros {#1}\Z
824 }%
825 \def\XINT_jrr_start #1#2/#3\Z
826 {%
827     \if 0\XINT_isOne {#3}\xint_afterfi
828         {\xint_UDsignfork
829             #1\XINT_jrr_negative
830             -{\XINT_jrr_nonneg #1}%
831             \krof}%
832     \else
833         \xint_afterfi{\XINT_jrr_denomiseone #1}%
834     \fi
835     #2\Z {#3}%
836 }%
837 \def\XINT_jrr_denomiseone #1\Z #2{ #1/1}% changed in 1.08
838 \def\XINT_jrr_negative #1\Z #2{\XINT_jrr_D #1\Z #2\Z -}%
839 \def\XINT_jrr_nonneg #1\Z #2{\XINT_jrr_D #1\Z #2\Z \space}%
840 \def\XINT_jrr_D #1#2\Z #3#4\Z
841 {%

```



```

842 \xint_UDzerosfork
843   #3#1\XINT_jrr_indeterminate
844   #30\XINT_jrr_divisionbyzero
845   #10\XINT_jrr_zero
846   00\XINT_jrr_loop_a
847 \krof
848 {#3#4}{#1#2}1001%
849 }%
850 \def\XINT_jrr_indeterminate #1#2#3#4#5#6#7%
851 {%
852   \XINT_signalcondition{DivisionUndefined}{0/0 indeterminate fraction.}{#{0/1}%
853 }%
854 \def\XINT_jrr_divisionbyzero #1#2#3#4#5#6#7%
855 {%
856   \XINT_signalcondition{DivisionByZero}{Division by zero: #7#2/0.}{#{0/1}%
857 }%
858 \def\XINT_jrr_zero #1#2#3#4#5#6#7{ 0/1}% changed in 1.08
859 \def\XINT_jrr_loop_a #1#2%
860 {%
861   \expandafter\XINT_jrr_loop_b
862   \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
863 }%
864 \def\XINT_jrr_loop_b #1#2#3#4#5#6#7%
865 {%
866   \expandafter \XINT_jrr_loop_c \expandafter
867     {\romannumeral0\xintiadd{\XINT_mul_fork #4\xint:#1\xint:}{#6}}%
868     {\romannumeral0\xintiadd{\XINT_mul_fork #5\xint:#1\xint:}{#7}}%
869     {#2}{#3}{#4}{#5}%
870 }%
871 \def\XINT_jrr_loop_c #1#2%
872 {%
873   \expandafter \XINT_jrr_loop_d \expandafter{#2}{#1}%
874 }%
875 \def\XINT_jrr_loop_d #1#2#3#4%
876 {%
877   \XINT_jrr_loop_e #3\Z {#4}{#2}{#1}%
878 }%
879 \def\XINT_jrr_loop_e #1#2\Z
880 {%
881   \xint_gob_til_zero #1\XINT_jrr_loop_exit0\XINT_jrr_loop_a {#1#2}%
882 }%
883 \def\XINT_jrr_loop_exit0\XINT_jrr_loop_a #1#2#3#4#5#6%
884 {%
885   \XINT_irr_finish {#3}{#4}%
886 }%

```

8.33 \xintTFrac

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

zero) if $x < 0$, or rather always positive. There should be in fact such a thing for each rounding function, trunc, round, floor, ceil.

```

887 \def\xintTFrac {\romannumeral0\xinttfrac }%
888 \def\xinttfrac #1{\expandafter\XINT_tfrac_fork\romannumeral0\xintraawithzeros {#1}\Z }%
889 \def\XINT_tfrac_fork #1%
890 {%
891     \xint_UDzerominusfork
892     #1-\XINT_tfrac_zero
893     0#1{\xintiopp\XINT_tfrac_P }%
894     0-{\XINT_tfrac_P #1}%
895     \krof
896 }%
897 \def\XINT_tfrac_zero #1\Z { 0/1[0]}%
898 \def\XINT_tfrac_P #1/#2\Z {\expandafter\XINT_rez_AB
899     \romannumeral0\xintiirem{#1}{#2}\Z {0}{#2}}%

```

8.34 \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 `\xintexpr`) 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.

```

900 \def\xintTrunc {\romannumeral0\xinttrunc }%
901 \def\xintiTrunc {\romannumeral0\xintitrunc}%
902 \def\xinttrunc #1{\expandafter\XINT_trunc\the\numexpr#1.\XINT_trunc_G}%
903 \def\xintitrunc #1{\expandafter\XINT_trunc\the\numexpr#1.\XINT_itrunc_G}%
904 \def\XINT_trunc #1.#2#3%
905 {%
906     \expandafter\XINT_trunc_a\romannumeral0\XINT_infrac{#3}{#1.#2}%
907 }%
908 \def\XINT_trunc_a #1#2#3#4.#5%
909 {%

```

```

910 \if0\XINT_Sgn#2\xint:\xint_dothis\XINT_trunc_zero\fi
911 \if1\XINT_is_One#3XY\xint_dothis\XINT_trunc_sp_b\fi
912 \xint_orthat\XINT_trunc_b #1+#4.{#2}{#3}#5#4.%
913 }%
914 \def\XINT_trunc_zero #1.#2.{ 0}%
915 \def\XINT_trunc_b {\expandafter\XINT_trunc_B\the\numexpr}%
916 \def\XINT_trunc_sp_b {\expandafter\XINT_trunc_sp_B\the\numexpr}%
917 \def\XINT_trunc_B #1%
918 {%
919 \xint_UDsignfork
920 #1\XINT_trunc_C
921 -\XINT_trunc_D
922 \krof #1%
923 }%
924 \def\XINT_trunc_sp_B #1%
925 {%
926 \xint_UDsignfork
927 #1\XINT_trunc_sp_C
928 -\XINT_trunc_sp_D
929 \krof #1%
930 }%
931 \def\XINT_trunc_C -#1.#2#3%
932 {%
933 \expandafter\XINT_trunc_CE
934 \romannumeral0\XINT_dsx_addzeros{#1}#3;.{#2}%
935 }%
936 \def\XINT_trunc_CE #1.#2{\XINT_trunc_E #2.{#1}}%

937 \def\XINT_trunc_sp_C -#1.#2#3{\XINT_trunc_sp_Ca #2.#1.}%
938 \def\XINT_trunc_sp_Ca #1%
939 {%
940 \xint_UDsignfork
941 #1{\XINT_trunc_sp_Cb -}%
942 -{\XINT_trunc_sp_Cb \space#1}%
943 \krof
944 }%
945 \def\XINT_trunc_sp_Cb #1#2.#3.%
946 {%
947 \expandafter\XINT_trunc_sp_Cc
948 \romannumeral0\expandafter\XINT_split_fromright_a
949 \the\numexpr#3-\numexpr\XINT_length_loop
950 #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
951 \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
952 \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
953 .#2\xint_bye2345678\xint_bye..#1%
954 }%
955 \def\XINT_trunc_sp_Cc #1%
956 {%
957 \if.#1\xint_dothis{\XINT_trunc_sp_Cd 0.}\fi
958 \xint_orthat {\XINT_trunc_sp_Cd #1}%
959 }%
960 \def\XINT_trunc_sp_Cd #1.#2.#3%
961 {%

```

```

962 \XINT_trunc_sp_F #3#1.%
963 }%
964 \def\XINT_trunc_D #1.#2%
965 {%
966 \expandafter\XINT_trunc_E
967 \romannumeral0\XINT_dsx_addzeros {#1}#2;.%
968 }%
969 \def\XINT_trunc_sp_D #1.#2#3%
970 {%
971 \expandafter\XINT_trunc_sp_E
972 \romannumeral0\XINT_dsx_addzeros {#1}#2;.%
973 }%
974 \def\XINT_trunc_E #1%
975 {%
976 \xint_UDsignfork
977 #1{\XINT_trunc_F -}%
978 -{\XINT_trunc_F \space#1}%
979 \krof
980 }%
981 \def\XINT_trunc_sp_E #1%
982 {%
983 \xint_UDsignfork
984 #1{\XINT_trunc_sp_F -}%
985 -{\XINT_trunc_sp_F\space#1}%
986 \krof
987 }%
988 \def\XINT_trunc_F #1#2.#3#4%
989 {\expandafter#4\romannumeral`&&\expandafter\xint_firstoftwo
990 \romannumeral0\XINT_div_prepare {#3}{#2}.#1}%
991 \def\XINT_trunc_sp_F #1#2.#3{#3#2.#1}%
992 \def\XINT_itrunc_G #1#2.#3#4.%
993 {%
994 \if#10\xint_dothis{ 0}\fi
995 \xint_orthat{#3#1}#2%
996 }%
997 \def\XINT_trunc_G #1.#2#3#4.%
998 {%
999 \xint_gob_til_minus#3\XINT_trunc_Hc-%
1000 \expandafter\XINT_trunc_H
1001 \the\numexpr\romannumeral0\xintlength {#1}-#3#4.#3#4.{#1}#2%
1002 }%
1003 \def\XINT_trunc_Hc-\expandafter\XINT_trunc_H
1004 \the\numexpr\romannumeral0\xintlength #1.-#2.#3#4{#4#3e#2}%
1005 \def\XINT_trunc_H #1.#2.%
1006 {%
1007 \ifnum #1 > \xint_c \xint_dothis{\XINT_trunc_Ha {#2}}\fi
1008 \xint_orthat {\XINT_trunc_Hb {-#1}}% -0,--1,--2, ....
1009 }%
1010 \def\XINT_trunc_Ha%
1011 {%
1012 \expandafter\XINT_trunc_Haa\romannumeral0\xintdecsplit
1013 }%

```

```

1014 \def\XINT_trunc_Haa #1#2#3{#3#1.#2}%
1015 \def\XINT_trunc_Hb #1#2#3%
1016 {%
1017     \expandafter #3\expandafter0\expandafter.%
1018     \romannumeral\xintreplicate{#1}0#2%
1019 }%

```

8.35 \xintTTrunc

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

```

1020 \def\xintTTrunc {\romannumeral0\xintttrunc }%
1021 \def\xintttrunc {\xintitrunc\xint_c_}%

```

8.36 \xintNum

```

1022 \let\xintnum \xintttrunc

```

8.37 \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.

```

1023 \def\xintRound {\romannumeral0\xintround }%
1024 \def\xintiRound {\romannumeral0\xintiround }%
1025 \def\xintround #1{\expandafter\XINT_round\the\numexpr #1.\XINT_round_A}%
1026 \def\xintiround #1{\expandafter\XINT_round\the\numexpr #1.\XINT_iround_A}%
1027 \def\XINT_round #1.{\expandafter\XINT_round_aa\the\numexpr #1+\xint_c_i.#1.}%
1028 \def\XINT_round_aa #1.#2.#3#4%
1029 {%
1030     \expandafter\XINT_round_a\romannumeral0\XINT_infrac{#4}#1.#3#2.%
1031 }%
1032 \def\XINT_round_a #1#2#3#4.%
1033 {%
1034     \if0\XINT_Sgn#2\xint:\xint_dothis\XINT_trunc_zero\fi
1035     \if1\XINT_is_One#3XY\xint_dothis\XINT_trunc_sp_b\fi
1036     \xint_orthat\XINT_trunc_b #1+#4.{#2}{#3}%
1037 }%
1038 \def\XINT_round_A{\expandafter\XINT_trunc_G\romannumeral0\XINT_round_B}%
1039 \def\XINT_iround_A{\expandafter\XINT_itrunc_G\romannumeral0\XINT_round_B}%
1040 \def\XINT_round_B #1.%
1041     {\XINT_dsrr #1\xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax.}%

```

8.38 \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.

```

1042 \def\xintXTrunc #1#2%
1043 {%
1044     \expandafter\XINT_xtrunc_a
1045     \the\numexpr #1\expandafter.\romannumeral0\xintra
1046 }%
1047 \def\XINT_xtrunc_a #1.% ?? faire autre chose
1048 {%
1049     \expandafter\XINT_xtrunc_b\the\numexpr\ifnum#1<\xint_c_i \xint_c_i-\fi #1.%
1050 }%

1051 \def\XINT_xtrunc_b #1.#2{\XINT_xtrunc_c #2{#1}}%

1052 \def\XINT_xtrunc_c #1%
1053 {%
1054     \xint_UDzerominusfork
1055     #1-\XINT_xtrunc_zero
1056     0#1{-\XINT_xtrunc_d {}}%
1057     0-{\XINT_xtrunc_d #1}%
1058     \krof
1059 }%[
1060 \def\XINT_xtrunc_zero #1#2]{0.\romannumeral\xintreplicate{#1}0}%

1061 \def\XINT_xtrunc_d #1#2#3/#4[#5]%
1062 {%
1063     \XINT_xtrunc_prepare_a#4\R\R\R\R\R\R\R {10}0000001\W
1064     !{#4};{#5}{#2}{#1#3}%
1065 }%
1066 \def\XINT_xtrunc_prepare_a #1#2#3#4#5#6#7#8#9%
1067 {%
1068     \xint_gob_til_R #9\XINT_xtrunc_prepare_small\R
1069     \XINT_xtrunc_prepare_b #9%
1070 }%
1071 \def\XINT_xtrunc_prepare_small\R #1!#2;%
1072 {%
1073     \ifcase #2
1074     \or\expandafter\XINT_xtrunc_BisOne
1075     \or\expandafter\XINT_xtrunc_BisTwo
1076     \or
1077     \or\expandafter\XINT_xtrunc_BisFour
1078     \or\expandafter\XINT_xtrunc_BisFive
1079     \or
1080     \or
1081     \or\expandafter\XINT_xtrunc_BisEight
1082     \fi\XINT_xtrunc_BisSmall {#2}%
1083 }%

1084 \def\XINT_xtrunc_BisOne\XINT_xtrunc_BisSmall #1#2#3#4%
1085     {\XINT_xtrunc_sp_e {#2}{#4}{#3}}%
1086 \def\XINT_xtrunc_BisTwo\XINT_xtrunc_BisSmall #1#2#3#4%
1087 {%
1088     \expandafter\XINT_xtrunc_sp_e\expandafter

```

```

1089     {\the\numexpr #2-\xint_c_i\expandafter}\expandafter
1090     {\romannumeral0\xintiimul 5{#4}}{#3}%
1091 }%
1092 \def\xINT_xtrunc_BisFour\xINT_xtrunc_BisSmall #1#2#3#4%
1093 {%
1094     \expandafter\xINT_xtrunc_sp_e\expandafter
1095     {\the\numexpr #2-\xint_c_ii\expandafter}\expandafter
1096     {\romannumeral0\xintiimul {25}{#4}}{#3}%
1097 }%
1098 \def\xINT_xtrunc_BisFive\xINT_xtrunc_BisSmall #1#2#3#4%
1099 {%
1100     \expandafter\xINT_xtrunc_sp_e\expandafter
1101     {\the\numexpr #2-\xint_c_i\expandafter}\expandafter
1102     {\romannumeral0\xintdouble {#4}}{#3}%
1103 }%
1104 \def\xINT_xtrunc_BisEight\xINT_xtrunc_BisSmall #1#2#3#4%
1105 {%
1106     \expandafter\xINT_xtrunc_sp_e\expandafter
1107     {\the\numexpr #2-\xint_c_iii\expandafter}\expandafter
1108     {\romannumeral0\xintiimul {125}{#4}}{#3}%
1109 }%
1110 \def\xINT_xtrunc_BisSmall #1%
1111 {%
1112     \expandafter\xINT_xtrunc_e\expandafter
1113     {\expandafter\xINT_xtrunc_small_a
1114     \the\numexpr #1/\xint_c_ii\expandafter
1115     .\the\numexpr \xint_c_x^viii+#1!}%
1116 }%

1117 \def\xINT_xtrunc_small_a #1.#2!#3%
1118 {%
1119     \expandafter\xINT_div_small_b\the\numexpr #1\expandafter
1120     \xint:\the\numexpr #2\expandafter!%
1121     \romannumeral0\xINT_div_small_ba #3\R\R\R\R\R\R\R\{10}0000001\W
1122     #3\xINT_sepbyviii_Z_end 2345678\relax
1123 }%

1124 \def\xINT_xtrunc_prepare_b
1125     {\expandafter\xINT_xtrunc_prepare_c\romannumeral0\xINT_zeroes_forviii }%
1126 \def\xINT_xtrunc_prepare_c #1!%
1127 {%
1128     \xINT_xtrunc_prepare_d #1.00000000!{#1}%
1129 }%
1130 \def\xINT_xtrunc_prepare_d #1#2#3#4#5#6#7#8#9%
1131 {%
1132     \expandafter\xINT_xtrunc_prepare_e
1133     \xint_gob_til_dot #1#2#3#4#5#6#7#8#9!%
1134 }%
1135 \def\xINT_xtrunc_prepare_e #1!#2!#3#4%
1136 {%
1137     \xINT_xtrunc_prepare_f #4#3\X {#1}{#3}%
1138 }%
1139 \def\xINT_xtrunc_prepare_f #1#2#3#4#5#6#7#8#9\X

```

```

1140 {%
1141   \expandafter\XINT_xtrunc_prepare_g\expandafter
1142   \XINT_div_prepare_g
1143   \the\numexpr #1#2#3#4#5#6#7#8+\xint_c_i\expandafter
1144   \xint:\the\numexpr (#1#2#3#4#5#6#7#8+\xint_c_i)/\xint_c_ii\expandafter
1145   \xint:\the\numexpr #1#2#3#4#5#6#7#8\expandafter
1146   \xint:\romannumeral0\XINT_sepandrev_andcount
1147   #1#2#3#4#5#6#7#8#9\XINT_rsepbyviii_end_A 2345678%
1148   \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
1149   \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
1150   \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_W
1151   \X
1152 }%

1153 \def\XINT_xtrunc_prepare_g #1;{\XINT_xtrunc_e {#1}}%

1154 \def\XINT_xtrunc_e #1#2%
1155 {%
1156   \ifnum #2<\xint_c_
1157     \expandafter\XINT_xtrunc_I
1158   \else
1159     \expandafter\XINT_xtrunc_II
1160   \fi #2\xint:{#1}%
1161 }%

1162 \def\XINT_xtrunc_I -#1\xint:#2#3#4%
1163 {%
1164   \expandafter\XINT_xtrunc_I_a\romannumeral0#2{#4}{#2}{#1}{#3}%
1165 }%

1166 \def\XINT_xtrunc_I_a #1#2#3#4#5%
1167 {%
1168   \expandafter\XINT_xtrunc_I_b\the\numexpr #4-#5\xint:#4\xint:{#5}{#2}{#3}{#1}%
1169 }%

1170 \def\XINT_xtrunc_I_b #1%
1171 {%
1172   \xint_UDsignfork
1173   #1\XINT_xtrunc_IA_c
1174   -\XINT_xtrunc_IB_c
1175   \krof #1%
1176 }%

1177 \def\XINT_xtrunc_IA_c -#1\xint:#2\xint:#3#4#5#6%
1178 {%
1179   \expandafter\XINT_xtrunc_IA_d
1180   \the\numexpr#2-\xintLength{#6}\xint:{#6}%
1181   \expandafter\XINT_xtrunc_IA_xd
1182   \the\numexpr (#1+\xint_c_ii^v)/\xint_c_ii^vi-\xint_c_i\xint:#1\xint:{#5}{#4}%
1183 }%

1184 \def\XINT_xtrunc_IA_d #1%
1185 {%
1186   \xint_UDsignfork
1187   #1\XINT_xtrunc_IAA_e

```



```

1188     -\XINT_xtrunc_IAB_e
1189     \krof #1%
1190 }%

1191 \def\XINT_xtrunc_IAB_e -#1\xint:#2%
1192 {%
1193     \romannumeral0\XINT_split_fromleft
1194     #1.#2\xint_gobble_i\xint_bye2345678\xint_bye..%
1195 }%

1196 \def\XINT_xtrunc_IAB_e #1\xint:#2%
1197 {%
1198     0.\romannumeral\XINT_rep#1\endcsname0#2%
1199 }%

1200 \def\XINT_xtrunc_IA_xd #1\xint:#2\xint:%
1201 {%
1202     \expandafter\XINT_xtrunc_IA_xe\the\numexpr #2-\xint_c_ii^vi*#1\xint:#1\xint:%
1203 }%

1204 \def\XINT_xtrunc_IA_xe #1\xint:#2\xint:#3#4%
1205 {%
1206     \XINT_xtrunc_loop {#2}{#4}{#3}{#1}%
1207 }%

1208 \def\XINT_xtrunc_IB_c #1\xint:#2\xint:#3#4#5#6%
1209 {%
1210     \expandafter\XINT_xtrunc_IB_d
1211     \romannumeral0\XINT_split_xfork #1.#6\xint_bye2345678\xint_bye..{#3}%
1212 }%

1213 \def\XINT_xtrunc_IB_d #1.#2.#3%
1214 {%
1215     \expandafter\XINT_xtrunc_IA_d\the\numexpr#3-\xintLength {#1}\xint:{#1}%
1216 }%

1217 \def\XINT_xtrunc_II #1\xint:%
1218 {%
1219     \expandafter\XINT_xtrunc_II_a\romannumeral\xintreplicate{#1}0\xint:%
1220 }%
1221 \def\XINT_xtrunc_II_a #1\xint:#2#3#4%
1222 {%
1223     \expandafter\XINT_xtrunc_II_b
1224     \the\numexpr (#3+\xint_c_ii^v)/\xint_c_ii^vi-\xint_c_i\expandafter\xint:%
1225     \the\numexpr #3\expandafter\xint:\romannumeral0#2{#4#1}{#2}%
1226 }%
1227 \def\XINT_xtrunc_II_b #1\xint:#2\xint:%
1228 {%
1229     \expandafter\XINT_xtrunc_II_c\the\numexpr #2-\xint_c_ii^vi*#1\xint:#1\xint:%
1230 }%

1231 \def\XINT_xtrunc_II_c #1\xint:#2\xint:#3#4#5%
1232 {%
1233     #3.\XINT_xtrunc_loop {#2}{#4}{#5}{#1}%
1234 }%

```

[illegible]

```

1285 \def\XINT_xtrunc_sp_IA_b -#1\xint:#2\xint:#3#4%
1286 {%
1287   \expandafter\XINT_xtrunc_sp_IA_c
1288   \the\numexpr#2-\xintlength{#4}\xint:{#4}\romannumeral\XINT_rep#1\endcsname0%
1289 }%

1290 \def\XINT_xtrunc_sp_IA_c #1%
1291 {%
1292   \xint_UDsignfork
1293   #1\XINT_xtrunc_sp_IAA
1294   -\XINT_xtrunc_sp_IAB
1295   \krof #1%
1296 }%

1297 \def\XINT_xtrunc_sp_IAA -#1\xint:#2%
1298 {%
1299   \romannumeral0\XINT_split_fromleft
1300   #1.#2\xint_gobble_i\xint_bye2345678\xint_bye. .%
1301 }%

1302 \def\XINT_xtrunc_sp_IAB #1\xint:#2%
1303 {%
1304   0.\romannumeral\XINT_rep#1\endcsname0#2%
1305 }%

1306 \def\XINT_xtrunc_sp_IB_b #1\xint:#2\xint:#3#4%
1307 {%
1308   \expandafter\XINT_xtrunc_sp_IB_c
1309   \romannumeral0\XINT_split_xfork #1.#4\xint_bye2345678\xint_bye. .{#3}%
1310 }%

1311 \def\XINT_xtrunc_sp_IB_c #1.#2.#3%
1312 {%
1313   \expandafter\XINT_xtrunc_sp_IA_c\the\numexpr#3-\xintlength {#1}\xint:{#1}%
1314 }%

1315 \def\XINT_xtrunc_sp_II #1\xint:#2#3%
1316 {%
1317   #2\romannumeral\XINT_rep#1\endcsname0.\romannumeral\XINT_rep#3\endcsname0%
1318 }%

```

8.39 \xintAdd

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

```

1319 \def\xintAdd {\romannumeral0\xintadd}%
1320 \def\xintadd #1{\expandafter\XINT_fadd\romannumeral0\xintraw {#1}}%
1321 \def\XINT_fadd #1{\xint_gob_til_zero #1\XINT_fadd_Azero 0\XINT_fadd_a #1}%
1322 \def\XINT_fadd_Azero #1{\xintraw}%
1323 \def\XINT_fadd_a #1/#2[#3]#4%
1324   {\expandafter\XINT_fadd_b\romannumeral0\xintraw {#4}{#3}{#1}{#2}}%
1325 \def\XINT_fadd_b #1{\xint_gob_til_zero #1\XINT_fadd_Bzero 0\XINT_fadd_c #1}%
1326 \def\XINT_fadd_Bzero #1[#2]#3#4{ #3/#4[#2]}%
1327 \def\XINT_fadd_c #1/#2[#3]#4%
1328 {%

```

```

1329 \expandafter\XINT_fadd_Aa\the\numexpr #4-#3.{#3}{#4}{#1}{#2}%
1330 }%
1331 \def\XINT_fadd_Aa #1%
1332 {%
1333 \xint_UDzerominusfork
1334 #1-\XINT_fadd_B
1335 0#1\XINT_fadd_Bb
1336 0-\XINT_fadd_Ba
1337 \krof #1%
1338 }%
1339 \def\XINT_fadd_B #1.#2#3#4#5#6#7{\XINT_fadd_C {#4}{#5}{#7}{#6}[#3]}%
1340 \def\XINT_fadd_Ba #1.#2#3#4#5#6#7%
1341 {%
1342 \expandafter\XINT_fadd_C\expandafter
1343 {\romannumeral0\XINT_dsx_addzeros {#1}#6;}%
1344 {#7}{#5}{#4}[#2]}%
1345 }%
1346 \def\XINT_fadd_Bb -#1.#2#3#4#5#6#7%
1347 {%
1348 \expandafter\XINT_fadd_C\expandafter
1349 {\romannumeral0\XINT_dsx_addzeros {#1}#4;}%
1350 {#5}{#7}{#6}[#3]}%
1351 }%
1352 \def\XINT_fadd_iszero #1[#2]{ 0/1[0]}% ou [#2] origine1?
1353 \def\XINT_fadd_C #1#2#3%
1354 {%
1355 \expandafter\XINT_fadd_D_b
1356 \romannumeral0\XINT_div_prepare{#2}{#3}{#2}{#2}{#3}{#1}%
1357 }%

```

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.

```

1358 \def\XINT_fadd_D_a #1#2%
1359 {%
1360 \expandafter\XINT_fadd_D_b
1361 \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
1362 }%
1363 \def\XINT_fadd_D_b #1#2{\XINT_fadd_D_c #2\Z}%
1364 \def\XINT_fadd_D_c #1#2\Z
1365 {%
1366 \xint_gob_til_zero #1\XINT_fadd_D_exit0\XINT_fadd_D_a {#1#2}%
1367 }%
1368 \def\XINT_fadd_D_exit0\XINT_fadd_D_a #1#2#3%
1369 {%
1370 \expandafter\XINT_fadd_E
1371 \romannumeral0\xintiiquo {#3}{#2}.{#2}%
1372 }%
1373 \def\XINT_fadd_E #1.#2#3%
1374 {%
1375 \expandafter\XINT_fadd_F
1376 \romannumeral0\xintiimul{#1}{#3}.{\xintiiquo{#3}{#2}}{#1}%
1377 }%
1378 \def\XINT_fadd_F #1.#2#3#4#5%

```

```

1379 {%
1380     \expandafter\XINT_fadd_G
1381     \romannumeral0\xintiiadd{\xintiiMul{#2}{#4}}{\xintiiMul{#3}{#5}}/#1%
1382 }%
1383 \def\XINT_fadd_G #1{%
1384 \def\XINT_fadd_G ##1{\if0##1\expandafter\XINT_fadd_iszero\fi#1##1}%
1385 }\XINT_fadd_G{ }%

```

8.40 \xintSub

Since 1.3 will use least common multiple of denominators.

```

1386 \def\xintSub {\romannumeral0\xintsub }%
1387 \def\xintsub #1{\expandafter\XINT_fsub\romannumeral0\xintra{#1}}%
1388 \def\XINT_fsub #1{\xint_gob_til_zero #1\XINT_fsub_Azero 0\XINT_fsub_a #1}%
1389 \def\XINT_fsub_Azero #1]{\xintopp }%
1390 \def\XINT_fsub_a #1/#2[#3]#4%
1391     {\expandafter\XINT_fsub_b\romannumeral0\xintra{#4}{#3}{#1}{#2}}%
1392 \def\XINT_fsub_b #1{\xint_UDzerominusfork
1393     #1-\XINT_fadd_Bzero
1394     0#1\XINT_fadd_c
1395     0-\XINT_fadd_c -#1}%
1396 \kro{ }%

```

8.41 \xintSum

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

Empty items 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.

```

1397 \def\xintSum {\romannumeral0\xintsum }%
1398 \def\xintsum #1{\expandafter\XINT_sum\romannumeral`&&@#1^}%
1399 \def\XINT_Sum{\romannumeral0\XINT_sum}%
1400 \def\XINT_sum#1%
1401 {%
1402     \xint_gob_til_^ #1\XINT_sum_empty ^%
1403     \expandafter\XINT_sum_loop\romannumeral0\xintra{#1}\xint:
1404 }%
1405 \def\XINT_sum_empty ^#1\xint:{ 0/1[0]}%
1406 \def\XINT_sum_loop #1\xint:#2%
1407 {%
1408     \xint_gob_til_^ #2\XINT_sum_end ^%
1409     \expandafter\XINT_sum_loop
1410     \romannumeral0\xintadd{#1}{\romannumeral0\xintra{#2}}\xint:
1411 }%
1412 \def\XINT_sum_end ^#1\xintadd #2#3\xint:{ #2}%

```

8.42 \xintMul

```

1413 \def\xintMul {\romannumeral0\xintmul }%
1414 \def\xintmul #1{\expandafter\XINT_fm{#1}}%
1415 \def\XINT_fm #1{\xint_gob_til_zero #1\XINT_fm{#1}}%
1416 \def\XINT_fm{#1}#2.#3%

```

```

1417 {\expandafter\XINT_fmulo\romannumeral0\xintra {#3}#1[#2.]}%
1418 \def\XINT_fmulo #1{\xint_gob_til_zero #1\XINT_fmulo_zero 0\XINT_fmulo_c #1}%
1419 \def\XINT_fmulo_c #1/#2[#3]#4/#5[#6.]}%
1420 {%
1421   \expandafter\XINT_fmulo_d
1422   \expandafter{\the\numexpr #3+#6\expandafter}%
1423   \expandafter{\romannumeral0\xintiimulo {#5}{#2}}%
1424   {\romannumeral0\xintiimulo {#4}{#1}}%
1425 }%
1426 \def\XINT_fmulo_d #1#2#3%
1427 {%
1428   \expandafter \XINT_fmulo_e \expandafter{#3}{#1}{#2}%
1429 }%
1430 \def\XINT_fmulo_e #1#2{\XINT_outfrac {#2}{#1}}%
1431 \def\XINT_fmulo_zero #1.#2{ 0/1[0]}%

```

8.43 \xintSqr

1.1 modifs comme xintMul.

```

1432 \def\xintSqr {\romannumeral0\xintsqr }%
1433 \def\xintsqr #1{\expandafter\XINT_fsqr\romannumeral0\xintra {#1}}%
1434 \def\XINT_fsqr #1{\xint_gob_til_zero #1\XINT_fsqr_zero 0\XINT_fsqr_a #1}%
1435 \def\XINT_fsqr_a #1/#2[#3]}%
1436 {%
1437   \expandafter\XINT_fsqr_b
1438   \expandafter{\the\numexpr #3+#3\expandafter}%
1439   \expandafter{\romannumeral0\xintiisqr {#2}}%
1440   {\romannumeral0\xintiisqr {#1}}%
1441 }%
1442 \def\XINT_fsqr_b #1#2#3{\expandafter \XINT_fmulo_e \expandafter{#3}{#1}{#2}}%
1443 \def\XINT_fsqr_zero #1{ 0/1[0]}%

```

8.44 \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.

```

1444 \def\xintipow #1#2%
1445 {%
1446   \expandafter\xint_pow\the\numexpr \xintNum{#2}\expandafter
1447   .\romannumeral0\xintnum{#1}\xint:
1448 }%
1449 \def\xintPow {\romannumeral0\xintpow }%
1450 \def\xintpow #1%
1451 {%
1452   \expandafter\XINT_fpow\expandafter {\romannumeral0\XINT_infrac {#1}}%
1453 }%
1454 \def\XINT_fpow #1#2%
1455 {%
1456   \expandafter\XINT_fpow_fork\the\numexpr \xintNum{#2}\relax\Z #1%
1457 }%

```

```

1458 \def\XINT_fpow_fork #1#2\Z
1459 {%
1460     \xint_UDzerominusfork
1461     #1-\XINT_fpow_zero
1462     0#1\XINT_fpow_neg
1463     0-\XINT_fpow_pos #1}%
1464     \krof
1465     {#2}%
1466 }%
1467 \def\XINT_fpow_zero #1#2#3#4{ 1/1[0]}%
1468 \def\XINT_fpow_pos #1#2#3#4#5%
1469 {%
1470     \expandafter\XINT_fpow_pos_A\expandafter
1471     {\the\numexpr #1#2*#3\expandafter}\expandafter
1472     {\romannumeral0\xintiipow {#5}{#1#2}}%
1473     {\romannumeral0\xintiipow {#4}{#1#2}}%
1474 }%
1475 \def\XINT_fpow_neg #1#2#3#4%
1476 {%
1477     \expandafter\XINT_fpow_pos_A\expandafter
1478     {\the\numexpr -#1*#2\expandafter}\expandafter
1479     {\romannumeral0\xintiipow {#3}{#1}}%
1480     {\romannumeral0\xintiipow {#4}{#1}}%
1481 }%
1482 \def\XINT_fpow_pos_A #1#2#3%
1483 {%
1484     \expandafter\XINT_fpow_pos_B\expandafter {#3}{#1}{#2}%
1485 }%
1486 \def\XINT_fpow_pos_B #1#2{\XINT_outfrac {#2}{#1}}%

```

8.45 \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.

```

1487 \def\xintFac {\romannumeral0\xintfac}%
1488 \def\xintfac #1{\expandafter\XINT_fac_fork\the\numexpr\xintNum{#1}.[0]}%

```

8.46 \xintBinomial

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

```

1489 \def\xintBinomial {\romannumeral0\xintbinomial}%
1490 \def\xintbinomial #1#2%
1491 {%
1492     \expandafter\XINT_binom_pre
1493     \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.[0]%
1494 }%

```

8.47 \xintPFactorial

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

```

1495 \def\xintipfactorial #1#2%
1496 {%
1497     \expandafter\XINT_pfac_fork
1498     \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.%
1499 }%
1500 \def\xintPFactorial {\romannumeral0\xintpfactorial}%
1501 \def\xintpfactorial #1#2%
1502 {%
1503     \expandafter\XINT_pfac_fork
1504     \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.[0]%
1505 }%

```

8.48 \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).

```

1506 \def\xintPrd {\romannumeral0\xintprd}%
1507 \def\xintprd #1{\expandafter\XINT_prd\romannumeral`&&@#1^}%
1508 \def\XINT_Prd{\romannumeral0\XINT_prd}%
1509 \def\XINT_prd#1%
1510 {%
1511     \xint_gob_til_^ #1\XINT_prd_empty ^%
1512     \expandafter\XINT_prd_loop\romannumeral0\xintra{#1}\xint:
1513 }%
1514 \def\XINT_prd_empty ^#1\xint:{ 1/1[0]}%
1515 \def\XINT_prd_loop #1\xint:#2%
1516 {%
1517     \xint_gob_til_^ #2\XINT_prd_end ^%
1518     \expandafter\XINT_prd_loop
1519     \romannumeral0\xintmul{#1}{\romannumeral0\xintra{#2}}\xint:
1520 }%
1521 \def\XINT_prd_end ^#1\xintmul #2#3\xint:{ #2}%

```

8.49 \xintDiv

```

1522 \def\xintDiv {\romannumeral0\xintdiv}%
1523 \def\xintdiv #1%
1524 {%
1525     \expandafter\XINT_fdiv\expandafter {\romannumeral0\XINT_infrac {#1}}%
1526 }%
1527 \def\XINT_fdiv #1#2%
1528     {\expandafter\XINT_fdiv_A\romannumeral0\XINT_infrac {#2}#1}%
1529 \def\XINT_fdiv_A #1#2#3#4#5#6%
1530 {%
1531     \expandafter\XINT_fdiv_B
1532     \expandafter{\the\numexpr #4-#1\expandafter}%
1533     \expandafter{\romannumeral0\xintiimul {#2}{#6}}%
1534     {\romannumeral0\xintiimul {#3}{#5}}%
1535 }%
1536 \def\XINT_fdiv_B #1#2#3%
1537 {%

```



```

1538 \expandafter\XINT_fdiv_C
1539 \expandafter{#3}{#1}{#2}%
1540 }%
1541 \def\XINT_fdiv_C #1#2{\XINT_outfrac {#2}{#1}}%

```

8.50 \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.

```

1542 \def\xintDivFloor {\romannumeral0\xintdivfloor }%
1543 \def\xintdivfloor #1#2{\xintifloor{\xintDiv {#1}{#2}}}%

```

8.51 \xintDivTrunc

1.1. \xintttrunc rather than \xintitrunc0 in 1.1a

```

1544 \def\xintDivTrunc {\romannumeral0\xintdivtrunc }%
1545 \def\xintdivtrunc #1#2{\xintttrunc {\xintDiv {#1}{#2}}}%

```

8.52 \xintDivRound

1.1

```

1546 \def\xintDivRound {\romannumeral0\xintdivround }%
1547 \def\xintdivround #1#2{\xintiround 0{\xintDiv {#1}{#2}}}%

```

8.53 \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).

```

1548 \def\xintModTrunc {\romannumeral0\xintmodtrunc }%
1549 \def\xintmodtrunc #1{\expandafter\XINT_modtrunc_a\romannumeral0\xintra{#1}.}%
1550 \def\XINT_modtrunc_a #1#2.#3%
1551 {\expandafter\XINT_modtrunc_b\expandafter #1\romannumeral0\xintra{#3}#2.}%
1552 \def\XINT_modtrunc_b #1#2% #1 de A, #2 de B.
1553 {%
1554 \if0#2\xint_dothis{\XINT_modtrunc_divbyzero #1#2}\fi
1555 \if0#1\xint_dothis\XINT_modtrunc_aiszero\fi
1556 \if-#2\xint_dothis{\XINT_modtrunc_bneg #1}\fi
1557 \xint_orthat{\XINT_modtrunc_bpos #1#2}%
1558 }%
1559 \def\XINT_modtrunc_divbyzero #1#2[#3]#4.%
1560 {%
1561 \XINT_signalcondition{DivisionByZero}{Division by zero: #1#4/(#2[#3]).}{\{ 0/1[0]\}}%
1562 }%
1563 \def\XINT_modtrunc_aiszero #1.{ 0/1[0]}%
1564 \def\XINT_modtrunc_bneg #1%
1565 {%
1566 \xint_UDsignfork
1567 #1{\xintiiopp\XINT_modtrunc_pos }%%
1568 -{\XINT_modtrunc_pos #1}%

```

```

1569 \krof
1570 }%
1571 \def\XINT_modtrunc_bpos #1%
1572 {%
1573 \xint_UDsignfork
1574 #1{\xintiiopp\XINT_modtrunc_pos {}}%
1575 -{\XINT_modtrunc_pos #1}%
1576 \krof
1577 }%

    Attention. This crucially uses that xint's \xintiiE{x}{e} is defined to return x unchanged if e
    is negative (and x extended by e zeroes if e >= 0).

1578 \def\XINT_modtrunc_pos #1#2/#3[#4]#5/#6[#7].%
1579 {%
1580 \expandafter\XINT_modtrunc_pos_a
1581 \the\numexpr#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_modtrunc_pos_a #1.#2#3#4{\xintiirem {#3}{#4}/#2[#1]}%

```

8.54 \xintDivMod

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

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

```

1587 \def\xintDivMod {\romannumeral0\xintdivmod }%
1588 \def\xintdivmod #1{\expandafter\XINT_divmod_a\romannumeral0\xintra{#1}.}%
1589 \def\XINT_divmod_a #1#2.#3%
1590 {\expandafter\XINT_divmod_b\expandafter #1\romannumeral0\xintra{#3}#2.}%
1591 \def\XINT_divmod_b #1#2% #1 de A, #2 de B.
1592 {%
1593 \if0#2\xint_dothis{\XINT_divmod_divbyzero #1#2}\fi
1594 \if0#1\xint_dothis\XINT_divmod_aiszero\fi
1595 \if-#2\xint_dothis{\XINT_divmod_bneg #1}\fi
1596 \xint_orthat{\XINT_divmod_bpos #1#2}%
1597 }%
1598 \def\XINT_divmod_divbyzero #1#2[#3]#4.%
1599 {%
1600 \XINT_signalcondition{DivisionByZero}{Division by zero: #1#4/(#2[#3]).}{}%
1601 {{0}{0/1[0]}}% à revoir...
1602 }%
1603 \def\XINT_divmod_aiszero #1.{{0}{0/1[0]}}%
1604 \def\XINT_divmod_bneg #1% f // -g = (-f) // g, f % -g = -((-f) % g)
1605 {%
1606 \expandafter\XINT_divmod_bneg_finish
1607 \romannumeral0\xint_UDsignfork
1608 #1{\XINT_divmod_bpos {}}%
1609 -{\XINT_divmod_bpos {-#1}}%
1610 \krof
1611 }%

```

```

1612 \def\XINT_divmod_bneg_finish#1#2%
1613 {%
1614     \expandafter\xint_exchangetwo_keepbraces\expandafter
1615     {\romannumeral0\xintiioopp#2}{#1}%
1616 }%
1617 \def\XINT_divmod_bpos #1#2/#3[#4]#5/#6[#7].%
1618 {%
1619     \expandafter\XINT_divmod_bpos_a
1620     \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.%
1621     \romannumeral0\expandafter\XINT_mod_D_b
1622     \romannumeral0\XINT_div_prepare{#3}{#6}{#3}{#3}{#6}%
1623     {#1#5}{#7-#4}{#2}{#4-#7}%
1624 }%
1625 \def\XINT_divmod_bpos_a #1.#2#3#4%
1626 {%
1627     \expandafter\XINT_divmod_bpos_finish
1628     \romannumeral0\xintiidivision{#3}{#4}{/#2[#1]}%
1629 }%
1630 \def\XINT_divmod_bpos_finish #1#2#3{#1}{#2#3}%

```

8.55 \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.

```

1631 \def\xintMod {\romannumeral0\xintmod }%
1632 \def\xintmod #1{\expandafter\XINT_mod_a\romannumeral0\xintra{#1}.}%
1633 \def\XINT_mod_a #1#2.#3%
1634     {\expandafter\XINT_mod_b\expandafter #1\romannumeral0\xintra{#3}#2.}%
1635 \def\XINT_mod_b #1#2% #1 de A, #2 de B.
1636 {%
1637     \if0#2\xint_dothis{\XINT_mod_divbyzero #1#2}\fi
1638     \if0#1\xint_dothis\XINT_mod_aiszero\fi
1639     \if-#2\xint_dothis{\XINT_mod_bneg #1}\fi
1640     \xint_orthat{\XINT_mod_bpos #1#2}%
1641 }%

```

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

```

1642 \let\XINT_mod_divbyzero\XINT_modtrunc_divbyzero
1643 \let\XINT_mod_aiszero \XINT_modtrunc_aiszero
1644 \def\XINT_mod_bneg #1% f % -g = - ((-f) % g), for g > 0
1645 {%
1646     \xintiioopp\xint_UDsignfork
1647     #1{\XINT_mod_bpos {}}%
1648     -{\XINT_mod_bpos {-#1}}%
1649     \krof
1650 }%
1651 \def\XINT_mod_bpos #1#2/#3[#4]#5/#6[#7].%
1652 {%
1653     \expandafter\XINT_mod_bpos_a
1654     \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.%
1655     \romannumeral0\expandafter\XINT_mod_D_b

```

```

1656 \romannumeral0\XINT_div_prepare{#3}{#6}{#3}{#3}{#6}%
1657 {#1#5}{#7-#4}{#2}{#4-#7}%
1658 }%
1659 \def\XINT_mod_D_a #1#2%
1660 {%
1661 \expandafter\XINT_mod_D_b
1662 \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
1663 }%
1664 \def\XINT_mod_D_b #1#2{\XINT_mod_D_c #2\Z}%
1665 \def\XINT_mod_D_c #1#2\Z
1666 {%
1667 \xint_gob_til_zero #1\XINT_mod_D_exit0\XINT_mod_D_a {#1#2}%
1668 }%
1669 \def\XINT_mod_D_exit0\XINT_mod_D_a #1#2#3%
1670 {%
1671 \expandafter\XINT_mod_E
1672 \romannumeral0\xintiipro {#3}{#2}.{#2}%
1673 }%
1674 \def\XINT_mod_E #1.#2#3%
1675 {%
1676 \expandafter\XINT_mod_F
1677 \romannumeral0\xintiimul{#1}{#3}.{\xintiiQuo{#3}{#2}}{#1}%
1678 }%
1679 \def\XINT_mod_F #1.#2#3#4#5#6#7%
1680 {%
1681 {#1}{\xintiiE{\xintiiMul{#4}{#3}}{#5}}%
1682 {\xintiiE{\xintiiMul{#6}{#2}}{#7}}%
1683 }%
1684 \def\XINT_mod_bpos_a #1.#2#3#4{\xintiiirem {#3}{#4}/#2[#1]}%

```

8.56 \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.

```

1685 \def\xintIsOne {\romannumeral0\xintisone }%
1686 \def\xintisone #1{\expandafter\XINT_fracisone
1687 \romannumeral0\xintrawwithzeros{#1}\Z }%
1688 \def\XINT_fracisone #1/#2\Z
1689 {\if0\xintiiCmp {#1}{#2}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%

```

8.57 \xintGeq

```

1690 \def\xintGeq {\romannumeral0\xintgeq }%
1691 \def\xintgeq #1%
1692 {%
1693 \expandafter\XINT_fgeq\expandafter {\romannumeral0\xintabs {#1}}%
1694 }%
1695 \def\XINT_fgeq #1#2%
1696 {%
1697 \expandafter\XINT_fgeq_A \romannumeral0\xintabs {#2}#1%
1698 }%
1699 \def\XINT_fgeq_A #1%

```

```

1700 {%
1701   \xint_gob_til_zero #1\XINT_fgeq_Zii 0%
1702   \XINT_fgeq_B #1%
1703 }%
1704 \def\XINT_fgeq_Zii 0\XINT_fgeq_B #1[#2]#3[#4]{ 1}%
1705 \def\XINT_fgeq_B #1/#2[#3]#4#5/#6[#7]%
1706 {%
1707   \xint_gob_til_zero #4\XINT_fgeq_Zi 0%
1708   \expandafter\XINT_fgeq_C\expandafter
1709   {\the\numexpr #7-#3\expandafter}\expandafter
1710   {\romannumeral0\xintiimul {#4#5}{#2}}%
1711   {\romannumeral0\xintiimul {#6}{#1}}%
1712 }%
1713 \def\XINT_fgeq_Zi 0#1#2#3#4#5#6#7{ 0}%
1714 \def\XINT_fgeq_C #1#2#3%
1715 {%
1716   \expandafter\XINT_fgeq_D\expandafter
1717   {#3}{#1}{#2}%
1718 }%
1719 \def\XINT_fgeq_D #1#2#3%
1720 {%
1721   \expandafter\XINT_cntSgnFork\romannumeral`&&\expandafter\XINT_cntSgn
1722   \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\xint:
1723   { 0}{\XINT_fgeq_E #2\Z {#3}{#1}}{ 1}%
1724 }%
1725 \def\XINT_fgeq_E #1%
1726 {%
1727   \xint_UDsignfork
1728   #1\XINT_fgeq_Fd
1729   -{\XINT_fgeq_Fn #1}%
1730   \krof
1731 }%

1732 \def\XINT_fgeq_Fd #1\Z #2#3%
1733 {%
1734   \expandafter\XINT_fgeq_Fe
1735   \romannumeral0\XINT_dsx_addzeros {#1}#3;\xint:#2\xint:
1736 }%
1737 \def\XINT_fgeq_Fe #1\xint:#2#3\xint:{\XINT_geq_plusplus #2#1\xint:#3\xint:}%
1738 \def\XINT_fgeq_Fn #1\Z #2#3%
1739 {%
1740   \expandafter\XINT_fgeq_Fo
1741   \romannumeral0\XINT_dsx_addzeros {#1}#2;\xint:#3\xint:
1742 }%
1743 \def\XINT_fgeq_Fo #1#2\xint:#3\xint:{\XINT_geq_plusplus #1#3\xint:#2\xint:}%

```

8.58 \xintMax

```

1744 \def\xintMax {\romannumeral0\xintmax }%
1745 \def\xintmax #1%
1746 {%
1747   \expandafter\XINT_fmax\expandafter {\romannumeral0\xintraw {#1}}%
1748 }%

```

```

1749 \def\XINT_fmax #1#2%
1750 {%
1751     \expandafter\XINT_fmax_A\romannumeral0\xintra{#2}#1%
1752 }%
1753 \def\XINT_fmax_A #1#2/#3[#4]#5#6/#7[#8]%
1754 {%
1755     \xint_UDsignsfork
1756     #1#5\XINT_fmax_minusminus
1757     -#5\XINT_fmax_firstneg
1758     #1-\XINT_fmax_secondneg
1759     --\XINT_fmax_nonneg_a
1760     \krof
1761     #1#5{#2/#3[#4]}{#6/#7[#8]}%
1762 }%
1763 \def\XINT_fmax_minusminus --%
1764     {\expandafter-\romannumeral0\XINT_fmin_nonneg_b }%
1765 \def\XINT_fmax_firstneg #1-#2#3{ #1#2}%
1766 \def\XINT_fmax_secondneg -#1#2#3{ #1#3}%
1767 \def\XINT_fmax_nonneg_a #1#2#3#4%
1768 {%
1769     \XINT_fmax_nonneg_b {#1#3}{#2#4}%
1770 }%
1771 \def\XINT_fmax_nonneg_b #1#2%
1772 {%
1773     \if0\romannumeral0\XINT_fgeq_A #1#2%
1774         \xint_afterfi{ #1}%
1775     \else \xint_afterfi{ #2}%
1776     \fi
1777 }%

```

8.59 \xintMaxof

1.2l 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).

```

1778 \def\xintMaxof {\romannumeral0\xintmaxof }%
1779 \def\xintmaxof #1{\expandafter\XINT_maxof\romannumeral`&&#1^}%
1780 \def\XINT_Maxof{\romannumeral0\XINT_maxof}%
1781 \def\XINT_maxof#1%
1782 {%
1783     \xint_gob_til_ ^ #1\XINT_maxof_empty ^%
1784     \expandafter\XINT_maxof_loop\romannumeral0\xintra{#1}\xint:
1785 }%
1786 \def\XINT_maxof_empty ^#1\xint:{ 0/1[0]}%
1787 \def\XINT_maxof_loop #1\xint:#2%
1788 {%
1789     \xint_gob_til_ ^ #2\XINT_maxof_e ^%
1790     \expandafter\XINT_maxof_loop
1791     \romannumeral0\xintmax{#1}{\romannumeral0\xintra{#2}}\xint:
1792 }%
1793 \def\XINT_maxof_e ^#1\xintmax #2#3\xint:{ #2}%

```

8.60 \xintMin

```

1794 \def\xintMin {\romannumeral0\xintmin }%
1795 \def\xintmin #1%
1796 {%
1797   \expandafter\XINT_fmin\expandafter {\romannumeral0\xintraw {#1}}%
1798 }%
1799 \def\XINT_fmin #1#2%
1800 {%
1801   \expandafter\XINT_fmin_A\romannumeral0\xintraw {#2}#1%
1802 }%
1803 \def\XINT_fmin_A #1#2/#3[#4]#5#6/#7[#8]%
1804 {%
1805   \xint_UDsignsfork
1806     #1#5\XINT_fmin_minusminus
1807     -#5\XINT_fmin_firstneg
1808     #1-\XINT_fmin_secondneg
1809     --\XINT_fmin_nonneg_a
1810   \krof
1811   #1#5{#2/#3[#4]}{#6/#7[#8]}%
1812 }%
1813 \def\XINT_fmin_minusminus --%
1814   {\expandafter-\romannumeral0\XINT_fmax_nonneg_b }%
1815 \def\XINT_fmin_firstneg #1-#2#3{ -#3}%
1816 \def\XINT_fmin_secondneg -#1#2#3{ -#2}%
1817 \def\XINT_fmin_nonneg_a #1#2#3#4%
1818 {%
1819   \XINT_fmin_nonneg_b {#1#3}{#2#4}%
1820 }%
1821 \def\XINT_fmin_nonneg_b #1#2%
1822 {%
1823   \if0\romannumeral0\XINT_fgeq_A #1#2%
1824     \xint_afterfi{ #2}%
1825   \else \xint_afterfi{ #1}%
1826   \fi
1827 }%

```

8.61 \xintMinof

1.2l 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).

```

1828 \def\xintMinof {\romannumeral0\xintminof }%
1829 \def\xintminof #1{\expandafter\XINT_minof\romannumeral`&&#1^}%
1830 \def\XINT_Minof{\romannumeral0\XINT_minof}%
1831 \def\XINT_minof#1%
1832 {%
1833   \xint_gob_til_ ^ #1\XINT_minof_empty ^%
1834   \expandafter\XINT_minof_loop\romannumeral0\xintraw{#1}\xint:
1835 }%
1836 \def\XINT_minof_empty ^#1\xint:{ 0/1[0]}%
1837 \def\XINT_minof_loop #1\xint:#2%
1838 {%
1839   \xint_gob_til_ ^ #2\XINT_minof_e ^%

```

```

1840 \expandafter\XINT_minof_loop\romannumeral0\xintmin{#1}{\romannumeral0\xintraw{#2}}\xint:
1841 }%
1842 \def\XINT_minof_e ^#1\xintmin #2#3\xint:{ #2}%

```

8.62 \xintCmp

```

1843 \def\xintCmp {\romannumeral0\xintcmp }%
1844 \def\xintcmp #1%
1845 {%
1846 \expandafter\XINT_fcmp\expandafter {\romannumeral0\xintraw {#1}}%
1847 }%
1848 \def\XINT_fcmp #1#2%
1849 {%
1850 \expandafter\XINT_fcmp_A\romannumeral0\xintraw {#2}#1%
1851 }%
1852 \def\XINT_fcmp_A #1#2/#3[#4]#5#6/#7[#8]%
1853 {%
1854 \xint_UDsignsfork
1855 #1#5\XINT_fcmp_minusminus
1856 -#5\XINT_fcmp_firstneg
1857 #1-\XINT_fcmp_secondneg
1858 --\XINT_fcmp_nonneg_a
1859 \krof
1860 #1#5{#2/#3[#4]}{#6/#7[#8]}%
1861 }%
1862 \def\XINT_fcmp_minusminus --#1#2{\XINT_fcmp_B #2#1}%
1863 \def\XINT_fcmp_firstneg #1-#2#3{ -1}%
1864 \def\XINT_fcmp_secondneg -#1#2#3{ 1}%
1865 \def\XINT_fcmp_nonneg_a #1#2%
1866 {%
1867 \xint_UDzerosfork
1868 #1#2\XINT_fcmp_zerozero
1869 0#2\XINT_fcmp_firstzero
1870 #10\XINT_fcmp_secondzero
1871 00\XINT_fcmp_pos
1872 \krof
1873 #1#2%
1874 }%
1875 \def\XINT_fcmp_zerozero #1#2#3#4{ 0}%
1876 \def\XINT_fcmp_firstzero #1#2#3#4{ -1}%
1877 \def\XINT_fcmp_secondzero #1#2#3#4{ 1}%
1878 \def\XINT_fcmp_pos #1#2#3#4%
1879 {%
1880 \XINT_fcmp_B #1#3#2#4%
1881 }%
1882 \def\XINT_fcmp_B #1/#2[#3]#4/#5[#6]%
1883 {%
1884 \expandafter\XINT_fcmp_C\expandafter
1885 {\the\numexpr #6-#3\expandafter}\expandafter
1886 {\romannumeral0\xintiimul {#4}{#2}}%
1887 {\romannumeral0\xintiimul {#5}{#1}}%
1888 }%
1889 \def\XINT_fcmp_C #1#2#3%

```



```

1890 {%
1891     \expandafter\XINT_fcmp_D\expandafter
1892     {#3}{#1}{#2}%
1893 }%
1894 \def\XINT_fcmp_D #1#2#3%
1895 {%
1896     \expandafter\XINT_cntSgnFork\romannumeral`&&\expandafter\XINT_cntSgn
1897     \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\xint:
1898     { -1}{\XINT_fcmp_E #2\Z {#3}{#1}}{ 1}%
1899 }%
1900 \def\XINT_fcmp_E #1%
1901 {%
1902     \xint_UDsignfork
1903     #1\XINT_fcmp_Fd
1904     -{\XINT_fcmp_Fn #1}%
1905     \krof
1906 }%

1907 \def\XINT_fcmp_Fd #1\Z #2#3%
1908 {%
1909     \expandafter\XINT_fcmp_Fe
1910     \romannumeral0\XINT_dsx_addzeros {#1}#3;\xint:#2\xint:
1911 }%
1912 \def\XINT_fcmp_Fe #1\xint:#2#3\xint:{\XINT_cmp_plusplus #2#1\xint:#3\xint:}%
1913 \def\XINT_fcmp_Fn #1\Z #2#3%
1914 {%
1915     \expandafter\XINT_fcmp_Fo
1916     \romannumeral0\XINT_dsx_addzeros {#1}#2;\xint:#3\xint:
1917 }%
1918 \def\XINT_fcmp_Fo #1#2\xint:#3\xint:{\XINT_cmp_plusplus #1#3\xint:#2\xint:}%

```

8.63 \xintAbs

```

1919 \def\xintAbs {\romannumeral0\xintabs }%
1920 \def\xintabs #1{\expandafter\XINT_abs\romannumeral0\xintraw {#1}}%

```

8.64 \xintOpp

```

1921 \def\xintOpp {\romannumeral0\xintopp }%
1922 \def\xintopp #1{\expandafter\XINT_opp\romannumeral0\xintraw {#1}}%

```

8.65 \xintInv

1.3d (2019/01/06).

```

1923 \def\xintInv {\romannumeral0\xintinv }%
1924 \def\xintinv #1{\expandafter\XINT_inv\romannumeral0\xintraw {#1}}%
1925 \def\XINT_inv #1%
1926 {%
1927     \xint_UDzerominusfork
1928     #1-\XINT_inv_iszero
1929     0#1\XINT_inv_a
1930     0-{\XINT_inv_a {}}%
1931     \krof #1%

```

```

1932 }%
1933 \def\XINT_inv_iszero #1]%
1934   {\XINT_signalcondition{DivisionByZero}{Inverse of zero: inv(#1)}.}{\{ 0/1[0]}}}%
1935 \def\XINT_inv_a #1#2/#3[#4#5]%
1936 {%
1937   \xint_UDzerominusfork
1938     #4-\XINT_inv_expiszero
1939     0#4\XINT_inv_b
1940     0-\XINT_inv_b -#4}%
1941   \krof #5.{#1#3/#2}%
1942 }%
1943 \def\XINT_inv_expiszero #1.#2{ #2[0]}%
1944 \def\XINT_inv_b #1.#2{ #2[#1]}%

```

8.66 \xintSgn

```

1945 \def\xintSgn {\romannumeral0\xintsgn }%
1946 \def\xintsgn #1{\expandafter\XINT_sgn\romannumeral0\xintra {#1}\xint:}%

```

8.67 \xintGCD

1.4 (2020/01/31). 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.

1.4d (2021/03/29). Somewhat strangely `\xintGCD` was formerly `\xintGCDof` used with only two arguments, as the latter directly implemented a fractionl gcd algorithm using `\xintMod` repeatedly for two arguments.

Now `\xintGCD` contains the pairwise gcd routine and `\xintGCDof` is only a wrapper. And the pairwise gcd is reduced to integer-only computations to hopefully reduce fraction overhead.

Each input is filtered via `\xintPIrr` and `\xintREZ` to reduce size of manipulate integers in algebra.

But hesitation about applying `\xintPIrr` to output, and/or `\xintREZ`. (as it is applied on input).

But as the code is now used for frational lcm's we actually need to do some reduction of output else lcm's of integers will not be necessarily printed by `\xinteval` as integers.

Well finally I apply `\xintIrr` (but not `\xintREZ` to output). Hesitations here (thinking of inputs with large [n] parts, the output will have many zeros). So I do this only for the user macro but the core routine as used by `\xintGCDof` will not do it.

Also at [1.4d](#) the code uses `\expanded`.

```

1947 \def\xintGCD {\romannumeral0\xintgcd}%
1948 \def\xintgcd #1%
1949 {%
1950   \expandafter\XINT_fgcd_in
1951   \romannumeral0\xintrez{\xintPIrr{\xintAbs{#1}}}\xint:
1952 }%
1953 \def\XINT_fgcd_in #1#2\xint:#3%
1954 {%
1955   \expandafter\XINT_fgcd_out
1956   \romannumeral0\expandafter\XINT_fgcd_chkzeros\expandafter#1%
1957   \romannumeral0\xintrez{\xintPIrr{\xintAbs{#3}}}\xint:#1#2\xint:
1958 }%
1959 \def\XINT_fgcd_out#1[#2]{\xintirr{#1[#2]}[0]}%
1960 \def\XINT_fgcd_chkzeros #1#2%

```

```

1961 {%
1962   \xint_UDzerofork
1963     #1\XINT_fgcd_aiszero
1964     #2\XINT_fgcd_biszero
1965     0\XINT_fgcd_main
1966   \krof #2%
1967}%
1968 \def\XINT_fgcd_aiszero #1\xint:#2\xint:{ #1}%
1969 \def\XINT_fgcd_biszero #1\xint:#2\xint:{ #2}%
1970 \def\XINT_fgcd_main #1/#2[#3]\xint:#4/#5[#6]\xint:
1971 {%
1972   \expandafter\XINT_fgcd_a
1973     \romannumeral0\XINT_gcd_loop #2\xint:#5\xint:\xint:
1974     #2\xint:#5\xint:#1\xint:#4\xint:#3.#6.%
1975}%
1976 \def\XINT_fgcd_a #1\xint:#2\xint:
1977 {%
1978   \expandafter\XINT_fgcd_b
1979     \romannumeral0\xintiigo{#2}{#1}\xint:#1\xint:#2\xint:
1980}%
1981 \def\XINT_fgcd_b #1\xint:#2\xint:#3\xint:#4\xint:#5\xint:#6\xint:#7.#8.%
1982 {%
1983   \expanded{
1984     \xintiigcd{\xintiiE{\xintiiMul{#5}{\xintiiQuo{#4}{#2}}}{#7-#8}}%
1985       {\xintiiE{\xintiiMul{#6}{#1}}{#8-#7}}%
1986     /\xintiiMul{#1}{#4}%
1987     [\ifnum#7>#8 #8\else #7\fi]%
1988   }%
1989}%

```

8.68 \xintGCDof

1.4 (2020/01/31). This inherits from former non public [xintexpr](#) macro called `\xintGCDof:csv`, which handled comma separated items.

It handles fractions presented as braced items and is the support macro for the `gcd()` function in `\xintexpr` and `\xintfloatexpr`. The support macro for the `gcd()` function in `\xintiiexpr` is `\xint-iiGCDof`, from [xint](#).

An empty input is allowed but I have some hesitations on the return value of 1.

1.4d (2021/03/29). Sadly the 1.4 version had multiple problems:

- broken if first argument vanished,
- broken if some argument was not in strict format, for example had leading chains of signs or zeros (`\xintGCDof{2}{03}`). This bug originates in the fact the original macro was used only in [xintexpr](#) sanitized context.

Also, output is now always an irreducible fraction (ending with `[0]`).

```

1990 \def\xintGCDof {\romannumeral0\xintgcdof}%
1991 \def\xintgcdof #1{\expandafter\XINT_fgcdof\romannumeral`&&@#1}%
1992 \def\XINT_GCDof{\romannumeral0\XINT_fgcdof}%
1993 \def\XINT_fgcdof #1%
1994 {%
1995   \expandafter\XINT_fgcdof_chkempty\romannumeral`&&@#1\xint:

```

```

1996 }%
1997 \def\XINT_fgcdof_chkempty #1%
1998 {%
1999     \xint_gob_til_^\#1\XINT_fgcdof_empty ^\XINT_fgcdof_in #1%
2000 }%
2001 \def\XINT_fgcdof_empty #1\xint:{ 1/1[0]}% hesitation, should it be infinity? 0?
2002 \def\XINT_fgcdof_in #1\xint:
2003 {%
2004     \expandafter\XINT_fgcd_out
2005     \romannumeral0\expandafter\XINT_fgcdof_loop
2006     \romannumeral0\xintrez{\xintPIrr{\xintAbs{#1}}}\xint:
2007 }%
2008 \def\XINT_fgcdof_loop #1\xint:#2%
2009 {%
2010     \expandafter\XINT_fgcdof_chkend\romannumeral`&&@#2\xint:#1\xint:\xint:
2011 }%
2012 \def\XINT_fgcdof_chkend #1%
2013 {%
2014     \xint_gob_til_^\#1\XINT_fgcdof_end ^\XINT_fgcdof_loop_pair #1%
2015 }%
2016 \def\XINT_fgcdof_end #1\xint:#2\xint:\xint:{ #2}%
2017 \def\XINT_fgcdof_loop_pair #1\xint:#2%
2018 {%
2019     \expandafter\XINT_fgcdof_loop
2020     \romannumeral0\expandafter\XINT_fgcd_chkzeros\expandafter#2%
2021     \romannumeral0\xintrez{\xintPIrr{\xintAbs{#1}}}\xint:#2%
2022 }%

```

8.69 \xintLCM

Same comments as for \xintGCD. Entirely redone for 1.4d. Well, actually we can express it in terms of fractional gcd.

```

2023 \def\xintLCM {\romannumeral0\xintlcm}%
2024 \def\xintlcm #1%
2025 {%
2026     \expandafter\XINT_flcm_in
2027     \romannumeral0\xintrez{\xintPIrr{\xintAbs{#1}}}\xint:
2028 }%
2029 \def\XINT_flcm_in #1#2\xint:#3%
2030 {%
2031     \expandafter\XINT_fgcd_out
2032     \romannumeral0\expandafter\XINT_flcm_chkzeros\expandafter#1%
2033     \romannumeral0\xintrez{\xintPIrr{\xintAbs{#3}}}\xint:#1#2\xint:
2034 }%
2035 \def\XINT_flcm_chkzeros #1#2%
2036 {%
2037     \xint_UDzerofork
2038     #1\XINT_flcm_zero
2039     #2\XINT_flcm_zero
2040     0\XINT_flcm_main
2041     \krof #2%
2042 }%

```

```

2043 \def\XINT_flgcm_zero #1\xint:#2\xint:{ 0/1[0]}%
2044 \def\XINT_flgcm_main #1/#2[#3]\xint:#4/#5[#6]\xint:
2045 {%
2046     \xintinv
2047     {%
2048         \romannumeral0\XINT_fgcd_main #2/#1[-#3]\xint:#5/#4[-#6]\xint:
2049     }%
2050 }%

```

8.70 \xintLCMof

See comments for `\xintGCDof`. `xint` provides the integer only `\xintiilCMof`.

adly, although a public `xintfrac` macro, it did not (since 1.4) sanitize its arguments like other `xintfrac` macros.

```

2051 \def\xintLCMof {\romannumeral0\xintlcmof}%
2052 \def\xintlcmof #1{\expandafter\XINT_flgcmof\romannumeral`&&@#1^}%
2053 \def\XINT_LCMof{\romannumeral0\XINT_flgcmof}%
2054 \def\XINT_flgcmof #1%
2055 {%
2056     \expandafter\XINT_flgcmof_chkempty\romannumeral`&&@#1\xint:
2057 }%
2058 \def\XINT_flgcmof_chkempty #1%
2059 {%
2060     \xint_gob_til_`^#1\XINT_flgcmof_empty ^\XINT_flgcmof_in #1%
2061 }%
2062 \def\XINT_flgcmof_empty #1\xint:{ 0/1[0]}% hesitation
2063 \def\XINT_flgcmof_in #1\xint:
2064 {%
2065     \expandafter\XINT_fgcd_out
2066     \romannumeral0\expandafter\XINT_flgcmof_loop
2067     \romannumeral0\xintrez{\xintPIrr{\xintAbs{#1}}}\xint:
2068 }%
2069 \def\XINT_flgcmof_loop #1\xint:#2%
2070 {%
2071     \expandafter\XINT_flgcmof_chkend\romannumeral`&&@#2\xint:#1\xint:\xint:
2072 }%
2073 \def\XINT_flgcmof_chkend #1%
2074 {%
2075     \xint_gob_til_`^#1\XINT_flgcmof_end ^\XINT_flgcmof_loop_pair #1%
2076 }%
2077 \def\XINT_flgcmof_end #1\xint:#2\xint:\xint:{ #2}%
2078 \def\XINT_flgcmof_loop_pair #1\xint:#2%
2079 {%
2080     \expandafter\XINT_flgcmof_chkzero
2081     \romannumeral0\expandafter\XINT_flgcmof_chkzeros\expandafter#2%
2082     \romannumeral0\xintrez{\xintPIrr{\xintAbs{#1}}}\xint:#2%
2083 }%
2084 \def\XINT_flgcmof_chkzero #1%
2085 {%
2086     \xint_gob_til_zero#1\XINT_flgcmof_zero0\XINT_flgcmof_loop#1%
2087 }%
2088 \def\XINT_flgcmof_zero#1^{ 0/1[0]}%

```

8.71 Floating point macros

For a long time the float routines dating back to releases 1.07/1.08a (May-June 2013) were not modified.

Since 1.2f (March 2016) the four operations first round their arguments to `\xinttheDigits`-floats (or `P`-floats), not `(\xinttheDigits+2)`-floats or `(P+2)`-floats as was the case with earlier releases.

The four operations addition, subtraction, multiplication, division have always produced the correct rounding of the theoretical exact value to `P` or `\xinttheDigits` digits when the inputs are decimal numbers with at most `P` digits, and arbitrary decimal exponent part.

From 1.08a to 1.2j, `\xintFloat` (and `\XINTinFloat` which is used to parse inputs to other float macros) handled a fractional input `A/B` via an initial replacement to `A'/B'` where `A'` and `B'` were `A` and `B` truncated to `Q+2` digits (where asked-for precision is `Q`), and then they correctly rounded `A/B` to `Q` digits. But this meant that this rounding of the input could differ (by up to one unit in the last place) from the correct rounding of the original `A/B` to the asked-for number of digits (which until 1.2f in uses as auxiliary to the macros for the basic operations was 2 more than the prevailing precision).

Since 1.2k all inputs are correctly rounded to the asked-for number of digits (this was, I think, the case in the 1.07 release -- there are no code comments -- but was, afaicr, not very efficiently done, and this is why the 1.08a release opted for truncation of the numerator and denominator.)

Notice that in float expressions, the `/` is treated as operator, hence the above discussion makes a difference only for the special input form `qfloat(A/B)` or for an `\xintexpr A/B\relax` embedded in the float expression, with `A` or `B` having more digits than the prevailing float precision.

Internally there is no inner representation of `P`-floats as such !!!!!

The input parser will again compute the length of the mantissa on each use !!! This is obviously something that must be improved upon before implementation of higher functions.

Currently, special tricks are used to quickly recognize inputs having no denominators, or fractions whose numerators and denominators are not too long compared to the target precision `P`, and in particular `P`-floats or quotients of two such.

Another long-standing issue is that float multiplication will first compute the `2P` or `2P-2` digits of the exact product, and then round it to `P` digits. This is sub-optimal for large `P` particularly as the multiplication algorithm is basically the schoolbook one, hence worse than quadratic in the `TEX` implementation which has extra cost of fetching long sequences of tokens.

Changes at 1.4e (done 2021/04/15; undone 2021/04/29)

Macros named `\XINTinFloat<name>` are not public user-level but were designed a long time ago for `\xintfloatexpr` context as a very preliminary step towards attempting to preserve some internal format, here `A[N]` type.

When `<name>` is lowercased it means it needs a `\romannumeral0` trigger (`\XINTinfloatS` keeps an uppercase `S`).

Most were coded to check for an optional argument `[D]`, and to use `D=\XINTdigits` in its place if absent but it turned out only `\XINTinfloatpow`, `\XINTinfloatmul`, `\XINTinfloatadd` were actually used with an optional argument and this happened only in macros from the very old `xintseries.sty`, so I changed all of them to not check for optional argument `[D]` anymore, keeping only some private interface for the `xintseries.sty` use case. Some required being used with `[D]`, some still had names ending in "digits" indicating they would use `\XINTdigits` always.

Indeed basically all algebra is done "exactly" and the `[D]` governs rules of float-rounding on input and output.

During development of 1.4e we fleetingly experimented with letting the value used in place of `D` be `\XINTdigitsex` to 1.4e, i.e. `\XINTdigits` with guard digits, a situation which was motivated

by the implementation of trigonometrical functions at high level, i.e. using `\xintdeffloatfunc` which had no mechanism to make intermediate calculations with guard digits.

Simply doing everything "as is" but with 2 guard digits proved very good (surprisingly efficient, even) to the trigonometrical functions. However using them systematically raises many issues (for example, the correct rounding at P digits is destroyed if we obtain it a D=P+2 then round from P+2 to P digits so we definitely can not do this as default, so some interface is needed to define intermediate functions only using such guard digits and keeping them in their output).

Finally, an approach limited to the `xinttrig.sty` scope was used and I removed all `\XINTdigitsx` related matters from 1.4e. But this left some modifications of the interfaces of the "float" macros here which this list tries to document, mainly for the author's benefit.

Macros always using `\XINTdigits` and now not allowing [P] option

```
\XINTinFloatAdd
\XINTinFloatSub
\XINTinFloatMul
\XINTinFloatSqr
\XINTinFloatInv
\XINTinFloatDiv
\XINTinFloatPow
\XINTinFloatPower
\XINTinFloatPFactorial
\XINTinFloatBinomial
```

Macros which already did not allow [P] option prior to 1.4e refactoring

```
\XINTinFloatFrac (renamed from \XINTinFloatFracdigits)
\XINTinFloatE
\XINTinFloatMod
\XINTinFloatDivFloor
\XINTinFloatDivMod
```

Macros requiring a [P]. Some of the "_wopt" named macros are renamings of macros formerly requiring [P].

```
\XINTinFloat
\XINTinFloatS
\XINTFloatiLogTen
\XINTinRandomFloatS (this one has only the [P] mandatory argument)
\XINTinFloatFac
\XINTinFloatSqrt
\XINTinFloatAdd_wopt, \XINTinfloatadd_wopt
\XINTinFloatSub_wopt, \XINTinfloatsub_wopt
\XINTinFloatMul_wopt, \XINTinfloatmul_wopt
\XINTinFloatSqr_wopt
\XINTinfloatpow_wopt (not FloatPow)
\XINTinFloatDiv_wopt
\XINTinFloatInv_wopt
```

Specially named macros indicating usage of `\XINTdigits`

```
\XINTinFloatdigits
\XINTinFloatSdigits
\XINTFloatiLogTendigits
\XINTinRandomFloatSdigits
\XINTinFloatFacdigits
\XINTinFloatSqrtdigits
```

8.72 \xintDigits, \xintSetDigits

1.3f allows `\xintDigits=` in place of `\xintDigits:=` syntax. It defines `\xintDigits*[:]=` which reloads `xinttrig.sty`. Perhaps this should be default, well.

During 1.4e development I added an interface for guard digits, but I decided to drop inclusion from 1.4e release because there were pending issues both in documentation and functionalities for which I did not have time left.

1.4e fixes the issue that `\xinttheDigits` could not be used in the right hand side of `\xintDigits*[:]=...` or inside the argument to `\xintSetDigits`.

```
2089 \mathchardef\XINTdigits 16
2090 \chardef\XINTguarddigits 0
2091 \def\xinttheDigits {\number\XINTdigits}%
2092 %\def\xinttheGuardDigits{\number\XINTguarddigits}%
2093 \def\xinttheGuardDigits{0}% in case used in some of my test files
2094 \def\xintDigits #1={\afterassignment\xintDigits_i\mathchardef\XINT_digits=}%
2095 \def\xintDigits_i#1%
2096 {%
2097   \let\XINTdigits\XINT_digits
2098 }%
2099 \def\xintSetDigits #1%
2100 {%
2101   \mathchardef\XINT_digits=\numexpr#1\relax
2102   \let\XINTdigits=\XINT_digits
2103 }%
```

8.73 \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.

```

2104 \def\xintFloat    {\romannumeral0\xintfloat}%
2105 \def\xintfloat #1{\XINT_float_chkopt #1\xint:}%
2106 \def\XINT_float_chkopt #1%
2107 {%
2108     \ifx [#1\expandafter\XINT_float_opt
2109         \else\expandafter\XINT_float_noopt
2110     \fi #1%
2111}%
2112 \def\XINT_float_noopt #1\xint:%
2113 {%
2114     \expandafter\XINT_float_post
2115     \romannumeral0\XINTinfloat[\XINTdigits]{#1}\XINTdigits.%
2116}%
2117 \def\XINT_float_opt [\xint:#1]%
2118 {%
2119     \expandafter\XINT_float_opt_a\the\numexpr #1.%
2120}%
2121 \def\XINT_float_opt_a #1.#2%
2122 {%
2123     \expandafter\XINT_float_post
2124     \romannumeral0\XINTinfloat[#1]{#2}#1.%
2125}%
2126 \def\XINT_float_post #1%
2127 {%
2128     \xint_UDzerominusfork
2129     #1-\XINT_float_zero
2130     0#1\XINT_float_neg
2131     0-\XINT_float_pos
2132     \krof #1%
2133}%[
2134 \def\XINT_float_zero #1]#2.{ 0.e0}%
2135 \def\XINT_float_neg-\{\expandafter-\romannumeral0\XINT_float_pos}%
2136 \def\XINT_float_pos #1#2[#3]#4.%
2137 {%
2138     \expandafter\XINT_float_pos_done\the\numexpr#3+#4-\xint_c_i.#1.#2;%
2139}%
2140 \def\XINT_float_pos_done #1.#2;{ #2e#1}%

```

8.74 \XINTinFloat, \XINTinFloatS, \XINTiLogTen

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

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

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

(but the zero value must always be checked for, as it outputs 0[0])

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 `\XINTFloatiLogTen`.

```
2141 \def\XINTinFloat {\romannumeral0\XINTinfloat}%
2142 \def\XINTinfloat
2143   {\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).

```
2144 \def\XINT_infloat_clean #1%
2145   {\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

```
2146 \def\XINT_infloat_clean_a !#1.#2[#3]%
2147 {%
2148   \expandafter\XINT_infloat_done
2149   \the\numexpr #3-#1\expandafter.%
2150   \romannumeral0\XINT_dsx_addzeros {#1}#2;;%
2151 }%
2152 \def\XINT_infloat_done #1.#2;{ #2[#1]}%
```

variant which allows output with shorter mantissas.

```
2153 \def\XINTinFloatS {\romannumeral0\XINTinfloatS}%
2154 \def\XINTinfloatS
2155   {\expandafter\XINT_infloatS_clean\romannumeral0\XINT_infloat}%
2156 \def\XINT_infloatS_clean #1%
2157   {\if #1!\xint_dothis\XINT_infloatS_clean_a\fi\xint_orthat{ }#1}%
2158 \def\XINT_infloatS_clean_a !#1.{ }%
```

1.3e ajoute `\XINTFloatiLogTen`. Le comportement pour un input nul est non encore finalisé. Il changera lorsque NaN, +Inf, -Inf existeront.

```
2159 \def\XINTFloatiLogTen {\the\numexpr\XINTfloatilogten}%
2160 \def\XINTfloatilogten [#1]#2%
2161   {\expandafter\XINT_floatilogten\romannumeral0\XINT_infloat[#1]{#2}#1.%}
2162 \def\XINTFloatiLogTendigits{\the\numexpr\XINTfloatilogten[\XINTdigits]}%
2163 \def\XINT_floatilogten #1{%
2164   \if #10\xint_dothis\XINT_floatilogten_z\fi
2165   \if #1!\xint_dothis\XINT_floatilogten_a\fi
2166   \xint_orthat\XINT_floatilogten_b #1%
2167 }%
2168 \def\XINT_floatilogten_z 0[0]#1.{-7FFF8000\relax}%
```

```
2169 \def\XINT_floatilogten_a !#1.#2[#3]#4.{#3-#1+#4-1\relax}%
2170 \def\XINT_floatilogten_b #1[#2]#3.{#2+#3-1\relax}%
```

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

```
2171 \def\XINT_infloat [#1]#2%
2172 {%
2173     \expandafter\XINT_infloat_a\the\numexpr #1\expandafter.%
2174     \romannumeral0\XINT_infrac {#2}%
2175 }%
```

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

```
2176 \def\XINT_infloat_a #1.#2#3#4%
2177 {%
```

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

```
2178     \if1\XINT_is_One#4XY%
2179     \expandafter\XINT_infloat_sp
2180     \else\expandafter\XINT_infloat_fork
2181     \fi #3.{#1}{#2}{#4}%
2182 }%
```

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.

```
2183 \def\XINT_infloat_sp #1%
2184 {%
2185     \xint_UDzerominusfork
2186     #1-\XINT_infloat_spzero
2187     0#1\XINT_infloat_spneg
2188     0-\XINT_infloat_sppos
2189     \krof #1%
2190 }%
```

Attention surtout pas 0/1[0] ici.

```
2191 \def\XINT_infloat_spzero 0.#1#2#3{ 0[0]}%
2192 \def\XINT_infloat_spneg-%
2193     {\expandafter\XINT_infloat_spnegend\romannumeral0\XINT_infloat_sppos}%
2194 \def\XINT_infloat_spnegend #1%
2195     {\if#1!\expandafter\XINT_infloat_spneg_needzeros\fi -#1}%
2196 \def\XINT_infloat_spneg_needzeros -!#1.{!#1.-}%
```

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

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

```
2197 \def\XINT_infloat_sppos #1.#2#3#4%
2198 {%
2199     \expandafter\XINT_infloat_sp_b\the\numexpr#2-\xintLength{#1}.#1.#2.#3.%
2200 }%
```

#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.

```
2201 \def\XINT_infloat_sp_b #1%
2202 {%
2203     \xint_UDzerominusfork
2204     #1-\XINT_infloat_sp_quick
2205     0#1\XINT_infloat_sp_c
2206     0-\XINT_infloat_sp_needzeros
```

```
2207 \krof #1%
2208 }%
```

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

```
2209 \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.

```
2210 \def\xINT_infloat_sp_needzeros #1.#2.#3.#4.{!#1.#2[#4]}%
```

$L-P=\#1$. $A=\#2$. $P=\#4$. $N=\#5$.

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`.

```
2211 \def\xINT_infloat_sp_c -#1.#2#3.#4.#5.%
```

```
2212 {%
```

```
2213 \expandafter\xINT_infloat_Y
```

```
2214 \the\numexpr #5+#1\expandafter.%
```

```
2215 \romannumeral0\expandafter\xINT_infloat_sp_round
```

```
2216 \romannumeral0\xINT_split_fromleft
```

```
2217 (\xint_c_i+#4).#2#3\xint_bye2345678\xint_bye..#2%
```

```
2218 }%
```

```
2219 \def\xINT_infloat_sp_round #1.#2.%
```

```
2220 {%
```

```
2221 \XINT_dsrr#1\xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax.%
```

```
2222 }%
```

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.

```
2223 \def\xINT_infloat_fork #1%
```

```
2224 {%
```

```
2225 \xint_UDsignfork
```

```
2226 #1\xINT_infloat_J
```

```
2227 -\XINT_infloat_K
```

```
2228 \krof #1%
```

```
2229 }%
```

```
2230 \def\xINT_infloat_J-{\expandafter-\romannumeral0\xINT_infloat_K }%
```

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

```
2231 \def\xINT_infloat_K #1.#2%
```

```
2232 {%
```

```
2233 \expandafter\xINT_infloat_L
```

```
2234 \the\numexpr\xintLength{#1}\expandafter.\the\numexpr #2+\xint_c_iv.{#1}\{#2}%
```

```
2235 }%
```

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

```
2236 \def\xINT_infloat_L #1.#2.%
```

```
2237 {%
```

```
2238 \ifnum #1>#2
```

```
2239 \expandafter\xINT_infloat_Ma
```

```

2240     \else
2241         \expandafter\XINT_infloat_Mb
2242     \fi #1.#2.%
2243 }%

    |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}

2244 \def\XINT_infloat_Ma #1.#2.#3%
2245 {%
2246     \expandafter\XINT_infloat_MtoN\expandafter-\expandafter0\expandafter.%
2247     \romannumeral0\XINT_split_fromleft#2.#3\xint_bye2345678\xint_bye..%
2248     #2.#1.{#3}%
2249 }%

    |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}

2250 \def\XINT_infloat_Mb #1.#2.#3%
2251 {%
2252     \expandafter\XINT_infloat_MtoN\the\numexpr#2-#1.%
2253     #3..#2.#1.{#3}%
2254 }%

    input u.A''.junk.P+4.|A|. {A}{P}{n}{B}
    output |B|.P+4.{B}u.A''.P.|A|.n.{A}{B}

2255 \def\XINT_infloat_MtoN #1.#2.#3.#4.#5.#6#7#8#9%
2256 {%
2257     \expandafter\XINT_infloat_N
2258     \the\numexpr\xintLength{#9}.#4.{#9}#1.#2.#7.#5.#8.{#6}{#9}%
2259 }%
2260 \def\XINT_infloat_N #1.#2.%
2261 {%
2262     \ifnum #1>#2
2263         \expandafter\XINT_infloat_Oa
2264     \else
2265         \expandafter\XINT_infloat_Ob
2266     \fi #1.#2.%
2267 }%

    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}

2268 \def\XINT_infloat_Oa #1.#2.#3%
2269 {%
2270     \expandafter\XINT_infloat_P\expandafter-\expandafter0\expandafter.%
2271     \romannumeral0\XINT_split_fromleft#2.#3\xint_bye2345678\xint_bye..%
2272     #1.%
2273 }%

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

2274 \def\XINT_infloat_Ob #1.#2.#3%
2275 {%
2276     \expandafter\XINT_infloat_P\the\numexpr#2-#1.#3..#1.%
2277 }%

```

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.

2278 \def\xINT_infloat_P #1#2.#3.#4.#5.#6#7.#8.#9.%

2279 {%

2280 \csname XINT_infloat_Q\if-#1\else\if-#6\else q\fi\fi\expandafter\endcsname

2281 \romannumeral0\xinti quo

2282 {\romannumeral0\xINT_dsx_addzerosnofuss

2283 {#6#7-#1#2+#9+\xint_c_iii\if-#1\else\if-#6\else-\xint_c_ii\fi\fi}#8;}%

2284 {#3}.#9.#5.%

2285 }%

«quick» branch.

2286 \def\xINT_infloat_Qq #1.#2.%

2287 {%

2288 \expandafter\xINT_infloat_Rq

2289 \romannumeral0\xINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..#2.%

2290 }%

2291 \def\xINT_infloat_Rq #1.#2#3.%

2292 {%

2293 \ifnum#2<\xint_c_v

2294 \expandafter\xINT_infloat_SEq

2295 \else\expandafter\xINT_infloat_SUP

2296 \fi

2297 {\if.#3.\xint_c_\else\xint_c_i\fi}#1.%

2298 }%

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

2299 \def\xINT_infloat_Q #1.#2.%

2300 {%

2301 \expandafter\xINT_infloat_R

2302 \romannumeral0\xINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..#2.%

2303 }%

2304 \def\xINT_infloat_R #1.#2#3#4#5.%

2305 {%

2306 \if.#5.\expandafter\xINT_infloat_Sa\else\expandafter\xINT_infloat_Sb\fi

2307 #2#3#4#5.#1.%

2308 }%

trailing digits.Q.P.|B|.A|.n.{A}{B}

#1=trailing digits (they may have leading zeros.)

2309 \def\xINT_infloat_Sa #1.%

2310 {%

2311 \ifnum#1>500 \xint_dothis\xINT_infloat_SUP\fi

2312 \ifnum#1<499 \xint_dothis\xINT_infloat_SEq\fi

2313 \xint_orthat\xINT_infloat_X\xint_c_

2314 }%

2315 \def\xINT_infloat_Sb #1.%

2316 {%

2317 \ifnum#1>5009 \xint_dothis\xINT_infloat_SUP\fi

```

2318 \ifnum#1<4990 \xint_dothis\XINT_infloat_SEq\fi
2319 \xint_orthat\XINT_infloat_X\xint_c_i
2320 }%

    epsilon #2=Q.#3=P.#4=|B|. #5=|A|. #6=n.{A}{B}
    exposant final est n+|A|-|B|-P+epsilon
2321 \def\XINT_infloat_SEq #1#2.#3.#4.#5.#6.#7#8%
2322 {%
2323 \expandafter\XINT_infloat_SY
2324 \the\numexpr #6+#5-#4-#3+#1.#2.%
2325 }%
2326 \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.
2327 \def\XINT_infloat_SUP #1#2#3.#4.#5.#6.#7.#8#9%
2328 {%
2329 \expandafter\XINT_infloat_Y
2330 \the\numexpr#7+#6-#5-#4+#1\expandafter.%
2331 \romannumeral0\xintinc{#2#3}.#2%
2332 }%

    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
2333 \def\XINT_infloat_X #1#2#3.#4.#5.#6.#7.#8#9%
2334 {%
2335 \expandafter\XINT_infloat_Y
2336 \the\numexpr #7+#6-#5-#4+#1\expandafter.%
2337 \romannumeral`&&\romannumeral0\xintiiflt
2338 {\xintDSH{#6-#5-#4+#1}{\xintDouble{#8}}}%
2339 {\xintiiMul{\xintInc{\xintDouble{#2#3}}}{#9}}%
2340 \xint_firstofone
2341 \xintinc{#2#3}.#2%
2342 }%

    check for rounding up to next power of ten.
2343 \def\XINT_infloat_Y #1{%
2344 \def\XINT_infloat_Y ##1.##2##3.##4%
2345 {%
2346 \if##49\if##21\expandafter\expandafter\expandafter\XINT_infloat_Z\fi\fi
2347 #1##2##3[##1]%
2348 }}\XINT_infloat_Y{ }%

    #1=1, #2=0.
2349 \def\XINT_infloat_Z #1#2#3[#4]%
2350 {%
2351 \expandafter\XINT_infloat_ZZ\the\numexpr#4+\xint_c_i.#3.%
2352 }%
2353 \def\XINT_infloat_ZZ #1.#2.{ #1#2[#1]}%

```

8.75 \xintPFloat, \xintPFloatE

xint has not yet incorporated a general formatter as it was not a priority during development and external solutions exist (I did not check for a while but I think LaTeX3 has implemented a general formatter in the printf or Python ".format" spirit)

But when one starts using really the package, especially in an interactive way (xintsession 2021), one needs the default output to be as nice as possible.

The \xintPFloat macro was added at 1.1 as a "prettifying printer" for floats, basically influenced by Maple.

The rules were:

0. The input is float-rounded to either Digits or the optional argument
 1. zero is printed as "0."
 2. x.yz...eK is printed "as is" if $K > 5$ or $K < -5$.
 3. if $-5 \leq K \leq 5$, fixed point decimal notation is used.
 4. in cases 2. and 3., no trimming of trailing zeroes.
- 1.4b added \xintPFloatE to customize whether to use e or E.
- 1.4e, with some hesitation, decided to make a breaking change and to modify the behaviour.

The new rules:

0. The input is float-rounded to either Digits or the optional argument
 1. zero is printed as 0.0
 2. x.yz...eK is printed in decimal fixed point if $-4 \leq K \leq +5$ (notice the change, formerly $K = -5$ used fixed point notation in output) else it is printed in scientific notation
 3. trailing zeros of the mantissa are trimmed always
 4. in case of decimal fixed point for an integer, there is a trailing ".0"
 5. in case of scientific notation with a one-digit trimmed mantissa there is an added ".0"
- too

Further, \xintPFloatE can now grab the scientific exponent K which is presented to it as explicit tokens (digit tokens, at least one, and an optional minus sign) delimited by a dot. It is thus now possible to customize at will for example adding a + sign in case of positive scientific exponent.

The macro must be f-expandable.

```

2354 \def\xintPFloat {\romannumeral0\xintpfloat }%
2355 \def\xintpfloat #1{\XINT_pfloat_chkopt #1\xint:}%
2356 \def\xintPFloat_wopt
2357 {%
2358   \romannumeral0\expandafter\XINT_pfloat\romannumeral0\XINTinfloatS
2359 }%
2360 \def\XINT_pfloat_chkopt #1%
2361 {%
2362   \ifx [#1\expandafter\XINT_pfloat_opt
2363     \else\expandafter\XINT_pfloat_noopt
2364   \fi #1%
2365 }%
2366 \def\XINT_pfloat_noopt #1\xint:%
2367 {%
2368   \expandafter\XINT_pfloat\romannumeral0\XINTinfloatS[\XINTdigits]{#1}%
2369 }%
2370 \def\XINT_pfloat_opt [\xint:#1]%
2371 {%
2372   \expandafter\XINT_pfloat\romannumeral0\XINTinfloatS[#1]%
2373 }%
2374 \def\XINT_pfloat#1%
2375 {%
2376   \expandafter\XINT_pfloat_fork\romannumeral0\xintrez{#1}}%
```



```

2377 }%
2378 \def\XINT_pfloat_fork#1%
2379 {%
2380     \xint_UDzerominusfork
2381     #1-\XINT_pfloat_zero
2382     0#1\XINT_pfloat_neg
2383     0-\XINT_pfloat_pos
2384     \krof #1%
2385 }%
2386 \def\XINT_pfloat_zero#1{ 0.0}%
2387 \def\XINT_pfloat_neg-{\expandafter-\romannumeral0\XINT_pfloat_pos}%
2388 \def\XINT_pfloat_pos#1/1[#2]%
2389 {%
2390     \expandafter\XINT_pfloat_a\the\numexpr\xintLength{#1}.*
2391     #2.#1.%
2392 }%
2393 \def\XINT_pfloat_a #1.#2#3.%
2394 {%
2395     \expandafter\XINT_pfloat_b\the\numexpr#1+#2#3-\xint_c_i.%
2396     #2#1.%
2397 }%
2398 \def\XINT_pfloat_b #1.#2%
2399 {%
2400     \ifnum #1>\xint_c_v \xint_dothis\XINT_pfloat_sci\fi
2401     \ifnum #1<-\xint_c_iv \xint_dothis\XINT_pfloat_sci\fi
2402     \ifnum #1<\xint_c_ \xint_dothis\XINT_pfloat_N\fi
2403     \if-#2\xint_dothis\XINT_pfloat_P\fi
2404     \xint_orthat\XINT_pfloat_Ps
2405     #1.%
2406 }%

    #1 is the scientific exponent, #2 is the length of trimmed mantissa.
    \xintPFloatE can be replaced by any f-expandable macro with a dot-delimited argument.

2407 \def\XINT_pfloat_sci #1.#2.%
2408 {%
2409     \ifnum#2=\xint_c_i\expandafter\XINT_pfloat_sci_i\expandafter\fi
2410     \expandafter\XINT_pfloat_sci_a\romannumeral`&&\xintPFloatE #1.%
2411 }%
2412 \def\XINT_pfloat_sci_a #1.#2#3.{ #2.#3#1}%

    #1#2=\fi\XINT_pfloat_sci_a
    1-digit mantissa, hesitation between d.0eK or deK

2413 \edef\XINT_pfloat_sci_i #1#2#3.#4.{#1\space#4.0#3}%
2414 \def\xintPFloatE{e}%
2415 \def\XINT_pfloat_N#1.#2.#3.%
2416 {%
2417     \csname XINT_pfloat_N_\romannumeral-#1\endcsname #3%
2418 }%
2419 \def\XINT_pfloat_N_i { 0.}%
2420 \def\XINT_pfloat_N_ii { 0.0}%
2421 \def\XINT_pfloat_N_iii{ 0.00}%
2422 \def\XINT_pfloat_N_iv { 0.000}%
2423 \def\XINT_pfloat_P #1.#2.#3.%

```

```

2424 {%
2425   \csname XINT_pfloat_P_\romannumeral#1\endcsname #3%
2426 }%
2427 \def\XINT_pfloat_P_ #1{ #1.}%
2428 \def\XINT_pfloat_P_i #1#2{ #1#2.}%
2429 \def\XINT_pfloat_P_ii #1#2#3{ #1#2#3.}%
2430 \def\XINT_pfloat_P_iii#1#2#3#4{ #1#2#3#4.}%
2431 \def\XINT_pfloat_P_iv #1#2#3#4#5{ #1#2#3#4#5.}%
2432 \def\XINT_pfloat_P_v #1#2#3#4#5#6{ #1#2#3#4#5#6.}%
2433 \def\XINT_pfloat_Ps #1.#2.#3.%
2434 {%
2435   \csname XINT_pfloat_Ps_\romannumeral#1\endcsname #300000.%
2436 }%
2437 \def\XINT_pfloat_Ps_ #1#2.{ #1.0}%
2438 \def\XINT_pfloat_Ps_i #1#2#3.{ #1#2.0}%
2439 \def\XINT_pfloat_Ps_ii #1#2#3#4.{ #1#2#3.0}%
2440 \def\XINT_pfloat_Ps_iii#1#2#3#4#5.{ #1#2#3#4.0}%
2441 \def\XINT_pfloat_Ps_iv #1#2#3#4#5#6.{ #1#2#3#4#5.0}%
2442 \def\XINT_pfloat_Ps_v #1#2#3#4#5#6#7.{ #1#2#3#4#5#6.0}%

```

8.76 \XINTinFloatFrac

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 context and issues with macro names.

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.

1.4e renames it back to \XINTinFloatFrac because of all such similarly named macros also using \XINTdigits forcedly.

```

2443 \def\XINTinFloatFrac {\romannumeral0\XINTinfloatfrac}%
2444 \def\XINTinfloatfrac #1%
2445 {%
2446   \expandafter\XINT_infloatfrac_a\expandafter {\romannumeral0\xinttfrac{#1}}%
2447 }%
2448 \def\XINT_infloatfrac_a {\XINTinfloat[\XINTdigits]}%

```

8.77 \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.

See general introduction for important changes at 1.4e relative to the \XINTinFloat<name> macros.

```

2449 \def\xintFloatAdd {\romannumeral0\xintfloatadd}%
2450 \def\xintfloatadd #1{\XINT_fladd_chkopt \xintfloat #1\xint:}%
2451 \def\XINTinFloatAdd{\romannumeral0\XINTinfloatadd}%

```

```

2452 \def\XINTinfloatadd{\XINT_fladd_opt_a\XINTdigits.\XINTinfloatS}%
2453 \def\XINTinFloatAdd_wopt{\romannumeral0\XINTinfloatadd_wopt}%
2454 \def\XINTinfloatadd_wopt[#1]{\expandafter\XINT_fladd_opt_a\the\numexpr#1.\XINTinfloatS}%
2455 \def\XINT_fladd_chkopt #1#2%
2456 {%
2457   \ifx [#2\expandafter\XINT_fladd_opt
2458     \else\expandafter\XINT_fladd_noopt
2459   \fi #1#2%
2460 }%
2461 \def\XINT_fladd_noopt #1#2\xint:#3%
2462 {%
2463   #1[\XINTdigits]%
2464   {\expandafter\XINT_FL_add_a
2465     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{#3}}%
2466 }%
2467 \def\XINT_fladd_opt #1[\xint:#2]%#3#4%
2468 {%
2469   \expandafter\XINT_fladd_opt_a\the\numexpr #2.#1%
2470 }%
2471 \def\XINT_fladd_opt_a #1.#2#3#4%
2472 {%
2473   #2[#1]{\expandafter\XINT_FL_add_a\romannumeral0\XINTinfloat[#1]{#3}#1.{#4}}%
2474 }%
2475 \def\XINT_FL_add_a #1%
2476 {%
2477   \xint_gob_til_zero #1\XINT_FL_add_zero 0\XINT_FL_add_b #1%
2478 }%

2479 \def\XINT_FL_add_zero #1.#2{#2}%[
2480 \def\XINT_FL_add_b #1]#2.#3%
2481 {%
2482   \expandafter\XINT_FL_add_c\romannumeral0\XINTinfloat[#2]{#3}#2.#1}%
2483 }%

2484 \def\XINT_FL_add_c #1%
2485 {%
2486   \xint_gob_til_zero #1\XINT_FL_add_zero 0\XINT_FL_add_d #1%
2487 }%

2488 \def\XINT_FL_add_d #1[#2]#3.#4[#5]%
2489 {%
2490   \ifnum\numexpr #2-#3-#5>\xint_c_\xint_dothis\xint_firstoftwo\fi
2491   \ifnum\numexpr #5-#3-#2>\xint_c_\xint_dothis\xint_secondoftwo\fi
2492   \xint_orthat\xintAdd {#1[#2]}{#4[#5]}%
2493 }%

```

8.78 \xintFloatSub, \XINTinFloatSub

First done 1.07.

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

```

2494 \def\xintFloatSub {\romannumeral0\xintfloatsub}%
2495 \def\xintfloatsub #1{\XINT_flsub_chkopt \xintfloat #1\xint:}%
2496 \def\XINTinFloatSub{\romannumeral0\XINTinfloatsub}%

```

```

2497 \def\XINTinfloatsub{\XINT_fsub_opt_a\XINTdigits.\XINTinfloatS}%
2498 \def\XINTinFloatSub_wopt{\romannumeral0\XINTinfloatsub_wopt}%
2499 \def\XINTinfloatsub_wopt[#1]{\expandafter\XINT_fsub_opt_a\the\numexpr#1.\XINTinfloatS}%
2500 \def\XINT_fsub_chkopt #1#2%
2501 {%
2502     \ifx [#2\expandafter\XINT_fsub_opt
2503         \else\expandafter\XINT_fsub_noopt
2504     \fi #1#2%
2505 }%
2506 \def\XINT_fsub_noopt #1#2\xint:#3%
2507 {%
2508     #1[\XINTdigits]%
2509     {\expandafter\XINT_FL_add_a
2510         \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{\xintOpp{#3}}}%
2511 }%
2512 \def\XINT_fsub_opt #1[\xint:#2]%#3#4%
2513 {%
2514     \expandafter\XINT_fsub_opt_a\the\numexpr #2.#1%
2515 }%
2516 \def\XINT_fsub_opt_a #1.#2#3#4%
2517 {%
2518     #2[#1]{\expandafter\XINT_FL_add_a\romannumeral0\XINTinfloat[#1]{#3}#1.{\xintOpp{#4}}}%
2519 }%

```

8.79 \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).

```

2520 \def\xintFloatMul {\romannumeral0\xintfloatmul}%
2521 \def\xintfloatmul #1{\XINT_fmul_chkopt \xintfloat #1\xint:%}
2522 \def\XINTinFloatMul{\romannumeral0\XINTinfloatmul}%
2523 \def\XINTinfloatmul{\XINT_fmul_opt_a\XINTdigits.\XINTinfloatS}%
2524 \def\XINTinFloatMul_wopt{\romannumeral0\XINTinfloatmul_wopt}%
2525 \def\XINTinfloatmul_wopt[#1]{\expandafter\XINT_fmul_opt_a\the\numexpr#1.\XINTinfloatS}%
2526 \def\XINT_fmul_chkopt #1#2%
2527 {%
2528     \ifx [#2\expandafter\XINT_fmul_opt
2529         \else\expandafter\XINT_fmul_noopt
2530     \fi #1#2%
2531 }%
2532 \def\XINT_fmul_noopt #1#2\xint:#3%
2533 {%
2534     #1[\XINTdigits]%
2535     {\expandafter\XINT_FL_mul_a
2536         \romannumeral0\XINTinfloatS[\XINTdigits]{#2}\XINTdigits.{#3}}%
2537 }%
2538 \def\XINT_fmul_opt #1[\xint:#2]%#3#4%

```

```

2539 {%
2540     \expandafter\XINT_flmul_opt_a\the\numexpr #2.#1%
2541 }%
2542 \def\XINT_flmul_opt_a #1.#2#3#4%
2543 {%
2544     #2[#1]{\expandafter\XINT_FL_mul_a\romannumeral0\XINTinfloatS[#1]{#3}#1.{#4}}%
2545 }%
2546 \def\XINT_FL_mul_a #1[#2]#3.#4%
2547 {%
2548     \expandafter\XINT_FL_mul_b\romannumeral0\XINTinfloatS[#3]{#4}#1[#2]%
2549 }%

2550 \def\XINT_FL_mul_b #1[#2]#3[#4]{\xintiiMul{#3}{#1}/1[#4+#2]}%

```

8.80 \xintFloatSqr, \XINTinFloatSqr

Added only at 1.4e, strangely \xintFloatSqr had never been defined so far.

An \XINTinFloatSqr{#1} was defined in xintexpr.sty directly as \XINTinFloatMul[\XINTdigits]{#1}{#1}, to support the sqr() function. The {#1}{#1} causes no problem as #1 in this context is always pre-expanded so we don't need to worry about this, and the \xintdeffloatfunc mechanism should hopefully take care to add the needed argument pre-expansion if need be.

Anyway let's do this finally properly here.

```

2551 \def\xintFloatSqr {\romannumeral0\xintfloatsqr}%
2552 \def\xintfloatsqr #1{\XINT_fl_sqr_chkopt \xintfloat #1\xint:}%
2553 \def\XINTinFloatSqr{\romannumeral0\XINTinfloatsqr}%
2554 \def\XINTinfloatsqr{\XINT_fl_sqr_opt_a\XINTdigits.\XINTinfloatS}%
2555 \def\XINT_fl_sqr_chkopt #1#2%
2556 {%
2557     \ifx [#2\expandafter\XINT_fl_sqr_opt
2558     \else\expandafter\XINT_fl_sqr_noopt
2559     \fi #1#2%
2560 }%
2561 \def\XINT_fl_sqr_noopt #1#2\xint:
2562 {%
2563     #1[\XINTdigits]%
2564     {\expandafter\XINT_FL_sqr_a\romannumeral0\XINTinfloatS[\XINTdigits]{#2}}%
2565 }%
2566 \def\XINT_fl_sqr_opt #1[\xint:#2]%
2567 {%
2568     \expandafter\XINT_fl_sqr_opt_a\the\numexpr #2.#1%
2569 }%
2570 \def\XINT_fl_sqr_opt_a #1.#2#3%
2571 {%
2572     #2[#1]{\expandafter\XINT_FL_sqr_a\romannumeral0\XINTinfloatS[#1]{#3}}%
2573 }%
2574 \def\XINT_FL_sqr_a #1[#2]{\xintiiSqr{#1}/1[#2+#2]}%
2575 \def\XINTinFloatSqr_wopt[#1]#2{\XINTinFloatS[#1]{\expandafter\XINT_FL_sqr_a\romannumeral0\XINTinfloatS[

```

8.81 \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.

```
2576 \def\XINTinFloatInv#1{\XINTinFloatS[\XINTdigits]{\xintInv{#1}}}%
2577 \def\XINTinFloatInv_wopt[#1]#2{\XINTinFloatS[#1]{\xintInv{#2}}}%
```

8.82 \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.

```
2578 \def\xintFloatDiv {\romannumeral0\xintfloatdiv}%
2579 \def\xintfloatdiv #1{\XINT_fldiv_chkopt \xintfloat #1\xint:}%
2580 \def\XINTinFloatDiv{\romannumeral0\XINTinfloatdiv}%
2581 \def\XINTinfloatdiv{\XINT_fldiv_opt_a\XINTdigits.\XINTinfloatS}%
2582 \def\XINTinFloatDiv_wopt[#1]{\romannumeral0\XINT_fldiv_opt_a#1.\XINTinfloatS}%
2583 \def\XINT_fldiv_chkopt #1#2%
2584 {%
2585   \ifx [#2\expandafter\XINT_fldiv_opt
2586     \else\expandafter\XINT_fldiv_noopt
2587   \fi #1#2%
2588 }%
```

1.4g adds here intercept of second argument being zero, else a low level error will arise at later stage from the the fall-back value returned by core iidivision being 0 and not having expected number of digits at \XINT_infloat_Qq and split from left returning some empty value breaking the \ifnum test in \XINT_infloat_Rq.

```
2589 \def\XINT_fldiv_noopt #1#2\xint:#3%
2590 {%
2591   #1[\XINTdigits]%
2592   {\expandafter\XINT_FL_div_aa
2593     \romannumeral0\XINTinfloatS[\XINTdigits]{#3}\XINTdigits.{#2}}%
2594 }%
2595 \def\XINT_FL_div_aa #1%
2596 {%
2597   \xint_gob_til_zero#1\XINT_FL_div_Bzero0\XINT_FL_div_a #1%
2598 }%
2599 \def\XINT_FL_div_Bzero0\XINT_FL_div_a#1[#2]#3.#4%
2600 {%
2601   \XINT_signalcondition{DivisionByZero}{Division by zero (#1[#2]) of #4}{\{ 0[0]\}}%
2602 }%
2603 \def\XINT_fldiv_opt #1[\xint:#2]#3#4%
2604 {%
2605   \expandafter\XINT_fldiv_opt_a\the\numexpr #2.#1%
2606 }%
```

Also here added early check at 1.4g if divisor is zero.

```
2607 \def\XINT_fldiv_opt_a #1.#2#3#4%
2608 {%
2609   #2[#1]{\expandafter\XINT_FL_div_aa\romannumeral0\XINTinfloatS[#1]{#4}#1.{#3}}%
2610 }%
2611 \def\XINT_FL_div_a #1[#2]#3.#4%
```

```

2612 {%
2613     \expandafter\XINT_FL_div_b\romannumeral0\XINTinfloatS[#3]{#4}/#1e#2%
2614 }%

2615 \def\XINT_FL_div_b #1[#2]{#1e#2}%

```

8.83 \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).

```

2616 \def\xintFloatPow {\romannumeral0\xintfloatpow}%
2617 \def\xintfloatpow #1{\XINT_flpow_chkopt \xintfloat #1\xint:}%
2618 \def\XINTinFloatPow{\romannumeral0\XINTinfloatpow}%
2619 \def\XINTinfloatpow{\XINT_flpow_opt_a\XINTdigits.\XINTinfloatS}%
2620 \def\XINTinfloatpow_wopt[#1]{\expandafter\XINT_flpow_opt_a\the\numexpr#1.\XINTinfloatS}%
2621 \def\XINT_flpow_chkopt #1#2%
2622 {%
2623     \ifx [#2\expandafter\XINT_flpow_opt
2624     \else\expandafter\XINT_flpow_noopt
2625     \fi
2626     #1#2%
2627 }%
2628 \def\XINT_flpow_noopt #1#2\xint:#3%
2629 {%
2630     \expandafter\XINT_flpow_checkB_a
2631     \the\numexpr #3.\XINTdigits.{#2}{#1[\XINTdigits]]}%
2632 }%
2633 \def\XINT_flpow_opt #1[\xint:#2]%
2634 {%
2635     \expandafter\XINT_flpow_opt_a\the\numexpr #2.#1%
2636 }%
2637 \def\XINT_flpow_opt_a #1.#2#3#4%
2638 {%
2639     \expandafter\XINT_flpow_checkB_a\the\numexpr #4.#1.{#3}{#2[#1]}%
2640 }%
2641 \def\XINT_flpow_checkB_a #1%
2642 {%
2643     \xint_UDzerominusfork
2644     #1-\XINT_flpow_BisZero
2645     0#1{\XINT_flpow_checkB_b -}%
2646     0-{\XINT_flpow_checkB_b }{#1}%
2647     \krof
2648 }%
2649 \def\XINT_flpow_BisZero .#1.#2#3{#3{1[0]}}%

2650 \def\XINT_flpow_checkB_b #1#2.#3.%
2651 {%
2652     \expandafter\XINT_flpow_checkB_c

```

```

2653 \the\numexpr\xintLength{#2}+\xint_c_iii.#3.#2.{#1}%
2654 }%
2655 \def\XINT_flpow_checkB_c #1.#2.%
2656 {%
2657 \expandafter\XINT_flpow_checkB_d\the\numexpr#1+#2.#1.#2.%
2658 }%

1.2f rounds input to P digits, first.
2659 \def\XINT_flpow_checkB_d #1.#2.#3.#4.#5#6%
2660 {%
2661 \expandafter \XINT_flpow_aa
2662 \romannumeral0\XINTinfloat [#3]{#6}{#2}{#1}{#4}{#5}%
2663 }%

2664 \def\XINT_flpow_aa #1[#2]#3%
2665 {%
2666 \expandafter\XINT_flpow_ab\the\numexpr #2-#3\expandafter.%
2667 \romannumeral\XINT_rep #3\endcsname0.#1.%
2668 }%

2669 \def\XINT_flpow_ab #1.#2.#3.{\XINT_flpow_a #3#2[#1]}%

2670 \def\XINT_flpow_a #1%
2671 {%
2672 \xint_UDzerominusfork
2673 #1-\XINT_flpow_zero
2674 0#1{\XINT_flpow_b \iftrue}%
2675 0-{\XINT_flpow_b \iffalse#1}%
2676 \krof
2677 }%
2678 \def\XINT_flpow_zero #1[#2]#3#4#5#6%
2679 {%
2680 #6{\if 1#51\xint_dothis {0[0]}\fi
2681 \xint_orthat
2682 {\XINT_signalcondition{DivisionByZero}{0 raised to power -#4.}}{0[0]}}%
2683 }%
2684 }%

2685 \def\XINT_flpow_b #1#2[#3]#4#5%
2686 {%
2687 \XINT_flpow_loopI #5.#3.#2.#4.{#1\ifodd #5 \xint_c_i\fi\fi}%
2688 }%

2689 \def\XINT_flpow_truncate #1.#2.#3.%
2690 {%
2691 \expandafter\XINT_flpow_truncate_a
2692 \romannumeral0\XINT_split_fromleft
2693 #3.#2\xint_bye2345678\xint_bye..#1.#3.%
2694 }%

2695 \def\XINT_flpow_truncate_a #1.#2.#3.{#3+\xintLength{#2}.#1.%
2696 \def\XINT_flpow_loopI #1.%
2697 {%
2698 \ifnum #1=\xint_c_i\expandafter\XINT_flpow_ItoIII\fi

```



```

2699 \ifodd #1
2700 \expandafter\XINT_flpow_loopI_odd
2701 \else
2702 \expandafter\XINT_flpow_loopI_even
2703 \fi
2704 #1.%
2705 }%

2706 \def\XINT_flpow_ItoIII\ifodd #1\fi #2.#3.#4.#5.#6%
2707 {%
2708 \expandafter\XINT_flpow_III\the\numexpr #6+\xint_c_.#3.#4.#5.%
2709 }%

2710 \def\XINT_flpow_loopI_even #1.#2.#3.%#4.%
2711 {%
2712 \expandafter\XINT_flpow_loopI
2713 \the\numexpr #1/\xint_c_ii\expandafter.%
2714 \the\numexpr\expandafter\XINT_flpow_truncate
2715 \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.%
2716 }%
2717 \def\XINT_flpow_loopI_odd #1.#2.#3.#4.%
2718 {%
2719 \expandafter\XINT_flpow_loopII
2720 \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter.%
2721 \the\numexpr\expandafter\XINT_flpow_truncate
2722 \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#2.#3.%
2723 }%
2724 \def\XINT_flpow_loopII #1.%
2725 {%
2726 \ifnum #1 = \xint_c_i\expandafter\XINT_flpow_ItoIII\fi
2727 \ifodd #1
2728 \expandafter\XINT_flpow_loopII_odd
2729 \else
2730 \expandafter\XINT_flpow_loopII_even
2731 \fi
2732 #1.%
2733 }%
2734 \def\XINT_flpow_loopII_even #1.#2.#3.%#4.%
2735 {%
2736 \expandafter\XINT_flpow_loopII
2737 \the\numexpr #1/\xint_c_ii\expandafter.%
2738 \the\numexpr\expandafter\XINT_flpow_truncate
2739 \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.%
2740 }%
2741 \def\XINT_flpow_loopII_odd #1.#2.#3.#4.#5.#6.%
2742 {%
2743 \expandafter\XINT_flpow_loopII_odda
2744 \the\numexpr\expandafter\XINT_flpow_truncate
2745 \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2746 #1.#2.#3.%
2747 }%
2748 \def\XINT_flpow_loopII_odda #1.#2.#3.#4.#5.#6.%
2749 {%

```

```

2750 \expandafter\XINT_flpow_loopII
2751 \the\numexpr #4/\xint_c_ii-\xint_c_i\expandafter.%
2752 \the\numexpr\expandafter\XINT_flpow_truncate
2753 \the\numexpr\xint_c_ii*#5\expandafter.\romannumeral0\xintiisqr{#6}.#3.%
2754 #1.#2.%
2755 }%

2756 \def\XINT_flpow_IItoIII\ifodd #1\fi #2.#3.#4.#5.#6.#7.#8%
2757 {%
2758 \expandafter\XINT_flpow_III\the\numexpr #8+\xint_c_\expandafter.%
2759 \the\numexpr\expandafter\XINT_flpow_truncate
2760 \the\numexpr#3+#6\expandafter.\romannumeral0\xintiimul{#4}{#7}.#5.%
2761 }%

```

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).

```

2762 \def\XINT_flpow_III #1.#2.#3.#4.#5%
2763 {%
2764 \expandafter\XINT_flpow_IIIend
2765 \xint_UDsignfork
2766 #5{{1/#3[-#2]}}%
2767 -{{#3[#2]}}%
2768 \krof #1%
2769 }%

2770 \def\XINT_flpow_IIIend #1#2#3%
2771 {#3{\if#21\xint_afterfi{\expandafter-\romannumeral`&&@\fi#1}}%

```

8.84 `\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 `\XINTinFloatPowerH` (now moved to [xintlog](#), and renamed). It truncated the exponent to an integer of half-integer, and in the latter case use Square-root extraction. At 1.2k this was improved as 1.2f stupidly rounded to Digits before, not after the square root extraction, 1.2k kept 3 guard digits for this last step. And the initial step was changed to a rounding rather than truncating.

Until 1.4e this `\XINTinFloatPowerH` was the macro for a^b in expressions, but of course it behaved strangely for b not an integer or an half-integer! At 1.4e, the non-integer, non-half-integer exponents will be handled via `log10()` and `pow10()` support macros, see [xintlog](#). The code has now been relocated there.

```

2772 \def\xintFloatPower {\romannumeral0\xintfloatpower}%
2773 \def\xintfloatpower #1{\XINT_flpower_chkopt \xintfloat #1\xint:}%
2774 \def\XINTinFloatPower{\romannumeral0\XINTinfloatpower }%
2775 \def\XINTinfloatpower{\XINT_flpower_opt_a\XINTdigits.\XINTinfloatS}%

    Start of macro. Check for optional argument.

2776 \def\XINT_flpower_chkopt #1#2%
2777 {%
2778     \ifx [#2\expandafter\XINT_flpower_opt
2779         \else\expandafter\XINT_flpower_noopt
2780         \fi
2781     #1#2%
2782 }%
2783 \def\XINT_flpower_noopt #1#2\xint:#3%
2784 {%
2785     \expandafter\XINT_flpower_checkB_a
2786     \romannumeral0\xintnum{#3}.\XINTdigits.{#2}{#1[\XINTdigits]}%
2787 }%
2788 \def\XINT_flpower_opt #1[\xint:#2]%
2789 {%
2790     \expandafter\XINT_flpower_opt_a\the\numexpr #2.#1%
2791 }%
2792 \def\XINT_flpower_opt_a #1.#2#3#4%
2793 {%
2794     \expandafter\XINT_flpower_checkB_a
2795     \romannumeral0\xintnum{#4}.#1.{#3}{#2[#1]}%
2796 }%
2797 \def\XINT_flpower_checkB_a #1%
2798 {%
2799     \xint_UDzerominusfork
2800     #1-{\XINT_flpower_BisZero 0}%
2801     0#1{\XINT_flpower_checkB_b -}%
2802     0-{\XINT_flpower_checkB_b }{#1}%
2803     \krof
2804 }%
2805 \def\XINT_flpower_BisZero 0.#1.#2#3{#3{1[0]}}%
2806 \def\XINT_flpower_checkB_b #1#2.#3.%
2807 {%
2808     \expandafter\XINT_flpower_checkB_c
2809     \the\numexpr\xintLength{#2}+\xint_c_iii.#3.#2.{#1}%
2810 }%

2811 \def\XINT_flpower_checkB_c #1.#2.%
2812 {%
2813     \expandafter\XINT_flpower_checkB_d\the\numexpr#1+#2.#1.#2.%
2814 }%

2815 \def\XINT_flpower_checkB_d #1.#2.#3.#4.#5#6%
2816 {%
2817     \expandafter \XINT_flpower_aa
2818     \romannumeral0\XINTinfloat [#3]{#6}{#2}{#1}{#4}{#5}%
2819 }%

2820 \def\XINT_flpower_aa #1[#2]#3%

```

```

2821 {%
2822   \expandafter\XINT_flpower_ab\the\numexpr #2-#3\expandafter.%
2823   \romannumeral\XINT_rep #3\endcsname0.#1.%
2824 }%
2825 \def\XINT_flpower_ab #1.#2.#3.{\XINT_flpower_a #3#2[#1]}%
2826 \def\XINT_flpower_a #1%
2827 {%
2828   \xint_UDzerominusfork
2829   #1-\XINT_flpow_zero
2830   0#1{\XINT_flpower_b \iftrue}%
2831   0-{\XINT_flpower_b \iffalse#1}%
2832   \krof
2833 }%
2834 \def\XINT_flpower_b #1#2[#3]#4#5%
2835 {%
2836   \XINT_flpower_loopI #5.#3.#2.#4.{#1\xintiOdd{#5}\fi}%
2837 }%
2838 \def\XINT_flpower_loopI #1.%
2839 {%
2840   \if1\XINT_isOne {#1}\xint_dothis\XINT_flpower_ItoIII\fi
2841   \ifodd\xintLDg{#1} %<- intentional space
2842   \xint_dothis{\expandafter\XINT_flpower_loopI_odd}\fi
2843   \xint_orthat{\expandafter\XINT_flpower_loopI_even}%
2844   \romannumeral0\XINT_half
2845   #1\xint_bye\xint_Bye345678\xint_bye
2846   *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax.%
2847 }%
2848 \def\XINT_flpower_ItoIII #1.#2.#3.#4.#5%
2849 {%
2850   \expandafter\XINT_flpow_III\the\numexpr #5+\xint_c_.#2.#3.#4.%
2851 }%
2852 \def\XINT_flpower_loopI_even #1.#2.#3.#4.%
2853 {%
2854   \expandafter\XINT_flpower_toloopI
2855   \the\numexpr\expandafter\XINT_flpow_truncate
2856   \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#1.%
2857 }%
2858 \def\XINT_flpower_toloopI #1.#2.#3.#4.{\XINT_flpower_loopI #4.#1.#2.#3.}%
2859 \def\XINT_flpower_loopI_odd #1.#2.#3.#4.%
2860 {%
2861   \expandafter\XINT_flpower_toloopII
2862   \the\numexpr\expandafter\XINT_flpow_truncate
2863   \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.%
2864   #1.#2.#3.%
2865 }%
2866 \def\XINT_flpower_toloopII #1.#2.#3.#4.{\XINT_flpower_loopII #4.#1.#2.#3.}%
2867 \def\XINT_flpower_loopII #1.%
2868 {%
2869   \if1\XINT_isOne{#1}\xint_dothis\XINT_flpower_ItoIII\fi
2870   \ifodd\xintLDg{#1} %<- intentional space
2871   \xint_dothis{\expandafter\XINT_flpower_loopII_odd}\fi
2872   \xint_orthat{\expandafter\XINT_flpower_loopII_even}%

```

```

2873 \romannumeral0\XINT_half#1\xint_bye\xint_Bye345678\xint_bye
2874 *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax.%
2875 }%
2876 \def\XINT_flpower_loopII_even #1.#2.#3.#4.%
2877 {%
2878 \expandafter\XINT_flpower_toloopII
2879 \the\numexpr\expandafter\XINT_flpow_truncate
2880 \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#1.%
2881 }%
2882 \def\XINT_flpower_loopII_odd #1.#2.#3.#4.#5.#6.%
2883 {%
2884 \expandafter\XINT_flpower_loopII_odda
2885 \the\numexpr\expandafter\XINT_flpow_truncate
2886 \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2887 #1.#2.#3.%
2888 }%
2889 \def\XINT_flpower_loopII_odda #1.#2.#3.#4.#5.#6.%
2890 {%
2891 \expandafter\XINT_flpower_toloopII
2892 \the\numexpr\expandafter\XINT_flpow_truncate
2893 \the\numexpr\xint_c_ii*#5\expandafter.\romannumeral0\xintiisqr{#6}.#3.%
2894 #4.#1.#2.%
2895 }%
2896 \def\XINT_flpower_IItoIII #1.#2.#3.#4.#5.#6.#7%
2897 {%
2898 \expandafter\XINT_flpow_III\the\numexpr #7+\xint_c_\expandafter.%
2899 \the\numexpr\expandafter\XINT_flpow_truncate
2900 \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2901 }%

```

8.85 \xintFloatFac, \XINTFloatFac

Done at 1.2.

At 1.3e \XINTinFloatFac uses \XINTinFloatS for output.

1.4e adds some overhead for individual evaluations in float context as it obeys the guard digits for the default target precision. It is a waste for individual evaluation of one factorial...

```

2902 \def\xintFloatFac {\romannumeral0\xintfloatfac}%
2903 \def\xintfloatfac #1{\XINT_flfac_chkopt \xintfloat #1\xint:}%
2904 \def\XINTinFloatFac{\romannumeral0\XINTinfloatfac}%
2905 \def\XINTinfloatfac[#1]{\expandafter\XINT_flfac_opt_a\the\numexpr#1.\XINTinfloatS}%
2906 \def\XINTinFloatFacdigits{\romannumeral0\XINT_flfac_opt_a\XINTdigits.\XINTinfloatS}%
2907 \def\XINT_flfac_chkopt #1#2%
2908 {%
2909 \ifx [#2\expandafter\XINT_flfac_opt
2910 \else\expandafter\XINT_flfac_noopt
2911 \fi
2912 #1#2%
2913 }%
2914 \def\XINT_flfac_noopt #1#2\xint:
2915 {%
2916 \expandafter\XINT_FL_fac_fork_a
2917 \the\numexpr \xintNum{#2}.\xint_c_i \XINTdigits\XINT_FL_fac_out{#1[\XINTdigits]]}%

```

```

2918 }%
2919 \def\XINT_flgfac_opt #1[\xint:#2]%
2920 {%
2921   \expandafter\XINT_flgfac_opt_a\the\numexpr #2.#1%
2922 }%
2923 \def\XINT_flgfac_opt_a #1.#2#3%
2924 {%
2925   \expandafter\XINT_FL_fac_fork_a\the\numexpr \xintNum{#3}.\xint_c_i {#1}\XINT_FL_fac_out{#2[#1]}%
2926 }%
2927 \def\XINT_FL_fac_fork_a #1%
2928 {%
2929   \xint_UDzerominusfork
2930   #1-\XINT_FL_fac_iszero
2931   0#1\XINT_FL_fac_isneg
2932   0-\XINT_FL_fac_fork_b #1}%
2933 \krof
2934 }%
2935 \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.

```

2936 \def\XINT_FL_fac_isneg #1.#2#3#4#5%
2937 {%
2938   #5{\XINT_signalcondition{InvalidOperation}
2939     {Factorial argument is negative: -#1.}{\ 0[0]}}%
2940 }%
2941 \def\XINT_FL_fac_fork_b #1.%
2942 {%
2943   \ifnum #1>\xint_c_x^viii_mone\xint_dothis\XINT_FL_fac_toobig\fi
2944   \ifnum #1>\xint_c_x^iv\xint_dothis\XINT_FL_fac_vbig \fi
2945   \ifnum #1>465 \xint_dothis\XINT_FL_fac_big\fi
2946   \ifnum #1>101 \xint_dothis\XINT_FL_fac_med\fi
2947   \xint_orthat\XINT_FL_fac_small
2948   #1.%
2949 }%
2950 \def\XINT_FL_fac_toobig #1.#2#3#4#5%
2951 {%
2952   #5{\XINT_signalcondition{InvalidOperation}
2953     {Factorial argument is too large: #1>=10^8.}{\ 0[0]}}%
2954 }%

```

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

We then want $\text{ceil}((P+k+n)/8)$. Using \numexpr rounding division (ARRRRRGGGGHHHH), 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 $Q \leq (P+13+k)/8$.

The routine actually computes $8(Q-1)$ for use in `\XINT_FL_fac_addzeros`.

With 1.2f the formula is $m=P+9+k$, $Q<-(P+12+k)/8$, and we use now $4=12-8$ rather than the earlier $5=13-8$. Whatever happens, the value computed in `\XINT_FL_fac_increaseP` is at least 8. There will always be an extra block.

Note: with `Digits:=32`; Maple gives for $200!$:

```
> factorial(200.);
```

```
375
```

```
0.78865786736479050355236321393218 10
```

My 1.2f routine (and also 1.2) outputs:

```
7.8865786736479050355236321393219e374
```

and this is the correct rounding because for 40 digits it computes

```
7.886578673647905035523632139321850622951e374
```

Maple's result (contrarily to xint) is thus not the correct rounding but still it is less than 0.6 ulp wrong.

```
2955 \def\XINT_FL_fac_vbig
2956   {\expandafter\XINT_FL_fac_vbigloop_a
2957     \the\numexpr \XINT_FL_fac_increaseP \xint_c_i   }%
2958 \def\XINT_FL_fac_big
2959   {\expandafter\XINT_FL_fac_bigloop_a
2960     \the\numexpr \XINT_FL_fac_increaseP \xint_c_ii  }%
2961 \def\XINT_FL_fac_med
2962   {\expandafter\XINT_FL_fac_medloop_a
2963     \the\numexpr \XINT_FL_fac_increaseP \xint_c_iii }%
2964 \def\XINT_FL_fac_small
2965   {\expandafter\XINT_FL_fac_smallloop_a
2966     \the\numexpr \XINT_FL_fac_increaseP \xint_c_iv   }%
2967 \def\XINT_FL_fac_increaseP #1#2.#3#4%
2968 {%
2969   #2\expandafter.\the\numexpr\xint_c_viii*%
2970   ((\xint_c_iv+#4+\expandafter\XINT_FL_fac_countdigits
2971     \the\numexpr #2/(#1*#3)\relax 87654321\Z)/\xint_c_viii).%
2972 }%
2973 \def\XINT_FL_fac_countdigits #1#2#3#4#5#6#7#8{\XINT_FL_fac_countdone }%
2974 \def\XINT_FL_fac_countdone  #1#2\Z {#1}%
2975 \def\XINT_FL_fac_out #1;![#2]#3%
2976   {#3{\romannumeral0\XINT_mul_out
2977     #1;!1\R!1\R!1\R!1\R!1\R!%
2978     1\R!1\R!1\R!1\R!1\R!\W [#2]}}%
2979 \def\XINT_FL_fac_vbigloop_a #1.#2.%
2980 {%
2981   \XINT_FL_fac_bigloop_a \xint_c_x^iv.#2.%
2982   {\expandafter\XINT_FL_fac_vbigloop_loop\the\numexpr 100010001\expandafter.%
2983     \the\numexpr \xint_c_x^viii+#1.}%
2984 }%
2985 \def\XINT_FL_fac_vbigloop_loop #1.#2.%
2986 {%
2987   \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2988   \expandafter\XINT_FL_fac_vbigloop_loop
2989   \the\numexpr #1+\xint_c_i\expandafter.%
2990   \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_mul #1!%
2991 }%
2992 \def\XINT_FL_fac_bigloop_a #1.%
```

```

2993 {%
2994     \expandafter\XINT_FL_fac_bigloop_b \the\numexpr
2995     #1+\xint_c_i-\xint_c_ii*((#1-464)/\xint_c_ii).#1.%
2996 }%
2997 \def\XINT_FL_fac_bigloop_b #1.#2.#3.%
2998 {%
2999     \expandafter\XINT_FL_fac_medloop_a
3000     \the\numexpr #1-\xint_c_i.#3.{\XINT_FL_fac_bigloop_loop #1.#2.}%
3001 }%
3002 \def\XINT_FL_fac_bigloop_loop #1.#2.%
3003 {%
3004     \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
3005     \expandafter\XINT_FL_fac_bigloop_loop
3006     \the\numexpr #1+\xint_c_ii\expandafter.%
3007     \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_bigloop_mul #1!%
3008 }%
3009 \def\XINT_FL_fac_bigloop_mul #1!%
3010 {%
3011     \expandafter\XINT_FL_fac_mul
3012     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
3013 }%
3014 \def\XINT_FL_fac_medloop_a #1.%
3015 {%
3016     \expandafter\XINT_FL_fac_medloop_b
3017     \the\numexpr #1+\xint_c_i-\xint_c_iii*((#1-100)/\xint_c_iii).#1.%
3018 }%
3019 \def\XINT_FL_fac_medloop_b #1.#2.#3.%
3020 {%
3021     \expandafter\XINT_FL_fac_smallloop_a
3022     \the\numexpr #1-\xint_c_i.#3.{\XINT_FL_fac_medloop_loop #1.#2.}%
3023 }%
3024 \def\XINT_FL_fac_medloop_loop #1.#2.%
3025 {%
3026     \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
3027     \expandafter\XINT_FL_fac_medloop_loop
3028     \the\numexpr #1+\xint_c_iii\expandafter.%
3029     \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_medloop_mul #1!%
3030 }%
3031 \def\XINT_FL_fac_medloop_mul #1!%
3032 {%
3033     \expandafter\XINT_FL_fac_mul
3034     \the\numexpr
3035     \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
3036 }%
3037 \def\XINT_FL_fac_smallloop_a #1.%
3038 {%
3039     \csname
3040         XINT_FL_fac_smallloop_\the\numexpr #1-\xint_c_iv*(#1/\xint_c_iv)\relax
3041     \endcsname #1.%
3042 }%
3043 \expandafter\def\csname XINT_FL_fac_smallloop_1\endcsname #1.#2.%
3044 {%

```



```

3045 \XINT_FL_fac_addzeros #2.1000000001!.{2.#1.}{#2}%
3046 }%
3047 \expandafter\def\csname XINT_FL_fac_smallloop_-2\endcsname #1.#2.%
3048 {%
3049 \XINT_FL_fac_addzeros #2.1000000002!.{3.#1.}{#2}%
3050 }%
3051 \expandafter\def\csname XINT_FL_fac_smallloop_-1\endcsname #1.#2.%
3052 {%
3053 \XINT_FL_fac_addzeros #2.1000000006!.{4.#1.}{#2}%
3054 }%
3055 \expandafter\def\csname XINT_FL_fac_smallloop_0\endcsname #1.#2.%
3056 {%
3057 \XINT_FL_fac_addzeros #2.1000000024!.{5.#1.}{#2}%
3058 }%
3059 \def\XINT_FL_fac_addzeros #1.%
3060 {%
3061 \ifnum #1=\xint_c_viii \expandafter\XINT_FL_fac_addzeros_exit\fi
3062 \expandafter\XINT_FL_fac_addzeros
3063 \the\numexpr #1-\xint_c_viii.1000000000!%
3064 }%

```

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.

```

3065 \def\XINT_FL_fac_addzeros_exit #1.#2.#3#4{\XINT_FL_fac_smallloop_loop #3#21;![ -#4]}%
3066 \def\XINT_FL_fac_smallloop_loop #1.#2.%
3067 {%
3068 \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
3069 \expandafter\XINT_FL_fac_smallloop_loop
3070 \the\numexpr #1+\xint_c_iv\expandafter.%
3071 \the\numexpr #2\expandafter.\romannumeral0\XINT_FL_fac_smallloop_mul #1!%
3072 }%
3073 \def\XINT_FL_fac_smallloop_mul #1!%
3074 {%
3075 \expandafter\XINT_FL_fac_mul
3076 \the\numexpr
3077 \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
3078 }%[[
3079 \def\XINT_FL_fac_loop_exit #1!#2]#3{#3#2]}%
3080 \def\XINT_FL_fac_mul 1#1!%
3081 {\expandafter\XINT_FL_fac_mul_a\the\numexpr\XINT_FL_fac_smallmul 10!{#1}}%
3082 \def\XINT_FL_fac_mul_a #1-#2%
3083 {%
3084 \if#21\xint_afterfi{\expandafter\space\xint_gob_til_exclam}\else
3085 \expandafter\space\fi #11;!%
3086 }%
3087 \def\XINT_FL_fac_minimulwc_a #1#2#3#4#5!#6#7#8#9%
3088 {%
3089 \XINT_FL_fac_minimulwc_b {#1#2#3#4}{#5}{#6#7#8#9}%
3090 }%
3091 \def\XINT_FL_fac_minimulwc_b #1#2#3#4!#5%
3092 {%
3093 \expandafter\XINT_FL_fac_minimulwc_c

```

```

3094 \the\numexpr \xint_c_x^ix+#5+#2*#4!{{#1}{#2}{#3}{#4}}%
3095 }%
3096 \def\XINT_FL_fac_minimulwc_c 1#1#2#3#4#5#6!#7%
3097 {%
3098 \expandafter\XINT_FL_fac_minimulwc_d {#1#2#3#4#5}#7{#6}%
3099 }%
3100 \def\XINT_FL_fac_minimulwc_d #1#2#3#4#5%
3101 {%
3102 \expandafter\XINT_FL_fac_minimulwc_e
3103 \the\numexpr \xint_c_x^ix+#1+#2*#5+#3*#4!{{#2}{#4}}%
3104 }%
3105 \def\XINT_FL_fac_minimulwc_e 1#1#2#3#4#5#6!#7#8#9%
3106 {%
3107 1#6#9\expandafter!%
3108 \the\numexpr\expandafter\XINT_FL_fac_smallmul
3109 \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#7*#8!%
3110 }%
3111 \def\XINT_FL_fac_smallmul 1#1!#21#3!%
3112 {%
3113 \xint_gob_til_sc #3\XINT_FL_fac_smallmul_end;%
3114 \XINT_FL_fac_minimulwc_a #2!#3!{{#1}{#2}}%
3115 }%

```

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.)

```

3116 \def\XINT_FL_fac_smallmul_end;\XINT_FL_fac_minimulwc_a #1!;!#2#3[#4]%
3117 {%
3118 \ifnum #2=\xint_c_
3119 \expandafter\xint_firstoftwo\else
3120 \expandafter\xint_secondoftwo
3121 \fi
3122 {-2\relax[#4]}%
3123 {1#2\expandafter!\expandafter-\expandafter1\expandafter
3124 [\the\numexpr #4+\xint_c_viii]}%
3125 }%

```

8.86 \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`.

1.4e extends the precision in floating point context adding some overhead but well.

```

3126 \def\xintFloatPFactorial {\romannumeral0\xintfloatpfactorial}%
3127 \def\xintfloatpfactorial #1{\XINT_flpfac_chkopt \xintfloat #1\xint:%}%
3128 \def\XINTinFloatPFactorial{\romannumeral0\XINTinfloatpfactorial}%
3129 \def\XINTinfloatpfactorial{\XINT_flpfac_opt_a\XINTdigits.\XINTinfloatS}%
3130 \def\XINT_flpfac_chkopt #1#2%
3131 {%
3132 \ifx [#2\expandafter\XINT_flpfac_opt
3133 \else\expandafter\XINT_flpfac_noopt

```

```

3134 \fi
3135 #1#2%
3136 }%
3137 \def\XINT_flpfac_noopt #1#2\xint:#3%
3138 {%
3139 \expandafter\XINT_FL_pfac_fork
3140 \the\numexpr \xintNum{#2}\expandafter.%
3141 \the\numexpr \xintNum{#3}.\xint_c_i{\XINTdigits}{#1[\XINTdigits]]}%
3142 }%
3143 \def\XINT_flpfac_opt #1[\xint:#2]%
3144 {%
3145 \expandafter\XINT_flpfac_opt_a\the\numexpr #2.#1%
3146 }%
3147 \def\XINT_flpfac_opt_a #1.#2#3#4%
3148 {%
3149 \expandafter\XINT_FL_pfac_fork
3150 \the\numexpr \xintNum{#3}\expandafter.%
3151 \the\numexpr \xintNum{#4}.\xint_c_i{#1}{#2[#1]]}%
3152 }%
3153 \def\XINT_FL_pfac_fork #1#2.#3#4.%
3154 {%
3155 \unless\ifnum #1#2<#3#4 \xint_dothis\XINT_FL_pfac_one\fi
3156 \if-#3\xint_dothis\XINT_FL_pfac_neg \fi
3157 \if-#1\xint_dothis\XINT_FL_pfac_zero\fi
3158 \ifnum #3#4>\xint_c_x^viii_mone\xint_dothis\XINT_FL_pfac_outofrange\fi
3159 \xint_orthat \XINT_FL_pfac_increaseP #1#2.#3#4.%
3160 }%
3161 \def\XINT_FL_pfac_outofrange #1.#2.#3#4#5%
3162 {%
3163 #5{\XINT_signalcondition{InvalidOperation}
3164 {pFactorial with too large argument: #2 >= 10^8.}}{ 0[0]}}%
3165 }%
3166 \def\XINT_FL_pfac_one #1.#2.#3#4#5{#5{1[0]}}%
3167 \def\XINT_FL_pfac_zero #1.#2.#3#4#5{#5{0[0]}}%
3168 \def\XINT_FL_pfac_neg -#1.-#2.%
3169 {%
3170 \ifnum #1>\xint_c_x^viii\xint_dothis\XINT_FL_pfac_outofrange\fi
3171 \xint_orthat {%
3172 \ifodd\numexpr#2-#1\relax\xint_afterfi{\expandafter-\romannumeral`&&@}\fi
3173 \expandafter\XINT_FL_pfac_increaseP}%
3174 \the\numexpr #2-\xint_c_i\expandafter.\the\numexpr#1-\xint_c_i.%
3175 }%

```

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 #3.10^{ -#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.

```

3176 \def\XINT_FL_pfac_increaseP #1.#2.#3#4%
3177 {%
3178 \expandafter\XINT_FL_pfac_a

```

```

3179 \the\numexpr \xint_c_viii*((\xint_c_iv+#4+\expandafter
3180 \XINT_FL_fac_countdigits\the\numexpr (#2-#1-\xint_c_i)%
3181 /\ifnum #2>\xint_c_x^iv #3\else(#3*\xint_c_ii)\fi\relax
3182 87654321\Z)/\xint_c_viii).#1.#2.%
3183 }%
3184 \def\XINT_FL_pfac_a #1.#2.#3.%
3185 {%
3186 \expandafter\XINT_FL_pfac_b\the\numexpr \xint_c_i+#2\expandafter.%
3187 \the\numexpr#3\expandafter.%
3188 \romannumeral0\XINT_FL_pfac_addzeroes #1.10000000011;![-#1]%
3189 }%
3190 \def\XINT_FL_pfac_addzeroes #1.%
3191 {%
3192 \ifnum #1=\xint_c_viii \expandafter\XINT_FL_pfac_addzeroes_exit\fi
3193 \expandafter\XINT_FL_pfac_addzeroes\the\numexpr #1-\xint_c_viii.100000000!%
3194 }%
3195 \def\XINT_FL_pfac_addzeroes_exit #1.{ }%
3196 \def\XINT_FL_pfac_b #1.%
3197 {%
3198 \ifnum #1>9999 \xint_dothis\XINT_FL_pfac_vbigloop \fi
3199 \ifnum #1>463 \xint_dothis\XINT_FL_pfac_bigloop \fi
3200 \ifnum #1>98 \xint_dothis\XINT_FL_pfac_medloop \fi
3201 \xint_orthat\XINT_FL_pfac_smallloop #1.%
3202 }%
3203 \def\XINT_FL_pfac_smallloop #1.#2.%
3204 {%
3205 \ifcase\numexpr #2-#1\relax
3206 \expandafter\XINT_FL_pfac_end_
3207 \or \expandafter\XINT_FL_pfac_end_i
3208 \or \expandafter\XINT_FL_pfac_end_ii
3209 \or \expandafter\XINT_FL_pfac_end_iii
3210 \else\expandafter\XINT_FL_pfac_smallloop_a
3211 \fi #1.#2.%
3212 }%
3213 \def\XINT_FL_pfac_smallloop_a #1.#2.%
3214 {%
3215 \expandafter\XINT_FL_pfac_smallloop_b
3216 \the\numexpr #1+\xint_c_iv\expandafter.%
3217 \the\numexpr #2\expandafter.%
3218 \romannumeral0\expandafter\XINT_FL_fac_mul
3219 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
3220 }%
3221 \def\XINT_FL_pfac_smallloop_b #1.%
3222 {%
3223 \ifnum #1>98 \expandafter\XINT_FL_pfac_medloop \else
3224 \expandafter\XINT_FL_pfac_smallloop \fi #1.%
3225 }%
3226 \def\XINT_FL_pfac_medloop #1.#2.%
3227 {%
3228 \ifcase\numexpr #2-#1\relax
3229 \expandafter\XINT_FL_pfac_end_
3230 \or \expandafter\XINT_FL_pfac_end_i

```

```

3231 \or \expandafter\XINT_FL_pfac_end_ii
3232 \else\expandafter\XINT_FL_pfac_medloop_a
3233 \fi #1.#2.%
3234 }%
3235 \def\XINT_FL_pfac_medloop_a #1.#2.%
3236 {%
3237 \expandafter\XINT_FL_pfac_medloop_b
3238 \the\numexpr #1+\xint_c_iii\expandafter.%
3239 \the\numexpr #2\expandafter.%
3240 \romannumeral0\expandafter\XINT_FL_fac_mul
3241 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
3242 }%
3243 \def\XINT_FL_pfac_medloop_b #1.%
3244 {%
3245 \ifnum #1>463 \expandafter\XINT_FL_pfac_bigloop \else
3246 \expandafter\XINT_FL_pfac_medloop \fi #1.%
3247 }%
3248 \def\XINT_FL_pfac_bigloop #1.#2.%
3249 {%
3250 \ifcase\numexpr #2-#1\relax
3251 \expandafter\XINT_FL_pfac_end_
3252 \or \expandafter\XINT_FL_pfac_end_i
3253 \else\expandafter\XINT_FL_pfac_bigloop_a
3254 \fi #1.#2.%
3255 }%
3256 \def\XINT_FL_pfac_bigloop_a #1.#2.%
3257 {%
3258 \expandafter\XINT_FL_pfac_bigloop_b
3259 \the\numexpr #1+\xint_c_ii\expandafter.%
3260 \the\numexpr #2\expandafter.%
3261 \romannumeral0\expandafter\XINT_FL_fac_mul
3262 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
3263 }%
3264 \def\XINT_FL_pfac_bigloop_b #1.%
3265 {%
3266 \ifnum #1>9999 \expandafter\XINT_FL_pfac_vbigloop \else
3267 \expandafter\XINT_FL_pfac_bigloop \fi #1.%
3268 }%
3269 \def\XINT_FL_pfac_vbigloop #1.#2.%
3270 {%
3271 \ifnum #2=#1
3272 \expandafter\XINT_FL_pfac_end_
3273 \else\expandafter\XINT_FL_pfac_vbigloop_a
3274 \fi #1.#2.%
3275 }%
3276 \def\XINT_FL_pfac_vbigloop_a #1.#2.%
3277 {%
3278 \expandafter\XINT_FL_pfac_vbigloop
3279 \the\numexpr #1+\xint_c_i\expandafter.%
3280 \the\numexpr #2\expandafter.%
3281 \romannumeral0\expandafter\XINT_FL_fac_mul
3282 \the\numexpr \xint_c_x^viii+#1!%

```

```

3283 }%
3284 \def\XINT_FL_pfac_end_iii #1.#2.%
3285 {%
3286     \expandafter\XINT_FL_fac_out
3287     \romannumeral0\expandafter\XINT_FL_fac_mul
3288     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
3289 }%
3290 \def\XINT_FL_pfac_end_ii #1.#2.%
3291 {%
3292     \expandafter\XINT_FL_fac_out
3293     \romannumeral0\expandafter\XINT_FL_fac_mul
3294     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
3295 }%
3296 \def\XINT_FL_pfac_end_i #1.#2.%
3297 {%
3298     \expandafter\XINT_FL_fac_out
3299     \romannumeral0\expandafter\XINT_FL_fac_mul
3300     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
3301 }%
3302 \def\XINT_FL_pfac_end_ #1.#2.%
3303 {%
3304     \expandafter\XINT_FL_fac_out
3305     \romannumeral0\expandafter\XINT_FL_fac_mul
3306     \the\numexpr \xint_c_x^viii+#1!%
3307 }%

```

8.87 \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.

1.4e: same remarks as for factorial and partial factorial about added overhead due to extra guard digits.

```

3308 \def\xintFloatBinomial {\romannumeral0\xintfloatbinomial}%
3309 \def\xintfloatbinomial #1{\XINT_flbinom_chkopt \xintfloat #1\xint:}%
3310 \def\XINTinFloatBinomial{\romannumeral0\XINTinfloatbinomial }%
3311 \def\XINTinfloatbinomial{\XINT_flbinom_opt\XINTinfloatS[\xint:\XINTdigits]}%
3312 \def\XINT_flbinom_chkopt #1#2%
3313 {%
3314     \ifx [#2\expandafter\XINT_flbinom_opt
3315     \else\expandafter\XINT_flbinom_noopt
3316     \fi #1#2%
3317 }%
3318 \def\XINT_flbinom_noopt #1#2\xint:#3%
3319 {%
3320     \expandafter\XINT_FL_binom_a
3321     \the\numexpr\xintNum{#2}\expandafter.\the\numexpr\xintNum{#3}.\XINTdigits.#1%

```

```

3322 }%
3323 \def\XINT_flbinom_opt #1[\xint:#2]#3#4%
3324 {%
3325     \expandafter\XINT_FL_binom_a
3326     \the\numexpr\xintNum{#3}\expandafter.\the\numexpr\xintNum{#4}\expandafter.%
3327     \the\numexpr #2.#1%
3328 }%
3329 \def\XINT_FL_binom_a #1.#2.%
3330 {%
3331     \expandafter\XINT_FL_binom_fork \the\numexpr #1-#2.#2.#1.%
3332 }%
3333 \def\XINT_FL_binom_fork #1#2.#3#4.#5#6.%
3334 {%
3335     \if-#5\xint_dothis \XINT_FL_binom_neg\fi
3336     \if-#1\xint_dothis \XINT_FL_binom_zero\fi
3337     \if-#3\xint_dothis \XINT_FL_binom_zero\fi
3338     \if0#1\xint_dothis \XINT_FL_binom_one\fi
3339     \if0#3\xint_dothis \XINT_FL_binom_one\fi
3340     \ifnum #5#6>\xint_c_x^viii_mone \xint_dothis\XINT_FL_binom_toobig\fi
3341     \ifnum #1#2>#3#4 \xint_dothis\XINT_FL_binom_ab \fi
3342     \xint_orthat\XINT_FL_binom_aa
3343     #1#2.#3#4.#5#6.%
3344 }%
3345 \def\XINT_FL_binom_neg #1.#2.#3.#4.#5%
3346 {%
3347     #5[#4]{\XINT_signalcondition{InvalidOperation}
3348             {Binomial with negative argument: #3.}}{ 0[0]}}%
3349 }%
3350 \def\XINT_FL_binom_toobig #1.#2.#3.#4.#5%
3351 {%
3352     #5[#4]{\XINT_signalcondition{InvalidOperation}
3353             {Binomial with too large argument: #3 >= 10^8.}}{ 0[0]}}%
3354 }%
3355 \def\XINT_FL_binom_one #1.#2.#3.#4.#5{#5[#4]{1[0]}}%
3356 \def\XINT_FL_binom_zero #1.#2.#3.#4.#5{#5[#4]{0[0]}}%
3357 \def\XINT_FL_binom_aa #1.#2.#3.#4.#5%
3358 {%
3359     #5[#4]{\xintDiv{\XINT_FL_pfac_increaseP
3360                 #2.#3.\xint_c_iv{#4+\xint_c_i}{\XINTinfloat[#4+\xint_c_iii]}}%
3361             {\XINT_FL_fac_fork_b
3362             #1.\xint_c_iv{#4+\xint_c_i}\XINT_FL_fac_out{\XINTinfloat[#4+\xint_c_iii]}}}%
3363 }%
3364 \def\XINT_FL_binom_ab #1.#2.#3.#4.#5%
3365 {%
3366     #5[#4]{\xintDiv{\XINT_FL_pfac_increaseP
3367                 #1.#3.\xint_c_iv{#4+\xint_c_i}{\XINTinfloat[#4+\xint_c_iii]}}%
3368             {\XINT_FL_fac_fork_b
3369             #2.\xint_c_iv{#4+\xint_c_i}\XINT_FL_fac_out{\XINTinfloat[#4+\xint_c_iii]}}}%
3370 }%

```

8.88 \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.

1.4e (2021/04/15) great hesitation about what to do regarding guard digits. This will spoil the guaranteed "correct-rounding" property for individual calculations... but is interesting for precision as soon as the square root is embedded into some larger calculation. Annoying. But there is \xintexpr which I can left configured to use strictly \xintDigits in contrast to \xintfloatexpr. Ah ok and there will always be sqrt(x,\xinttheDigits) syntax if one wants. And finally I keep sqrt() acting the same in expr and floatexpr.

Attention that at 1.4e \XINTinFloatSqrt is defined to be used ONLY with optional argument.

```

3371 \def\xintFloatSqrt {\romannumeral0\xintfloatsqrt}%
3372 \def\xintfloatsqrt #1{\XINT_flsqrt_chkopt \xintfloat #1\xint:}%
3373 \def\XINTinFloatSqrt{\romannumeral0\XINTinfloatsqrt}%
3374 \def\XINTinfloatsqrt[#1]{\expandafter\XINT_flsqrt_opt_a\the\numexpr#1.\XINTinfloatS}%
3375 \def\XINTinFloatSqrtdigits{\romannumeral0\XINT_flsqrt_opt_a\XINTdigits.\XINTinfloatS}%
3376 \def\XINT_flsqrt_chkopt #1#2%
3377 {%
3378   \ifx [#2\expandafter\XINT_flsqrt_opt
3379     \else\expandafter\XINT_flsqrt_noopt
3380   \fi #1#2%
3381 }%
3382 \def\XINT_flsqrt_noopt #1#2\xint:%
3383 {%
3384   \expandafter\XINT_FL_sqrt_a
3385     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.#1%
3386 }%
3387 \def\XINT_flsqrt_opt #1[\xint:#2]%#3%
3388 {%
3389   \expandafter\XINT_flsqrt_opt_a\the\numexpr #2.#1%
3390 }%
3391 \def\XINT_flsqrt_opt_a #1.#2#3%
3392 {%
3393   \expandafter\XINT_FL_sqrt_a\romannumeral0\XINTinfloat[#1]{#3}#1.#2%
3394 }%
3395 \def\XINT_FL_sqrt_a #1%
3396 {%
3397   \xint_UDzerominusfork
3398   #1-\XINT_FL_sqrt_iszero
3399   0#1\XINT_FL_sqrt_isneg
3400   0-{\XINT_FL_sqrt_pos #1}%
3401   \krof
3402 }%[
3403 \def\XINT_FL_sqrt_iszero #1]#2.#3{#3[#2]{0[0]}}%

```



```

3404 \def\XINT_FL_sqrt_isneg #1]#2.#3%
3405 {%
3406   #3[#2]{\XINT_signalcondition{InvalidOperation}
3407           {Square root of negative: -#1].}}{ 0[0]}}%
3408 }%

3409 \def\XINT_FL_sqrt_pos #1[#2]#3.%
3410 {%
3411   \expandafter\XINT_flsqrt
3412   \the\numexpr #3\ifodd #2 \xint_dothis {+\xint_c_iii.(#2+\xint_c_i).0}\fi
3413   \xint_orthat {+\xint_c_ii.#2.}}#100.#3.%
3414 }%

3415 \def\XINT_flsqrt #1.#2.%
3416 {%
3417   \expandafter\XINT_flsqrt_a
3418   \the\numexpr #2/\xint_c_ii-(#1-\xint_c_i)/\xint_c_ii.#1.%
3419 }%

3420 \def\XINT_flsqrt_a #1.#2.#3#4.#5.%
3421 {%
3422   \expandafter\XINT_flsqrt_b
3423   \the\numexpr (#2-\xint_c_i)/\xint_c_ii\expandafter.%
3424   \romannumeral0\XINT_sqrt_start #2.#4#3.#5.#2.#4#3.#5.#1.%
3425 }%

3426 \def\XINT_flsqrt_b #1.#2#3%
3427 {%
3428   \expandafter\XINT_flsqrt_c
3429   \romannumeral0\xintiisub
3430   {\XINT_dsx_addzeros {#1}#2;}%
3431   {\xintiiDivRound{\XINT_dsx_addzeros {#1}#3;}%
3432           {\XINT_dbl#2\xint_bye2345678\xint_bye*\xint_c_ii\relax}}.%
3433 }%

3434 \def\XINT_flsqrt_c #1.#2.%
3435 {%
3436   \expandafter\XINT_flsqrt_d
3437   \romannumeral0\XINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..%
3438 }%

3439 \def\XINT_flsqrt_d #1.#2#3.%
3440 {%
3441   \ifnum #2=\xint_c_v
3442   \expandafter\XINT_flsqrt_f\else\expandafter\XINT_flsqrt_finish\fi
3443   #2#3.#1.%
3444 }%

3445 \def\XINT_flsqrt_finish #1#2.#3.#4.#5.#6.#7.#8{#8[#6]{#3#1[#7]}}%

3446 \def\XINT_flsqrt_f 5#1.%
3447   {\expandafter\XINT_flsqrt_g\romannumeral0\xintinum{#1}\relax.%
3448 \def\XINT_flsqrt_g #1#2#3.{\ifrelax#2\xint_dothis{\XINT_flsqrt_h #1}\fi
3449   \xint_orthat{\XINT_flsqrt_finish 5.}}%
3450 \def\XINT_flsqrt_h #1{\ifnum #1<\xint_c_iii\xint_dothis{\XINT_flsqrt_again}\fi
3451   \xint_orthat{\XINT_flsqrt_finish 5.}}%

```

```

3452 \def\XINT_flsqrt_again #1.#2.%
3453 {%
3454   \expandafter\XINT_flsqrt_again_a\the\numexpr #2+\xint_c_viii.%
3455 }%

3456 \def\XINT_flsqrt_again_a #1.#2.#3.%
3457 {%
3458   \expandafter\XINT_flsqrt_b
3459   \the\numexpr (#1-\xint_c_i)/\xint_c_ii\expandafter.%
3460   \romannumeral0\XINT_sqrt_start #1.#2000000000.#3.%
3461   #1.#2000000000.#3.%
3462 }%

```

8.89 \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. With input 0 it produces on output an 0[N], not 0[0].

```

3463 \def\xintFloatE   {\romannumeral0\xintfloate }%
3464 \def\xintfloate #1{\XINT_floate_chkopt #1\xint:}%
3465 \def\XINT_floate_chkopt #1%
3466 {%
3467   \ifx [#1\expandafter\XINT_floate_opt
3468   \else\expandafter\XINT_floate_noopt
3469   \fi #1%
3470 }%
3471 \def\XINT_floate_noopt #1\xint:%
3472 {%
3473   \expandafter\XINT_floate_post
3474   \romannumeral0\XINTinfloat[\XINTdigits]{#1}\XINTdigits.%
3475 }%
3476 \def\XINT_floate_opt [\xint:#1]%
3477 {%
3478   \expandafter\XINT_floate_opt_a\the\numexpr #1.%
3479 }%
3480 \def\XINT_floate_opt_a #1.#2%
3481 {%
3482   \expandafter\XINT_floate_post
3483   \romannumeral0\XINTinfloat[#1]{#2}#1.%
3484 }%
3485 \def\XINT_floate_post #1%
3486 {%
3487   \xint_UDzerominusfork
3488   #1-\XINT_floate_zero
3489   0#1\XINT_floate_neg
3490   0-\XINT_floate_pos
3491   \krof #1%
3492 }%[
3493 \def\XINT_floate_zero #1]#2.#3{ 0.e0}%
3494 \def\XINT_floate_neg-{\expandafter-\romannumeral0\XINT_floate_pos}%
3495 \def\XINT_floate_pos #1#2[#3]#4.#5%

```

```

3496 {%
3497     \expandafter\XINT_float_pos_done\the\numexpr#3+#4+#5-\xint_c_i.#1.#2;%
3498 }%
3499 \def\XINTinFloatE {\romannumeral0\XINTinfloate }%
3500 \def\XINTinfloate
3501     {\expandafter\XINT_infloate\romannumeral0\XINTinfloat[\XINTdigits]]}%
3502 \def\XINT_infloate #1[#2]#3%
3503     {\expandafter\XINT_infloate_end\the\numexpr #3+#2.{#1}}%
3504 \def\XINT_infloate_end #1.#2{ #2[#1]}%

```

8.90 \XINTinFloatMod

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

```

3505 \def\XINTinFloatMod {\romannumeral0\XINTinfloatmod [\XINTdigits]]}%
3506 \def\XINTinfloatmod [#1]#2#3%
3507 {%
3508     \XINTinfloat[#1]{\xintMod
3509         {\romannumeral0\XINTinfloat[#1]{#2}}}%
3510     {\romannumeral0\XINTinfloat[#1]{#3}}}%
3511 }%

```

8.91 \XINTinFloatDivFloor

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

```

3512 \def\XINTinFloatDivFloor {\romannumeral0\XINTinfloatdivfloor [\XINTdigits]]}%
3513 \def\XINTinfloatdivfloor [#1]#2#3%
3514 {%
3515     \xintdivfloor
3516         {\romannumeral0\XINTinfloat[#1]{#2}}}%
3517     {\romannumeral0\XINTinfloat[#1]{#3}}}%
3518 }%

```

8.92 \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.

```

3519 \def\XINTinFloatDivMod {\romannumeral0\XINTinfloatdivmod [\XINTdigits]]}%
3520 \def\XINTinfloatdivmod [#1]#2#3%
3521 {%
3522     \expandafter\XINT_infloatdivmod
3523     \romannumeral0\xintdivmod
3524         {\romannumeral0\XINTinfloat[#1]{#2}}}%
3525     {\romannumeral0\XINTinfloat[#1]{#3}}}%
3526     {#1}%

```

```
3527 }%
3528 \def\XINT_infloatdivmod #1#2#3{\expanded{{#1}{\XINTinFloat[#3]{#2}}}}%
```

8.93 \xintiffloatInt

1.3a for ifint() function in \xintfloatexpr.

```
3529 \def\xintiffloatInt {\romannumeral0\xintiffloatint}%
3530 \def\xintiffloatint #1{\expandafter\XINT_iffloatint
3531     \romannumeral0\xintrez{\XINTinFloatS[\XINTdigits]{#1}}}%
3532 \def\XINT_iffloatint #1#2/1[#3]%
3533 {%
3534     \if 0#1\xint_dothis\xint_stop_atfirstoftwo\fi
3535     \ifnum#3<\xint_c_\xint_dothis\xint_stop_atsecondoftwo\fi
3536     \xint_orthat\xint_stop_atfirstoftwo
3537 }%
```

8.94 \xintFloatIsInt

1.3d for isint() function in \xintfloatexpr.

```
3538 \def\xintFloatIsInt {\romannumeral0\xintfloatisint}%
3539 \def\xintfloatisint #1{\expandafter\XINT_iffloatint
3540     \romannumeral0\xintrez{\XINTinFloatS[\XINTdigits]{#1}}10}%

```

8.95 \xintFloatIntType

1.4e for fractional powers. Expands to \xint_c_mone if argument is not an integer, to \xint_c_ if it is an even integer and to \xint_c_i if it is an odd integer.

```
3541 \def\xintFloatIntType {\romannumeral`&&\xintfloatinttype}%
3542 \def\xintfloatinttype #1%
3543 {%
3544     \expandafter\XINT_floatinttype
3545     \romannumeral0\xintrez{\XINTinFloatS[\XINTdigits]{#1}}}%
3546 }%
3547 \def\XINT_floatinttype #1#2/1[#3]%
3548 {%
3549     \if 0#1\xint_dothis\xint_c_\fi
3550     \ifnum#3<\xint_c_\xint_dothis\xint_c_mone\fi
3551     \ifnum#3>\xint_c_\xint_dothis\xint_c_\fi
3552     \ifodd\xintLDg{#1#2} \xint_dothis\xint_c_i\fi
3553     \xint_orthat\xint_c_
3554 }%
```

8.96 \XINTinFloatdigits, \XINTinFloatSdigits

For \xintNewExpr/\xintdeffloatfunc matters, mainly.

```
3555 \def\XINTinFloatdigits {\XINTinFloat [\XINTdigits]}%
3556 \def\XINTinFloatSdigits{\XINTinFloatS[\XINTdigits]}%
```

8.97 (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.

```

3557 \def\XINTinRandomFloatS{\romannumeral0\XINTinrandomfloatS}%
3558 \def\XINTinRandomFloatSdigits{\XINTinRandomFloatS[\XINTdigits]}%
3559 \def\XINTinrandomfloatS[#1]%
3560 {%
3561     \expandafter\XINT_inrandomfloatS\the\numexpr\xint_c_-(#1)\xint:
3562 }%
3563 \def\XINT_inrandomfloatS-#1\xint:
3564 {%
3565     \expandafter\XINT_inrandomfloatS_a
3566     \romannumeral0\xintrandomdigits{#1}[-#1]%
3567 }%

```

We add one macro to handle a tiny bit faster 90% of cases, after all we also use one extra macro for the completely improbable all 0 case.

```

3568 \def\XINT_inrandomfloatS_a#1%
3569 {%
3570     \if#10\xint_dothis{\XINT_inrandomfloatS_b}\fi
3571     \xint_orthat{ #1}%
3572 }%[
3573 \def\XINT_inrandomfloatS_b#1%
3574 {%
3575     \if#1[\xint_dothis{\XINT_inrandomfloatS_zero}\fi% ]
3576     \if#10\xint_dothis{\XINT_inrandomfloatS_b}\fi
3577     \xint_orthat{ #1}%
3578 }%[
3579 \def\XINT_inrandomfloatS_zero#1{ 0[0]}%

```

8.98 (WIP) \XINTinRandomFloatSixteen

1.3b. Support for qrand() function.

```

3580 \def\XINTinRandomFloatSixteen%
3581 {%
3582     \romannumeral0\expandafter\XINT_inrandomfloatS_a
3583     \romannumeral`&&\expandafter\XINT_eightrandomdigits
3584     \romannumeral`&&\XINT_eightrandomdigits[-16]%
3585 }%

```

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

```
3586 \let\XINTinFloatMaxof\XINT_Maxof
3587 \let\XINTinFloatMinof\XINT_Minof
3588 \let\XINTinFloatSum\XINT_Sum
3589 \let\XINTinFloatPrd\XINT_Prd
3590 \XINTrestorecatcodesendinginput%
```

9 Package *xintseries* implementation

.1	Catcodes, ε -TeX and reload detection . . .	279	.7	<code>\xintRationalSeries</code>	282
.2	Package identification	280	.8	<code>\xintRationalSeriesX</code>	283
.3	<code>\xintSeries</code>	280	.9	<code>\xintFxFtPowerSeries</code>	284
.4	<code>\xintiSeries</code>	280	.10	<code>\xintFxFtPowerSeriesX</code>	285
.5	<code>\xintPowerSeries</code>	281	.11	<code>\xintFloatPowerSeries</code>	285
.6	<code>\xintPowerSeriesX</code>	282	.12	<code>\xintFloatPowerSeriesX</code>	287

The commenting is currently (2021/06/11) 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 [2021/06/11 v1.4i 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,rationfunction,x

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

```

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

```

```

203     }%
204   \fi
205 }%
206 \def\XINT_ratseriesx_pre #1#2#3#4#5%
207 {%
208   \XINT_ratseries_loop {#2}{1}{#3}{#4{#1}}{#5{#1}}%
209 }%

```

9.9 \xintFxFtPowerSeries

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\xintFxFtPowerSeries {\romannumeral0\xintfxptpowerseries }%
211 \def\xintfxptpowerseries #1#2%
212 {%
213   \expandafter\XINT_fppowseries\expandafter
214   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
215 }%
216 \def\XINT_fppowseries #1#2#3#4#5%
217 {%
218   \ifnum #2<#1
219     \xint_afterfi { 0}%
220   \else
221     \xint_afterfi
222     {\expandafter\XINT_fppowseries_loop_pre\expandafter
223      {\romannumeral0\xinttrunc {#5}{\xintPow {#4}{#1}}}%
224      {#1}{#4}{#2}{#3}{#5}%
225     }%
226   \fi
227 }%
228 \def\XINT_fppowseries_loop_pre #1#2#3#4#5#6%
229 {%
230   \ifnum #4>#2 \else\XINT_fppowseries_dont_i \fi
231   \expandafter\XINT_fppowseries_loop_i\expandafter
232   {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
233   {\romannumeral0\xintitrunc {#6}{\xintMul {#5{#2}}{#1}}}%
234   {#1}{#3}{#4}{#5}{#6}%
235 }%
236 \def\XINT_fppowseries_dont_i \fi\expandafter\XINT_fppowseries_loop_i
237   {\fi \expandafter\XINT_fppowseries_dont_ii }%
238 \def\XINT_fppowseries_dont_ii #1#2#3#4#5#6#7{\xinttrunc {#7}{#2[-#7]}}%
239 \def\XINT_fppowseries_loop_i #1#2#3#4#5#6#7%
240 {%
241   \ifnum #5>#1 \else \XINT_fppowseries_exit_i \fi
242   \expandafter\XINT_fppowseries_loop_ii\expandafter
243   {\romannumeral0\xinttrunc {#7}{\xintMul {#3}{#4}}}%
244   {#1}{#4}{#2}{#5}{#6}{#7}%
245 }%
246 \def\XINT_fppowseries_loop_ii #1#2#3#4#5#6#7%
247 {%

```

```

248 \expandafter\XINT_fppowseries_loop_i\expandafter
249 {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
250 {\romannumeral0\xintiiadd {#4}{\xintiTrunc {#7}{\xintMul {#6{#2}}{#1}}}%
251 {#1}{#3}{#5}{#6}{#7}%
252 }%
253 \def\XINT_fppowseries_exit_i\fi\expandafter\XINT_fppowseries_loop_ii
254 {\fi \expandafter\XINT_fppowseries_exit_ii }%
255 \def\XINT_fppowseries_exit_ii #1#2#3#4#5#6#7%
256 {%
257 \xinttrunc {#7}
258 {\xintiiadd {#4}{\xintiTrunc {#7}{\xintMul {#6{#2}}{#1}}}{-#7}}%
259 }%

```

9.10 \xintFxFtPowerSeriesX

a,b,coeff,x,D

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

```

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

```

9.11 \xintFloatPowerSeries

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

Usage of new names \XINTinfloatpow_wopt \XINTinfloatmul_wopt, \XINTinfloatadd_wopt to track xintfrac.sty changes at 1.4e.

```

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_wopt[#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_wopt[#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_wopt[#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_wopt[#7]{#4}%
339             {\XINTinfloatmul_wopt[#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_wopt[#7]{#6{#2}}{#1}}}%
347 }%

```

9.12 \xintFloatPowerSeriesX

1.08a

See \xintFloatPowerSeries for 1.4e comments.

```

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_wopt[#5]{#1}{#2}}%
384     {#2}{#1}{#3}{#4}{#5}%

```

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

385 }%

386 \XINTrestorecatcodesendinginput%

10 Package *xintcfrac* implementation

.1	Catcodes, ε -T _E X and reload detection . . .	289	.16	<code>\xintiGctoF</code>	301
.2	Package identification	290	.17	<code>\xintCtoCv</code> , <code>\xintCstoCv</code>	302
.3	<code>\xintCFrac</code>	290	.18	<code>\xintiCstoCv</code>	303
.4	<code>\xintGCFrac</code>	291	.19	<code>\xintGctoCv</code>	304
.5	<code>\xintGGCFrac</code>	293	.20	<code>\xintiGctoCv</code>	305
.6	<code>\xintGctoGCx</code>	294	.21	<code>\xintFtoCv</code>	306
.7	<code>\xintFtoCs</code>	294	.22	<code>\xintFtoCCv</code>	306
.8	<code>\xintFtoCx</code>	295	.23	<code>\xintCntoF</code>	306
.9	<code>\xintFtoC</code>	296	.24	<code>\xintGcntoF</code>	307
.10	<code>\xintFtoGC</code>	296	.25	<code>\xintCntoCs</code>	308
.11	<code>\xintFGtoC</code>	296	.26	<code>\xintCntoGC</code>	309
.12	<code>\xintFtoCC</code>	297	.27	<code>\xintGcntoGC</code>	309
.13	<code>\xintCtoF</code> , <code>\xintCstoF</code>	298	.28	<code>\xintCstoGC</code>	310
.14	<code>\xintiCstoF</code>	299	.29	<code>\xintGctoGC</code>	310
.15	<code>\xintGctoF</code>	300			

The commenting is currently (2021/06/11) 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 [2021/06/11 v1.4i 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\xintraewithzeros {#1}\Z
78    \hfill\relax
79 }%
80 \def\XINT_cfrac_A #1/#2\Z
81 {%
82    \expandafter\XINT_cfrac_B\romannumeral0\xintiidivision {#1}{#2}{#2}%
83 }%
84 \def\XINT_cfrac_B #1#2%
85 {%
86    \XINT_cfrac_C #2\Z {#1}%
87 }%
88 \def\XINT_cfrac_C #1%
89 {%
90    \xint_gob_til_zero #1\XINT_cfrac_integer 0\XINT_cfrac_D #1%
91 }%
92 \def\XINT_cfrac_integer 0\XINT_cfrac_D 0#1\Z #2#3#4#5{ #2}%
93 \def\XINT_cfrac_D #1\Z #2#3{\XINT_cfrac_loop_a {#1}{#3}{#1}{#2}}%
94 \def\XINT_cfrac_loop_a
95 {%
96    \expandafter\XINT_cfrac_loop_d\romannumeral0\XINT_div_prepare
97 }%
98 \def\XINT_cfrac_loop_d #1#2%
99 {%
100    \XINT_cfrac_loop_e #2.{#1}%
101 }%
102 \def\XINT_cfrac_loop_e #1%
103 {%
104    \xint_gob_til_zero #1\xint_cfrac_loop_exit0\XINT_cfrac_loop_f #1%
105 }%
106 \def\XINT_cfrac_loop_f #1.#2#3#4%
107 {%
108    \XINT_cfrac_loop_a {#1}{#3}{#1}{#2#4}%
109 }%
110 \def\xint_cfrac_loop_exit0\XINT_cfrac_loop_f #1.#2#3#4#5#6%
111    {\XINT_cfrac_T #5#6{#2#4}\Z }%
112 \def\XINT_cfrac_T #1#2#3#4%
113 {%
114    \xint_gob_til_Z #4\XINT_cfrac_end\Z\XINT_cfrac_T #1#2{#4+\cfrac{#11#2}{#3}}%
115 }%
116 \def\XINT_cfrac_end\Z\XINT_cfrac_T #1#2#3%
117 {%
118    \XINT_cfrac_end_b #3%
119 }%
120 \def\XINT_cfrac_end_b \Z+\cfrac#1#2{ #2}%

```

10.4 \xintGCFrac

Updated at 1.4g to follow-up on renaming of `\xintFrac` into `\xintTeXFrac`.

```

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{\xintTeXFrac{#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{\xintTeXFrac{#5}}%
166         \ifcase\xintSgn{#4}
167         +\or+\else-\fi
168         \cfrac{#1\xintTeXFrac{\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_ggfrac_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_fgtc_d\romannumeral0\XINT_div_prepare
361         {#4}{#5}{#4}{#1}{#2}{#3}%
362 }%

```

10.12 \xintFtoCC

```

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

```

```

403 \romannumeral0\xintraewithzeros {\xintAdd {1/2[0]}\{#1}\Z {#1}%
404 }%
405 \def\XINT_ftcc_loop_b #1/#2\Z
406 {%
407 \expandafter\XINT_ftcc_loop_c\expandafter
408 {\romannumeral0\xintiigo {#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{\xintraewithzeros {#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{\xintrowwithzeros {#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\xintrowwithzeros {#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\xintrowwithzeros {#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\xintraawithzeros {#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\xintraawithzeros {#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\xintraawithzeros {#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}}/%
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 \XINTrestorecatcodesendinginput%

```

11 Package **xintexpr** implementation

This is release 1.4i of 2021/06/11.

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	314
11.2	Old comments	315
11.3	Catcodes, ε -TeX and reload detection	316
11.4	Package identification	317
11.5	<code>\xintDigits*</code> , <code>\xintSetDigits*</code> , <code>\xintreloadscilibs</code>	317
11.6	<code>\XINTdigitsormax</code>	318
11.7	Support for output and transform of nested braced contents as core data type	318
11.7.1	Bracketed list rendering with prettifying of leaves from nested braced contents	318
11.7.2	Flattening nested braced contents	319
11.7.3	Braced contents rendering via a TeX alignment with prettifying of leaves	319
11.7.4	Transforming all leaves within nested braced contents	321
11.8	Top level user TeX interface: <code>\xinteval</code> , <code>\xintfloateval</code> , <code>\xintiieval</code>	321
11.8.1	<code>\xintexpr</code> , <code>\xintiexpr</code> , <code>\xintfloatexpr</code> , <code>\xintiieval</code>	322
11.8.2	<code>\XINT_expr_wrap</code> , <code>\XINT_iexpr_wrap</code> , <code>\XINT_flexpr_wrap</code>	323
11.8.3	<code>\XINTexprprint</code> , <code>\XINTiexprprint</code> , <code>\XINTiievalprint</code> , <code>\XINTflexprprint</code>	323
11.8.4	<code>\xintthe</code> , <code>\xintthealign</code> , <code>\xinttheexpr</code> , <code>\xinttheiexpr</code> , <code>\xintthefloatexpr</code> , <code>\xinttheiieval</code>	324
11.8.5	<code>\thexintexpr</code> , <code>\thexintiexpr</code> , <code>\thexintfloatexpr</code> , <code>\thexintiieval</code>	325
11.8.6	<code>\xintbareeval</code> , <code>\xintbarefloateval</code> , <code>\xintbareiieval</code>	325
11.8.7	<code>\xintthebareeval</code> , <code>\xintthebarefloateval</code> , <code>\xintthebareiieval</code>	325
11.8.8	<code>\xinteval</code> , <code>\xintiieval</code> , <code>\xintfloateval</code> , <code>\xintiieval</code>	325
11.8.9	<code>\xintboolexpr</code> , <code>\XINT_boolexpr_print</code> , <code>\xinttheboolexpr</code> , <code>\thexintboolexpr</code>	326
11.8.10	<code>\xintifboolexpr</code> , <code>\xintifboolfloatexpr</code> , <code>\xintifbooliieval</code>	326
11.8.11	<code>\xintifsgnexpr</code> , <code>\xintifsgnfloatexpr</code> , <code>\xintifsgniieval</code>	326
11.8.12	Small bits we have to put somewhere	326
11.9	Hooks into the numeric parser for usage by the <code>\xintdeffunc</code> symbolic parser	328
11.10	<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	329
11.11	<code>\XINT_expr_startint</code>	333
11.11.1	Integral part (skipping zeroes)	334
11.11.2	Fractional part	336
11.11.3	Scientific notation	337
11.11.4	Hexadecimal numbers	338
11.11.5	<code>\XINT_expr_startfunc</code> : collecting names of functions and variables	341
11.11.6	<code>\XINT_expr_func</code> : dispatch to variable replacement or to function execution	341
11.12	<code>\XINT_expr_op_`</code> : launch function or pseudo-function, or evaluate variable and insert operator of multiplication in front of parenthesized contents	342
11.13	<code>\XINT_expr_op_</code> : replace a variable by its value and then fetch next operator	343
11.14	<code>\XINT_expr_gettop</code> : fetch the next operator or closing parenthesis or end of expression	344
11.15	Expansion spanning; opening and closing parentheses	347
11.16	The comma as binary operator	349
11.17	The minus as prefix operator of variable precedence level	350
11.18	The <code>*</code> as Python-like «unpacking» prefix operator	351
11.19	Infix operators	351
11.19.1	<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>	352

11.19.2	.., ..[, and].. for a..b and a..[b]..c syntax	354
11.19.3	<, >, ==, <=, >=, != with Python-like chaining	356
11.19.4	Support macros for .., ..[and]..	358
11.20	Square brackets [] both as a container and a Python slicer	360
11.20.1	[...] as «oneple» constructor	360
11.20.2	[...] brackets and : operator for NumPy-like slicing and item indexing syntax	361
11.20.3	Macro layer implementing indexing and slicing	363
11.21	Support for raw A/B[N]	367
11.22	? as two-way and ?? as three-way «short-circuit» conditionals	368
11.23	! as postfix factorial operator	369
11.24	User defined variables	369
11.24.1	\xintdefvar, \xintdefiivar, \xintdeffloatvar	369
11.24.2	\xintunassignvar	373
11.25	Support for dummy variables	374
11.25.1	\xintnewdummy	374
11.25.2	\xintensuredummy, \xintrestorevariable	375
11.25.3	Checking (without expansion) that a symbolic expression contains correctly nested parentheses	376
11.25.4	Fetching balanced expressions E1, E2 and a variable name Name from E1, Name=E2)	377
11.25.5	Fetching a balanced expression delimited by a semi-colon	377
11.25.6	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()	378
11.25.7	Reserved dummy variables @, @1, @2, @3, @4, @@, @@(1), ..., @@@, @@@(1), ... for recursions	379
11.26	Pseudo-functions involving dummy variables and generating scalars or sequences	380
11.26.1	Comments	380
11.26.2	subs(): substitution of one variable	382
11.26.3	subsm(): simultaneous independent substitutions	383
11.26.4	subsn(): leaner syntax for nesting (possibly dependent) substitutions	384
11.26.5	seq(): sequences from assigning values to a dummy variable	386
11.26.6	iter()	387
11.26.7	add(), mul()	388
11.26.8	rseq()	389
11.26.9	iterr()	390
11.26.10	rrseq()	391
11.27	Pseudo-functions related to N-dimensional hypercubic lists	392
11.27.1	ndseq()	392
11.27.2	ndmap()	393
11.27.3	ndfillraw()	395
11.28	Other pseudo-functions: bool(), togl(), protect(), qraw(), qint(), qfrac(), qfloat(), qrand(), random(), rbit()	395
11.29	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(), iquo(), irem(), gcd(), lcm(), max(), min(), `+`(), `*`(), all(), any(), xor(), len(), first(), last(), reversed(), if(), ifint(), ifone(), ifsgn(), nuple(), unpack(), flat() and zip()	397
11.30	User declared functions	410
11.30.1	\xintdeffunc, \xintdefiifunc, \xintdeffloatfunc	411
11.30.2	\xintdefufunc, \xintdefiifunc, \xintdeffloatufunc	414
11.30.3	\xintunassignexprfunc, \xintunassigniexprfunc, \xintunassignfloatexprfunc	415

11.30.4	<code>\xintNewFunction</code>	416
11.30.5	Mysterious stuff	417
11.30.6	<code>\XINT_expr_redefinemacros</code>	429
11.30.7	<code>\xintNewExpr</code> , <code>\xintNewIExpr</code> , <code>\xintNewFloatExpr</code> , <code>\xintNewIIExpr</code>	430
11.30.8	<code>\ifxintexprsafecatcodes</code> , <code>\xintexprSafeCatcodes</code> , <code>\xintexprRestoreCatcodes</code>	432

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 much easier 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 `\XINTfstop \XINTexprprint .{{<num1>}}{<num2>}...` uses only normal catcodes: 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 initially launched because 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 [2021/06/11 v1.4i Expandable expression parser (JFB)]%
59 \catcode`! 11
60 \let\XINT_Cmp \xintiiCmp
61 \def\XINTfstop{\noexpand\XINTfstop}%

```

11.5 *\xintDigits**, *\xintSetDigits**, *\xintreloadscilibs*

1.3f. 1.4e added some *\xintGuardDigits* and *\XINTdigitsex* mechanism but it was finally removed, due to pending issues of user interface, functionality, and documentation (the worst part) for whose resolution no time was left.

```

62 \def\xintreloadscilibs{\xintreloadxintlog\xintreloadxinttrig}%
63 \def\xintDigits {\futurelet\XINT_token\xintDigits_i}%
64 \def\xintDigits_i#1={\afterassignment\xintDigits_j\mathchardef\XINT_digits=}%
65 \def\xintDigits_j#1%
66 {%
67     \let\XINTdigits=\XINT_digits
68     \ifx*\XINT_token\expandafter\xintreloadscilibs\fi
69 }%

```

```

70 \let\xintfracSetDigits\xintSetDigits
71 \def\xintSetDigits#1#{\if\relax\detokenize{#1}\relax\expandafter\xintfracSetDigits
72     \else\expandafter\xintSetDigits_a\fi}%
73 \def\xintSetDigits_a#1%
74 {%
75     \mathchardef\XINT_digits=\numexpr#1\relax
76     \let\XINTdigits\XINT_digits
77     \xintreloadscilibs
78 }%

```

11.6 \XINTdigitsormax

1.4f. To not let `xintlog` and `xinttrig` work with, and produce, long mantissas exceeding the supported range for accuracy of the math functions. The official maximal value is 62, let's set the cut-off at 64.

A priori, no need for `\expandafter`, always ends up expanded in `\numexpr` (I saw also in an `\edef` in `xinttrig` as argument to `\xintReplicate` prior to its `\numexpr`).

```

79 \def\XINTdigitsormax{\ifnum\XINTdigits>\xint_c_ii^vi\xint_c_ii^vi\else\XINTdigits\fi}%

```

11.7 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.

11.7.1 Bracketed list rendering with prettifying of leaves from nested braced contents

1.4 The braces in `\XINT:expr:toblistwith` are there because there is an `\expanded` trigger.

1.4d: support for `poexpr 0.8` polynomial type.

```

80 \def\XINT:expr:toblistwith#1#2%
81 {%
82     {\expandafter\XINT:expr:toblist_checkempty
83     \expanded{\noexpand#1!\expandafter}\detokenize{#2}^}%
84 }%
85 \def\XINT:expr:toblist_checkempty #1!#2%
86 {%
87     \if ^#2\expandafter\xint_gob_til_^\else\expandafter\XINT:expr:toblist_a\fi
88     #1!#2%
89 }%
90 \catcode`< 1 \catcode`> 2 \catcode`{ 12 \catcode`} 12
91 \def\XINT:expr:toblist_a #1{#2%
92 <%
93     \if{#2\xint_dothis<[\XINT:expr:toblist_a>\fi
94     \if P#2\xint_dothis<[\XINT:expr:toblist_pol>\fi
95     \xint_orthat\XINT:expr:toblist_b #1#2%
96 >%
97 \def\XINT:expr:toblist_pol #1!#2.{#3}}%
98 <%
99     pol([\XINT:expr:toblist_b #1!#3]^)]\XINT:expr:toblist_c #1!}%
100 >%
101 \def\XINT:expr:toblist_b #1!#2}%
102 <%

```

```

103 \if\relax#2\relax\xintexprEmptyItem\else#1<#2>\fi\XINT:expr:toblist_c #1!}%
104 >%
105 \def\XINT:expr:toblist_c #1}#2%
106 <%
107 \if ^#2\xint_dothis<\xint_gob_til_>\fi
108 \if{#2\xint_dothis<, \XINT:expr:toblist_a>\fi
109 \xint_orthat<]\XINT:expr:toblist_c>#1#2%
110 >%
111 \catcode`{ 1 \catcode`} 2 \catcode`< 12 \catcode`> 12

```

11.7.2 Flattening nested braced contents

1.4b I hesitated whether using this technique or some variation of the method of the ListSel macros. I chose this one which I downscaled from tobliswith, I will revisit later. I only have a few minutes right now.

Call form is \expanded\XINT:expr:flatten

See \XINT_expr_func_flat. I hesitated with «flattened», but short names are faster parsed.

```

112 \def\XINT:expr:flatten#1%
113 {%
114 {{\expandafter\XINT:expr:flatten_checkempty\detokenize{#1}^}}%
115 }%
116 \def\XINT:expr:flatten_checkempty #1%
117 {%
118 \if ^#1\expandafter\xint_gobble_i\else\expandafter\XINT:expr:flatten_a\fi
119 #1%
120 }%
121 \begingroup % should I check lccode s generally if corrupted context at load?
122 \catcode`[ 1 \catcode`] 2 \lccode`[{ \lccode`]`
123 \catcode`< 1 \catcode`> 2 \catcode`{ 12 \catcode`} 12
124 \lowercase<\endgroup
125 \def\XINT:expr:flatten_a {#1%
126 <%
127 \if{#1\xint_dothis<\XINT:expr:flatten_a>\fi
128 \xint_orthat\XINT:expr:flatten_b #1%
129 >%
130 \def\XINT:expr:flatten_b #1}%
131 <%
132 [#1]\XINT:expr:flatten_c }%
133 >%
134 \def\XINT:expr:flatten_c }#1%
135 <%
136 \if ^#1\xint_dothis<\xint_gobble_i>\fi
137 \if{#1\xint_dothis<\XINT:expr:flatten_a>\fi
138 \xint_orthat<\XINT:expr:flatten_c>#1%
139 >%
140 >% back to normal catcodes

```

11.7.3 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 \xintexpralign tab nor \xintexpralign inner comma or \xintexpralign outer comma but \xintexpralign inner sep, etc...

At 1.4c I remove the `\protected from \xintexpralignend`. I had made note a year ago that it served nothing. Let's trust myself on this one (risky one year later!) .

```

141 \catcode`& 4
142 \protected\def\xintexpralignbegin      {\halign\bgroup\tabskip2ex\hfil##&&&\hfil\cr}%
143 \def\xintexpralignend                  {\crcr\egroup}%
144 \protected\def\xintexpralignlinesep    {\,\cr}%
145 \protected\def\xintexpralignleftbracket {[%
146 \protected\def\xintexpralignrightbracket{]}%
147 \protected\def\xintexpralignleftsep    {&}%
148 \protected\def\xintexpralignrightsep   {&}%
149 \protected\def\xintexpraligninnersep    {,&}%
150 \catcode`& 7
151 \def\XINT:expr:toalignwith#1#2%
152 {%
153     {\expandafter\XINT:expr:toalign_checkempty
154     \expanded{\noexpand#1!\expandafter}\detokenize{#2}^\expandafter}%
155     \xintexpralignend
156 }%
157 \def\XINT:expr:toalign_checkempty #1#2%
158 {%
159     \if ^#2\expandafter\xint_gob_til_^\else\expandafter\XINT:expr:toalign_a\fi
160     #1!#2%
161 }%
162 \catcode`< 1 \catcode`> 2 \catcode`{ 12 \catcode`} 12
163 \def\XINT:expr:toalign_a #1#2%
164 <%
165     \if{#2\xint_dothis<\xintexpralignleftbracket\XINT:expr:toalign_a>\fi
166     \xint_orthat<\xintexpralignleftsep\XINT:expr:toalign_b>#1#2%
167 >%
168 \def\XINT:expr:toalign_b #1#2}%
169 <%
170     \if\relax#2\relax\xintexprEmptyItem\else#1<#2>\fi\XINT:expr:toalign_c #1!}%
171 >%
172 \def\XINT:expr:toalign_c #1#2%
173 <%
174     \if ^#2\xint_dothis<\xint_gob_til_^\fi
175     \if {#2\xint_dothis<\xintexpraligninnersep\XINT:expr:toalign_A>\fi
176     \xint_orthat<\xintexpralignrightsep\xintexpralignrightbracket\XINT:expr:toalign_C>#1#2%
177 >%
178 \def\XINT:expr:toalign_A #1#2%
179 <%
180     \if{#2\xint_dothis<\xintexpralignleftbracket\XINT:expr:toalign_A>\fi
181     \xint_orthat\XINT:expr:toalign_b #1#2%
182 >%
183 \def\XINT:expr:toalign_C #1#2%
184 <%
185     \if ^#2\xint_dothis<\xint_gob_til_^\fi
186     \if {#2\xint_dothis<\xintexpralignlinesep\XINT:expr:toalign_a>\fi
187     \xint_orthat<\xintexpralignrightbracket\XINT:expr:toalign_C>#1#2%
188 >%
189 \catcode`{ 1 \catcode`} 2 \catcode`< 12 \catcode`> 12

```


11.7.4 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 inside an added brace pair:

```
\def\foo#1{\xintiRound{0}{#1}}
```

As the things will expand inside expanded, propagating expansion is not an issue.

This code is used by \xintexpr and \xintfloatexpr in case of optional argument and by the «Universal functions».

```
190 \def\XINT:expr:mapwithin#1#2%
191 {%
192   {\expandafter\XINT:expr:mapwithin_checkempty
193     \expanded{\noexpand#1!\expandafter}\detokenize{#2}^}%
194 }%
195 \def\XINT:expr:mapwithin_checkempty #1!#2%
196 {%
197   \if ^#2\expandafter\xint_gob_til_^\else\expandafter\XINT:expr:mapwithin_a\fi
198   #1!#2%
199 }%
200 \begingroup % should I check lccode s generally if corrupted context at load?
201 \catcode`[ 1 \catcode`] 2 \lccode`[{ \lccode`]` }
202 \catcode`< 1 \catcode`> 2 \catcode`{ 12 \catcode`} 12
203 \lowercase<\endgroup
204 \def\XINT:expr:mapwithin_a #1{#2%
205 <%
206   \if{#2\xint_dothis<[\iffalse]\fi\XINT:expr:mapwithin_a>\fi%
207   \xint_orthat\XINT:expr:mapwithin_b #1#2%
208 >%
209 \def\XINT:expr:mapwithin_b #1!#2}%
210 <%
211   #1<#2>\XINT:expr:mapwithin_c #1!}%
212 >%
213 \def\XINT:expr:mapwithin_c #1}{#2%
214 <%
215   \if ^#2\xint_dothis<\xint_gob_til_^\>\fi
216   \if{#2\xint_dothis<\XINT:expr:mapwithin_a>\fi%
217   \xint_orthat<[\iffalse[\fi]\XINT:expr:mapwithin_c>#1#2%
218 >%
219 >% back to normal catcodes
```

11.8 Top level user T_EX interface: \xinteval, \xintfloateval, \xintiieval

11.8.1	\xintexpr, \xintiexpr, \xintfloatexpr, \xintiieval	322
11.8.2	\XINT_expr_wrap, \XINT_iexpr_wrap, \XINT_flexpr_wrap	323
11.8.3	\XINTexprprint, \XINTiexprprint, \XINTiexprprint, \XINTflexprprint	323
11.8.4	\xintthe, \xintthealign, \xinttheexpr, \xinttheiexpr, \xintthefloatexpr, \xint-theiieval	324
11.8.5	\thexintexpr, \thexintiexpr, \thexintfloatexpr, \thexintiieval	325
11.8.6	\xintbareeval, \xintbarefloateval, \xintbareieval	325

11.8.7	<code>\xintthebareeval</code> , <code>\xintthebarefloateval</code> , <code>\xintthebareiieval</code>	325
11.8.8	<code>\xinteval</code> , <code>\xintieval</code> , <code>\xintfloateval</code> , <code>\xintiieval</code>	325
11.8.9	<code>\xintboolexpr</code> , <code>\XINT_boolexpr_print</code> , <code>\xinttheboolexpr</code> , <code>\thexintboolexpr</code> . . .	326
11.8.10	<code>\xintifboolexpr</code> , <code>\xintifboolfloatexpr</code> , <code>\xintifbooliiexpr</code>	326
11.8.11	<code>\xintifsgnexpr</code> , <code>\xintifsgnfloatexpr</code> , <code>\xintifsgniiexpr</code>	326
11.8.12	Small bits we have to put somewhere	326

11.8.1 `\xintexpr`, `\xintiexpr`, `\xintfloatexpr`, `\xintiieexpr`

`\xintiexpr` and `\xintfloatexpr` have an optional argument since 1.1.

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

Usage of `\xintiRound{0}` for `\xintiexpr` without optional `[D]` means that `\xintiexpr ... \relax` wrapper can be used to insert rounded-to-integers values in `\xintiieexpr` context: no post-fix `[0]` which would break it.

1.4a add support for the optional argument `[D]` for `\xintiexpr` being negative `D`, with same meaning as the 1.4a modified `\xintRound` from `xintfrac.sty`.

`\xintiexpr` mechanism was refactored at 1.4e so that rounding due to `[D]` optional argument uses raw format, not fixed point format on output, delegating fixed point conversion to an `\XINTiexprprint` now separated from `\XINTexprprint`.

In case of negative `[D]`, `\xintiexpr [D]... \relax` internally has the `[0]` post-fix so it can not be inserted as sub-expression in `\xintiieexpr` without a `num()` or `\xintiexpr ... \relax (extra) wrapper`.

```

220 \def\xintexpr      {\romannumeral0\xintexprpro    }%
221 \def\xintiexpr     {\romannumeral0\xintiexprpro   }%
222 \def\xintfloatexpr {\romannumeral0\xintfloatexprpro}%
223 \def\xintiieexpr   {\romannumeral0\xintiieexprpro }%
224 \def\xintexprpro   {\expandafter\XINT_expr_wrap\romannumeral0\xintbareeval }%
225 \def\xintiieexprpro {\expandafter\XINT_iieexpr_wrap\romannumeral0\xintbareiieval }%
226 \def\xintiexprpro #1%
227 {%
228   \ifx [#1\expandafter\XINT_iexpr_withopt\else\expandafter\XINT_iexpr_noopt
229   \fi #1%
230 }%
231 \def\XINT_iexpr_noopt
232 {%
233   \expandafter\XINT_iexpr_iiround\romannumeral0\xintbareeval
234 }%
235 \def\XINT_iexpr_iiround
236 {%
237   \expandafter\XINT_expr_wrap
238   \expanded
239   \XINT:NEhook:x:mapwithin\XINT:expr:mapwithin{\XINTiRoundzero_braced}%
240 }%
241 \def\XINTiRoundzero_braced#1{\xintiRound{0}{#1}}%
242 \def\XINT_iexpr_withopt [#1]%
243 {%
244   \expandafter\XINT_iexpr_round
245   \the\numexpr \xint_zapspaces #1 \xint_gobble_i\expandafter.%
246   \romannumeral0\xintbareeval
247 }%
248 \def\XINT_iexpr_round #1.%
249 {%

```

```

250 \ifnum#1=\xint_c_\xint_dothis{\XINT_iexpr_iiround}\fi
251 \xint_orthat{\XINT_iexpr_round_a #1.}%
252 }%
253 \def\XINT_iexpr_round_a #1.%
254 {%
255 \expandafter\XINT_iexpr_wrap
256 \expanded
257 \XINT:NEhook:x:mapwithin\XINT:expr:mapwithin{\XINTiRound_braced{#1}}%
258 }%
259 \def\XINTiRound_braced#1#2{{\xintiRound{#1}{#2}[\the\numexpr\ifnum#1<\xint_c_i0\else-#1\fi]}}%
260 \def\xintfloatexpro #1%
261 {%
262 \ifx [#1\expandafter\XINT_flexpr_withopt\else\expandafter\XINT_flexpr_noopt
263 \fi #1%
264 }%
265 \def\XINT_flexpr_noopt
266 {%
267 \expandafter\XINT_flexpr_wrap\the\numexpr\XINTdigits\expandafter.%
268 \romannumeral0\xintbarefloateval
269 }%
270 \def\XINT_flexpr_withopt [#1]%
271 {%
272 \expandafter\XINT_flexpr_withopt_a
273 \the\numexpr\xint_zapspaces #1 \xint_gobble_i\expandafter.%
274 \romannumeral0\xintbarefloateval
275 }%
276 \def\XINT_flexpr_withopt_a #1#2.%
277 {%
278 \expandafter\XINT_flexpr_withopt_b\the\numexpr\if#1-\XINTdigits\fi#1#2.%
279 }%
280 \def\XINT_flexpr_withopt_b #1.%
281 {%
282 \expandafter\XINT_flexpr_wrap
283 \the\numexpr#1\expandafter.%
284 \expanded
285 \XINT:NEhook:x:mapwithin\XINT:expr:mapwithin{\XINTinFloat_braced[#1]}%
286 }%
287 \def\XINTinFloat_braced[#1]#2{{\XINTinFloat[#1]{#2}}}%

```

11.8.2 \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.

1.4e has \XINT_iexpr_wrap separated from \XINT_expr_wrap, thus simplifying internal matters as output printer for \xintexpr will not have to handle fixed point input but only extended-raw type input (i.e. A, A/B, A[N] or A/B[N]).

```

288 \def\XINT_expr_wrap {\XINTfstop\XINTexprprint.}%
289 \def\XINT_iexpr_wrap {\XINTfstop\XINTiexprprint.}%
290 \def\XINT_iiexpr_wrap {\XINTfstop\XINTiiexprprint.}%
291 \def\XINT_flexpr_wrap {\XINTfstop\XINTflexprprint}%

```

11.8.3 \XINTexprprint, \XINTiexprprint, \XINTiiexprprint, \XINTflexprprint

Comments (still) currently under reconstruction.

1.4: this now requires `\expanded` context.

1.4e has a separate `\XINTiexprprint` and `\xintiexprPrintOne`.

1.4e has a breaking change of `\XINTfexprprint` and `\xintfloatexprPrintOne` which now requires `\xintfloatexprPrintOne[D]{x}` usage, with first argument in brackets.

```

292 \protected\def\XINTexprprint.%
293   {\XINT:NEhook:x:toblist\XINT:expr:toblistwith\xintexprPrintOne}%
294 \let\xintexprPrintOne\xintFracToSci
295 \protected\def\XINTiexprprint.%
296   {\XINT:NEhook:x:toblist\XINT:expr:toblistwith\xintiexprPrintOne}%
297 \let\xintiexprPrintOne\xintDecToString
298 \def\xintexprEmptyItem{[]}%
299 \protected\def\XINTiexprprint.%
300   {\XINT:NEhook:x:toblist\XINT:expr:toblistwith\xintiexprPrintOne}%
301 \let\xintiexprPrintOne\xint_firstofone
302 \protected\def\XINTfexprprint #1.%
303   {\XINT:NEhook:x:toblist\XINT:expr:toblistwith{\xintfloatexprPrintOne[#1]}}%
304 \let\xintfloatexprPrintOne\xintPFloat_wopt
305 \protected\def\XINTboolexprprint.%
306   {\XINT:NEhook:x:toblist\XINT:expr:toblistwith\xintboolexprPrintOne}%
307 \def\xintboolexprPrintOne#1{\xintiifNotZero{#1}{True}{False}}%

```

11.8.4 `\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...

The `\expanded` upfront ensures `\xintthe` mechanism does expand completely in two steps.

```

308 \def\xintthe      #1{\expanded\expandafter\xint_gobble_i\romannumeral`&&@#1}%
309 \def\xintthealign #1{\expanded\expandafter\xintexpralignbegin
310   \expanded\expandafter\XINT:expr:toalignwith
311   \romannumeral0\expandafter\expandafter\expandafter\expandafter
312   \expandafter\expandafter\expandafter\xint_gob_andstop_ii
313   \expandafter\xint_gobble_i\romannumeral`&&@#1}%
314 \def\xinttheexpr
315   {\expanded\expandafter\XINTexprprint\expandafter.\romannumeral0\xintbareeval}%
316 \def\xinttheiexpr

```

```

317   {\expanded\expandafter\xint_gobble_i\romannumeral`&&\xintiexpr}%
318 \def\xintthefloatexpr
319   {\expanded\expandafter\xint_gobble_i\romannumeral`&&\xintfloatexpr}%
320 \def\xinttheiexpr
321   {\expanded\expandafter\XINTiexprprint\expandafter.\romannumeral0\xintbareiieval}%

```

11.8.5 \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. I will remove them at 1.5.

```

322 \let\thexintexpr      \xintthexpr
323 \let\thexintiexpr     \xinttheiexpr
324 \let\thexintfloatexpr\xintthefloatexpr
325 \let\thexintiexpr     \xinttheiexpr

```

11.8.6 \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

```

326 \def\xintbareeval      {\XINT_expr_start }%
327 \def\xintbarefloateval{\XINT_flexpr_start}%
328 \def\xintbareiieval    {\XINT_iexpr_start}%

```

11.8.7 \xintthebareeval, \xintthebarefloateval, \xintthebareiieval

For matters of \XINT_NewFunc

```

329 \def\XINT_expr_unlock    {\expandafter\xint_firstofone\romannumeral`&&}%
330 \def\xintthebareeval     {\romannumeral0\expandafter\xint_stop_atfirstofone\romannumeral0\xintbareeval}%
331 \def\xintthebareiieval   {\romannumeral0\expandafter\xint_stop_atfirstofone\romannumeral0\xintbareiieval}%
332 \def\xintthebarefloateval{\romannumeral0\expandafter\xint_stop_atfirstofone\romannumeral0\xintbarefloateval}%
333 \def\xintthebareroundedfloateval
334 {%
335   \romannumeral0\expandafter\xintthebareroundedfloateval_a\romannumeral0\xintbarefloateval
336 }%
337 \def\xintthebareroundedfloateval_a
338 {%
339   \expandafter\xint_stop_atfirstofone
340   \expanded\XINT:NEhook:x:mapwithin\XINT:expr:mapwithin{\XINTinFloatSdigits_braced}%
341 }%
342 \def\XINTinFloatSdigits_braced#1{{\XINTinFloatS[\XINTdigits]{#1}}}%

```

11.8.8 \xinteval, \xintieval, \xintfloateval, \xintiieval

Refactored at 1.4.

The \expanded upfront ensures \xinteval still expands completely in two steps. No \romannumeral trigger here, in relation to the fact that \XINTexprprint is no f-expandable, only e-expandable.

(and attention that \xintexpr\relax is now legal, and an empty ope can be produced in output also from \xintexpr [17][1]\relax for example)

```

343 \def\xinteval #1%
344   {\expanded\expandafter\XINTexprprint\expandafter.\romannumeral0\xintbareeval#1\relax}%
345 \def\xintieval #1%

```

```

346   {\expanded\expandafter\xint_gobble_i\romannumeral`&&\xintiexpr#1\relax}%
347 \def\xintfloateval #1%
348   {\expanded\expandafter\xint_gobble_i\romannumeral`&&\xintfloatexpr#1\relax}%
349 \def\xintiieval #1%
350   {\expanded\expandafter\XINTiexprprint\expandafter.\romannumeral0\xintbareieval#1\relax}%

```

11.8.9 \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.

```

351 \def\xintboolexpr
352 {%
353   \romannumeral0\expandafter\XINT_boolexpr_done\romannumeral0\xintexpr
354 }%
355 \def\XINT_boolexpr_done #1.{\XINTfstop\XINTboolexprprint.}%
356 \def\xinttheboolexpr
357 {%
358   \expanded\expandafter\XINTboolexprprint\expandafter.\romannumeral0\xintbareeval
359 }%
360 \let\thexintboolexpr\xinttheboolexpr

```

11.8.10 \xintifboolexpr, \xintifboolfloatexpr, \xintifboolliexpr

They do not accept comma separated expressions input.

```

361 \def\xintifboolexpr      #1{\romannumeral0\xintiiifnotzero {\xinttheexpr #1\relax}}%
362 \def\xintifboolfloatexpr #1{\romannumeral0\xintiiifnotzero {\xintthefloatexpr #1\relax}}%
363 \def\xintifboolliexpr    #1{\romannumeral0\xintiiifnotzero {\xinttheiiexpr #1\relax}}%

```

11.8.11 \xintifsgnexpr, \xintifsgnfloatexpr, \xintifsgniiexpr

1.3d (2019/01/06).

They do not accept comma separated expressions.

```

364 \def\xintifsgnexpr      #1{\romannumeral0\xintiiifsgn {\xinttheexpr #1\relax}}%
365 \def\xintifsgnfloatexpr #1{\romannumeral0\xintiiifsgn {\xintthefloatexpr #1\relax}}%
366 \def\xintifsgniiexpr    #1{\romannumeral0\xintiiifsgn {\xinttheiiexpr #1\relax}}%

```

11.8.12 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...).

```
367 \def\xINT_embrace#1{{#1}}%
368 \def\xint_gob_til_! #1!{}% ! with catcode 11
369 \def\xintError:noopening
370 {%
371     \XINT_expandableerror{Extra }. This is serious and prospects are bleak.}%
372 }%
```

`\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!

```
373 \def\xintthecoords#1%
374     {\romannumeral`&&\expandafter\xINT_thecoords_a\romannumeral0#1}%
375 \def\xINT_thecoords_a #1#2.#3% #2.=\XINTfloatprint<digits>. etc...
376     {\expanded{\expandafter\xINT_thecoords_b\expanded#2.{#3},!,!,^}}%
377 \def\xINT_thecoords_b #1#2,#3#4,%
378     {\xint_gob_til_! #3\xINT_thecoords_c ! (#1#2, #3#4)\XINT_thecoords_b }%
379 \def\xINT_thecoords_c #1^{}%%
```

`\xintthespaceseparated` 1.4a This is a utility macro which was distributed previously separately for usage with PSTricks `\listplot`

```
380 \def\xintthespaceseparated#1%
381     {\expanded\expandafter\xintthespaceseparated_a\romannumeral0#1}%
382 \def\xintthespaceseparated_a #1#2.#3%
383     {{\expandafter\xintthespaceseparated_b\expanded#2.{#3},!,!,!,!,!,!,!,!,^}}%
384 \def\xintthespaceseparated_b #1,#2,#3,#4,#5,#6,#7,#8,#9,%
385     {\xint_gob_til_! #9\xintthespaceseparated_c !%
386         #1#2#3#4#5#6#7#8#9%
387         \xintthespaceseparated_b}%%
```

1.4c I add a space here to stop the `\romannumeral`&&` in case of empty input. But this space induces an extra un-needed space token after 9, 18, 27,... items before the last group of less than 9 items.

Fix (at 1.4h) is simple because I already use `\expanded` anyhow: I don't need at all the `\romannumeral`&&` which was first in `\xintthespaceseparated`, let's move the first `\expanded` which was in `\xintthespaceseparated_a` to `\xintthespaceseparated`, and remove the extra space here in `_c`.

(alternative would have been to put the space after `#1` and accept a systematic trailing space, at least it is more aesthetic).

Again, I did have a test file, but it was not incorporated in my test suite, so I discovered the problem accidentally by compiling all files in an archive.

```
388 \def\xintthespaceseparated_c !#1!#2^{#1}%%
```


11.9 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».

```
389 \let\xint:NEhook:f:one:from:one\expandafter
390 \let\xint:NEhook:f:one:from:one:direct\empty
391 \let\xint:NEhook:f:one:from:two\expandafter
392 \let\xint:NEhook:f:one:from:two:direct\empty
393 \let\xint:NEhook:x:one:from:two\empty
394 \let\xint:NEhook:f:one:and:opt:direct \empty
395 \let\xint:NEhook:f:tacitzeroifone:direct \empty
396 \let\xint:NEhook:f:iitacitzeroifone:direct \empty
397 \let\xint:NEhook:x:select:obey\empty
398 \let\xint:NEhook:x:listsell\empty
399 \let\xint:NEhook:f:reverse\empty
```

At 1.4 it was `\def\xint:NEhook:f:from:delim:u #1#2^{#1#2}` which was trick to allow automatic unpacking of a nutple argument to multi-arguments functions such as `gcd()` or `max()`. But this sacrificed the usage with a single numeric argument.

1.4i (2021/06/11).

More sophisticated code to check if the argument ople was actually a single number. Notice that this forces numeric types to actually use catcode 12 tokens, and `polexpr` diverges a bit using `P`, but actually always testing with `\if not \ifx`.

This is used by `gcd()`, `lcm()`, `max()`, `min()`, ``+`()`, ``*`()`, `all()`, `any()`, `xor()`.

The `nil` and `None` will give the same result due to the initial brace stripping done by `\xint:NEhook:f:from:delim:` (there was even a prior brace stripping to provide the `#2` which is empty here for the `nil` and `{}` for the `None`).

```
400 \def\xint:NEhook:f:from:delim:u #1#2^%
401 {%
402   \expandafter\xint_foof_checkifnumber\expandafter#1\string#2^%
403 }%
404 \def\xint_foof_checkifnumber#1#2%
405 {%
406   \expandafter#1%
407   \romannumeral0\expanded{\if ^#2^else
408     \if\bgroup#2\noexpand\xint_foof_no\else
409     \noexpand\xint_foof_yes#2\fi\fi}%
410 }%
411 \def\xint_foof_yes#1^{{#1}^}%
412 \def\xint_foof_no{\expandafter{\iffalse}\fi}%

```

1.4i (2021/06/11).

Same changes as for the other multiple arguments functions, making them again usable with a single numeric input.

Was at 1.4 `\def\xint:NEhook:f:noeval:from:braced:u#1#2^{#1{#2}}` which is not compatible with a single numeric input.

Used by `len()`, `first()`, `last()` but it is a potential implementation bug that the three share this as the location where expansion takes places is one level deeper for the support macro of `len()`.

The `None` is here handled as `nil`, i.e. it is unpacked, which is fine as the documentations says nutples are unpacked.

This is somewhat complicated here due to the macro interfaces of `\xintFirstOne`, `\xintLastOne` and `\xintLength` which used a braced not a delimited argument, this forces here some re-grab which adds overhead.

To avoid this one would like to move the check to inside {#2}, but then there is a difficulty caused by the nil (or None), if we apply a string technique as it will hit the closing brace.

```

413 \def\XINT:NEhook:f:LFL #1{\expandafter#1\expandafter}%
414 \def\XINT:NEhook:r:check #1^%
415 {%
416   \expandafter\XINT:NEhook:r:check_a\string#1^%
417 }%
418 \let\XINT:NEsaved:r:check \XINT:NEhook:r:check
419 \def\XINT:NEhook:r:check_a #1%
420 {%
421   \if ^#1\xint_dothis\xint_c_\fi
422   \if bgroupp#1\xint_dothis\XINT:NEhook:r:check_no\fi
423   \xint_orthat{\XINT:NEhook:r:check_yes#1}%
424 }%
425 \def\XINT:NEhook:r:check_no
426 {%
427   \expandafter\XINT:NEhook:r:check_no_b
428   \expandafter\xint_c_\expandafter{\iffalse}\fi
429 }%
430 \def\XINT:NEhook:r:check_no_b#1^{#1}%
431 \def\XINT:NEhook:r:check_yes#1^{\xint_c_{#1}}%
432 \let\XINT:NEhook:branch\expandafter
433 \let\XINT:NEhook:seqx\empty
434 \let\XINT:NEhook:iter\expandafter
435 \let\XINT:NEhook:opx\empty
436 \let\XINT:NEhook:rseq\expandafter
437 \let\XINT:NEhook:iterr\expandafter
438 \let\XINT:NEhook:rrseq\expandafter
439 \let\XINT:NEhook:x:toblist\empty
440 \let\XINT:NEhook:x:mapwithin\empty
441 \let\XINT:NEhook:x:ndmapx\empty

```

11.10 \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_getop 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_getop. 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.

The \expandafter chain in \XINT_expr_put_op_first is an overhead related to an 1.4 attempt, the "varvalue" mechanism. I.e.: expansion of \XINT_expr_var_foo is {\XINT_expr_varvalue_foo } and then for example \XINT_expr_varvalue_foo expands to {4/1[0]}. The mechanism was originally conceived to have only one token with idea its makes things faster. But the xintfrac macros break

with syntax such as `\xintMul\foo\bar` and `\foo` expansion giving braces. So at 1.4c I added here these `\expandafter`, but this is REALLY not satisfactory because the `\expandafter` are needed it seems only for this variable "varvalue" mechanism.

See also the discussion of `\XINT_expr_op__` which distinguishes variables from functions.

After a 1.4g refactoring it would be possible to drop here the `\expandafter` if the `\XINT_expr_var_foo` macro was defined to f-expand to {actual expanded value (as ople)} for example `explicit {{3}}`. I have to balance the relative weights of doing always the `\expandafter` but they are needed only for the case the value was encapsulated in a variable, and of never doing the `\expandafter` and ensure f-expansion of the `_var_foo` gives explicit value (now that the refactoring let it be f-expanded, and the case of fake variables omit and abort in particular was safely separated instead of being treated like other and imposing restrictions on general variable handling), and then there is the overhead of possibly moving around many digits in the #1 of `\XINT_expr_put_op_first`.

```

442 \def\XINT_expr_getnext #1%
443 {%
444   \expandafter\XINT_expr_put_op_first\romannumeral`&&@%
445   \expandafter\XINT_expr_getnext_a\romannumeral`&&@#1%
446 }%
447 \def\XINT_expr_put_op_first #1#2#3{\expandafter#2\expandafter#3\expandafter{#1}}%
448 \def\XINT_expr_getnext_a #1%
449 {%
450   \ifx\relax #1\xint_dothis\XINT_expr_foundprematureend\fi
451   \ifx\XINTfstop#1\xint_dothis\XINT_expr_subexpr\fi
452   \ifcat\relax#1\xint_dothis\XINT_expr_countetc\fi
453   \xint_orthat{}\XINT_expr_getnextfork #1%
454 }%
455 \def\XINT_expr_foundprematureend\XINT_expr_getnextfork #1{{}\xint_c_\relax}%
456 \def\XINT_expr_subexpr #1.#2%
457 {%
458   \expanded{\unexpanded{{#2}}\expandafter}\romannumeral`&&\XINT_expr_getop
459 }%

```

1.2 adds `\ht`, `\dp`, `\wd` and the eTeX font things. 1.4 avoids big nested `\if`'s, simply for code readability.

This "fetch as number" is dangerous as long as list is not complete... at 1.4g I belatedly add `\catcode`

```

460 \def\XINT_expr_countetc\XINT_expr_getnextfork#1%
461 {%
462   \if0\ifx\count#11\fi
463   \ifx\numexpr#11\fi
464   \ifx\catcode#11\fi
465   \ifx\dimen#11\fi
466   \ifx\dimexpr#11\fi
467   \ifx\skip#11\fi
468   \ifx\glueexpr#11\fi
469   \ifx\fontdimen#11\fi
470   \ifx\ht#11\fi
471   \ifx\dp#11\fi
472   \ifx\wd#11\fi
473   \ifx\fontcharht#11\fi
474   \ifx\fontcharwd#11\fi
475   \ifx\fontchardp#11\fi
476   \ifx\fontcharic#11\fi
477   0\expandafter\XINT_expr_fetch_as_number\fi

```

```

478 \expandafter\XINT_expr_getnext_a\number #1%
479 }%
480 \def\XINT_expr_fetch_as_number
481 \expandafter\XINT_expr_getnext_a\number #1%
482 {%
483 \expanded{{{ \number#1}}\expandafter}\romannumeral`&&\XINT_expr_getop
484 }%

```

This is the key initial dispatch component. It has been refactored at 1.4g to give priority to identifying letter and digit tokens first. It thus combines former `\XINT_expr_getnextfork`, `\XINT_expr_scan_nbr_or_func` and `\XINT_expr_scanfunc`. A branch of the latter having become `\XINT_expr_startfunc`. The handling of non-catcode 11 underscore `_` has changed: it is now skipped completely like the `+`. Formerly it would cause an infinite loop because it triggered first insertion of a nil variable, (being confused with a possible operator at a location where one looks for a value), then tacit multiplication (being now interpreted as starting some name), and then it came back to `getnextfork` creating loop. The `@` of catcode 12 could have caused the same issue if it was not handled especially because it is used in the syntax as special variable for recursion hence was recognized even if of catcode 12. Anyway I could have handled the `_` like the `@`, to avoid this problem of infinite loop with a non-letter underscore used as first character but decided finally to have it be ignored (it is already ignored if among digits, but it can be a constituent of a function of variable name). It is not ignored of course if of catcode 11. It may then start a variable or function name, but only for use by the package (by `polexpr` for example), not by users.

Then the matter is handed over to specialized routines: gathering digits of a number (inclusive of a decimal mark, an exponential part) or letters of a function or variable. And we have to intercept some tokens to implement various functionalities.

In each `dothis/orthat` structure, the first encountered branches are usually handled slower than the next, because `\if..\fi` test cost less than grabbing tokens. The exception is in the first one where letters pass through slightly faster than digits, presumably because the `\ifnum` test is more costly. Prior to this 1.4g refactoring the case of a starting letter of a variable or function name was handled last, it is now handled first. Now, this is only first letter...

Here are the various possibilities in order that they are appear below (the indicative order of speed of treatment is given as a number).

- 1 tokens of catcode letter start a variable or function name
- 2 digits (I apply `\string` for the test, but I will have to review, it seems natural anyhow to require digits to be of catcode 12 and this is in fact basically done by the package, `\numexpr` does not work if not the case.),
- 7 support for Python-like `*` "unpacking" unary operator (added at 1.4),
- 6 support for `[` as opener for the `[..]` nutple constructor (1.4),
- 5 support for the minus as unary operator of variable precedence,
- 4 support for `@` as first character of special variables even if not letter,
- 3 support for opening parentheses (possibly triggering tacit multiplication),
- 13 support for skipping over ignored `+` character,
- 12 support for numbers starting with a decimal point,
- 11 support for the ``+`()` and ``*`()` functions,
- 10 support for the `!()` function,
- 9 support for the `?()` function,
- 8 support for `"` for input of hexadecimal numbers. But [xintbinhex](#) must be loaded explicitly by user.
- 17 support for `\xintdeffunc` via special handling of `#` token,
- 16 support for ignoring `_` if not of catcode 11 and at start of numbers or names (this 1.4g change fixes `\xinteval{4}` creating infinite loop)
- 15 support for inserting "nil" in front of operators, as needed in particular for the Python slicing syntax. This covers the comma, the `:`, the `]` and the `)` and also the `;` although I don't think

using ; to delimit nil is licit.

-14 support for inserting 0 as missing value if / or ^ are encountered directly. This 1.4g changes avoids \xinteval{/3} causing unrecoverable low level errors from \xintDiv receiving only one argument.

I did not see here other bad syntax to protect.

The handling of "nil" insertion penalizes Python slicing but anyway time differences in the 14-15-16-17 group are less than 5%. The alternative will be to do some positive test for the targets (:,], the comma and closing parenthesis) and do this in the prior group but this then penalizes others. Anyway. This is all negligible compared to actual computations...

Note: the above may not be in sync with code as it is extremely time-consuming to maintain correspondence in case of re-factoring.

```

485 \def\xINT_expr_getnextfork #1%
486 {%
487   \ifcat a#1\xint_dothis\xINT_expr_startfunc\fi
488   \ifnum \xint_c_ix<1\string#1 \xint_dothis\xINT_expr_startint\fi
489   \xint_orthat\xINT_expr_getnextfork_a #1%
490 }%
491 \def\xINT_expr_getnextfork_a #1%
492 {%
493   \if#1*\xint_dothis {{}}\xint_c_ii^v 0}\fi
494   \if#1[\xint_dothis {{}}\xint_c_ii^v \XINT_expr_itself_obracket}\fi
495   \if#1-\xint_dothis {{}}{-}\fi
496   \if#1@\xint_dothis{\XINT_expr_startfunc @}\fi
497   \if#1(\xint_dothis {{}}\xint_c_ii^v {})\fi
498   \xint_orthat{\XINT_expr_getnextfork_b#1}%
499 }%
500 \catcode96 11 % `
501 \def\xINT_expr_getnextfork_b #1%
502 {%
503   \if#1+\xint_dothis \XINT_expr_getnext_a\fi
504   \if#1.\xint_dothis \XINT_expr_startdec\fi
505   \if#1`\xint_dothis {\XINT_expr_onliteral_`}\fi
506   \if#1!\xint_dothis {\XINT_expr_startfunc !}\fi
507   \if#1?\xint_dothis {\XINT_expr_startfunc ?}\fi
508   \if#1"\xint_dothis \XINT_expr_starthex\fi
509   \xint_orthat{\XINT_expr_getnextfork_c#1}%
510 }%
511 \def\xINT_tmpa #1{%
512 \def\xINT_expr_getnextfork_c ##1%
513 {%
514   \if##1#1\xint_dothis \XINT_expr_getmacropar\fi
515   \if##1_\xint_dothis \XINT_expr_getnext_a\fi
516   \if0\if##1/1\fi\if##1^1\fi0\xint_dothis{\XINT_expr_insertnil##1}\fi
517   \xint_orthat{\XINT_expr_missing_arg##1}%
518 }%
519 }\expandafter\xINT_tmpa\string#%

```

The ` syntax is here used for special constructs like `+`(..), `*`(..) where + or * will be treated as functions. Current implementation picks only one token (could have been braced stuff), here it will be + or *, and via \XINT_expr_op_` this then becomes a suitable \XINT_{expr|iiexpr|fexpr}_func_+ (or *). Documentation says to use `+`(...), but `+(...) is also valid. The opening parenthesis must be there, it is not allowed to require some expansion.

```

520 \def\xINT_expr_onliteral_` #1#2({{#1}\xint_c_ii^v `}%

```

521 \catcode96 12 % `

Prior to 1.4g, I was using a \lowercase technique to insert the catcode 12 #, but this is a bit risky when one does not ensure a priori control of all lccodes.

522 \def\XINT_tmpa #1{%

523 \def\XINT_expr_getmacropar ##1%

524 {%

525 \expandafter{\expandafter{\expandafter#1\expandafter

526 ##1\expandafter}\expandafter}\romannumeral`&&\XINT_expr_getop

527 }%

528 }\expandafter\XINT_tmpa\string#%

529 \def\XINT_expr_insertnil #1%

530 {%

531 \expandafter{\expandafter}\romannumeral`&&\XINT_expr_getop_a#1%

532 }%

533 \def\XINT_expr_missing_arg#1%

534 {%

535 \expanded{\XINT_expandableerror{Expected a value, got nothing before `#1'. Inserting 0.}\{{0}}\expand

536 \romannumeral`&&\XINT_expr_getop_a#1%

537 }%

11.11 \XINT_expr_startint

11.11.1	Integral part (skipping zeroes)	334
11.11.2	Fractional part	336
11.11.3	Scientific notation	337
11.11.4	Hexadecimal numbers	338
11.11.5	\XINT_expr_startfunc: collecting names of functions and variables	341
11.11.6	\XINT_expr_func: dispatch to variable replacement or to function execution	341

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.

Attention at this location #1 was of catcode 12 in all versions prior to 1.4.

We assume anyhow that catcodes of digits are 12...

```

538 \def\XINT_expr_startint #1%
539 {%
540     \if #10\expandafter\XINT_expr_gobz_a\else\expandafter\XINT_expr_scanint_a\fi #1%
541 }%
542 \def\XINT_expr_scanint_a #1#2%
543     {\expanded\bgroup{{\iffalse}}\fi #1% spare a \string
544     \expandafter\XINT_expr_scanint_main\romannumeral`&&@#2}%
545 \def\XINT_expr_gobz_a #1#2%
546     {\expanded\bgroup{{\iffalse}}\fi
547     \expandafter\XINT_expr_gobz_scanint_main\romannumeral`&&@#2}%
548 \def\XINT_expr_startdec #1%
549     {\expanded\bgroup{{\iffalse}}\fi
550     \expandafter\XINT_expr_scandec_a\romannumeral`&&@#1}%

```

11.11.1 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.

(reference to `\string` is obsolete: it is only used in the test but the tokens are not submitted to `\string` anymore)

```

551 \def\XINT_expr_scanint_main #1%
552 {%
553     \ifcat \relax #1\expandafter\XINT_expr_scanint_hit_cs \fi
554     \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanint_next\fi
555     #1\XINT_expr_scanint_again
556 }%
557 \def\XINT_expr_scanint_again #1%
558 {%
559     \expandafter\XINT_expr_scanint_main\romannumeral`&&@#1%
560 }%

```

1.4f had `_getop` here, but let's jump directly to `_getop_a`.

```

561 \def\XINT_expr_scanint_hit_cs \ifnum#1\fi#2\XINT_expr_scanint_again
562 {%
563     \iffalse{{{\fi}}\expandafter}\romannumeral`&&@\XINT_expr_getop_a#2%
564 }%

```

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 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.

1.4f had `_getop` here, but let's jump directly to `_getop_a`.

```
565 \def\xINT_expr_scanint_next #1\xINT_expr_scanint_again
566 {%
567   \if    [#1\xint_dothis\xINT_expr_rawxintfrac\fi
568   \if    _#1\xint_dothis\xINT_expr_scanint_again\fi
569   \if    e#1\xint_dothis{[\the\numexpr0\xINT_expr_scanexp_a +}\fi
570   \if    E#1\xint_dothis{[\the\numexpr0\xINT_expr_scanexp_a +}\fi
571   \if    .#1\xint_dothis{\XINT_expr_startdec_a .}\fi
572   \xint_orthat
573   {\iffalse{ {\fi}}\expandafter}\romannumeral`&&\XINT_expr_getop_a#1}%
574 }%
575 \def\xINT_expr_rawxintfrac
576 {%
577   \iffalse{ {\fi}}\expandafter}\csname XINT_expr_precedence_&&\endcsname&&%
578 }%
579 \def\xINT_expr_gobz_scanint_main #1%
580 {%
581   \ifcat \relax #1\expandafter\xINT_expr_gobz_scanint_hit_cs\fi
582   \ifnum\xint_c_x<1|string#1 \else\expandafter\xINT_expr_gobz_scanint_next\fi
583   #1\xINT_expr_scanint_again
584 }%
585 \def\xINT_expr_gobz_scanint_again #1%
586 {%
587   \expandafter\xINT_expr_gobz_scanint_main\romannumeral`&&#1%
588 }%
```

1.4f had `_getop` here, but let's jump directly to `_getop_a`.

```
589 \def\xINT_expr_gobz_scanint_hit_cs\ifnum#1\fi#2\xINT_expr_scanint_again
590 {%
591   0\iffalse{ {\fi}}\expandafter}\romannumeral`&&\XINT_expr_getop_a#2%
592 }%
593 \def\xINT_expr_gobz_scanint_next #1\xINT_expr_scanint_again
594 {%
595   \if    [#1\xint_dothis{\expandafter0\xINT_expr_rawxintfrac}\fi
596   \if    _#1\xint_dothis\xINT_expr_gobz_scanint_again\fi
597   \if    e#1\xint_dothis{0[\the\numexpr0\xINT_expr_scanexp_a +}\fi
598   \if    E#1\xint_dothis{0[\the\numexpr0\xINT_expr_scanexp_a +}\fi
599   \if    .#1\xint_dothis{\XINT_expr_gobz_startdec_a .}\fi
600   \if    0#1\xint_dothis\xINT_expr_gobz_scanint_again\fi
601   \xint_orthat
602   {0\iffalse{ {\fi}}\expandafter}\romannumeral`&&\XINT_expr_getop_a#1}%
603 }%
```


11.11.2 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 ?

1.4f had _getop here, but let's jump directly to _getop_a.

```

604 \def\XINT_expr_startdec_a .#1%
605 {%
606   \expandafter\XINT_expr_scandec_a\romannumeral`&&@#1%
607 }%
608 \def\XINT_expr_scandec_a #1%
609 {%
610   \if .#1\xint_dothis{\iffalse{ {\fi}}\expandafter}%
611   \romannumeral`&&\XINT_expr_getop_a..}\fi
612   \xint_orthat {\XINT_expr_scandec_main 0.#1}%
613 }%
614 \def\XINT_expr_gobz_startdec_a .#1%
615 {%
616   \expandafter\XINT_expr_gobz_scandec_a\romannumeral`&&@#1%
617 }%
618 \def\XINT_expr_gobz_scandec_a #1%
619 {%
620   \if .#1\xint_dothis
621   {0\iffalse{ {\fi}}\expandafter}\romannumeral`&&\XINT_expr_getop_a..}\fi
622   \xint_orthat {\XINT_expr_gobz_scandec_main 0.#1}%
623 }%
624 \def\XINT_expr_scandec_main #1.#2%
625 {%
626   \ifcat \relax #2\expandafter\XINT_expr_scandec_hit_cs\fi
627   \ifnum\xint_c_ix<1|string#2 \else\expandafter\XINT_expr_scandec_next\fi
628   #2\expandafter\XINT_expr_scandec_again\the\numexpr #1-\xint_c_i.%
629 }%
630 \def\XINT_expr_scandec_again #1.#2%
631 {%
632   \expandafter\XINT_expr_scandec_main
633   \the\numexpr #1\expandafter.\romannumeral`&&@#2%
634 }%

```

1.4f had _getop here, but let's jump directly to _getop_a.

```

635 \def\XINT_expr_scandec_hit_cs\ifnum#1\fi
636   #2\expandafter\XINT_expr_scandec_again\the\numexpr#3-\xint_c_i.%
637 {%
638   [#3]\iffalse{ {\fi}}\expandafter}\romannumeral`&&\XINT_expr_getop_a#2%
639 }%
640 \def\XINT_expr_scandec_next #1#2\the\numexpr#3-\xint_c_i.%
641 {%
642   \if _#1\xint_dothis{\XINT_expr_scandec_again#3.}\fi
643   \if e#1\xint_dothis{[\the\numexpr#3\XINT_expr_scanexp_a +}\fi
644   \if E#1\xint_dothis{[\the\numexpr#3\XINT_expr_scanexp_a +}\fi
645   \xint_orthat
646   {[#3]\iffalse{ {\fi}}\expandafter}\romannumeral`&&\XINT_expr_getop_a#1}%

```



```

647 }%
648 \def\XINT_expr_gobz_scandec_main #1.#2%
649 {%
650   \ifcat \relax #2\expandafter\XINT_expr_gobz_scandec_hit_cs\fi
651   \ifnum\xint_c_ix<1\string#2 \else\expandafter\XINT_expr_gobz_scandec_next\fi
652   \if0#2\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
653   {\expandafter\XINT_expr_gobz_scandec_main}%
654   {#2\expandafter\XINT_expr_scandec_again}\the\numexpr#1-\xint_c_i.%
655 }%

```

1.4f had `_getop` here, but let's jump directly to `_getop_a`.

```

656 \def\XINT_expr_gobz_scandec_hit_cs \ifnum#1\fi\if0#2#3\xint_c_i.%
657 {%
658   0[0]\iffalse{{\fi}}\expandafter\romannumeral`&&\XINT_expr_getop_a#2%
659 }%
660 \def\XINT_expr_gobz_scandec_next\if0#1#2\fi #3\numexpr#4-\xint_c_i.%
661 {%
662   \if _#1\xint_dothis{\XINT_expr_gobz_scandec_main #4.}\fi
663   \if e#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
664   \if E#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +}\fi
665   \xint_orthat
666   {0[0]\iffalse{{\fi}}\expandafter\romannumeral`&&\XINT_expr_getop_a#1}%
667 }%

```

11.11.3 Scientific notation

Some pluses and minuses are allowed at the start of the scientific part, however not later, and no parenthesis.

ATTENTION! `1e\numexpr2+3\relax` or `1e\xintexpr i\relax`, `i=1..5` are not allowed and `1e1\numexpr2\relax` does `1e1 * \numexpr2\relax`. Use `\the\numexpr`, `\xinttheiexpr`, etc...

```

668 \def\XINT_expr_scanexp_a #1#2%
669 {%
670   #1\expandafter\XINT_expr_scanexp_main\romannumeral`&&#2%
671 }%
672 \def\XINT_expr_scanexp_main #1%
673 {%
674   \ifcat \relax #1\expandafter\XINT_expr_scanexp_hit_cs\fi
675   \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanexp_next\fi
676   #1\XINT_expr_scanexp_again
677 }%
678 \def\XINT_expr_scanexp_again #1%
679 {%
680   \expandafter\XINT_expr_scanexp_main_b\romannumeral`&&#1%
681 }%

```

1.4f had `_getop` here, but let's jump directly to `_getop_a`.

```

682 \def\XINT_expr_scanexp_hit_cs\ifnum#1\fi#2\XINT_expr_scanexp_again
683 {%
684   ]\iffalse{{\fi}}\expandafter\romannumeral`&&\XINT_expr_getop_a#2%
685 }%
686 \def\XINT_expr_scanexp_next #1\XINT_expr_scanexp_again
687 {%
688   \if _#1\xint_dothis \XINT_expr_scanexp_again \fi

```

```

689 \if +#1\xint_dothis {\XINT_expr_scanexp_a +}\fi
690 \if -#1\xint_dothis {\XINT_expr_scanexp_a -}\fi
691 \xint_orthat
692 {} \iffalse{{\fi}}\expandafter\romannumeral`&&\XINT_expr_getop_a#1}%
693 }%
694 \def\XINT_expr_scanexp_main_b #1%
695 {%
696 \ifcat \relax #1\expandafter\XINT_expr_scanexp_hit_cs_b\fi
697 \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanexp_next_b\fi
698 #1\XINT_expr_scanexp_again_b
699 }%

```

1.4f had `_getop` here, but let's jump directly to `_getop_a`.

```

700 \def\XINT_expr_scanexp_hit_cs_b\ifnum#1\fi#2\XINT_expr_scanexp_again_b
701 {%
702 } \iffalse{{\fi}}\expandafter\romannumeral`&&\XINT_expr_getop_a#2%
703 }%
704 \def\XINT_expr_scanexp_again_b #1%
705 {%
706 \expandafter\XINT_expr_scanexp_main_b\romannumeral`&&#1%
707 }%
708 \def\XINT_expr_scanexp_next_b #1\XINT_expr_scanexp_again_b
709 {%
710 \if _#1\xint_dothis\XINT_expr_scanexp_again\fi
711 \xint_orthat
712 {} \iffalse{{\fi}}\expandafter\romannumeral`&&\XINT_expr_getop_a#1}%
713 }%

```

11.11.4 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.

(some above remarks have been obsoleted for some long time, no more applied `\string` since 1.4)

Notice that internal representation adds a `[N]` part only in case input used "DDD.dddd form, for compatibility with `\xintiexpr` which is not compatible with such internal representation.

At 1.4g a very long-standing bug was fixed: input such as `"\foo` broke the parser because (incredibly) the `\foo` token was picked up unexpanded and ended up as is in an `\ifcat` !

Another long-standing bug was fixed at 1.4g: contrarily to the decimal case, here in the hexadecimal input leading zeros were not trimmed. This was ok, because formerly `\xintHexToDec` trimmed leading zeros, but at 1.2m 2017/07/31 `xintbinhex.sty` was modified and this ceased being the case. But I forgot to upgrade the parser here at that time. Leading zeros would in many circumstances (presence of a fractional part, or `\xintiexpr` context) lead to wrong results. Leading zeros are now trimmed during input.

```

714 \def\XINT_expr_hex_in #1.#2#3;%
715 {%
716 \expanded{{\if#2>%
717 \xintHexToDec{#1}%
718 \else
719 \xintiiMul{\xintiiPow{625}{\xintLength{#3}}}{\xintHexToDec{#1#3}}}%

```

```

720      [\the\numexpr-4*\xintLength{#3}]]%
721      \fi}}\expandafter}\romannumeral`&&\XINT_expr_getop
722 }%

    Let's not forget to grab-expand next token first as is normal rule of operation. Formerly called
    \XINT_expr_scanhex_I and had " upfront.

723 \def\XINT_expr_starthex #1%
724 {%
725     \expandafter\XINT_expr_hex_in\expanded\bgroup
726     \expandafter\XINT_expr_scanhexIgobz_a\romannumeral`&&#1%
727 }%
728 \def\XINT_expr_scanhexIgobz_a #1%
729 {%
730     \ifcat #1\relax
731         0.>;\iffalse{\fi\expandafter}\expandafter\xint_gobble_i\fi
732     \XINT_expr_scanhexIgobz_aa #1%
733 }%
734 \def\XINT_expr_scanhexIgobz_aa #1%
735 {%
736     \if\ifnum`#1>`0
737         \ifnum`#1>`9
738         \ifnum`#1>`@
739         \ifnum`#1>`F
740         0\else1\fi\else0\fi\else1\fi\else0\fi 1%
741         \xint_dothis\XINT_expr_scanhexI_b
742     \fi
743     \if 0#1\xint_dothis\XINT_expr_scanhexIgobz_bgob\fi
744     \if _#1\xint_dothis\XINT_expr_scanhexIgobz_bgob\fi
745     \if .#1\xint_dothis\XINT_expr_scanhexIgobz_toII\fi
746     \xint_orthat
747     {\XINT_expandableerror
748         {Expected an hexadecimal digit but got `#1'. Using `0'.}%
749     0.>;\iffalse{\fi}}%
750     #1%
751 }%
752 \def\XINT_expr_scanhexIgobz_bgob #1#2%
753 {%
754     \expandafter\XINT_expr_scanhexIgobz_a\romannumeral`&&#2%
755 }%
756 \def\XINT_expr_scanhexIgobz_toII .#1%
757 {%
758     0..\expandafter\XINT_expr_scanhexII_a\romannumeral`&&#1%
759 }%
760 \def\XINT_expr_scanhexI_a #1%
761 {%
762     \ifcat #1\relax
763         .>;\iffalse{\fi\expandafter}\expandafter\xint_gobble_i\fi
764     \XINT_expr_scanhexI_aa #1%
765 }%
766 \def\XINT_expr_scanhexI_aa #1%
767 {%
768     \if\ifnum`#1>`/
769         \ifnum`#1>`9

```

```

770     \ifnum`#1>`@
771     \ifnum`#1>`F
772     0\else1\fi\else0\fi\else1\fi\else0\fi 1%
773     \expandafter\XINT_expr_scanhexI_b
774     \else
775     \if_#1\xint_dothis{\expandafter\XINT_expr_scanhexI_bgob}\fi
776     \if_#1\xint_dothis{\expandafter\XINT_expr_scanhexI_toII}\fi
777     \xint_orthat {.;\iffalse{\fi\expandafter}}}%
778     \fi
779     #1%
780 }%
781 \def\XINT_expr_scanhexI_b #1#2%
782 {%
783     #1\expandafter\XINT_expr_scanhexI_a\romannumeral`&&@#2%
784 }%
785 \def\XINT_expr_scanhexI_bgob #1#2%
786 {%
787     \expandafter\XINT_expr_scanhexI_a\romannumeral`&&@#2%
788 }%
789 \def\XINT_expr_scanhexI_toII_#1%
790 {%
791     ..\expandafter\XINT_expr_scanhexII_a\romannumeral`&&@#1%
792 }%
793 \def\XINT_expr_scanhexII_a #1%
794 {%
795     \ifcat #1\relax\xint_dothis{;\iffalse{\fi}#1}\fi
796     \xint_orthat {\XINT_expr_scanhexII_aa #1}%
797 }%
798 \def\XINT_expr_scanhexII_aa #1%
799 {%
800     \if\ifnum`#1>`/
801     \ifnum`#1>`9
802     \ifnum`#1>`@
803     \ifnum`#1>`F
804     0\else1\fi\else0\fi\else1\fi\else0\fi 1%
805     \expandafter\XINT_expr_scanhexII_b
806     \else
807     \if_#1\xint_dothis{\expandafter\XINT_expr_scanhexII_bgob}\fi
808     \xint_orthat{;\iffalse{\fi\expandafter}}}%
809     \fi
810     #1%
811 }%
812 \def\XINT_expr_scanhexII_b #1#2%
813 {%
814     #1\expandafter\XINT_expr_scanhexII_a\romannumeral`&&@#2%
815 }%
816 \def\XINT_expr_scanhexII_bgob #1#2%
817 {%
818     \expandafter\XINT_expr_scanhexII_a\romannumeral`&&@#2%
819 }%

```

11.11.5 \XINT_expr_startfunc: collecting names of functions and variables

At 1.4 the first token left over 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 (edited: which however will cause certain breakage of the infix binary operators: I notice I did not insert None {{{} but nil {}, perhaps by oversight).

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.

Comments here have been removed because 1.4g did a refactoring and renamed \XINT_expr_scanfunc to \XINT_expr_startfunc, moving half of it earlier inside the getnextfork macros.

```
820 \def\XINT_expr_startfunc #1{\expandafter\XINT_expr_func\expanded\bgroup#1\XINT_expr_scanfunc_a}%
821 \def\XINT_expr_scanfunc_a #1%
822 {%
823   \expandafter\XINT_expr_scanfunc_b\romannumeral`&&@#1%
824 }%
```

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.

```
825 \def\XINT_expr_scanfunc_b #1%
826 {%
827   \ifcat \relax#1\xint_dothis{\iffalse{\fi}(_#1)\fi
828   \if (#1\xint_dothis{\iffalse{\fi}(`)\fi
829   \if 1\ifcat a#10\fi
830     \ifnum\xint_c_ix<1\string#1 0\fi
831     \if @#10\fi
832     \if _#10\fi
833     1%
834     \xint_dothis{\iffalse{\fi}(_#1)\fi
835   \xint_orthat {#1\XINT_expr_scanfunc_a}%
836 }%
```

11.11.6 \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.

```
837 \def\XINT_expr_func #1(#2{\if _#2\xint_dothis{\XINT_expr_op_{#1}}\fi
838                               \xint_orthat{#1}\xint_c_ii^v #2}}%
```

11.12 `\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) and trigger here the use of the suitable parser-dependant form. The former (pseudo functions and functions handling dummy variables) first trigger a parser independent mechanism.

With 1.2c "onliteral" 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 `onliteral_<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.

1.4i (2021/06/11) [commented 2021/06/11].

The 1.2c abuse of «onliteral» for both tacit multiplication in front of an opening parenthesis and «generic» functions or pseudo-functions meant that the latter were vulnerable against user redefinition of a function name as a variable name. This applied to `subs`, `subsm`, `subsn`, `seq`, `add`, `mul`, `ndseq`, `ndmap`, `ndfillraw`, `bool`, `togl`, `protect`, `qint`, `qfrac`, `qfloat`, `qraw`, `random`, `qrand`, `rbit` and the most susceptible in real life was probably "seq".

Now variables have an associated «var*» named macro, not «onliteral».

In passing I refactor here in a `\romannumeral` inspired way how `\csname` and TeX booleans are intertwined, minimizing `\expandafter` usage.

```
839 \def\XINT_tmpa #1#2#3{%
840   \def #1##1%
841     {%
842       \csname
843         XINT_\ifcsname XINT_#3_func_##1\endcsname
844         #3_func_##1\expandafter\endcsname\romannumeral`&&\expandafter#2%
845       \romannumeral\else
```

```

846 \ifcsname XINT_expr_onliteral_##1\endcsname
847   expr_onliteral_##1\expandafter\endcsname\romannumeral
848 \else
849 \ifcsname XINT_expr_var*_##1\endcsname
850   expr_var*_##1\expandafter\endcsname\romannumeral
851 \else
852   #3_func_\XINT_expr_unknown_function {##1}%
853   \expandafter\endcsname\romannumeral`&&\expandafter#2%
854 \romannumeral
855 \fi\fi\fi\xint_c_
856 }%
857 }%
858 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
859   \expandafter\XINT_tmpa
860       \csname XINT_#1_op_\` \expandafter\endcsname
861       \csname XINT_#1_oparen\endcsname
862       {#1}%
863 }%
864 \def\XINT_expr_unknown_function #1%
865   {\XINT_expandableerror{`#1' is unknown, say `Isome_func' or I use 0.}}%
866 \def\XINT_expr_func_ #1#2#3{#1#2{{0}}}%
867 \let\XINT_flexpr_func_\XINT_expr_func_
868 \let\XINT_iiexpr_func_\XINT_expr_func_

```

11.13 \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 fake variables which rely on possibility to modify incoming upfront tokens. If we did here something such as

```
_var_#1\expandafter\endcsname\romannumeral`^^@\XINT_expr_getop
```

the premature expansion of getop would break the var_omit and var_abort mechanism. Thus we revert to former code which locates an \XINT_expr_getop (call it _legacy) before the tokens from the variable expansion (in xintexpr < 1.4 the normal variables expanded to a single token so the overhead was not serious) so we can expand fake variables first.

Abusing variables to manipulate the incoming token stream is a bit bad, usually I prefer functions for this (such as the break() function) but then I have to define 3 macros for the 3 parsers.

This trick of fake variables puts thus a general overhead at various locations, and the situation here is REALLY not satisfactory. But 1.4 has (had) to be released now.

Even if I could put the \csname XINT_expr_var_foo\endcsname upfront, which would then be f-expanded, this would still need \XINT_expr_put_op_first to use its \expandafter's as long as \XINT_expr_var_foo expands to {\XINT_expr_varvalue_foo} with a not-yet expanded \XINT_expr_var_value.

I could let \XINT_expr_var_foo expand to \expandafter{\XINT_expr_varvalue_foo} allowing then (if it gets f-expanded) probably to drop the \expandafter in \XINT_expr_put_op_first. But I can not consider this option in the form

```
_var_foo\expandafter\endcsname\romannumeral`^^@\XINT_expr_getop
```

until the issue with fake variables such as omit and abort which must act before \XINT_expr_getop has some workaround. This could be implemented here with some extra branch, i.e. there would not be some \XINT_expr_var_omit but something else filtered out in the \else branch here.

The above comments mention only omit and abort, but the case of real dummy variables also needs

consideration.

At 1.4g, I test first for existence of `\XINT_expr_onliteral_foo`.

Updated for 1.4i: now rather existence of `\XINT_expr_var*_foo` is tested.

This is a trick which allows to distinguish actual or dummy variables from really fake variables omit and abort (must check if there are others). For the real or dummy variables we can trigger the expansion of the `\XINT_expr_getop` before the one of the variable. I could test for `varvalue_foo` but this applies only to real variables not dummy variables. Actual and dummy variables are thus handled slightly faster at 1.4g as there is less induced moving around (the `\expandafter` chain in `\XINT_expr_put_op_first` still applies at this stage, as I have not yet re-examined the var/varvalue mechanism). And the test for `var_foo` is moved directly inside the `\csname` construct in the `\else` branch which now handles together fake variables and non-existing variables.

I only have to make sure dummy variables are really safe being handled this way with the `getop` action having been done before they expand, but it looks ok. Attention it is crucial that if `\XINT_expr_getop` finds a `\relax` it inserts `\xint_c\relax` so the `\relax` token is still there!

With this refactoring the `\XINT_expr_getop_legacy` is applied only in case of non-existent variables or fake variables omit/abort or things such as `nil`, `None`, `false`, `true`, `False`, `True`.

If user in interactive mode fixes the variable name, the `\XINT_expr_var_foo` expanded once with `deliver{\XINT_expr_varvalue_foo}` (if not dummy), and the braces are maintained by `\XINT_expr_getop_legacy`.

```

869 \def\XINT_expr_op__ #1% op__ with two '_'s
870 {%
871   \ifcsname XINT_expr_var*_{#1}\endcsname
872     \csname XINT_expr_var_{#1}\expandafter\endcsname
873     \romannumeral`&&\expandafter\XINT_expr_getop
874   \else
875     \expandafter\expandafter\expandafter\XINT_expr_getop_legacy
876     \csname XINT_expr_var_{#1}\endcsname
877     \ifcsname XINT_expr_var_{#1}\endcsname_{#1}\else\XINT_expr_unknown_variable{#1}\fi
878     \expandafter\endcsname
879   \fi
880}%
881 \def\XINT_expr_unknown_variable #1%
882   {\XINT_expandableerror {\#1' unknown, say `Isome_var' or I use 0.}}%
883 \def\XINT_expr_var_{#1}%
884 \let\XINT_flexpr_op__ \XINT_expr_op__
885 \let\XINT_iexpr_op__ \XINT_expr_op__
886 \def\XINT_expr_getop_legacy #1%
887 {%
888   \expanded{\unexpanded{#1}\expandafter}\romannumeral`&&\XINT_expr_getop
889}%

```

11.14 `\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 $x/2\max(3,5)$, or $x/2\xintexpr 3\relax$.

In fact, finally I decide that all sorts of tacit multiplication will always use the higher precedence.

Indeed I hesitated somewhat: with the current code one does not know if `\XINT_expr_getop` as invoked after a closing parenthesis or because a number parsing ended, and I felt distinguishing

the two was unneeded extra stuff. This means cases like $(a+b)/(c+d)(e+f)$ will first multiply the last two parenthesized terms.

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.

Prior to 1.4g there was an `\if_#1\xint_dothis\xint_secondofthree\fi` because the `_` can be used to start names, for private use by package (for example by `polexpr`). But this test was silly because these usages are only with a `_` of catcode 11. And allowing non-catcode 11 `_` also to trigger tacit multiplication caused an infinite loop in collaboration with `\XINT_expr_scanfunc`, see explanations there (now removed after refactoring, see `\XINT_expr_startfunc`).

The situation with the `@` is different because we must allow it even as catcode 12 as a name, as it used in the syntax and must work the same if of catcode 11 or 12. No infinite loop because it is filtered out by one of the `\XINT_expr_getnextfork` macros.

The check for `:` to send it to thirdofthree "getop" branch is needed, last time I checked, because during some part of at least `\xintdeffunc`, some scantokens are done which need to work with the `:` of catcode 11, and it would be misconstrued to start a name if not filtered out.

```
890 \def\xint_expr_getop #1%
891 {%
892   \expandafter\xint_expr_getop_a\romannumeral`&&@#1%
893 }%
894 \catcode`* 11
895 \def\xint_expr_getop_a #1%
896 {%
897   \ifx \relax #1\xint_dothis\xint_firstofthree\fi
898   \ifcat \relax #1\xint_dothis\xint_secondofthree\fi
899   \ifnum\xint_c_ix<1\string#1 \xint_dothis\xint_secondofthree\fi
900   \if :#1\xint_dothis \xint_thirdofthree\fi
901   \if @#1\xint_dothis \xint_secondofthree\fi
902   \if (#1\xint_dothis \xint_secondofthree\fi %)
903   \ifcat a#1\xint_dothis \xint_secondofthree\fi
904   \xint_orthat \xint_thirdofthree
```

Formerly `\XINT_expr_foundend` as `firstofthree` but at 1.4g let's simply insert `\xint_c_` as the `#1` is `\relax` (and anyhow a place-holder according to remark in definition of `\XINT_expr_foundend`

```
905 \xint_c_
```

Tacit multiplication with higher precedence. Formerly `\XINT_expr_precedence_***` was used, renamed to `\XINT_expr_prec_tacit` at 1.4g in case a backport is done of the `\bnumdefinfix` from `bnumexpr`.

```
906 {\XINT_expr_prec_tacit *}%
```

This is only location which jumps to `\XINT_expr_getop_b`. At 1.4f and perhaps for old legacy reasons this was `\expandafter\xint_expr_getop_b \string#1` but I see no reason now for applying `\string` to `#1`. Removed at 1.4g. And the `#1` now moved out of the `secondofthree` and `thirdofthree` branches.

```
907 \XINT_expr_getop_b
908 #1%
909 }%
910 \catcode`* 12
```

`\relax` is a place holder here. At 1.4g, we don't use `\XINT_expr_foundend` anymore in `\XINT_expr_getop_a` which was slightly refactored, but it is used elsewhere.

Attention that keeping a `\relax` around if `\XINT_expr_getop` hits it is crucial to good functioning of dummy variables after 1.4g refactoring of `\XINT_expr_op__`, the `\relax` being used as delimiter by dummy variables, and `\XINT_expr_getop` is now expanded before the variable itself does its thing.

```
911 \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`.

Memo: 1.4g, the token fetched by `\XINT_expr_getop_b` has not anymore been previously submitted in `\XINT_expr_getop_a` to `\string`.

```
912 \def\XINT_expr_getop_b#1{\def\XINT_expr_getop_b ##1%
913 {%
914   \if &&@##1\xint_dothis{#1&&}\fi
915   \if '##1\xint_dothis{\XINT_expr_binopwrld }\fi
916   \if ?##1\xint_dothis{\XINT_expr_precedence_? ?}\fi
917   \xint_orthat      {\XINT_expr_scanop_a ##1}%
918 }}\expandafter\XINT_expr_getop_b\csname XINT_expr_precedence_&&\endcsname
919 \def\XINT_expr_binopwrld #1'%
920 {%
921   \expandafter\XINT_expr_foundop_a
922   \csname XINT_expr_itself_\xint_zapspaces #1 \xint_gobble_i\endcsname
923 }%
924 \def\XINT_expr_scanop_a #1#2%
925 {%
926   \expandafter\XINT_expr_scanop_b\expandafter#1\romannumeral`&&@#2%
927 }%
```

Multi-character operators have an associated itself macro at each stage of decomposition starting at two characters. Here, nothing imposes to the operator characters not to be of catcode letter, this constraint applies only on the first character and is done via `\XINT_expr_getop_a`, to handle in particular tacit multiplication in front of variable or function names.

But it would be dangerous to allow letters in operator characters, again due to existence of variables and functions, and anyhow there is no user interface to add such custom operators. However in `bnumexpr`, such a constraint does not exist.

I don't worry too much about efficiency here... and at 1.4g I have re-written for code readability only. Once we see that `#1#2` is not a candidate to be or start an operator, we need to check if single-character operator `#1` is really an operator and this is done via the existence of the precedence token.

Unfortunately the 1.4g refactoring of the scanop macros had a bad bug: `\XINT_expr_scanop_c` inserted `\romannumeral`^^@` in stream but did not grab a token first so a space would stop the `\romannumeral` and then the `#2` in `\XINT_expr_scanop_d` was not pre-expanded and ended up alone in `\ifcat`. It is too distant in the past the time when I wrote the core of `xintexpr` in 2013... older and dumber now.

```
928 \def\XINT_expr_scanop_b #1#2%
929 {%
930   \unless\ifcat#2\relax
931     \ifcsname XINT_expr_itself_#1#2\endcsname
932     \XINT_expr_scanop_c
933   \fi\fi
934   \XINT_expr_foundop_a #1#2%
935 }%
```

```

936 \def\XINT_expr_scanop_c #1#2#3#4#5#6% #1#2=\fi\fi
937 {%
938     #1#2%
939     \expandafter\XINT_expr_scanop_d\csname XINT_expr_itself_#4#5\expandafter\endcsname
940     \romannumeral`&&@#6%
941 }%
942 \def\XINT_expr_scanop_d #1#2%
943 {%
944     \unless\ifcat#2\relax
945         \ifcsname XINT_expr_itself_#1#2\endcsname
946             \XINT_expr_scanop_c
947         \fi\fi
948     \XINT_expr_foundop #1#2%
949 }%
950 \def\XINT_expr_foundop_a #1%
951 {%
952     \ifcsname XINT_expr_precedence_#1\endcsname
953         \csname XINT_expr_precedence_#1\expandafter\endcsname
954         \expandafter #1%
955     \else
956         \expandafter\XINT_expr_getop\romannumeral`&&@%
957         \xint_afterfi{\XINT_expandableerror
958             {Expected an operator but got `#1'. Ignoring.}}%
959     \fi
960 }%
961 \def\XINT_expr_foundop #1{\csname XINT_expr_precedence_#1\endcsname #1}%

```

11.15 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`.

```

962 \catcode`) 11
963 \def\XINT_tmpa #1#2#3#4#5#6%
964 {%
965     \def#1% start
966     {%
967         \expandafter#2\romannumeral`&&\XINT_expr_getnext
968     }%
969     \def#2##1% check
970     {%

```

```

971      \xint_UDsignfork
972      ##1{\expandafter#3\romannumeral`&&@#4}%
973      -{#3##1}%
974      \krof
975  }%
976  \def#3##1##2%  checkp
977  {%
978      \ifcase ##1%
979          \expandafter\xINT_expr_done
980          \or\expandafter#5%
981          \else
982              \expandafter#3\romannumeral`&&@\csname XINT_#6_op_##2\expandafter\endcsname
983          \fi
984  }%
985  \def#5%
986  {%
987      \XINT_expandableerror
988      {Extra ) removed. Hit <return>, fingers crossed.}%
989      \expandafter#2\romannumeral`&&@\expandafter\xINT_expr_put_op_first
990      \romannumeral`&&@\XINT_expr_getop_legacy
991  }%
992 }%
993 \let\xINT_expr_done\space
994 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
995     \expandafter\xINT_tmpa
996     \csname XINT_#1_start\expandafter\endcsname
997     \csname XINT_#1_check\expandafter\endcsname
998     \csname XINT_#1_checkp\expandafter\endcsname
999     \csname XINT_#1_op_-xii\expandafter\endcsname
1000     \csname XINT_#1_extra_\expandafter\endcsname
1001     {#1}%
1002 }%

```

Here also we take some shortcuts relative to general philosophy and have no explicit exec macro.

```

1003 \def\xINT_tmpa #1#2#3#4#5#6#7%
1004 {%
1005     \def #1##1% op_(
1006     {%
1007         \expandafter #4\romannumeral`&&@\XINT_expr_getnext
1008     }%
1009     \def #2##1% op_)
1010     {%
1011         \expanded{\unexpanded{\XINT_expr_put_op_first{##1}}\expandafter}\romannumeral`&&@\XINT_expr_geto
1012     }%
1013     \def #3% oparen
1014     {%
1015         \expandafter #4\romannumeral`&&@\XINT_expr_getnext
1016     }%
1017     \def #4##1% check-
1018     {%
1019         \xint_UDsignfork
1020         ##1{\expandafter#5\romannumeral`&&@#6}%
1021         -{#5##1}%

```

```

1022     \krof
1023 }%
1024 \def #5##1##2% checkp
1025 {%
1026     \ifcase ##1\expandafter\XINT_expr_missing_
1027     \or \csname XINT_#7_op_##2\expandafter\endcsname
1028     \else
1029     \expandafter #5\romannumeral`&&\csname XINT_#7_op_##2\expandafter\endcsname
1030     \fi
1031 }%
1032 }%
1033 \def\XINT_expr_missing_
1034 {\XINT_expandableerror{End of expression found, but some ) was missing there.}%
1035 \xint_c_ \XINT_expr_done }%
1036 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1037     \expandafter\XINT_tmpa
1038     \csname XINT_#1_op_(\expandafter\endcsname
1039     \csname XINT_#1_op_)\expandafter\endcsname
1040     \csname XINT_#1_oparen\expandafter\endcsname
1041     \csname XINT_#1_check_)\expandafter\endcsname
1042     \csname XINT_#1_checkp_)\expandafter\endcsname
1043     \csname XINT_#1_op_-xii\endcsname
1044     {#1}%
1045 }%
1046 \let\XINT_expr_precedence_)\xint_c_i
1047 \catcode`) 12

```

11.16 The comma as binary operator

New with 1.09a. Refactored at 1.4.

```

1048 \def\XINT_tmpa #1#2#3#4#5#6%
1049 {%
1050     \def #1##1% \XINT_expr_op_,
1051     {%
1052         \expanded{\unexpanded{#2{##1}}\expandafter}%
1053         \romannumeral`&&\expandafter#3\romannumeral`&&\XINT_expr_getnext
1054     }%
1055     \def #2##1##2##3##4{##2##3{##1##4}}% \XINT_expr_exec_,
1056     \def #3##1% \XINT_expr_check_-,
1057     {%
1058         \xint_UDsignfork
1059         ##1{\expandafter#4\romannumeral`&&@#5}%
1060         -{#4##1}%
1061     \krof
1062 }%
1063 \def #4##1##2% \XINT_expr_checkp_,
1064 {%
1065     \ifnum ##1>\xint_c_iii
1066     \expandafter#4%
1067     \romannumeral`&&\csname XINT_#6_op_##2\expandafter\endcsname
1068     \else
1069     \expandafter##1\expandafter##2%

```

```

1070     \fi
1071   }%
1072 }%
1073 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
1074 \expandafter\XINT_tmpa
1075   \csname XINT_#1_op_,\expandafter\endcsname
1076   \csname XINT_#1_exec_,\expandafter\endcsname
1077   \csname XINT_#1_check_,\expandafter\endcsname
1078   \csname XINT_#1_checkp_,\expandafter\endcsname
1079   \csname XINT_#1_op_-xii\endcsname {#1}%
1080 }%
1081 \expandafter\let\csname XINT_expr_precedence_,\endcsname\xint_c_iii

```

11.17 The minus as prefix operator of variable precedence level

Inherits the precedence level of the previous infix operator, if the latter has at least the precedence level of binary + and -, i.e. currently 12.

Refactored at 1.4.

At 1.4g I belatedly observe that I have been defining architecture for op_-xvi but such operator can never be created, because there are no infix operators of precedence level 16. Perhaps in the past this was really needed? But now such 16 is precedence level of tacit multiplication which is implemented simply by the \XINT_expr_prec_tacit token, there is no macro check_-*** which would need an op_-xvi.

For the record: at least one scenario exists which creates tacit multiplication in front of a unary -, it is 2\count0 which first generates tacit multiplication then applies \number to \count0, but the operator is still *, so this triggers only \XINT_expr_op_-xiv, not -xvi.

At 1.4g we need 17 and not 18 anymore as the precedence of unary minus following power operators ^ and **. The needed \xint_c_xvii creation was added to xintkernel.sty.

```

1082 \def\XINT_tmpb #1#2#3#4#5#6#7%
1083 {%
1084   \def #1% \XINT_expr_op_-<level>
1085   {%
1086     \expandafter #2\romannumeral`&&\expandafter#3%
1087     \romannumeral`&&\XINT_expr_getnext
1088   }%
1089   \def #2##1##2##3% \XINT_expr_exec_-<level>
1090   {%
1091     \expandafter ##1\expandafter ##2\expandafter
1092     {%
1093       \romannumeral`&&\XINT:NEhook:f:one:from:one
1094       {\romannumeral`&&#7##3}%
1095     }%
1096   }%
1097   \def #3##1% \XINT_expr_check_-<level>
1098   {%
1099     \xint_UDsignfork
1100     ##1{\expandafter #4\romannumeral`&&#1}%
1101     -{#4##1}%
1102     \krof
1103   }%
1104   \def #4##1##2% \XINT_expr_checkp_-<level>
1105   {%

```

```

1106     \ifnum ##1>#5%
1107     \expandafter #4%
1108     \romannumeral`&&@\csname XINT_#6_op_##2\expandafter\endcsname
1109     \else
1110     \expandafter ##1\expandafter ##2%
1111     \fi
1112 }%
1113}%
1114\def\XINT_tmpa #1#2#3%
1115{%
1116    \expandafter\XINT_tmppb
1117    \csname XINT_#1_op_#3\expandafter\endcsname
1118    \csname XINT_#1_exec_#3\expandafter\endcsname
1119    \csname XINT_#1_check_#3\expandafter\endcsname
1120    \csname XINT_#1_checkp_#3\expandafter\endcsname
1121    \csname xint_c_#3\endcsname {#1}#2%
1122}%
1123\xintApplyInline{\XINT_tmpa {expr}\xintOpp}{\xii}{xiv}{xvii}}%
1124\xintApplyInline{\XINT_tmpa {flexpr}\xintOpp}{\xii}{xiv}{xvii}}%
1125\xintApplyInline{\XINT_tmpa {iiexpr}\xintiiOpp}{\xii}{xiv}{xvii}}%

```

11.18 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.

```

1126\def\XINT_tmpa#1#2#3%
1127{%
1128    \def#1##1{\expandafter#2\romannumeral`&&@\XINT_expr_getnext}%
1129    \def#2##1##2%
1130    {%
1131        \ifnum ##1>\xint_c_xx
1132        \expandafter #2%
1133        \romannumeral`&&@\csname XINT_#3_op_##2\expandafter\endcsname
1134        \else
1135        \expandafter##1\expandafter##2\romannumeral0\expandafter\XINT:NEhook:unpack
1136        \fi
1137    }%
1138}%
1139\def\XINT:NEhook:unpack{\xint_stop_atfirstofone}%
1140\xintFor* #1 in {{expr}{flexpr}{iiexpr}}:
1141    {\expandafter\XINT_tmpa\csname XINT_#1_op_0\expandafter\endcsname
1142     \csname XINT_#1_until_unpack\endcsname {#1}}%

```

11.19 Infix operators

11.19.1	&&, , //, /:, +, -, *, /, ^, **, 'and', 'or', 'xor', and 'mod'	352
11.19.2	.., ..[, and].. for a..b and a..[b]..c syntax	354
11.19.3	<, >, ==, <=, >=, != with Python-like chaining	356
11.19.4	Support macros for .., ..[and]..	358

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.

At 1.4b we must make sure that the `!` in expansion of `\XINT_expr_itself_!` is of catcode 12 and not of catcode 11. This is because implementation of chaining of comparison operators proceeds via inserting the `itself` macro directly into upcoming token stream, whereas formerly such `itself` macros would be expanded only in a `\csname...\endcsname` context.

```
1143 \catcode\& 12 \catcode\! 12
1144 \xintFor* #1 in {{==}{!=}{<=}{>=}}{&&}{||}{//}{/:}{..}{.[]}{.[]}{.[]}}%
1145   \do {\expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}}%
1146 \catcode\& 7 \catcode\! 11
```

11.19.1 &&, ||, //, /:, +, -, *, /, ^, **, 'and', 'or', 'xor', and 'mod'

Single character boolean operators `&` and `|` had been deprecated since 1.1 and finally got removed (together with `=` comparison test) from syntax at 1.4g.

Also, at 1.4g I finally decide to enact the switch to right associativity for the power operators `^` and `**`.

This goes via inserting into the `checkp` macros not anymore the precedence `chardef` token (which now only serves as left precedence, inserted in the token stream) but in its place an `\xint_c<roman>` token holding the right precedence. Which is also transmitted to spanned unary minus operators.

Here only levels 12, 14, and 17 are created as right precedences.

#6 and #7 got permuted and the new #7 is directly a control sequence. Also #3 and #4 are now integers which need `\romannumeral`. The change in `\XINT_expr_defbin_c` does not propagate as it is re-defined shortly thereafter.

```
1147 \def\XINT_expr_defbin_c #1#2#3#4#5#6#7#8%
1148 {%
1149   \def #1#1% \XINT_expr_op_<op>
1150   {%
1151     \expanded{\unexpanded{#2{##1}}\expandafter}%
1152     \romannumeral`&&\expandafter#3\romannumeral`&&\XINT_expr_getnext
1153   }%
1154   \def #2#1#2#3#4% \XINT_expr_exec_<op>
1155   {%
1156     \expandafter##2\expandafter##3\expandafter
1157     {\romannumeral`&&\XINT:NEhook:f:one:from:two{\romannumeral`&&#7##1##4}}%
1158   }%
1159   \def #3#1% \XINT_expr_check_-_<op>
1160   {%
1161     \xint_UDsignfork
1162     ##1{\expandafter#4\romannumeral`&&#5}%
1163     -{#4##1}%
1164     \krof
1165   }%
1166   \def #4#1#2% \XINT_expr_checkp_<op>
1167   {%
1168     \ifnum ##1>#6%
1169       \expandafter#4%
1170       \romannumeral`&&\csname XINT_#8_op_##2\expandafter\endcsname
1171     \else
1172       \expandafter ##1\expandafter ##2%
1173     \fi
```



```

1174 }%
1175 }%
1176 \def\XINT_expr_defbin_b #1#2#3#4#5%
1177 {%
1178 \expandafter\XINT_expr_defbin_c
1179 \csname XINT_#1_op_#2\expandafter\endcsname
1180 \csname XINT_#1_exec_#2\expandafter\endcsname
1181 \csname XINT_#1_check_#2\expandafter\endcsname
1182 \csname XINT_#1_checkp_#2\expandafter\endcsname
1183 \csname XINT_#1_op_-\romannumeral\ifnum#4>12 #4\else12\fi\expandafter\endcsname
1184 \csname xint_c_-\romannumeral#4\endcsname
1185 #5%
1186 {#1}%
1187 \expandafter % done 3 times but well
1188 \let\csname XINT_expr_precedence_#2\expandafter\endcsname
1189 \csname xint_c_-\romannumeral#3\endcsname
1190 }%
1191 \XINT_expr_defbin_b {expr} {||} {6} {6} \xintOR
1192 \XINT_expr_defbin_b {flexpr}{||} {6} {6} \xintOR
1193 \XINT_expr_defbin_b {iiexpr}{||} {6} {6} \xintOR
1194 \catcode`& 12
1195 \XINT_expr_defbin_b {expr} {&&} {8} {8} \xintAND
1196 \XINT_expr_defbin_b {flexpr}{&&} {8} {8} \xintAND
1197 \XINT_expr_defbin_b {iiexpr}{&&} {8} {8} \xintAND
1198 \catcode`& 7
1199 \XINT_expr_defbin_b {expr} {xor}{6} {6} \xintXOR
1200 \XINT_expr_defbin_b {flexpr}{xor}{6} {6} \xintXOR
1201 \XINT_expr_defbin_b {iiexpr}{xor}{6} {6} \xintXOR
1202 \XINT_expr_defbin_b {expr} {///  

1203 \XINT_expr_defbin_b {flexpr}{///  

1204 \XINT_expr_defbin_b {iiexpr}{///  

1205 \XINT_expr_defbin_b {expr} {/:}  

1206 \XINT_expr_defbin_b {flexpr}{/:}  

1207 \XINT_expr_defbin_b {iiexpr}{/:}  

1208 \XINT_expr_defbin_b {expr} + {12}{12}\xintAdd  

1209 \XINT_expr_defbin_b {flexpr} + {12}{12}\XINTinFloatAdd  

1210 \XINT_expr_defbin_b {iiexpr} + {12}{12}\xintiiAdd  

1211 \XINT_expr_defbin_b {expr} - {12}{12}\xintSub  

1212 \XINT_expr_defbin_b {flexpr} - {12}{12}\XINTinFloatSub  

1213 \XINT_expr_defbin_b {iiexpr} - {12}{12}\xintiiSub  

1214 \XINT_expr_defbin_b {expr} * {14}{14}\xintMul  

1215 \XINT_expr_defbin_b {flexpr} * {14}{14}\XINTinFloatMul  

1216 \XINT_expr_defbin_b {iiexpr} * {14}{14}\xintiiMul  

1217 \let\XINT_expr_prec_tacit \xint_c_xvi  

1218 \XINT_expr_defbin_b {expr} / {14}{14}\xintDiv  

1219 \XINT_expr_defbin_b {flexpr} / {14}{14}\XINTinFloatDiv  

1220 \XINT_expr_defbin_b {iiexpr} / {14}{14}\xintiiDivRound

```

At 1.4g, right associativity is implemented via a lowered right precedence here.

```

1221 \XINT_expr_defbin_b {expr} ^ {18}{17}\xintPow
1222 \XINT_expr_defbin_b {flexpr} ^ {18}{17}\XINTinFloatSciPow
1223 \XINT_expr_defbin_b {iiexpr} ^ {18}{17}\xintiiPow

```

1.4g This is a trick (which was in old version of bnumexpr, I wonder why I did not have it here) but it will make error messages in case of **<token> confusing. The ^ here is of catcode 11 but it does not matter.

```
1224 \expandafter\def\csname XINT_expr_itself_**\endcsname{^}%
1225 \catcode\& 12
```

For this which contributes to implementing 'and', 'or', etc... see \XINT_expr_binopwrd.

```
1226 \xintFor #1 in {and,or,xor,mod} \do
1227 {%
1228   \expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}%
1229 }%
1230 \expandafter\let\csname XINT_expr_precedence_and\expandafter\endcsname
1231   \csname XINT_expr_precedence_&\endcsname
1232 \expandafter\let\csname XINT_expr_precedence_or\expandafter\endcsname
1233   \csname XINT_expr_precedence_|\endcsname
1234 \expandafter\let\csname XINT_expr_precedence_mod\expandafter\endcsname
1235   \csname XINT_expr_precedence_/\endcsname
1236 \xintFor #1 in {expr, flexpr, iexpr} \do
1237 {%
1238   \expandafter\let\csname XINT_#1_op_and\expandafter\endcsname
1239     \csname XINT_#1_op_&\endcsname
1240   \expandafter\let\csname XINT_#1_op_or\expandafter\endcsname
1241     \csname XINT_#1_op_|\endcsname
1242   \expandafter\let\csname XINT_#1_op_mod\expandafter\endcsname
1243     \csname XINT_#1_op_/\endcsname
1244 }%
1245 \catcode\& 7
```

11.19.2 .., ..[, and].. for a..b and a..[b]..c syntax

The 1.4 exec_...[macros (which do no further expansion!) had silly \expandafter doing nothing for the sole reason of sharing a common \XINT_expr_defbin_c as used previously for the +, - etc... operators. At 1.4b we take the time to set things straight and do other similar simplifications.

```
1246 \def\XINT_expr_defbin_c #1#2#3#4#5#6#7%
1247 {%
1248   \def #1##1% \XINT_expr_op...[
1249     {%
1250       \expanded{\unexpanded{#2{##1}}\expandafter}%
1251       \romannumeral`&&\expandafter#3\romannumeral`&&\XINT_expr_getnext
1252     }%
1253   \def #2##1##2##3##4% \XINT_expr_exec...[
1254     {%
1255       ##2##3{{##1##4}}%
1256     }%
1257   \def #3##1% \XINT_expr_check-...[
1258     {%
1259       \xint_UDsignfork
1260       ##1{\expandafter#4\romannumeral`&&@#5}%
1261       -{#4##1}%
1262       \krof
1263     }%
1264   \def #4##1##2% \XINT_expr_checkp...[
1265     {%
```

```

1266 \ifnum ##1>#6%
1267 \expandafter#4%
1268 \romannumeral`&&@\csname XINT_#7_op_##2\expandafter\endcsname
1269 \else
1270 \expandafter ##1\expandafter ##2%
1271 \fi
1272 }%
1273 }%
1274 \def\XINT_expr_defbin_b #1%
1275 {%
1276 \expandafter\XINT_expr_defbin_c
1277 \csname XINT_#1_op_..[\expandafter\endcsname
1278 \csname XINT_#1_exec_..[\expandafter\endcsname
1279 \csname XINT_#1_check_..[\expandafter\endcsname
1280 \csname XINT_#1_checkp_..[\expandafter\endcsname
1281 \csname XINT_#1_op_-xii\expandafter\endcsname
1282 \csname XINT_expr_precedence_..[\endcsname
1283 {#1}%
1284 }%
1285 \XINT_expr_defbin_b {expr}%
1286 \XINT_expr_defbin_b {flexpr}%
1287 \XINT_expr_defbin_b {iiexpr}%
1288 \expandafter\let\csname XINT_expr_precedence_..[\endcsname\xint_c_vi
1289 \def\XINT_expr_defbin_c #1#2#3#4#5#6#7#8%
1290 {%
1291 \def #1##1% \XINT_expr_op_<op>
1292 {%
1293 \expanded{\unexpanded{#2{##1}}\expandafter}%
1294 \romannumeral`&&@\expandafter#3\romannumeral`&&@\XINT_expr_getnext
1295 }%
1296 \def #2##1##2##3##4% \XINT_expr_exec_<op>
1297 {%
1298 \expandafter##2\expandafter##3\expanded
1299 {{\XINT:NEhook:x:one:from:two#8##1##4}}%
1300 }%
1301 \def #3##1% \XINT_expr_check_<op>
1302 {%
1303 \xint_UDsignfork
1304 ##1{\expandafter#4\romannumeral`&&@#5}%
1305 -{#4##1}%
1306 \krof
1307 }%
1308 \def #4##1##2% \XINT_expr_checkp_<op>
1309 {%
1310 \ifnum ##1>#6%
1311 \expandafter#4%
1312 \romannumeral`&&@\csname XINT_#7_op_##2\expandafter\endcsname
1313 \else
1314 \expandafter ##1\expandafter ##2%
1315 \fi
1316 }%
1317 }%

```

```

1318 \def\XINT_expr_defbin_b #1#2#3%
1319 {%
1320   \expandafter\XINT_expr_defbin_c
1321   \csname XINT_#1_op_#2\expandafter\endcsname
1322   \csname XINT_#1_exec_#2\expandafter\endcsname
1323   \csname XINT_#1_check_#2\expandafter\endcsname
1324   \csname XINT_#1_checkp_#2\expandafter\endcsname
1325   \csname XINT_#1_op_#2\expandafter\endcsname
1326   \csname XINT_expr_precedence_#2\endcsname
1327   {#1}#3%
1328   \expandafter\let
1329   \csname XINT_expr_precedence_#2\expandafter\endcsname\xint_c_vi
1330 }%
1331 \XINT_expr_defbin_b {expr} {..}\xintSeq:tl:x
1332 \XINT_expr_defbin_b {flexpr} {..}\xintSeq:tl:x
1333 \XINT_expr_defbin_b {iiexpr} {..}\xintiiSeq:tl:x
1334 \XINT_expr_defbin_b {expr} {}..\xintSeqB:tl:x
1335 \XINT_expr_defbin_b {flexpr}{}..\xintSeqB:tl:x
1336 \XINT_expr_defbin_b {iiexpr}{}..\xintiiSeqB:tl:x

```

11.19.3 <, >, ==, <=, >=, != with Python-like chaining

1.4b This is preliminary implementation of chaining of comparison operators like Python and (I think) l3fp do. I am not too happy with how many times the (second) operand (already evaluated) is fetched.

```

1337 \def\XINT_expr_defbin_d #1#2%
1338 {%
1339   \def #1##1##2##3##4% \XINT_expr_exec_<op>
1340   {%
1341     \expandafter##2\expandafter##3\expandafter
1342     {\romannumeral`&&\XINT:NEhook:f:one:from:two{\romannumeral`&&#2##1##4}}%
1343   }%
1344 }%
1345 \def\XINT_expr_defbin_c #1#2#3#4#5#6#7#8#9%
1346 {%
1347   \def #1##1% \XINT_expr_op_<op>
1348   {%
1349     \expanded{\unexpanded{#2{##1}}\expandafter}%
1350     \romannumeral`&&\expandafter#7%
1351     \romannumeral`&&\expandafter#3\romannumeral`&&\XINT_expr_getnext
1352   }%
1353   \def #3##1% \XINT_expr_check_<op>
1354   {%
1355     \xint_UDsignfork
1356     ##1{\expandafter#4\romannumeral`&&#5}%
1357     -{#4##1}%
1358     \krof
1359   }%
1360   \def #4##1##2% \XINT_expr_checkp_<op>
1361   {%
1362     \ifnum ##1>#6%
1363       \expandafter#4%

```

```

1364     \romannumeral`&&@\csname XINT_#9_op_##2\expandafter\endcsname
1365     \else
1366     \expandafter ##1\expandafter ##2%
1367     \fi
1368 }%
1369 \let #6\xint_c_x
1370 \def #7##1% \XINT_expr_checkc_<op>
1371 {%
1372     \ifnum ##1=\xint_c_x\expandafter#8\fi ##1%
1373 }%
1374 \edef #8##1##2##3% \XINT_expr_execc_<op>
1375 {%
1376     \csname XINT_#9_precedence_\string&\string&\endcsname
1377     \expandafter\noexpand\csname XINT_#9_itself_\string&\string&\endcsname
1378     {##3}%
1379     \XINTfstop.{##3}##2%
1380 }%
1381 \XINT_expr_defbin_d #2% \XINT_expr_exec_<op>
1382 }%
1383 \def\XINT_expr_defbin_b #1#2%#3%
1384 {%
1385     \expandafter\XINT_expr_defbin_c
1386     \csname XINT_#1_op_#2\expandafter\endcsname
1387     \csname XINT_#1_exec_#2\expandafter\endcsname
1388     \csname XINT_#1_check_#2\expandafter\endcsname
1389     \csname XINT_#1_checkp_#2\expandafter\endcsname
1390     \csname XINT_#1_op_-xii\expandafter\endcsname
1391     \csname XINT_expr_precedence_#2\expandafter\endcsname
1392     \csname XINT_#1_checkc_#2\expandafter\endcsname
1393     \csname XINT_#1_execc_#2\endcsname
1394     {#1}%#3%
1395 }%

```

Single character comparison operator = had been deprecated for many years (but I can't find since when precisely; & and | were deprecated at 1.1 as since in CHANGES.md) and it finally got removed from syntax at 1.4g, as well as & and |.

Attention that third token here is left in stream by defbin_b, then also by defbin_c and is picked up as #2 of defbin_d. Had to work around TeX accepting only 9 arguments. Why did it not start counting at #0 like all decent mathematicians do?

```

1396 \XINT_expr_defbin_b {expr} <\xintLt
1397 \XINT_expr_defbin_b {flexpr}<\xintLt
1398 \XINT_expr_defbin_b {iiexpr}<\xintiilt
1399 \XINT_expr_defbin_b {expr} >\xintGt
1400 \XINT_expr_defbin_b {flexpr}>\xintGt
1401 \XINT_expr_defbin_b {iiexpr}>\xintiiGt
1402 \XINT_expr_defbin_b {expr} {==}\xintEq
1403 \XINT_expr_defbin_b {flexpr}{==}\xintEq
1404 \XINT_expr_defbin_b {iiexpr}{==}\xintiiEq
1405 \XINT_expr_defbin_b {expr} {<=}\xintLtorEq
1406 \XINT_expr_defbin_b {flexpr}{<=}\xintLtorEq
1407 \XINT_expr_defbin_b {iiexpr}{<=}\xintiiLtorEq
1408 \XINT_expr_defbin_b {expr} {>=}\xintGtorEq
1409 \XINT_expr_defbin_b {flexpr}{>=}\xintGtorEq

```

```

1410 \XINT_expr_defbin_b {iiexpr}{>=}\xintiiGtorEq
1411 \XINT_expr_defbin_b {expr} {!=}\xintNotEq
1412 \XINT_expr_defbin_b {flexpr}{!=}\xintNotEq
1413 \XINT_expr_defbin_b {iiexpr}{!=}\xintiiNotEq

```

11.19.4 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.

```

1414 \def\xintSeq:tl:x #1#2%
1415 {%
1416     \expandafter\XINT_Seq:tl:x
1417     \the\numexpr \xintiCeil{#1}\expandafter.\the\numexpr \xintiFloor{#2}.%
1418 }%
1419 \def\XINT_Seq:tl:x #1.#2.%
1420 {%
1421     \ifnum #2=#1 \xint_dothis\XINT_Seq:tl:x_z\fi
1422     \ifnum #2<#1 \xint_dothis\XINT_Seq:tl:x_n\fi
1423     \xint_orthat\XINT_Seq:tl:x_p
1424     #1.#2.%
1425 }%
1426 \def\XINT_Seq:tl:x_z #1.#2.{\#1/1[0]}}%
1427 \def\XINT_Seq:tl:x_p #1.#2.%
1428 {%
1429     {\#1/1[0]}\ifnum #1=#2 \XINT_Seq:tl:x_e\fi
1430     \expandafter\XINT_Seq:tl:x_p \the\numexpr #1+\xint_c_i.#2.%
1431 }%
1432 \def\XINT_Seq:tl:x_n #1.#2.%
1433 {%
1434     {\#1/1[0]}\ifnum #1=#2 \XINT_Seq:tl:x_e\fi
1435     \expandafter\XINT_Seq:tl:x_n \the\numexpr #1-\xint_c_i.#2.%
1436 }%
1437 \def\XINT_Seq:tl:x_e#1#2.#3.{#1}%

```

\xintiiSeq:tl:x

```

1438 \def\xintiiSeq:tl:x #1#2%
1439 {%
1440     \expandafter\XINT_iiSeq:tl:x
1441     \the\numexpr \xintiCeil{#1}\expandafter.\the\numexpr \xintiFloor{#2}.%
1442 }%
1443 \def\XINT_iiSeq:tl:x #1.#2.%
1444 {%
1445     \ifnum #2=#1 \xint_dothis\XINT_iiSeq:tl:x_z\fi
1446     \ifnum #2<#1 \xint_dothis\XINT_iiSeq:tl:x_n\fi
1447     \xint_orthat\XINT_iiSeq:tl:x_p
1448     #1.#2.%
1449 }%
1450 \def\XINT_iiSeq:tl:x_z #1.#2.{\#1}%
1451 \def\XINT_iiSeq:tl:x_p #1.#2.%
1452 {%

```

```

1453     {#1}\ifnum #1=#2 \XINT_Seq:tl:x_e\fi
1454     \expandafter\XINT_iiSeq:tl:x_p \the\numexpr #1+\xint_c_i.#2.%
1455 }%
1456 \def\XINT_iiSeq:tl:x_n #1.#2.%
1457 {%
1458     {#1}\ifnum #1=#2 \XINT_Seq:tl:x_e\fi
1459     \expandafter\XINT_iiSeq:tl:x_n \the\numexpr #1-\xint_c_i.#2.%
1460 }%

```

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`, `\xintiiSeqA`

```

1461 \def\xintSeqA      {\expandafter\XINT_SeqA\romannumeral0\xintra}%
1462 \def\xintiiSeqA    #1{\expandafter\XINT_iiSeqA\romannumeral`&&@#1;}%
1463 \def\XINT_SeqA    #1]#2{\expandafter\XINT_SeqA_a\romannumeral0\xintra {#2}#1]}%
1464 \def\XINT_iiSeqA#1;#2{\expandafter\XINT_SeqA_a\romannumeral`&&@#2;#1;}%
1465 \def\XINT_SeqA_a #1{\xint_UDzerominusfork
1466                                     #1-{z}%
1467                                     0#1{n}%
1468                                     0-{p}%
1469                                     \krof #1}%

```

`\xintSeqB:tl:x` At 1.4, delayed expansion of start and step done here and not before, for matters of `\xintdeffunc` 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`.

```

1470 \def\xintSeqB:tl:x #1{\expandafter\XINT_SeqB:tl:x\romannumeral`&&@\xintSeqA#1}%
1471 \def\XINT_SeqB:tl:x #1{\csname XINT_SeqB#1:tl:x\endcsname}%
1472 \def\XINT_SeqBz:tl:x #1]#2]#3{{#2}}}%
1473 \def\XINT_SeqBp:tl:x #1]#2]#3{\expandafter\XINT_SeqBp:tl:x_a\romannumeral0\xintra{#3}#2]#1]}%
1474 \def\XINT_SeqBp:tl:x_a #1]#2]#3]%
1475 {%
1476     \xintifCmp{#1}]{#2}}%
1477     {{{#2}}}{#2}}\expandafter\XINT_SeqBp:tl:x_b\romannumeral0\xintadd{#3}]{#2}]{#1]#3}}%
1478 }%
1479 \def\XINT_SeqBp:tl:x_b #1]#2]#3]%
1480 {%
1481     \xintifCmp{#1}]{#2}}%
1482     {#1]}\expandafter\XINT_SeqBp:tl:x_b\romannumeral0\xintadd{#3}]{#1]#2]#3}]{#1}}}%
1483 }%
1484 \def\XINT_SeqBn:tl:x #1]#2]#3{\expandafter\XINT_SeqBn:tl:x_a\romannumeral0\xintra{#3}#2]#1]}%
1485 \def\XINT_SeqBn:tl:x_a #1]#2]#3]%
1486 {%
1487     \xintifCmp{#1}]{#2}}%
1488     {#2]}\expandafter\XINT_SeqBn:tl:x_b\romannumeral0\xintadd{#3}]{#2]#1]#3}]{#2}}}%
1489 }%
1490 \def\XINT_SeqBn:tl:x_b #1]#2]#3]%
1491 {%
1492     \xintifCmp{#1}]{#2}}%

```

```

1493      {{{#1}}}{{{#1}}}\expandafter\XINT_SeqBn:tl:x_b\romannumeral0\xintadd{#3}{{{#1}}#2]#3]]%
1494 }%

\XINTiiSeqB:tl:x
1495 \def\XINTiiSeqB:tl:x #1{\expandafter\XINT_iiSeqB:tl:x\romannumeral`&&\XINTiiSeqA#1}%
1496 \def\XINT_iiSeqB:tl:x #1{\csname XINT_iiSeqB#1:tl:x\endcsname}%
1497 \def\XINT_iiSeqBz:tl:x #1;#2;#3{{{#2}}}%
1498 \def\XINT_iiSeqBp:tl:x #1;#2;#3{\expandafter\XINT_iiSeqBp:tl:x_a\romannumeral`&&#3;#2;#1;}%
1499 \def\XINT_iiSeqBp:tl:x_a #1;#2;#3;%
1500 {%
1501     \XINTiiifCmp{#1}{#2}%
1502     {{{#2}}}{{{#2}}}\expandafter\XINT_iiSeqBp:tl:x_b\romannumeral0\xintiiadd{#3}{#2};#1;#3;}%
1503 }%
1504 \def\XINT_iiSeqBp:tl:x_b #1;#2;#3;%
1505 {%
1506     \XINTiiifCmp{#1}{#2}%
1507     {{{#1}}}\expandafter\XINT_iiSeqBp:tl:x_b\romannumeral0\xintiiadd{#3}{#1};#2;#3;{{{#1}}}%
1508 }%
1509 \def\XINT_iiSeqBn:tl:x #1;#2;#3{\expandafter\XINT_iiSeqBn:tl:x_a\romannumeral`&&#3;#2;#1;}%
1510 \def\XINT_iiSeqBn:tl:x_a #1;#2;#3;%
1511 {%
1512     \XINTiiifCmp{#1}{#2}%
1513     {{{#2}}}\expandafter\XINT_iiSeqBn:tl:x_b\romannumeral0\xintiiadd{#3}{#2};#1;#3;{{{#2}}}%
1514 }%
1515 \def\XINT_iiSeqBn:tl:x_b #1;#2;#3;%
1516 {%
1517     \XINTiiifCmp{#1}{#2}%
1518     {{{#1}}}{{{#1}}}\expandafter\XINT_iiSeqBn:tl:x_b\romannumeral0\xintiiadd{#3}{#1};#2;#3;}%
1519 }%

```

11.20 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.

11.20.1	[...] as «oneple» constructor	360
11.20.2	[...] brackets and : operator for NumPy-like slicing and item indexing syntax	361
11.20.3	Macro layer implementing indexing and slicing	363

11.20.1 [...] 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.

```

1520 \def\XINT_expr_itself_obracket{obracket}%
1521 \catcode`[ 11 \catcode`\[ 11
1522 \def\XINT_expr_defbin_c #1#2#3#4#5#6%
1523 {%
1524     \def #1##1%
1525     {%
1526         \expandafter#3\romannumeral`&&\XINT_expr_getnext
1527     }%

```



```

1528 \def #2##1% op_]
1529 {%
1530 \expanded{\unexpanded{\XINT_expr_put_op_first{##1}}\expandafter}%
1531 \romannumeral`&&\XINT_expr_getop
1532 }%
1533 \def #3##1% until_cbracket_a
1534 {%
1535 \xint_UDsignfork
1536 ##1{\expandafter#4\romannumeral`&&#5}% #5 = op_-xii
1537 -{#4##1}%
1538 \krof
1539 }%
1540 \def #4##1##2% until_cbracket_b
1541 {%
1542 \ifcase ##1\expandafter\XINT_expr_missing_]
1543 \or \expandafter\XINT_expr_missing_]
1544 \or \expandafter#2%
1545 \else
1546 \expandafter #4%
1547 \romannumeral`&&\csname XINT_#6_op_##2\expandafter\endcsname
1548 \fi
1549 }%
1550 }%
1551 \def\XINT_expr_defbin_b #1%
1552 {%
1553 \expandafter\XINT_expr_defbin_c
1554 \csname XINT_#1_op_obracket\expandafter\endcsname
1555 \csname XINT_#1_op_]\expandafter\endcsname
1556 \csname XINT_#1_until_cbracket_a\expandafter\endcsname
1557 \csname XINT_#1_until_cbracket_b\expandafter\endcsname
1558 \csname XINT_#1_op_-xii\endcsname
1559 {#1}%
1560 }%
1561 \XINT_expr_defbin_b {expr}%
1562 \XINT_expr_defbin_b {flexpr}%
1563 \XINT_expr_defbin_b {iiexpr}%
1564 \def\XINT_expr_missing_]
1565 {\XINT_expandableerror{Oops, looks like we are missing a ]. Aborting!}%
1566 \xint_c_ \XINT_expr_done}%
1567 \let\XINT_expr_precedence_]\xint_c_ii

```

11.20.2 [...] 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.

```

1568 \def\XINT_expr_defbin_c #1#2#3#4#5#6%
1569 {%
1570   \def #1##1% \XINT_expr_op_[
1571     {%
1572       \expanded{\unexpanded{#2{##1}}\expandafter}%
1573       \romannumeral`&&\expandafter#3\romannumeral`&&\XINT_expr_getnext
1574     }%
1575   \def #2##1##2##3##4% \XINT_expr_exec_]
1576   {%
1577     \expandafter\XINT_expr_put_op_first
1578     \expanded
1579     {%
1580       {\XINT:NEhook:x:lists\XINT_ListSel_top ##1__##4&({##1}\expandafter}%
1581       \expandafter
1582     }%
1583     \romannumeral`&&\XINT_expr_getop
1584   }%
1585   \def #3##1% \XINT_expr_check_]
1586   {%
1587     \xint_UDsignfork
1588     ##1{\expandafter#4\romannumeral`&&#5}%
1589     -{#4##1}%
1590     \krof
1591   }%
1592   \def #4##1##2% \XINT_expr_checkp_]
1593   {%
1594     \ifcase ##1\XINT_expr_missing_]
1595     \or \XINT_expr_missing_]
1596     \or \expandafter##1\expandafter##2%
1597     \else \expandafter#4%
1598       \romannumeral`&&\csname XINT_#6_op_##2\expandafter\endcsname
1599     \fi
1600   }%
1601 }%
1602 \let\XINT_expr_precedence_ \xint_c_xx
1603 \def\XINT_expr_defbin_b #1%
1604 {%
1605   \expandafter\XINT_expr_defbin_c
1606   \csname XINT_#1_op_[\expandafter\endcsname
1607   \csname XINT_#1_exec_]\expandafter\endcsname
1608   \csname XINT_#1_check_]\expandafter\endcsname
1609   \csname XINT_#1_checkp_]\expandafter\endcsname
1610   \csname XINT_#1_op_-xii\endcsname
1611   {#1}%
1612 }%
1613 \XINT_expr_defbin_b {expr}%
1614 \XINT_expr_defbin_b {flexpr}%
1615 \XINT_expr_defbin_b {iiexpr}%
1616 \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.

```

1617 \def\XINT_expr_defbin_c #1#2#3#4#5#6%
1618 {%
1619   \def #1##1% \XINT_expr_op_:
1620   {%
1621     \expanded{\unexpanded{#2{##1}}\expandafter}%
1622     \romannumeral`&&\expandafter#3\romannumeral`&&\XINT_expr_getnext
1623   }%
1624   \def #2##1##2##3##4% \XINT_expr_exec_:
1625   {%
1626     ##2##3{:#1{0};##4:_}%
1627   }%
1628   \def #3##1% \XINT_expr_check-_:
1629   {\xint_UDsignfork
1630     ##1{\expandafter#4\romannumeral`&&@#5}%
1631     -{#4##1}%
1632   \krof
1633   }%
1634   \def #4##1##2% \XINT_expr_checkp_:
1635   {%
1636     \ifnum ##1>\XINT_expr_precedence_:
1637       \expandafter #4\romannumeral`&&@%
1638       \csname XINT_#6_op_##2\expandafter\endcsname
1639     \else
1640       \expandafter##1\expandafter##2%
1641     \fi
1642   }%
1643 }%
1644 \let\XINT_expr_precedence_:\xint_c_vi
1645 \def\XINT_expr_defbin_b #1%
1646 {%
1647   \expandafter\XINT_expr_defbin_c
1648   \csname XINT_#1_op_:\expandafter\endcsname
1649   \csname XINT_#1_exec_:\expandafter\endcsname
1650   \csname XINT_#1_check-:\expandafter\endcsname
1651   \csname XINT_#1_checkp_:\expandafter\endcsname
1652   \csname XINT_#1_op_-xii\endcsname {#1}%
1653 }%
1654 \XINT_expr_defbin_b {expr}%
1655 \XINT_expr_defbin_b {flexpr}%
1656 \XINT_expr_defbin_b {iiexpr}%

```

11.20.3 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.

```

1657 \def\XINT_ListSel_deeper #1%
1658 {%
1659     \if :#1\xint_dothis\XINT_ListSel_slice_next\fi
1660     \xint_orthat {\XINT_ListSel_extract_next {#1}}%
1661 }%
1662 \def\XINT_ListSel_slice_next #1(
1663 {%
1664     \xintApply{\XINT_ListSel_recurse{#1}}%
1665 }%
1666 \def\XINT_ListSel_extract_next #1(
1667 {%
1668     \xintApplyUnbraced{\XINT_ListSel_recurse{#1}}%
1669 }%
1670 \def\XINT_ListSel_recurse #1#2%
1671 {%
1672     \XINT_ListSel_check #2__#1({#2}\expandafter\empty\empty
1673 }%
1674 \def\XINT_ListSel_check{\expandafter\XINT_ListSel_check_a \string}%
1675 \def\XINT_ListSel_check_a #1%
1676 {%
1677     \if #1\bgroup\xint_dothis\XINT_ListSel_check_is_ok\fi
1678     \xint_orthat\XINT_ListSel_check_leaf
1679 }%
1680 \def\XINT_ListSel_check_leaf #1\expandafter{\expandafter}%
1681 \def\XINT_ListSel_check_is_ok
1682 {%
1683     \expandafter\XINT_ListSel_check_is_ok_a\expandafter{\string}%
1684 }%
1685 \def\XINT_ListSel_check_is_ok_a #1__#2%
1686 {%
1687     \if :#2\xint_dothis{\XINT_ListSel_slice}\fi
1688     \xint_orthat {\XINT_ListSel_nthone {#2}}%
1689 }%
1690 \def\XINT_ListSel_top #1#2%
1691 {%
1692     \if _\noexpand#2%
1693         \expandafter\XINT_ListSel_top_one_or_none\string#1.\else
1694         \expandafter\XINT_ListSel_top_at_least_two\fi
1695 }%
1696 \def\XINT_ListSel_top_at_least_two #1__{\XINT_ListSel_top_ople}%
1697 \def\XINT_ListSel_top_one_or_none #1%
1698 {%

```

```

1699 \if #1\xint_dothis\XINT_ListSel_top_nil\fi
1700 \if #1.\xint_dothis\XINT_ListSel_top_nutple_a\fi
1701 \if #1\bgroup\xint_dothis\XINT_ListSel_top_nutple\fi
1702 \xint_orthat\XINT_ListSel_top_number
1703 }%
1704 \def\XINT_ListSel_top_nil #1\expandafter#2\expandafter{\fi\expandafter}%
1705 \def\XINT_ListSel_top_nutple
1706 {%
1707 \expandafter\XINT_ListSel_top_nutple_a\expandafter{\string}%
1708 }%
1709 \def\XINT_ListSel_top_nutple_a #1_#2#3(#4%
1710 {%
1711 \fi\if :#2\xint_dothis{\XINT_ListSel_slice #3(#4}}\fi
1712 \xint_orthat {\XINT_ListSel_nthone {#2}#3(#4}%
1713 }%
1714 \def\XINT_ListSel_top_number #1_{\fi\XINT_ListSel_top_ople}%
1715 \def\XINT_ListSel_top_ople #1%
1716 {%
1717 \if :#1\xint_dothis\XINT_ListSel_slice\fi
1718 \xint_orthat {\XINT_ListSel_nthone {#1}}%
1719 }%
1720 \def\XINT_ListSel_slice #1%
1721 {%
1722 \expandafter\XINT_ListSel_slice_a \expandafter{\romannumeral0\xintnum{#1}}%
1723 }%
1724 \def\XINT_ListSel_slice_a #1#2;#3#4%
1725 {%
1726 \if _#4\expandafter\XINT_ListSel_s_b
1727 \else\expandafter\XINT_ListSel_slice_b\fi
1728 #1;#3%
1729 }%
1730 \def\XINT_ListSel_s_b #1#2;#3#4%
1731 {%
1732 \if &#4\expandafter\XINT_ListSel_s_last\fi
1733 \XINT_ListSel_s_c #1{#1#2}{#4}%
1734 }%
1735 \def\XINT_ListSel_s_last\XINT_ListSel_s_c #1#2#3(#4%
1736 {%
1737 \if-#1\expandafter\xintKeep\else\expandafter\xintTrim\fi {#2}{#4}%
1738 }%
1739 \def\XINT_ListSel_s_c #1#2#3(#4%
1740 {%
1741 \expandafter\XINT_ListSel_deeper
1742 \expanded{\unexpanded{#3}(\expandafter)\expandafter{%
1743 \romannumeral0%
1744 \if-#1\expandafter\xintkeep\else\expandafter\xinttrim\fi {#2}{#4}}%
1745 }%

```

`\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.

```

1746 \def\XINT_ListSel_nthone #1#2%
1747 {%
1748     \if &#2\expandafter\XINT_ListSel_nthone_last\fi
1749     \XINT_ListSel_nthone_a {#1}{#2}%
1750 }%
1751 \def\XINT_ListSel_nthone_a #1#2(#3%
1752 {%
1753     \expandafter\XINT_ListSel_deeper
1754     \expanded{\unexpanded{#2}(\expandafter)\expandafter{%
1755         \romannumeral0\expandafter\XINT_nthonepy_a\the\numexpr\xintNum{#1}.{#3}}}%
1756 }%
1757 \def\XINT_ListSel_nthone_last\XINT_ListSel_nthone_a #1#2(%#3%
1758 {%
1759     \romannumeral0\expandafter\XINT_nthonepy_a\the\numexpr\xintNum{#1}.%{#3}%
1760 }%

```

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.

```

1761 \def\XINT_ListSel_slice_b #1;#2;#3%
1762 {%
1763     \if &#3\expandafter\XINT_ListSel_slice_last\fi
1764     \expandafter\XINT_ListSel_slice_c \expandafter{\romannumeral0\xintnum{#2}};#1;{#3}%
1765 }%
1766 \def\XINT_ListSel_slice_last\expandafter\XINT_ListSel_slice_c #1;#2;#3(%#4
1767 {%
1768     \expandafter\XINT_ListSel_slice_last_c #1;#2;%{#4}%
1769 }%
1770 \def\XINT_ListSel_slice_last_c #1;#2;#3%
1771 {%
1772     \romannumeral0\XINT_ListSel_slice_d #2;#1;{#3}%
1773 }%
1774 \def\XINT_ListSel_slice_c #1;#2;#3(%#4%
1775 {%
1776     \expandafter\XINT_ListSel_deeper
1777     \expanded{\unexpanded{#3}(\expandafter)\expandafter{%
1778         \romannumeral0\XINT_ListSel_slice_d #2;#1;{#4}}}%
1779 }%
1780 \def\XINT_ListSel_slice_d #1#2;#3#4;%
1781 {%
1782     \xint_UDsignsfork
1783     #1#3\XINT_ListSel_N:N
1784     #1-\XINT_ListSel_N:P
1785     -#3\XINT_ListSel_P:N
1786     --\XINT_ListSel_P:P
1787     \krof #1#2;#3#4;%

```

```

1788 }%
1789 \def\XINT_ListSel_P:P #1;#2;#3%
1790 {%
1791     \unless\ifnum #1<#2 \expandafter\xint_gob_andstop_iii\fi
1792     \xintkeep{#2-#1}{\xintTrim{#1}{#3}}%
1793 }%
1794 \def\XINT_ListSel_N:N #1;#2;#3%
1795 {%
1796     \expandafter\XINT_ListSel_N:N_a
1797     \the\numexpr #2-#1\expandafter;\the\numexpr#1+\xintLength{#3};{#3}%
1798 }%
1799 \def\XINT_ListSel_N:N_a #1;#2;#3%
1800 {%
1801     \unless\ifnum #1>\xint_c_ \expandafter\xint_gob_andstop_iii\fi
1802     \xintkeep{#1}{\xintTrim{\ifnum#2<\xint_c_\xint_c_\else#2\fi}{#3}}%
1803 }%
1804 \def\XINT_ListSel_N:P #1;#2;#3%
1805 {%
1806     \expandafter\XINT_ListSel_N:P_a
1807     \the\numexpr #1+\xintLength{#3};#2;{#3}%
1808 }%
1809 \def\XINT_ListSel_N:P_a #1#2;%
1810     {\if -#1\expandafter\XINT_ListSel_O:P\fi\XINT_ListSel_P:P #1#2;}%
1811 \def\XINT_ListSel_O:P\XINT_ListSel_P:P #1;{\XINT_ListSel_P:P 0;}%
1812 \def\XINT_ListSel_P:N #1;#2;#3%
1813 {%
1814     \expandafter\XINT_ListSel_P:N_a
1815     \the\numexpr #2+\xintLength{#3};#1;{#3}%
1816 }%
1817 \def\XINT_ListSel_P:N_a #1#2;#3;%
1818     {\if -#1\expandafter\XINT_ListSel_P:O\fi\XINT_ListSel_P:P #3;#1#2;}%
1819 \def\XINT_ListSel_P:O\XINT_ListSel_P:P #1;#2;{\XINT_ListSel_P:P #1;0;}%

```

11.21 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).

```

1820 \catcode\ 11
1821 \let\XINT_expr_precedence_&&@ \xint_c_xiv
1822 \def\XINT_expr_op_&&@ #1#2]%
1823 {%

```

```

1824 \expandafter\XINT_expr_put_op_first
1825 \expanded{{{xintE#1{\xint_zapspaces #2 \xint_gobble_i}}}%
1826 \expandafter}\romannumeral`&&\XINT_expr_getop
1827 }%
1828 \def\XINT_iiexpr_op_&& #1#2]%
1829 {%
1830 \expandafter\XINT_expr_put_op_first
1831 \expanded{{{xintiE#1{\xint_zapspaces #2 \xint_gobble_i}}}%
1832 \expandafter}\romannumeral`&&\XINT_expr_getop
1833 }%
1834 \def\XINT_flexpr_op_&& #1#2]%
1835 {%
1836 \expandafter\XINT_expr_put_op_first
1837 \expanded{{{XINTinFloatE#1{\xint_zapspaces #2 \xint_gobble_i}}}%
1838 \expandafter}\romannumeral`&&\XINT_expr_getop
1839 }%
1840 \catcode\ 12

```

11.22 ? as two-way and ?? as three-way «short-circuit» conditionals

Comments undergoing reconstruction.

```

1841 \let\XINT_expr_precedence_? \xint_c_xx
1842 \catcode`- 11
1843 \def\XINT_expr_op_? {\XINT_expr_op__? \XINT_expr_op_-xii}%
1844 \def\XINT_flexpr_op_?{\XINT_expr_op__? \XINT_flexpr_op_-xii}%
1845 \def\XINT_iiexpr_op_?{\XINT_expr_op__? \XINT_iiexpr_op_-xii}%
1846 \catcode`- 12
1847 \def\XINT_expr_op__? #1#2#3%
1848 {\XINT_expr_op__?_a #3!\xint_bye\XINT_expr_exec_? {#1}{#2}{#3}}%
1849 \def\XINT_expr_op__?_a #1{\expandafter\XINT_expr_op__?_b\detokenize{#1}}%
1850 \def\XINT_expr_op__?_b #1%
1851 {\if ?#1\expandafter\XINT_expr_op__?_c\else\expandafter\xint_bye\fi }%
1852 \def\XINT_expr_op__?_c #1{\xint_gob_til_! #1\XINT_expr_op_?? !\xint_bye}%
1853 \def\XINT_expr_op_?? !\xint_bye\xint_bye\XINT_expr_exec_?{\XINT_expr_exec_??}%
1854 \catcode`- 11
1855 \def\XINT_expr_exec_? #1#2%
1856 {%
1857 \expandafter\XINT_expr_check_-_after?\expandafter#1%
1858 \romannumeral`&&\expandafter\XINT_expr_getnext\romannumeral0\xintiifnotzero#2%
1859 }%
1860 \def\XINT_expr_exec_?? #1#2#3%
1861 {%
1862 \expandafter\XINT_expr_check_-_after?\expandafter#1%
1863 \romannumeral`&&\expandafter\XINT_expr_getnext\romannumeral0\xintiifsgn#2%
1864 }%
1865 \def\XINT_expr_check_-_after? #1{%
1866 \def\XINT_expr_check_-_after? ##1##2%
1867 {%
1868 \xint_UDsignfork
1869 ##2{##1}%
1870 #1{##2}%
1871 \krof

```



```
1872 }}\expandafter\XINT_expr_check-_-after?\string -%
1873 \catcode`- 12
```

11.23 ! as postfix factorial operator

```
1874 \let\XINT_expr_precedence_! \xint_c_xx
1875 \def\XINT_expr_op_! #1%
1876 {%
1877   \expandafter\XINT_expr_put_op_first
1878   \expanded{{\romannumeral`&&\XINT:NEhook:f:one:from:one
1879   {\romannumeral`&&\xintFac#1}}\expandafter}\romannumeral`&&\XINT_expr_getop
1880 }%
1881 \def\XINT_flexpr_op_! #1%
1882 {%
1883   \expandafter\XINT_expr_put_op_first
1884   \expanded{{\romannumeral`&&\XINT:NEhook:f:one:from:one
1885   {\romannumeral`&&\XINTinFloatFacdigits#1}}\expandafter}\romannumeral`&&\XINT_expr_getop
1886 }%
1887 \def\XINT_iiexpr_op_! #1%
1888 {%
1889   \expandafter\XINT_expr_put_op_first
1890   \expanded{{\romannumeral`&&\XINT:NEhook:f:one:from:one
1891   {\romannumeral`&&\xintiiFac#1}}\expandafter}\romannumeral`&&\XINT_expr_getop
1892 }%
```

At 1.4g, fix for input "x! == y" via a fake operator !=. The ! is of catcode 11 but this does not matter here. The definition of \XINT_expr_itself_!= is required by the functioning of the scanop macros.

We don't have to worry about "x! = y" as the single-character Boolean comparison = operator has been removed from syntax. Fixing it would have required obeying space tokens when parsing operators. For "x! == y" case, obeying space tokens would not solve "x!==y" input case anyhow.

```
1893 \expandafter
1894 \def\csname XINT_expr_precedence_!=\expandafter\endcsname
1895   \csname XINT_expr_itself_!=\endcsname {\XINT_expr_precedence_! !=}%
1896 \expandafter\def\csname XINT_expr_itself_!=\endcsname{!=}%
```

11.24 User defined variables

11.24.1	\xintdefvar, \xintdefiivar, \xintdeffloatvar	369
11.24.2	\xintunassignvar	373

11.24.1 \xintdefvar, \xintdefiivar, \xintdeffloatvar

1.1 (2014/10/28).

1.2p (2017/12/05) [commented 2017/12/01]. Extends \xintdefvar et al. to accept simultaneous assignments to multiple variables.

1.3c (2018/06/17) [commented 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 (2020/01/31) [commented 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.

1.4 (2020/01/31). Automatic unpacking in case of simultaneous assignments if the expression evaluates to a nutple.

Notes (added much later on 2021/06/10 during preparation of 1.4i):

1. the code did not try to intercept illicit syntax such as `\xintdefvar a,b,c:=<number>;`. It blindly «unpacked» the number handling it as if it was a nutple. The extended functionality added at 1.4i requires to check for such a situation, as the syntax is not illicit anymore.
2. the code was broken in case the expression to evaluate was an ople of length 10 or more, due to a silly mistake at some point during 1.4 development which replaced some `\ifnum` by an `\if`, perhaps due to mental confusion with the fact that functions can have at most 9 arguments, but here the code is about defining variables. Anyway this got fixed as corollary to the 1.4i extension.

1.4c (2021/02/20) [commented 2021/02/20]. One year later I realized I had broken tacit multiplication for situations such as `variable(1+2)`. As hinted at in comments above before 1.4 release I had been doing some deep refactoring here, which I cancelled almost completely in the end... but not quite, and as a result there was a problem that some macro holding braced contents was expanded to late, once it was in old core routines of `xintfrac` not expecting other things than digits. I do an emergency bugfix here with some `\expandafter`'s but I don't have the code in my brain at this time, and don't have the luxury now to invest into it. Let's hope this does not induce breakage elsewhere, and that the February 2020 1.4 did not break something else.

1.4e (2021/05/05) [commented 2021/04/17].

Modifies `\xintdeffloatvar` to round to the prevailing precision (formerly, any operation would induce rounding, but in case of things such as `\xintdeffloatvar foo:=\xintexpr 1/100!\relax`; there was no automatic rounding. One could use `0+` syntax to trigger it, and for oples, some trick like `\xintfloatexpr[\XINTdigits]...\relax` extra wrapper.

1.4g (2021/05/25) [commented 2021/05/22].

The `\expandafter\expandafter\expandafter et al. chain` which was kept by `\XINT_expr_defvar_one_b` for expanding only at time of use the `\XINT_expr_var_foo` in `\XINT_expr_onliteral_foo` were senseless overhead added at 1.4c. This is used only for real variables, not dummy variables or fake variables and it is simpler to have the `\XINT_expr_var_foo` pre-expanded. So let's use some `\edef` here.

The `\XINT_expr_onliteral_foo` is expanded as result of action of `\XINT_expr_op`` (or `\XINT_flexpr_op``, `\XINT_iiexpr_op``) which itself was triggered consuming already an `\XINT_expr_put_op_first`, so its expansion has to produce tokens as expected after `\XINT_expr_put_op_first`: `<precedence token><op token>{expanded value}`.

1.4i (2021/06/11) [commented 2021/06/10].

Implement extended notion of simultaneous assignments: if there are more variables than values, define the extra variables to be nil. If there are less variables than values let the last variable be defined as the ople concatenating all non reclaimed values.

If there are at least two variables, the right hand side, if it turns out to be a nutple, is (as since 1.4) automatically unpacked, then the above rules apply.

1.4i (2021/06/11) [commented 2021/06/11].

Fix the long-standing «seq renaming bug» via a change here of the name of auxiliary macro. Previously «onliteral_<varname>» now «var*_<varname>». I hesitated with using «var_varname*» rather.

Hesitated adding \XINT_expr_letvar_one (motivation: case of simultaneous assignments leading to defining «nil» variables). Finally, no.

```

1897 \catcode`* 11
1898 \def\XINT_expr_defvar_one #1#2%
1899 {%
1900   \XINT_global
1901   \expandafter\edef\csname XINT_expr_varvalue_#1\endcsname {#2}%
1902   \XINT_expr_defvar_one_b {#1}%
1903 }%
1904 \def\XINT_expr_defvar_one_b #1%
1905 {%
1906   \XINT_global
1907   \expandafter\edef\csname XINT_expr_var_#1\endcsname
1908     {\expandafter\noexpand\csname XINT_expr_varvalue_#1\endcsname}%
1909   \XINT_global
1910   \expandafter\edef\csname XINT_expr_var*_#1\endcsname
1911     {\XINT_expr_prec_tacit *\csname XINT_expr_var_#1\endcsname}%
1912   \ifxintverbose\xintMessage{xintexpr}{Info}
1913     {Variable #1 \ifxintglobaldefs globally \fi
1914     defined with value \csname XINT_expr_varvalue_#1\endcsname.}%
1915   \fi
1916 }%
1917 \catcode`* 12
1918 \catcode`~ 13
1919 \catcode`: 12
1920 \def\XINT_expr_defvar_getname #1:#2~%
1921 {%
1922   \endgroup
1923   \def\XINT_defvar_tmpa{#1}\edef\XINT_defvar_tmpc{\xintCSVLength{#1}}%
1924 }%
1925 \def\XINT_expr_defvar #1#2%
1926 {%
1927   \def\XINT_defvar_tmpa{#2}%
1928   \expandafter\XINT_expr_defvar_a\expanded{\unexpanded{#1}}\expandafter}%
1929   \romannumeral\XINT_expr_fetch_to_semicolon
1930 }%
1931 \def\XINT_expr_defvar_a #1#2%
1932 {%
1933   \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.

```

1934 \beginingroup\lccode`~: \lowercase{\let~}\empty
1935 \edef\XINT_defvar_tmpa{\XINT_defvar_tmpa}%
1936 \edef\XINT_defvar_tmpa{\xint_zapspaces_o\XINT_defvar_tmpa}%
1937 \expandafter\XINT_expr_defvar_getname
1938 \detokenize\expandafter{\XINT_defvar_tmpa}:~%
1939 \ifcase\XINT_defvar_tmpc\space
1940 \xintMessage {xintexpr}{Error}
1941 {Aborting: not allowed to declare variable with empty name.}%
1942 \or
1943 \XINT_global
1944 \expandafter
1945 \edef\csname XINT_expr_varvalue_\XINT_defvar_tmpa\endcsname{#1#2\relax}%
1946 \XINT_expr_defvar_one_b\XINT_defvar_tmpa
1947 \else
1948 \edef\XINT_defvar_tmpb{#1#2\relax}%
1949 \edef\XINT_defvar_tmpe{\expandafter\xintLength\expandafter{\XINT_defvar_tmpb}}%
1950 \ifnum\XINT_defvar_tmpe=\xint_c_i
1951 \oodef\XINT_defvar_tmpe{\expandafter\xint_firstofone\XINT_defvar_tmpe}%
1952 \if0\expandafter\expandafter\expandafter\XINT_defvar_checkifnutple
1953 \expandafter\string\XINT_defvar_tmpe _\xint_bye
1954 \oodef\XINT_defvar_tmpe{\expandafter{\XINT_defvar_tmpe}}%
1955 \else
1956 \edef\XINT_defvar_tmpe{\expandafter\xintLength\expandafter{\XINT_defvar_tmpe}}%
1957 \fi
1958 \fi
1959 \xintAssignArray\xintCSVtoList\XINT_defvar_tmpe\to\XINT_defvar_tmpevar
1960 \def\XINT_defvar_tmpe{1}%
1961 \expandafter\XINT_expr_defvar_multiple\XINT_defvar_tmpe\relax
1962 \fi
1963 }%
1964 \def\XINT_defvar_checkifnutple#1%
1965 {%
1966 \if#1_1\fi
1967 \if#1\bgroup1\fi
1968 0\xint_bye
1969 }%
1970 \def\XINT_expr_defvar_multiple
1971 {%
1972 \ifnum\XINT_defvar_tmpe<\XINT_defvar_tmpe\space
1973 \expandafter\XINT_expr_defvar_multiple_one
1974 \else
1975 \expandafter\XINT_expr_defvar_multiple_last\expandafter\empty
1976 \fi
1977 }%
1978 \def\XINT_expr_defvar_multiple_one
1979 {%
1980 \ifnum\XINT_defvar_tmpe>\XINT_defvar_tmpe\space
1981 \expandafter\XINT_expr_defvar_one
1982 \csname XINT_defvar_tmpevar\XINT_defvar_tmpe\endcsname{}%
1983 \edef\XINT_defvar_tmpe{\the\numexpr\XINT_defvar_tmpe+1}%
1984 \expandafter\XINT_expr_defvar_multiple

```

```

1985 \else
1986 \expandafter\XINT_expr_defvar_multiple_one_a
1987 \fi
1988 }%
1989 \def\XINT_expr_defvar_multiple_one_a #1%
1990 {%
1991 \expandafter\XINT_expr_defvar_one
1992 \csname XINT_defvar_tmpvar\XINT_defvar_tmpe\endcsname{{#1}}%
1993 \edef\XINT_defvar_tmpe{\the\numexpr\XINT_defvar_tmpe+1}%
1994 \XINT_expr_defvar_multiple
1995 }%
1996 \def\XINT_expr_defvar_multiple_last #1\relax
1997 {%
1998 \expandafter\XINT_expr_defvar_one
1999 \csname XINT_defvar_tmpvar\XINT_defvar_tmpe\endcsname{{#1}}%
2000 \xintRelaxArray\XINT_defvar_tmpvar
2001 \let\XINT_defvar_tmpe\empty
2002 \let\XINT_defvar_tmpe\empty
2003 \let\XINT_defvar_tmpe\empty
2004 \let\XINT_defvar_tmpe\empty
2005 \let\XINT_defvar_tmpe\empty
2006 }%
2007 \catcode`~ 3
2008 \catcode`.: 11

```

This SafeCatcodes is mainly in the hope that semi-colon ending the expression can still be sanitized.

Pre 1.4e definition:

```

\def\xintdeffloatvar {\xintexprSafeCatcodes\xintdeffloatvar_a}
\def\xintdeffloatvar_a #1={\XINT_expr_defvar\xintthebarefloateval{#1}}

```

This would keep the value (or values) with extra digits, now. If this is actually wanted one can use `\xintdefvar foo:=\xintfloatexpr...\relax`; syntax, but recalling that only operations trigger the rounding inside `\xintfloatexpr`. Some tricks are needed for no operations case if multiple or nested values. But for a single one, one can use simply the `float()` function.

```

2009 \def\xintdefvar {\xintexprSafeCatcodes\xintdefvar_a}%
2010 \def\xintdefvar_a#1={\XINT_expr_defvar\xintthebareeval{#1}}%
2011 \def\xintdefiivar {\xintexprSafeCatcodes\xintdefiivar_a}%
2012 \def\xintdefiivar_a#1={\XINT_expr_defvar\xintthebareiieval{#1}}%
2013 \def\xintdeffloatvar {\xintexprSafeCatcodes\xintdeffloatvar_a}%
2014 \def\xintdeffloatvar_a #1={\XINT_expr_defvar\xintthebareroundedfloateval{#1}}%

```

11.24.2 \xintunassignvar

1.2e (2015/11/22).

1.3d (2019/01/06). 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``.

1.4i (2021/06/11). Track [s/onliteral/var*/](#) change in macro names.

```

2015 \def\xintunassignvar #1{%
2016 \edef\XINT_unvar_tmpe{#1}%

```

```

2017 \edef\XINT_unvar_tmpa {\xint_zapspaces_o\XINT_unvar_tmpa}%
2018 \ifcsname XINT_expr_var_\XINT_unvar_tmpa\endcsname
2019 \ifnum\expandafter\xintLength\expandafter{\XINT_unvar_tmpa}=\@ne
2020 \expandafter\xintnewdummy\XINT_unvar_tmpa
2021 \else
2022 \XINT_global\expandafter
2023 \let\csname XINT_expr_varvalue_\XINT_unvar_tmpa\endcsname\xint_undefined
2024 \XINT_global\expandafter
2025 \let\csname XINT_expr_var_\XINT_unvar_tmpa\endcsname\xint_undefined
2026 \XINT_global\expandafter
2027 \let\csname XINT_expr_var*\XINT_unvar_tmpa\endcsname\xint_undefined
2028 \ifxintverbose\xintMessage {xintexpr}{Info}
2029 {Variable \XINT_unvar_tmpa\space has been
2030 \ifxintglobaldefs globally \fi ``unassigned''.}%
2031 \fi
2032 \fi
2033 \else
2034 \xintMessage {xintexpr}{Warning}
2035 {Error: there was no such variable \XINT_unvar_tmpa\space to unassign.}%
2036 \fi
2037 }%

```

11.25 Support for dummy variables

11.25.1	\xintnewdummy	374
11.25.2	\xintensuredummy, \xintrestorevariable	375
11.25.3	Checking (without expansion) that a symbolic expression contains correctly nested parentheses	376
11.25.4	Fetching balanced expressions E1, E2 and a variable name Name from E1, Name=E2) .	377
11.25.5	Fetching a balanced expression delimited by a semi-colon	377
11.25.6	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()	378
11.25.7	Reserved dummy variables @, @1, @2, @3, @4, @@, @@(1), ..., @@@, @@@(1), ... for recursions	379

11.25.1 \xintnewdummy

Comments under reconstruction.

1.4 adds multi-letter names as usable dummy variables!

1.4i (2021/06/11) [commented 2021/06/11].

s/onlital/var*/ to fix the «seq renaming bug».

```

2038 \catcode`* 11
2039 \def\XINT_expr_makedummy #1%
2040 {%
2041 \edef\XINT_tmpa{\xint_zapspaces #1 \xint_gobble_i}%
2042 \ifcsname XINT_expr_var_\XINT_tmpa\endcsname
2043 \XINT_global
2044 \expandafter\let\csname XINT_expr_var_\XINT_tmpa\old\expandafter\endcsname
2045 \csname XINT_expr_var_\XINT_tmpa\expandafter\endcsname
2046 \fi
2047 \ifcsname XINT_expr_var*\XINT_tmpa\endcsname

```

```

2048 \XINT_global
2049 \expandafter\let\csname XINT_expr_var*\XINT_tmpa/old\expandafter\endcsname
2050 \csname XINT_expr_var*\XINT_tmpa\expandafter\endcsname
2051 \fi
2052 \expandafter\XINT_global
2053 \expanded
2054 {\edef\expandafter\noexpand
2055 \csname XINT_expr_var*\XINT_tmpa\endcsname ##1\relax !\XINT_tmpa##2}%
2056 {{##2}##1\relax !\XINT_tmpa{##2}}%
2057 \expandafter\XINT_global
2058 \expanded
2059 {\edef\expandafter\noexpand
2060 \csname XINT_expr_var*\XINT_tmpa\endcsname ##1\relax !\XINT_tmpa##2}%
2061 {\XINT_expr_prec_tacit *{##2}(##1\relax !\XINT_tmpa{##2}}%)
2062 }%
2063 \xintApplyUnbraced \XINT_expr_makedummy {abcdefghijklmnopqrstuvwxyz}%
2064 \xintApplyUnbraced \XINT_expr_makedummy {ABCDEFGHIJKLMNOPQRSTUVWXYZ}%
2065 \def\xintnewdummy #1{%
2066 \XINT_expr_makedummy{#1}%
2067 \ifxintverbose\xintMessage {xintexpr}{Info}%
2068 {\XINT_tmpa\space now
2069 \ifxintglobaldefs globally \fi usable as dummy variable.}%
2070 \fi
2071 }%
2072 \catcode`* 12
2073 % \begin{macrocode}
2074 % The |nil| variable was need in |xint < 1.4| (with some other meaning)
2075 % in places the syntax could not allow emptiness, such as |,,|, and
2076 % other things, but at |1.4| meaning as changed.
2077 %
2078 % The other variables are new with |1.4|.
2079 % Don't use the |None|, it is tentative, and may be input as |[ ]|.
2080 %
2081 % Refactored at |1.4i| to define them as really genuine variables,
2082 % i.e. also with associated |var*| macros involved in tacit multiplication
2083 % (even though it will be broken with |nil|, and with |None| in |\xintiexpr|).
2084 % No real reason, because |\XINT_expr_op__| managed them fine even in absence
2085 % of |var*| macros.
2086 % \begin{macrocode}
2087 \XINT_expr_defvar_one{nil}{}%
2088 \XINT_expr_defvar_one{None}{}% ? tentative
2089 \XINT_expr_defvar_one{false}{{0}}% Maple, TeX
2090 \XINT_expr_defvar_one{true}{{1}}%
2091 \XINT_expr_defvar_one{False}{{0}}% Python
2092 \XINT_expr_defvar_one{True}{{1}}%

```

11.25.2 \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.

```

2093 \def\xintensuredummy #1{%
2094   \XINT_expr_makedummy{#1}%
2095   \ifxintverbose\xintMessage {xintexpr}{Info}%
2096     {\XINT_tmpa\space now
2097     \ifxintglobaldefs globally \fi usable as dummy variable.&&J
2098     Issue \string\xintrestorevariable{\XINT_tmpa} to restore former meaning.}%
2099   \fi
2100 }%
2101 \def\xintrestorevariablesilently #1{%
2102   \edef\XINT_tmpa{\xint_zapspace #1 \xint_gobble_i}%
2103   \ifcsname XINT_expr_var_\XINT_tmpa/old\endcsname
2104     \XINT_global
2105     \expandafter\let\csname XINT_expr_var_\XINT_tmpa\expandafter\endcsname
2106       \csname XINT_expr_var_\XINT_tmpa/old\expandafter\endcsname
2107   \fi
2108   \ifcsname XINT_expr_var*_\XINT_tmpa/old\endcsname
2109     \XINT_global
2110     \expandafter\let\csname XINT_expr_var*_\XINT_tmpa\expandafter\endcsname
2111       \csname XINT_expr_var*_\XINT_tmpa/old\expandafter\endcsname
2112   \fi
2113 }%
2114 \def\xintrestorevariable #1{%
2115   \xintrestorevariablesilently {#1}%
2116   \ifxintverbose\xintMessage {xintexpr}{Info}%
2117     {\XINT_tmpa\space
2118     \ifxintglobaldefs globally \fi restored to its earlier status, if any.}%
2119   \fi
2120 }%

```

11.25.3 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.

```

2121 \def\XINT_isbalanced_a #1({\XINT_isbalanced_b #1)\xint_bye }%
2122 \def\XINT_isbalanced_b #1)#2%
2123   {\xint_bye #2\XINT_isbalanced_c\xint_bye\XINT_isbalanced_error }%
2124   if #2 is not \xint_bye, a ) was found, but there was no (. Hence error -> -1
2124 \def\XINT_isbalanced_error #1)\xint_bye {\xint_c_mone}%
2125   #2 was \xint_bye, was there a ) in original #1?
2125 \def\XINT_isbalanced_c\xint_bye\XINT_isbalanced_error #1%
2126   {\xint_bye #1\XINT_isbalanced_yes\xint_bye\XINT_isbalanced_d #1}%
2127   #1 is \xint_bye, there was never ( nor ) in original #1, hence OK.
2127 \def\XINT_isbalanced_yes\xint_bye\XINT_isbalanced_d\xint_bye )\xint_bye {\xint_c_ }%
2128   #1 is not \xint_bye, there was indeed a ( in original #1. We check if we see a ). If we do, we then
2129   loop until no ( nor ) is to be found.

```



```

2128 \def\XINT_isbalanced_d #1)#2%
2129   {\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.
2130 \def\XINT_isbalanced_no\xint_bye #1\xint_bye\xint_bye {\xint_c_i }%

```

11.25.4 Fetching balanced expressions E1, E2 and a variable name Name from E1, Name=E2)

Multi-letter dummy variables added at 1.4.

```

2131 \def\XINT_expr_fetch_E_comma_V_equal_E_a #1#2,%
2132 {%
2133   \ifcase\XINT_isbalanced_a \relax #1#2(\xint_bye)\xint_bye
2134     \expandafter\XINT_expr_fetch_E_comma_V_equal_E_c
2135     \or\expandafter\XINT_expr_fetch_E_comma_V_equal_E_b
2136     \else\expandafter\xintError:noopening
2137   \fi {#1#2},%
2138 }%
2139 \def\XINT_expr_fetch_E_comma_V_equal_E_b #1,%
2140   {\XINT_expr_fetch_E_comma_V_equal_E_a {#1},}%
2141 \def\XINT_expr_fetch_E_comma_V_equal_E_c #1,#2#3=%
2142 {%
2143   \expandafter\XINT_expr_fetch_E_comma_V_equal_E_d\expandafter
2144   {\expanded{{\xint_zapspaces #2#3 \xint_gobble_i}}{#1}}}%
2145 }%
2146 \def\XINT_expr_fetch_E_comma_V_equal_E_d #1#2#3)%
2147 {%
2148   \ifcase\XINT_isbalanced_a \relax #2#3(\xint_bye)\xint_bye
2149     \or\expandafter\XINT_expr_fetch_E_comma_V_equal_E_e
2150     \else\expandafter\xintError:noopening
2151   \fi
2152   {#1}{#2#3}%
2153 }%
2154 \def\XINT_expr_fetch_E_comma_V_equal_E_e #1#2{\XINT_expr_fetch_E_comma_V_equal_E_d {#1}{#2}}}%

```

11.25.5 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.

```

2155 \def\XINT_expr_fetch_to_semicolon {\XINT_expr_fetch_to_semicolon_a {} \empty}%
2156 \def\XINT_expr_fetch_to_semicolon_a #1#2;%
2157 {%
2158   \ifcase\XINT_isbalanced_a \relax #1#2(\xint_bye)\xint_bye
2159     \xint_dothis{\expandafter\XINT_expr_fetch_to_semicolon_c}%
2160     \or\xint_dothis{\expandafter\XINT_expr_fetch_to_semicolon_b}%
2161     \else\expandafter\xintError:noopening
2162   \fi\xint_orthat{} \expandafter{#2}{#1}%
2163 }%
2164 \def\XINT_expr_fetch_to_semicolon_b #1#2{\XINT_expr_fetch_to_semicolon_a {#2#1;} \empty}%
2165 \def\XINT_expr_fetch_to_semicolon_c #1#2{\xint_c_{#2#1}}%

```

11.25.6 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.

```
2166 \expandafter\def\csname XINT_expr_itself_++\endcsname {++}%
2167 \expandafter\def\csname XINT_expr_itself_++)\endcsname {++)}%
2168 \expandafter\let\csname XINT_expr_precedence_++\endcsname \xint_c_i
2169 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
2170     \expandafter\def\csname XINT_#1_op_++\endcsname ##1##2\relax
2171     {\expandafter\XINT_expr_foundend
2172         \expanded{{+{\XINT:NEhook:f:one:from:one:direct\xintiCeil##1}}}%
2173     }%
2174 }%
```

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.

```
2175 \catcode`? 3
2176 \def\XINT_expr_func_break #1#2#3{#1#2{?#3}}%
2177 \catcode`? 11
2178 \let\XINT_flexpr_func_break \XINT_expr_func_break
2179 \let\XINT_iiexpr_func_break \XINT_expr_func_break
```

The omit and abort keywords Comments are currently undergoing reconstruction.

The mechanism is somewhat complex. The operator `!?` will fetch a dummy value `!` or `^` which is then recognized into the loops implementing the various `seq` etc... construct using dummy variables and implement `omit` and `abort`.

In May 2021 I realized that the January 2020 1.4 had broken `omit` and `abort` if used inside a `subs()`. The definition

```
\edef\XINT_expr_var_omit #1\relax !{1\string !?!\relax !}
```

conflicted with the 1.4 refactoring of «subs» and similar things which had replaced formerly clean-up macros (of `!` and what's next, as in now defunct `\def\XINT_expr_subx:_end #1!#2#3{#1}` which was involved in `subs` mechanism, and by the way would be incompatible with multi-letter dummy variables) by usage of an `\iffalse` as in `"\relax\iffalse\relax !"` to delimit a sub-expression, which was supposed to be clever (the `"\relax !"` being delimiter for dummy variables).

This `\iffalse` from `subs` mechanism ended up being gobbled by `omit/abort` thus inducing breakage.

Grabbing `\relax #!` would be a fix but looks a bit dangerous, as there can be a subexpression after the omit or abort bringing its own `\relax`, although this is very very unlikely.

I considered to modify the dummy variables delimiter from `\relax !` to `\xint_Bye !` for example but got afraid from the ramifications, as all structures handling dummy variables would have needed refactoring.

So finally things here remain unchanged and the refactoring to fix this breakage was done in `\XINT_allexpr_subsx` (and also `subsm`). Done at 1.4h. See `\XINT_allexpr_subsx` for comments.

```
2180 \edef\XINT_expr_var_omit #1\relax !{1\string !?!\relax !}%
2181 \edef\XINT_expr_var_abort #1\relax !{1\string !?^\relax !}%
2182 \def\XINT_expr_itself_! ? {! ?}%
2183 \def\XINT_expr_op_! ? #1#2\relax{\XINT_expr_foundend{#2}}%
2184 \let\XINT_iiexpr_op_! ? \XINT_expr_op_! ?
2185 \let\XINT_flexpr_op_! ? \XINT_expr_op_! ?
2186 \let\XINT_expr_precedence_! ? \xint_c_iv
```

The semi-colon Obsolete comments undergoing re-construction

```
2187 \xintFor #1 in {expr, flexpr, iiexpr} \do {%
2188     \expandafter\def\csname XINT_#1_op_;\endcsname {\xint_c_i ;}%
2189 }%
2190 \expandafter\let\csname XINT_expr_precedence_;\endcsname\xint_c_i
2191 \expandafter\def\csname XINT_expr_itself_;\endcsname {}%
2192 \expandafter\let\csname XINT_expr_precedence_;\endcsname\xint_c_i
```

11.25.7 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.

```
2193 \catcode`* 11
2194 \def\XINT_expr_var_@ #1~#2{{#2}#1~{#2}}%
2195 \def\XINT_expr_var*_@ #1~#2{\XINT_expr_prec_tacit *{#2}({#1~{#2}}}%
2196 \expandafter
2197 \def\csname XINT_expr_var_@1\endcsname #1~#2{{{#2}}#1~{#2}}%
2198 \expandafter
2199 \def\csname XINT_expr_var_@2\endcsname #1~#2#3{{{#3}}#1~{#2}{#3}}%
2200 \expandafter
2201 \def\csname XINT_expr_var_@3\endcsname #1~#2#3#4{{{#4}}#1~{#2}{#3}{#4}}%
2202 \expandafter
2203 \def\csname XINT_expr_var_@4\endcsname #1~#2#3#4#5{{{#5}}#1~{#2}{#3}{#4}{#5}}%
2204 \expandafter\def\csname XINT_expr_var_@1\endcsname #1~#2%
2205     {\XINT_expr_prec_tacit *{#2}({#1~{#2}}}%
2206 \expandafter\def\csname XINT_expr_var_@2\endcsname #1~#2#3%
2207     {\XINT_expr_prec_tacit *{#3}({#1~{#2}{#3}}}%
2208 \expandafter\def\csname XINT_expr_var_@3\endcsname #1~#2#3#4%
2209     {\XINT_expr_prec_tacit *{#4}({#1~{#2}{#3}{#4}}}%
2210 \expandafter\def\csname XINT_expr_var_@4\endcsname #1~#2#3#4#5%
2211     {\XINT_expr_prec_tacit *{#5}({#1~{#2}{#3}{#4}{#5}}}%
2212 \catcode`* 12
```

```

2213 \catcode`? 3
2214 \def\XINT_expr_func_@@ #1#2#3#4~#5?%
2215 {%
2216   \expandafter#1\expandafter#2\expandafter{\expandafter{%
2217     \romannumeral0\xintntheltnoexpand{\xintNum#3}{#5}}}\#4~#5?%
2218 }%
2219 \def\XINT_expr_func_@@@ #1#2#3#4~#5~#6?%
2220 {%
2221   \expandafter#1\expandafter#2\expandafter{\expandafter{%
2222     \romannumeral0\xintntheltnoexpand{\xintNum#3}{#6}}}\#4~#5~#6?%
2223 }%
2224 \def\XINT_expr_func_@@@ #1#2#3#4~#5~#6~#7?%
2225 {%
2226   \expandafter#1\expandafter#2\expandafter{\expandafter{%
2227     \romannumeral0\xintntheltnoexpand{\xintNum#3}{#7}}}\#4~#5~#6~#7?%
2228 }%
2229 \let\XINT_flexpr_func_@@\XINT_expr_func_@@
2230 \let\XINT_flexpr_func_@@@\XINT_expr_func_@@@
2231 \let\XINT_flexpr_func_@@@@\XINT_expr_func_@@@
2232 \def\XINT_iexpr_func_@@ #1#2#3#4~#5?%
2233 {%
2234   \expandafter#1\expandafter#2\expandafter{\expandafter{%
2235     \romannumeral0\xintntheltnoexpand{\xint_firstofone#3}{#5}}}\#4~#5?%
2236 }%
2237 \def\XINT_iexpr_func_@@@ #1#2#3#4~#5~#6?%
2238 {%
2239   \expandafter#1\expandafter#2\expandafter{\expandafter{%
2240     \romannumeral0\xintntheltnoexpand{\xint_firstofone#3}{#6}}}\#4~#5~#6?%
2241 }%
2242 \def\XINT_iexpr_func_@@@ #1#2#3#4~#5~#6~#7?%
2243 {%
2244   \expandafter#1\expandafter#2\expandafter{\expandafter{%
2245     \romannumeral0\xintntheltnoexpand{\xint_firstofone#3}{#7}}}\#4~#5~#6~#7?%
2246 }%
2247 \catcode`? 11

```

11.26 Pseudo-functions involving dummy variables and generating scalars or sequences

11.26.1	Comments	380
11.26.2	subs(): substitution of one variable	382
11.26.3	subsm(): simultaneous independent substitutions	383
11.26.4	subsn(): leaner syntax for nesting (possibly dependent) substitutions	384
11.26.5	seq(): sequences from assigning values to a dummy variable	386
11.26.6	iter()	387
11.26.7	add(), mul()	388
11.26.8	rseq()	389
11.26.9	iterr()	390
11.26.10	rrseq()	391

11.26.1 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 `\xintNew-Expr` !)

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 (super-

fluous 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 !

11.26.2 subs(): substitution of one variable

```
2248 \def\XINT_expr_onliteral_subs
2249 {%
2250     \expandafter\XINT_allexpr_subsx_f
2251     \romannumeral'&&\XINT_expr_fetch_E_comma_V_equal_E_a {%
2252 }%
2253 \def\XINT_allexpr_subsx_f #1#2{\xint_c_ii^v `{subsx}#2)\relax #1}%
2254 \def\XINT_expr_func_subsx #1#2{\XINT_allexpr_subsx \xintbareeval }%
2255 \def\XINT_flexpr_func_subsx #1#2{\XINT_allexpr_subsx \xintbarefloateval}%
2256 \def\XINT_iiexpr_func_subsx #1#2{\XINT_allexpr_subsx \xintbareiieval }%
```

#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

1.4 was doing something clever to get rid of the ! and tokens following it, via an `\iffalse...\fi` which erased them and propagated the expansion to trigger the `getopt`:

```
\expanded\bgroup\romannumeral0#1#4\relax \iffalse\relax !#3{#2}{\fi\expandafter}
```

But sadly, with a delay of more than one year later (right after having released 1.4g) I realized that this had broken `omit` and `abort` if inside a `subs`. As `omit` and `abort` would clean all up to `\relax !`, this meant here swallowing in particular the above `\iffalse`, leaving a dangling `\fi`. I had the files which show this bug already at time of 1.4 release but did not compile them, and they were not included in my test suite.

I hesitated with modifying the delimiter from `"\relax !<varname>"` (catcode 11 !) to `"\relax \xint_Bye<varname>"` for the dummy variables which would have allowed some trickery using

\xint_Bye...\xint_bye clean-up but got afraid from the breakage potential of such refactoring with many induced changes.

A variant like this:

```
\def\xint_allexpr_subsx #1#2#3#4
{
  \expandafter\xint_expr_clean_and_put_op_first
  \expanded
  {\romannumeral0#1#4\relax !#3{#2}\xint:\expandafter}\romannumeral`&&\xint_expr_getop
}
\def\xint_expr_clean_and_put_op_first #1#2\xint:#3#4{#3#4{#1}}
```

breaks nesting: the braces make variables encountered in #4 unable to match their definition. This would work:

```
\def\xint_allexpr_subsx #1#2#3#4
{
  \expandafter\xint_allexpr_subsx_clean\romannumeral0#1#4\relax !#3{#2}\xint:
}
\def\xint_allexpr_subsx_clean #1#2\xint:
{
  \expandafter\xint_expr_put_op_first
  \expanded{\unexpanded{#1}}\expandafter}\romannumeral`&&\xint_expr_getop
}

```

(not tested).

But in the end I decided to simply fix the first envisioned code above. This accepts expansion of supposedly inert #3{#2}. There is again the \iffalse but it is moved to the right. This change limits possibly hacky future developments. Done at 1.4h (2021/01/27).

No need for the \expandafter's from \xint_expr_put_op_first in \xint_expr_clean_and_put_op_first.

```
2257 \def\xint_allexpr_subsx #1#2#3#4%
2258 {%
2259   \expandafter\xint_expr_clean_and_put_op_first
2260   \expanded
2261   \bgroup\romannumeral0#1#4\relax !#3{#2}\xint:\iffalse{\fi\expandafter}%
2262   \romannumeral`&&\xint_expr_getop
2263 }%
2264 \def\xint_expr_clean_and_put_op_first #1#2\xint:#3#4{#3#4{#1}}%
```

11.26.3 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 subsm() the final parenthesis must be there from the start.

```
2265 \def\xint_expr_onliteral_subsm
2266 {%
2267   \expandafter\xint_allexpr_subsm_f
2268   \romannumeral`&&\xint_expr_fetch_E_comma_V_equal_E_a {}%
2269 }%
2270 \def\xint_allexpr_subsm_f #1#2{\xint_c_ii^v `{subsmx}#2)\relax #1}%
2271 \def\xint_expr_func_subsmx
2272 {%
2273   \expandafter\xint_allexpr_subsmx\expandafter\xintbareeval
2274   \expanded\bgroup{\iffalse}\fi\xint_allexpr_subsm_A\xint_expr_oparen
2275 }%
```



```

2276 \def\XINT_flexpr_func_subsmx
2277 {%
2278   \expandafter\XINT_allexpr_subsmx\expandafter\xintbarefloateval
2279   \expanded\bgroup{\iffalse}\fi\XINT_allexpr_subsm_A\XINT_flexpr_oparen
2280 }%
2281 \def\XINT_iiexpr_func_subsmx
2282 {%
2283   \expandafter\XINT_allexpr_subsmx\expandafter\xintbareiieval
2284   \expanded\bgroup{\iffalse}\fi\XINT_allexpr_subsm_A\XINT_iiexpr_oparen
2285 }%
2286 \def\XINT_allexpr_subsm_A #1#2#3%
2287 {%
2288   \ifx#2\xint_c_
2289     \expandafter\XINT_allexpr_subsm_done
2290   \else
2291     \expandafter\XINT_allexpr_subsm_B
2292   \fi #1%
2293 }%
2294 \def\XINT_allexpr_subsm_B #1#2#3#4=%
2295 {%
2296   {#2}\relax !\xint_zapspaces#3#4 \xint_gobble_i
2297   \expandafter\XINT_allexpr_subsm_A\expandafter#1\romannumeral`&&@#1%
2298 }%

    #1 = \xintbareeval, or \xintbarefloateval or \xintbareiieval
    #2 = evaluation of last variable assignment
2299 \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
    Refactored at 1.4h as for \XINT_allexpr_subsx, see comments there related to the omit/abort co-
    nundrum.
2300 \def\XINT_allexpr_subsmx #1#2#3#4%
2301 {%
2302   \expandafter\XINT_expr_clean_and_put_op_first
2303   \expanded
2304   \bgroup\romannumeral0#1#4\relax !#3#2\xint:\iffalse{\fi\expandafter}%
2305   \romannumeral`&&\XINT_expr_getop
2306 }%

```

11.26.4 subsn(): leaner syntax for nesting (possibly dependent) substitutions

New with 1.4. 2020/01/24

```

2307 \def\XINT_expr_onliteral_subsn
2308 {%
2309   \expandafter\XINT_allexpr_subsn_f
2310   \romannumeral`&&\XINT_expr_fetch_E_comma_V_equal_E_a {}%
2311 }%
2312 \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
 This one needed no rectoring at 1.4h to fix the omit/abort problem, as there was no \iffalse..\fi
 clean-up: the clean-up is done directly via \XINT_allexpr_subsnx_J.

I only added usage of \XINT_expr_put_op_first_noexpand. There may be other locations where it
 could be used, but I can't afford now reviewing usage. For next release after 1.4h bugfix.

```

2313 \def\XINT_allexpr_subsn_g #1#2#3%
2314 {%
2315     \expandafter\XINT_allexpr_subsn_h
2316     \expanded\bgroup{\iffalse}\fi\expandafter\XINT_allexpr_subsn_B
2317     \romannumeral\XINT_expr_fetch_to_semicolon #1=#3;\hbox=;;^{#2}%
2318 }%
2319 \def\XINT_allexpr_subsn_B #1{\XINT_allexpr_subsn_C #1\vbox}%
2320 \def\XINT_allexpr_subsn_C #1#2=#3\vbox
2321 {%
2322     \ifx\hbox#1\iffalse{\fi}\expandafter\else
2323     {\xint_zapspaces #1#2 \xint_gobble_i}};\unexpanded{{{#3}}}%
2324     \expandafter\XINT_allexpr_subsn_B
2325     \romannumeral\expandafter\XINT_expr_fetch_to_semicolon\fi
2326 }%
2327 \def\XINT_allexpr_subsn_h
2328 {%
2329     \xint_c_ii^v `{subsnx}\romannumeral0\xintreverseorder
2330 }%
2331 \def\XINT_expr_func_subsnx #1#2#3#4#5;#6%
2332 {%
2333     \xint_gob_til_ ^ #6\XINT_allexpr_subsnx_H ^%
2334     \expandafter\XINT_allexpr_subsnx\expandafter
2335     \xintbareeval\romannumeral0\xintbareeval #5\relax !#4{#3}\xintundefined
2336     {\relax !#4{#3}\relax !#6}%
2337 }%
2338 \def\XINT_iiexpr_func_subsnx #1#2#3#4#5;#6%
2339 {%
2340     \xint_gob_til_ ^ #6\XINT_allexpr_subsnx_H ^%
2341     \expandafter\XINT_allexpr_subsnx\expandafter
2342     \xintbareiieval\romannumeral0\xintbareiieval #5\relax !#4{#3}\xintundefined
2343     {\relax !#4{#3}\relax !#6}%
2344 }%
2345 \def\XINT_flexpr_func_subsnx #1#2#3#4#5;#6%
2346 {%
2347     \xint_gob_til_ ^ #6\XINT_allexpr_subsnx_H ^%
2348     \expandafter\XINT_allexpr_subsnx\expandafter
2349     \xintbarefloateval\romannumeral0\xintbarefloateval #5\relax !#4{#3}\xintundefined
2350     {\relax !#4{#3}\relax !#6}%
2351 }%
2352 \def\XINT_allexpr_subsnx #1#2!#3\xintundefined#4#5;#6%
2353 {%
2354     \xint_gob_til_ ^ #6\XINT_allexpr_subsnx_I ^%
2355     \expandafter\XINT_allexpr_subsnx\expandafter
2356     #1\romannumeral0#1#5\relax !#4{#2}\xintundefined
2357     {\relax !#4{#2}\relax !#6}%
2358 }%
2359 \def\XINT_allexpr_subsnx_H ^#1\romannumeral0#2#3!#4\xintundefined #5#6%
    
```

```

2360 {%
2361     \expandafter\XINT_allexpr_subsnx_J\romannumeral0#2#6#5%
2362 }%
2363 \def\XINT_allexpr_subsnx_I ^#1\romannumeral0#2#3\xintundefined #4#5%
2364 {%
2365     \expandafter\XINT_allexpr_subsnx_J\romannumeral0#2#5#4%
2366 }%
2367 \def\XINT_allexpr_subsnx_J #1#2^%
2368 {%
2369     \expandafter\XINT_expr_put_op_first_noexpand
2370     \expanded{\unexpanded{{#1}}\expandafter}\romannumeral`&&\XINT_expr_getop
2371 }%
2372 \def\XINT_expr_put_op_first_noexpand#1#2#3{#2#3{#1}}%

```

11.26.5 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.

```

2373 \def\XINT_allexpr_seq_f #1#2{\xint_c_ii^v `{seqx}#2)\relax #1}%
2374 \def\XINT_expr_onliteral_seq
2375 {\expandafter\XINT_allexpr_seq_f\romannumeral`&&\XINT_expr_fetch_E_comma_V_equal_E_a {}}%
2376 \def\XINT_expr_func_seqx #1#2{\XINT:NEhook:seqx\XINT_allexpr_seqx\xintbareeval}%
2377 \def\XINT_flexpr_func_seqx #1#2{\XINT:NEhook:seqx\XINT_allexpr_seqx\xintbarefloateval}%
2378 \def\XINT_iexpr_func_seqx #1#2{\XINT:NEhook:seqx\XINT_allexpr_seqx\xintbareiieval}%
2379 \def\XINT_allexpr_seqx #1#2#3#4%
2380 {%
2381     \expandafter\XINT_expr_put_op_first
2382     \expanded \bgroup {\iffalse}\fi\XINT_expr_seq:_b {#1#4\relax !#3}#2^%
2383     \XINT_expr_cb_and_getop
2384 }%
2385 \def\XINT_expr_cb_and_getop{\iffalse{\fi\expandafter}\romannumeral`&&\XINT_expr_getop}%

```

Comments undergoing reconstruction.

```

2386 \catcode`? 3
2387 \def\XINT_expr_seq:_b #1#2%
2388 {%
2389     \ifx +#2\xint_dothis\XINT_expr_seq:_Ca\fi
2390     \ifx !#2!\xint_dothis\XINT_expr_seq:_noop\fi
2391     \ifx ^#2\xint_dothis\XINT_expr_seq:_end\fi
2392     \xint_orthat{\XINT_expr_seq:_c}{#2}{#1}%
2393 }%
2394 \def\XINT_expr_seq:_noop #1{\XINT_expr_seq:_b}%
2395 \def\XINT_expr_seq:_end #1#2{\iffalse{\fi}}%
2396 \def\XINT_expr_seq:_c #1#2{\expandafter\XINT_expr_seq:_d\romannumeral0#2{{#1}}{{#2}}}%
2397 \def\XINT_expr_seq:_d #1{\ifx ^#1\xint_dothis\XINT_expr_seq:_abort\fi
2398     \ifx ?#1\xint_dothis\XINT_expr_seq:_break\fi
2399     \ifx !#1\xint_dothis\XINT_expr_seq:_omit\fi
2400     \xint_orthat{\XINT_expr_seq:_goon}{#1}}%
2401 \def\XINT_expr_seq:_abort #1!#2^{\iffalse{\fi}}%
2402 \def\XINT_expr_seq:_break #1!#2^{#1\iffalse{\fi}}%
2403 \def\XINT_expr_seq:_omit #1!#2#{\expandafter\XINT_expr_seq:_b\xint_gobble_i}%
2404 \def\XINT_expr_seq:_goon #1!#2#{#1\expandafter\XINT_expr_seq:_b\xint_gobble_i}%

```

```

2405 \def\XINT_expr_seq:_Ca #1#2#3{\XINT_expr_seq:_Cc#3.{#2}}%
2406 \def\XINT_expr_seq:_Cb #1{\expandafter\XINT_expr_seq:_Cc\the\numexpr#1+\xint_c_i.}%
2407 \def\XINT_expr_seq:_Cc #1.#2{\expandafter\XINT_expr_seq:_D\romannumeral0#2{{#1}}{{#1}}{{#2}}}%
2408 \def\XINT_expr_seq:_D #1{\ifx ^#1\xint_dothis\XINT_expr_seq:_abort\fi
2409         \ifx ?#1\xint_dothis\XINT_expr_seq:_break\fi
2410         \ifx !#1\xint_dothis\XINT_expr_seq:_omit\fi
2411         \xint_orthat{\XINT_expr_seq:_Goon {#1}}}%
2412 \def\XINT_expr_seq:_omit #1!#2#{\expandafter\XINT_expr_seq:_Cb\xint_gobble_i}%
2413 \def\XINT_expr_seq:_Goon #1!#2#{\#1\expandafter\XINT_expr_seq:_Cb\xint_gobble_i}%

```

11.26.6 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.

```

2414 \def\XINT_expr_func_iter {\XINT_allexpr_iter \xintbareeval }%
2415 \def\XINT_flexpr_func_iter {\XINT_allexpr_iter \xintbarefloateval }%
2416 \def\XINT_iexpr_func_iter {\XINT_allexpr_iter \xintbareiieval }%
2417 \def\XINT_allexpr_iter #1#2#3#4%
2418 {%
2419     \expandafter\XINT_expr_iterx
2420     \expandafter#1\expanded{\unexpanded{{#4}}\expandafter}%
2421     \romannumeral' &&@\XINT_expr_fetch_E_comma_V_equal_E_a {}%
2422 }%
2423 \def\XINT_expr_iterx #1#2#3#4%
2424 {%
2425     \XINT:NEhook:iter\XINT_expr_itery\romannumeral0#1(#4)\relax {#2}#3#1%
2426 }%
2427 \def\XINT_expr_itery #1#2#3#4#5%
2428 {%
2429     \expandafter\XINT_expr_put_op_first
2430     \expanded \bgroup {\iffalse}\fi
2431     \XINT_expr_iter:_b {#5#4\relax !#3}#1^~{#2}\XINT_expr_cb_and_getop
2432 }%
2433 \def\XINT_expr_iter:_b #1#2%
2434 {%
2435     \ifx +#2\xint_dothis\XINT_expr_iter:_Ca\fi
2436     \ifx !#2!\xint_dothis\XINT_expr_iter:_noop\fi
2437     \ifx ^#2\xint_dothis\XINT_expr_iter:_end\fi
2438     \xint_orthat{\XINT_expr_iter:_c}{#2}{#1}%
2439 }%
2440 \def\XINT_expr_iter:_noop #1{\XINT_expr_iter:_b }%
2441 \def\XINT_expr_iter:_end #1#2~#3{#3\iffalse}\fi}%
2442 \def\XINT_expr_iter:_c #1#2{\expandafter\XINT_expr_iter:_d\romannumeral0#2{{#1}}{{#2}}}%
2443 \def\XINT_expr_iter:_d #1{\ifx ^#1\xint_dothis\XINT_expr_iter:_abort\fi
2444         \ifx ?#1\xint_dothis\XINT_expr_iter:_break\fi

```

```

2445             \ifx !#1\xint_dothis\xINT_expr_iter:_omit\fi
2446             \xint_orthat{\XINT_expr_iter:_goon {#1}}}%
2447 \def\xINT_expr_iter:_abort #1!#2^~#3{#3\iffalse{\fi}}%
2448 \def\xINT_expr_iter:_break #1!#2^~#3{#1\iffalse{\fi}}%
2449 \def\xINT_expr_iter:_omit #1!#2#{\expandafter\xINT_expr_iter:_b\xint_gobble_i}%
2450 \def\xINT_expr_iter:_goon #1!#2#{\XINT_expr_iter:_goon_a {#1}}%
2451 \def\xINT_expr_iter:_goon_a #1!#2#3~#4{\XINT_expr_iter:_b #3~{#1}}%
2452 \def\xINT_expr_iter:_Ca #1#2#3{\XINT_expr_iter:_Cc#3.{#2}}%
2453 \def\xINT_expr_iter:_Cb #1{\expandafter\xINT_expr_iter:_Cc\the\numexpr#1+\xint_c_i.}%
2454 \def\xINT_expr_iter:_Cc #1.#2{\expandafter\xINT_expr_iter:_D\romannumeral0#2{{#1}}{{#1}}{{#2}}}%
2455 \def\xINT_expr_iter:_D #1{\ifx ^#1\xint_dothis\xINT_expr_iter:_abort\fi
2456             \ifx ?#1\xint_dothis\xINT_expr_iter:_break\fi
2457             \ifx !#1\xint_dothis\xINT_expr_iter:_omit\fi
2458             \xint_orthat{\XINT_expr_iter:_Goon {#1}}}%
2459 \def\xINT_expr_iter:_Omit #1!#2#{\expandafter\xINT_expr_iter:_Cb\xint_gobble_i}%
2460 \def\xINT_expr_iter:_Goon #1!#2#{\XINT_expr_iter:_Goon_a {#1}}%
2461 \def\xINT_expr_iter:_Goon_a #1#2#3~#4{\XINT_expr_iter:_Cb #3~{#1}}%

```

11.26.7 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.

```

2462 \def\xINT_expr_onliteral_add
2463 {\expandafter\xINT_allexpr_add_f\romannumeral`&&\XINT_expr_fetch_E_comma_V_equal_E_a {}}%
2464 \def\xINT_allexpr_add_f #1#2{\xint_c_ii^v `{opx}#2)\relax #1{+}{0}}%
2465 \def\xINT_expr_onliteral_mul
2466 {\expandafter\xINT_allexpr_mul_f\romannumeral`&&\XINT_expr_fetch_E_comma_V_equal_E_a {}}%
2467 \def\xINT_allexpr_mul_f #1#2{\xint_c_ii^v `{opx}#2)\relax #1{*}{1}}%
2468 \def\xINT_expr_func_opx {\XINT:NEhook:opx \XINT_allexpr_opx \xintbareeval }%
2469 \def\xINT_flexpr_func_opx {\XINT:NEhook:opx \XINT_allexpr_opx \xintbarefloateval}%
2470 \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.

```

2471 \def\xINT_allexpr_opx #1#2#3#4#5#6#7#8%
2472 {%
2473     \expandafter\xINT_expr_put_op_first
2474     \expanded \bgroup {\iffalse}\fi
2475     \XINT_expr_iter:_b {#1%
2476     \expandafter\xINT_allexpr_opx_ifnotomitted
2477     \romannumeral0#1#6\relax#7@\relax !#5}#4^~{{#8}}\XINT_expr_cb_and_getop
2478 }%

```

```

2479 \def\XINT_allexpr_opx_ifnotomitted #1%
2480 {%
2481     \ifx !#1\xint_dothis{@\relax}\fi
2482     \ifx ^#1\xint_dothis{\XINTfstop. ^\relax}\fi
2483     \if ?\xintFirstItem{#1}\xint_dothis{\XINT_allexpr_opx_break{#1}}\fi
2484     \xint_orthat{\XINTfstop. {#1}}%
2485 }%
2486 \def\XINT_allexpr_opx_break #1#2\relax
2487 {%
2488     break(\expandafter\XINTfstop\expandafter.\expandafter{\xint_gobble_i#1#2}\relax
2489 }%

```

11.26.8 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.

```

2490 \def\XINT_expr_func_rseq {\XINT_allexpr_rseq \xintbareeval }%
2491 \def\XINT_flexpr_func_rseq {\XINT_allexpr_rseq \xintbarefloateval }%
2492 \def\XINT_iexpr_func_rseq {\XINT_allexpr_rseq \xintbareiieval }%
2493 \def\XINT_allexpr_rseq #1#2#3#4%
2494 {%
2495     \expandafter\XINT_expr_rseqx
2496     \expandafter #1\expanded{\unexpanded{{#4}}\expandafter}%
2497     \romannumeral`&&@\XINT_expr_fetch_E_comma_V_equal_E_a {}%
2498 }%
2499 \def\XINT_expr_rseqx #1#2#3#4%
2500 {%
2501     \XINT:NEhook:rseq \XINT_expr_rseqy\romannumeral0#1(#4)\relax {#2}#3#1%
2502 }%
2503 \def\XINT_expr_rseqy #1#2#3#4#5%
2504 {%
2505     \expandafter\XINT_expr_put_op_first
2506     \expanded \bgroup {\iffalse}\fi
2507     #2%
2508     \XINT_expr_rseq:_b {#5#4}\relax !#3}#1^~{#2}\XINT_expr_cb_and_getop
2509 }%
2510 \def\XINT_expr_rseq:_b #1#2%
2511 {%
2512     \ifx +#2\xint_dothis\XINT_expr_rseq:_Ca\fi
2513     \ifx !#2!\xint_dothis\XINT_expr_rseq:_noop\fi
2514     \ifx ^#2\xint_dothis\XINT_expr_rseq:_end\fi
2515     \xint_orthat{\XINT_expr_rseq:_c}{#2}{#1}%
2516 }%
2517 \def\XINT_expr_rseq:_noop #1{\XINT_expr_rseq:_b }%
2518 \def\XINT_expr_rseq:_end #1#2~#3{\iffalse{\fi}}%
2519 \def\XINT_expr_rseq:_c #1#2{\expandafter\XINT_expr_rseq:_d\romannumeral0#2{{#1}}{{#2}}%
2520 \def\XINT_expr_rseq:_d #1{\ifx ^#1\xint_dothis\XINT_expr_rseq:_abort\fi
2521     \ifx ?#1\xint_dothis\XINT_expr_rseq:_break\fi
2522     \ifx !#1\xint_dothis\XINT_expr_rseq:_omit\fi
2523     \xint_orthat{\XINT_expr_rseq:_goon {#1}}}%

```

```

2524 \def\XINT_expr_rseq:_abort #1!#2^~#3{\iffalse{\fi}}%
2525 \def\XINT_expr_rseq:_break #1!#2^~#3{#1\iffalse{\fi}}%
2526 \def\XINT_expr_rseq:_omit #1!#2#{\expandafter\XINT_expr_rseq:_b\xint_gobble_i}%
2527 \def\XINT_expr_rseq:_goon #1!#2#{\XINT_expr_rseq:_goon_a {#1}}%
2528 \def\XINT_expr_rseq:_goon_a #1!#2#3~#4{#1\XINT_expr_rseq:_b #3~{#1}}%
2529 \def\XINT_expr_rseq:_Ca #1!#2#3{\XINT_expr_rseq:_Cc#3.{#2}}%
2530 \def\XINT_expr_rseq:_Cb #1{\expandafter\XINT_expr_rseq:_Cc\the\numexpr#1+\xint_c_i.}%
2531 \def\XINT_expr_rseq:_Cc #1.#2{\expandafter\XINT_expr_rseq:_D\romannumeral0#2{#{#1}}{#{#1}}{#{#2}}}%
2532 \def\XINT_expr_rseq:_D #1{\ifx ^#1\xint_dothis\XINT_expr_rseq:_abort\fi
2533 \ifx ?#1\xint_dothis\XINT_expr_rseq:_break\fi
2534 \ifx !#1\xint_dothis\XINT_expr_rseq:_omit\fi
2535 \xint_orthat{\XINT_expr_rseq:_Goon {#1}}}%
2536 \def\XINT_expr_rseq:_omit #1!#2#{\expandafter\XINT_expr_rseq:_Cb\xint_gobble_i}%
2537 \def\XINT_expr_rseq:_Goon #1!#2#{\XINT_expr_rseq:_Goon_a {#1}}%
2538 \def\XINT_expr_rseq:_Goon_a #1!#2#3~#4{#1\XINT_expr_rseq:_Cb #3~{#1}}%

```

11.26.9 iterr()

ATTENTION! at 1.4 the @ and @1 are not synonymous anymore. One *must* use @1 in iterr() context.

```

2539 \def\XINT_expr_func_iterr {\XINT_allexpr_iterr \xintbareeval }%
2540 \def\XINT_flexpr_func_iterr {\XINT_allexpr_iterr \xintbarefloateval }%
2541 \def\XINT_iexpr_func_iterr {\XINT_allexpr_iterr \xintbareiieval }%
2542 \def\XINT_allexpr_iterr #1!#2#3#4%
2543 {%
2544 \expandafter\XINT_expr_iterrx
2545 \expandafter #1\expanded{{\xintRevWithBraces{#4}}\expandafter}%
2546 \romannumeral`&&@\XINT_expr_fetch_E_comma_V_equal_E_a {}%
2547 }%
2548 \def\XINT_expr_iterrx #1!#2#3#4%
2549 {%
2550 \XINT:NEhook:iterr\XINT_expr_iterrry\romannumeral0#1(#4)\relax {#2}#3#1%
2551 }%
2552 \def\XINT_expr_iterrry #1!#2#3#4#5%
2553 {%
2554 \expandafter\XINT_expr_put_op_first
2555 \expanded \bgroup {\iffalse}\fi
2556 \XINT_expr_iterr:_b {#5#4\relax !#3}#1^~#20?\XINT_expr_cb_and_getop
2557 }%
2558 \def\XINT_expr_iterr:_b #1!#2%
2559 {%
2560 \ifx +#2\xint_dothis\XINT_expr_iterr:_Ca\fi
2561 \ifx !#2!\xint_dothis\XINT_expr_iterr:_noop\fi
2562 \ifx ^#2\xint_dothis\XINT_expr_iterr:_end\fi
2563 \xint_orthat{\XINT_expr_iterr:_c}{#2}{#1}%
2564 }%
2565 \def\XINT_expr_iterr:_noop #1{\XINT_expr_iterr:_b }%
2566 \def\XINT_expr_iterr:_end #1!#2~#3#4?{#{#3}\iffalse{\fi}}%
2567 \def\XINT_expr_iterr:_c #1!#2{\expandafter\XINT_expr_iterr:_d\romannumeral0#2{#{#1}}{#{#2}}}%
2568 \def\XINT_expr_iterr:_d #1{\ifx ^#1\xint_dothis\XINT_expr_iterr:_abort\fi
2569 \ifx ?#1\xint_dothis\XINT_expr_iterr:_break\fi
2570 \ifx !#1\xint_dothis\XINT_expr_iterr:_omit\fi
2571 \xint_orthat{\XINT_expr_iterr:_goon {#1}}}%

```

```

2572 \def\XINT_expr_iterr:_abort #1!#2^~#3?{\iffalse{\fi}}%
2573 \def\XINT_expr_iterr:_break #1!#2^~#3?{#1\iffalse{\fi}}%
2574 \def\XINT_expr_iterr:_omit #1!#2#{\expandafter\XINT_expr_iterr:_b\xint_gobble_i}%
2575 \def\XINT_expr_iterr:_goon #1!#2#{\XINT_expr_iterr:_goon_a{#1}}%
2576 \def\XINT_expr_iterr:_goon_a #1#2#3~#4?%
2577 {%
2578     \expandafter\XINT_expr_iterr:_b \expanded{\unexpanded{#3~}\xintTrim{-2}{#1#4}}0?%
2579 }%
2580 \def\XINT_expr_iterr:_Ca #1#2#3{\XINT_expr_iterr:_Cc#3.{#2}}%
2581 \def\XINT_expr_iterr:_Cb #1{\expandafter\XINT_expr_iterr:_Cc\the\numexpr#1+\xint_c_i.}%
2582 \def\XINT_expr_iterr:_Cc #1.#2{\expandafter\XINT_expr_iterr:_D\romannumeral0#2{{#1}}{{#1}}{{#2}}}%
2583 \def\XINT_expr_iterr:_D #1{\ifx ^#1\xint_dothis\XINT_expr_iterr:_abort\fi
2584     \ifx ?#1\xint_dothis\XINT_expr_iterr:_break\fi
2585     \ifx !#1\xint_dothis\XINT_expr_iterr:_Omit\fi
2586     \xint_orthat{\XINT_expr_iterr:_Goon {#1}}}%
2587 \def\XINT_expr_iterr:_Omit #1!#2#{\expandafter\XINT_expr_iterr:_Cb\xint_gobble_i}%
2588 \def\XINT_expr_iterr:_Goon #1!#2#{\XINT_expr_iterr:_Goon_a{#1}}%
2589 \def\XINT_expr_iterr:_Goon_a #1#2#3~#4?%
2590 {%
2591     \expandafter\XINT_expr_iterr:_Cb \expanded{\unexpanded{#3~}\xintTrim{-2}{#1#4}}0?%
2592 }%

```

11.26.10 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.

```

2593 \def\XINT_expr_func_rrseq {\XINT_allexpr_rrseq \xintbareeval }%
2594 \def\XINT_flexpr_func_rrseq {\XINT_allexpr_rrseq \xintbarefloateval }%
2595 \def\XINT_iexpr_func_rrseq {\XINT_allexpr_rrseq \xintbareiieval }%
2596 \def\XINT_allexpr_rrseq #1#2#3#4%
2597 {%
2598     \expandafter\XINT_expr_rrseqx\expandafter#1\expanded
2599     {\unexpanded{{#4}}{\xintRevWithBraces{#4}}\expandafter}%
2600     \romannumeral'&&\XINT_expr_fetch_E_comma_V_equal_E_a }%
2601 }%
2602 \def\XINT_expr_rrseqx #1#2#3#4#5%
2603 {%
2604     \XINT:NEhook:rrseq\XINT_expr_rrseqy\romannumeral0#1(#5)\relax {#2}{#3}#4#1%
2605 }%
2606 \def\XINT_expr_rrseqy #1#2#3#4#5#6%
2607 {%
2608     \expandafter\XINT_expr_put_op_first
2609     \expanded \bgroup {\iffalse}\fi
2610     #2\XINT_expr_rrseq:_b {#6#5}\relax !#4}#1^~#30?\XINT_expr_cb_and_getop
2611 }%
2612 \def\XINT_expr_rrseq:_b #1#2%
2613 {%
2614     \ifx +#2\xint_dothis\XINT_expr_rrseq:_Ca\fi
2615     \ifx !#2!\xint_dothis\XINT_expr_rrseq:_noop\fi
2616     \ifx ^#2\xint_dothis\XINT_expr_rrseq:_end\fi
2617     \xint_orthat{\XINT_expr_rrseq:_c}{#2}{#1}%

```



```

2618 }%
2619 \def\XINT_expr_rrseq:_noop #1{\XINT_expr_rrseq:_b }%
2620 \def\XINT_expr_rrseq:_end #1#2~#3?{\iffalse{\fi}}%
2621 \def\XINT_expr_rrseq:_c #1#2{\expandafter\XINT_expr_rrseq:_d\romannumeral0#2{{#1}}{{#2}}}%
2622 \def\XINT_expr_rrseq:_d #1{\ifx ^#1\xint_dothis\XINT_expr_rrseq:_abort\fi
2623         \ifx ?#1\xint_dothis\XINT_expr_rrseq:_break\fi
2624         \ifx !#1\xint_dothis\XINT_expr_rrseq:_omit\fi
2625         \xint_orthat{\XINT_expr_rrseq:_goon {#1}}}%
2626 \def\XINT_expr_rrseq:_abort #1!#2^~#3?{\iffalse{\fi}}%
2627 \def\XINT_expr_rrseq:_break #1!#2^~#3?{#1\iffalse{\fi}}%
2628 \def\XINT_expr_rrseq:_omit #1!#2#{\expandafter\XINT_expr_rrseq:_b\xint_gobble_i}%
2629 \def\XINT_expr_rrseq:_goon #1!#2#{\XINT_expr_rrseq:_goon_a {#1}}%
2630 \def\XINT_expr_rrseq:_goon_a #1#2#3~#4?%
2631 {%
2632     #1\expandafter\XINT_expr_rrseq:_b\expanded{\unexpanded{#3~}\xintTrim{-2}{#1#4}}0?%
2633 }%
2634 \def\XINT_expr_rrseq:_Ca #1#2#3{\XINT_expr_rrseq:_Cc#3.{#2}}%
2635 \def\XINT_expr_rrseq:_Cb #1{\expandafter\XINT_expr_rrseq:_Cc\the\numexpr#1+\xint_c_i.}%
2636 \def\XINT_expr_rrseq:_Cc #1.#2{\expandafter\XINT_expr_rrseq:_D\romannumeral0#2{{#1}}{{#1}}{{#2}}}%
2637 \def\XINT_expr_rrseq:_D #1{\ifx ^#1\xint_dothis\XINT_expr_rrseq:_abort\fi
2638         \ifx ?#1\xint_dothis\XINT_expr_rrseq:_break\fi
2639         \ifx !#1\xint_dothis\XINT_expr_rrseq:_Omit\fi
2640         \xint_orthat{\XINT_expr_rrseq:_Goon {#1}}}%
2641 \def\XINT_expr_rrseq:_Omit #1!#2#{\expandafter\XINT_expr_rrseq:_Cb\xint_gobble_i}%
2642 \def\XINT_expr_rrseq:_Goon #1!#2#{\XINT_expr_rrseq:_Goon_a {#1}}%
2643 \def\XINT_expr_rrseq:_Goon_a #1#2#3~#4?%
2644 {%
2645     #1\expandafter\XINT_expr_rrseq:_Cb\expanded{\unexpanded{#3~}\xintTrim{-2}{#1#4}}0?%
2646 }%
2647 \catcode`? 11

```

11.27 Pseudo-functions related to N-dimensional hypercubic lists

11.27.1 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 !

```

2648 \def\XINT_expr_onliteral_ndseq
2649 {%
2650     \expandafter\XINT_allexpr_ndseq_f
2651     \romannumeral`&&\XINT_expr_fetch_E_comma_V_equal_E_a {}%
2652 }%
2653 \def\XINT_allexpr_ndseq_f #1#2{\xint_c_ii^v `{\ndseqx}{#2})\relax #1}%
2654 \def\XINT_expr_func_ndseqx
2655 {%
2656     \expandafter\XINT_allexpr_ndseqx\expandafter\xintbareeval
2657     \expandafter{\romannumeral0\expandafter\xint_gobble_i\string}%
2658     \expandafter\xintrevwithbraces
2659     \expanded\bgroup{\iffalse}\fi\XINT_allexpr_ndseq_A\XINT_expr_oparen
2660 }%
2661 \def\XINT_flexpr_func_ndseqx

```



```

2662 {%
2663   \expandafter\XINT_allexpr_ndseqx\expandafter\xintbarefloateval
2664   \expandafter{\romannumeral0\expandafter\xint_gobble_i\string}%
2665   \expandafter\xintrevwithbraces
2666   \expanded\bgroup{\iffalse}\fi\XINT_allexpr_ndseq_A\XINT_flexpr_oparen
2667 }%
2668 \def\XINT_iiexpr_func_ndseqx
2669 {%
2670   \expandafter\XINT_allexpr_ndseqx\expandafter\xintbareiieval
2671   \expandafter{\romannumeral0\expandafter\xint_gobble_i\string}%
2672   \expandafter\xintrevwithbraces
2673   \expanded\bgroup{\iffalse}\fi\XINT_allexpr_ndseq_A\XINT_iiexpr_oparen
2674 }%
2675 \def\XINT_allexpr_ndseq_A #1#2#3%
2676 {%
2677   \ifx#2\xint_c_
2678     \expandafter\XINT_allexpr_ndseq_C
2679   \else
2680     \expandafter\XINT_allexpr_ndseq_B
2681   \fi #1%
2682 }%
2683 \def\XINT_allexpr_ndseq_B #1#2#3#4=%
2684 {%
2685   {#2}{\xint_zapspace#3#4 \xint_gobble_i}%
2686   \expandafter\XINT_allexpr_ndseq_A\expandafter#1\romannumeral`&&@#1%
2687 }%

#1 = \xintbareeval, or \xintbarefloateval or \xintbareiieval #2 = values for last coordinate
2688 \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
2689 \def\XINT_allexpr_ndseqx #1#2#3#4%
2690 {%
2691   \expandafter\XINT_expr_put_op_first
2692   \expanded
2693   \bgroup
2694     \romannumeral0#1\empty
2695     \expanded{\xintReplicate{\xintLength{{#3}#2}/2}{[seq(}%
2696       \unexpanded{#4}%
2697       \XINT_allexpr_ndseqx_a #2{#3}^^%
2698     }%
2699   \relax
2700   \iffalse{\fi\expandafter}\romannumeral`&&\XINT_expr_getop
2701 }%
2702 \def\XINT_allexpr_ndseqx_a #1#2%
2703 {%
2704   \xint_gob_til_ ^ #1\XINT_allexpr_ndseqx_e ^%
2705   \unexpanded{,#2=\XINTfstop.{#1})}\XINT_allexpr_ndseqx_a
2706 }%
2707 \def\XINT_allexpr_ndseqx_e ^#1\XINT_allexpr_ndseqx_a{%

```

11.27.2 ndmap()

New with 1.4. 2020/01/24.

```

2708 \def\XINT_expr_onliteral_ndmap #1,{\xint_c_ii^v `{ndmapx}\XINTfstop.{#1}};%
2709 \def\XINT_expr_func_ndmapx #1#2#3%
2710 {%
2711   \expandafter\XINT_allexpr_ndmapx
2712   \csname XINT_expr_func_\xint_zapspaces #3 \xint_gobble_i\endcsname
2713   \XINT_expr_oparen
2714 }%
2715 \def\XINT_flexpr_func_ndmapx #1#2#3%
2716 {%
2717   \expandafter\XINT_allexpr_ndmapx
2718   \csname XINT_flexpr_func_\xint_zapspaces #3 \xint_gobble_i\endcsname
2719   \XINT_flexpr_oparen
2720 }%
2721 \def\XINT_iiexpr_func_ndmapx #1#2#3%
2722 {%
2723   \expandafter\XINT_allexpr_ndmapx
2724   \csname XINT_iiexpr_func_\xint_zapspaces #3 \xint_gobble_i\endcsname
2725   \XINT_iiexpr_oparen
2726 }%
2727 \def\XINT_allexpr_ndmapx #1#2%
2728 {%
2729   \expandafter\XINT_expr_put_op_first
2730   \expanded\bgroup{\iffalse}\fi
2731   \expanded
2732     {\noexpand\XINT:NEhook:x:ndmapx
2733     \noexpand\XINT_allexpr_ndmapx_a
2734     \noexpand#1}\expandafter}%
2735   \expanded\bgroup\expandafter\XINT_allexpr_ndmap_A
2736     \expandafter#2\romannumeral`&&@#2%
2737 }%
2738 \def\XINT_allexpr_ndmap_A #1#2#3%
2739 {%
2740   \ifx#3;%
2741     \expandafter\XINT_allexpr_ndmap_B
2742   \else
2743     \xint_afterfi{\XINT_allexpr_ndmap_C#2#3}%
2744   \fi #1%
2745 }%
2746 \def\XINT_allexpr_ndmap_B #1#2%
2747 {%
2748   {#2}\expandafter\XINT_allexpr_ndmap_A\expandafter#1\romannumeral`&&@#1%
2749 }%
2750 \def\XINT_allexpr_ndmap_C #1#2#3#4%
2751 {%
2752   {#4}^\relax\iffalse{{{ \fi}}}%#1#2%
2753 }%
2754 \def\XINT_allexpr_ndmapx_a #1#2#3%
2755 {%
2756   \xint_gob_til_ ^ #3\XINT_allexpr_ndmapx_l ^%
2757   \XINT_allexpr_ndmapx_b #1{#2}{#3}%
2758 }%

```

```

2759 \def\XINT_allexpr_ndmapx_l ^#1\XINT_allexpr_ndmapx_b #2#3#4\relax
2760 {%
2761     #2\empty\xint_firstofone{#3}%
2762 }%
2763 \def\XINT_allexpr_ndmapx_b #1#2#3#4\relax
2764 {%
2765     {\iffalse}\fi\XINT_allexpr_ndmapx_c {#4\relax}#1{#2}#3^%
2766 }%
2767 \def\XINT_allexpr_ndmapx_c #1#2#3#4%
2768 {%
2769     \xint_gob_til_ ^ #4\XINT_allexpr_ndmapx_e ^%
2770     \XINT_allexpr_ndmapx_a #2{#3{#4}}#1%
2771     \XINT_allexpr_ndmapx_c {#1}#2{#3}%
2772 }%
2773 \def\XINT_allexpr_ndmapx_e ^#1\XINT_allexpr_ndmapx_c
2774     {\iffalse{\fi}\xint_gobble_iii}%

```

11.27.3 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.

```

2775 \def\XINT_expr_onliteral_ndfillraw #1,{\xint_c_ii^v `{ndfillrawx}\XINTfstop.{{#1}},}%
2776 \def\XINT_expr_func_ndfillrawx #1#2#3%
2777 {%
2778     \expandafter#1\expandafter#2\expanded{{{XINT_allexpr_ndfillrawx_a #3}}}%
2779 }%
2780 \let\XINT_iiexpr_func_ndfillrawx\XINT_expr_func_ndfillrawx
2781 \let\XINT_flexpr_func_ndfillrawx\XINT_expr_func_ndfillrawx
2782 \def\XINT_allexpr_ndfillrawx_a #1#2%
2783 {%
2784     \expandafter\XINT_allexpr_ndfillrawx_b
2785     \romannumeral0\xintApply{\xintNum}{#2}^{\relax {#1}}%
2786 }%
2787 \def\XINT_allexpr_ndfillrawx_b #1#2\relax#3%
2788 {%
2789     \xint_gob_til_ ^ #1\XINT_allexpr_ndfillrawx_c ^%
2790     \xintReplicate{#1}{{XINT_allexpr_ndfillrawx_b #2\relax {#3}}}%
2791 }%
2792 \def\XINT_allexpr_ndfillrawx_c ^\xintReplicate #1#2%
2793 {%
2794     \expandafter\XINT_allexpr_ndfillrawx_d\xint_firstofone #2%
2795 }%
2796 \def\XINT_allexpr_ndfillrawx_d\XINT_allexpr_ndfillrawx_b \relax #1{#1}%

```

11.28 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 (2015/10/10). Adds qint(), qfrac(), qfloat().

1.3c (2018/06/17). Adds qraw(). Useful to limit impact on TeX memory from abuse of \csname's storage when generating many comma separated values from a loop.

1.3e (2019/04/05). 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.

```

2797 \def\XINT_expr_onliteral_bool #1)%
2798   {\expandafter\XINT_expr_put_op_first\expanded{{{xintBool{#1}}}\expandafter
2799     }\romannumeral`&&\XINT_expr_getop}%
2800 \def\XINT_expr_onliteral_togl #1)%
2801   {\expandafter\XINT_expr_put_op_first\expanded{{{xintToggle{#1}}}\expandafter
2802     }\romannumeral`&&\XINT_expr_getop}%
2803 \def\XINT_expr_onliteral_protect #1)%
2804   {\expandafter\XINT_expr_put_op_first\expanded{{{detokenize{#1}}}\expandafter
2805     }\romannumeral`&&\XINT_expr_getop}%
2806 \def\XINT_expr_onliteral_qint #1)%
2807   {\expandafter\XINT_expr_put_op_first\expanded{{{xintiNum{#1}}}\expandafter
2808     }\romannumeral`&&\XINT_expr_getop}%
2809 \def\XINT_expr_onliteral_qfrac #1)%
2810   {\expandafter\XINT_expr_put_op_first\expanded{{{xintRaw{#1}}}\expandafter
2811     }\romannumeral`&&\XINT_expr_getop}%
2812 \def\XINT_expr_onliteral_qfloat #1)%
2813   {\expandafter\XINT_expr_put_op_first\expanded{{{XINTinFloatSdigits{#1}}}\expandafter
2814     }\romannumeral`&&\XINT_expr_getop}%
2815 \def\XINT_expr_onliteral_qraw #1)%
2816   {\expandafter\XINT_expr_put_op_first\expanded{#{#1}\expandafter
2817     }\romannumeral`&&\XINT_expr_getop}%
2818 \def\XINT_expr_onliteral_random #1)%
2819   {\expandafter\XINT_expr_put_op_first\expanded{{{XINTinRandomFloatSdigits}}\expandafter
2820     }\romannumeral`&&\XINT_expr_getop}%
2821 \def\XINT_expr_onliteral_grand #1)%
2822   {\expandafter\XINT_expr_put_op_first\expanded{{{XINTinRandomFloatSixteen}}\expandafter
2823     }\romannumeral`&&\XINT_expr_getop}%
2824 \def\XINT_expr_onliteral_rbit #1)%
2825   {\expandafter\XINT_expr_put_op_first\expanded{{{xintRandBit}}\expandafter
2826     }\romannumeral`&&\XINT_expr_getop}%

```

11.29 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(), iquo(), irem(), gcd(), lcm(), max(), min(), `+`(), `*`(), all(), any(), xor(), len(), first(), last(), reversed(), if(), ifint(), ifone(), ifsgn(), nuple(), unpack(), flat() and zip()

```

2827 \def\XINT:expr:f:one:and:opt #1#2#3!#4#5%
2828 {%
2829     \if\relax#3\relax\expandafter\xint_firstoftwo\else
2830         \expandafter\xint_secondoftwo\fi
2831     {#4}{#5[\xintNum{#2}]}{#1}%
2832 }%
2833 \def\XINT:expr:f:tacitzeroifone #1#2#3!#4#5%
2834 {%
2835     \if\relax#3\relax\expandafter\xint_firstoftwo\else
2836         \expandafter\xint_secondoftwo\fi
2837     {#4{0}}{#5{\xintNum{#2}}}{#1}%
2838 }%
2839 \def\XINT:expr:f:iitacitzeroifone #1#2#3!#4%
2840 {%
2841     \if\relax#3\relax\expandafter\xint_firstoftwo\else
2842         \expandafter\xint_secondoftwo\fi
2843     {#4{0}}{#4{#2}}{#1}%
2844 }%
2845 \def\XINT:expr_func_num #1#2#3%
2846 {%
2847     \expandafter #1\expandafter #2\expandafter{%
2848     \romannumeral`&&\XINT:NEhook:f:one:from:one
2849     {\romannumeral`&&\xintNum#3}}}%
2850 }%
2851 \let\XINT_flexpr_func_num\XINT_expr_func_num
2852 \let\XINT_iiexpr_func_num\XINT_expr_func_num
2853 \def\XINT:expr_func_reduce #1#2#3%
2854 {%
2855     \expandafter #1\expandafter #2\expandafter{%
2856     \romannumeral`&&\XINT:NEhook:f:one:from:one
2857     {\romannumeral`&&\xintIrr#3}}}%
2858 }%
2859 \let\XINT_flexpr_func_reduce\XINT_expr_func_reduce
2860 \def\XINT:expr_func_preduce #1#2#3%
2861 {%
2862     \expandafter #1\expandafter #2\expandafter{%
2863     \romannumeral`&&\XINT:NEhook:f:one:from:one
2864     {\romannumeral`&&\xintPIrr#3}}}%
2865 }%
2866 \let\XINT_flexpr_func_preduce\XINT_expr_func_preduce
2867 \def\XINT:expr_func_abs #1#2#3%
2868 {%
2869     \expandafter #1\expandafter #2\expandafter{%
2870     \romannumeral`&&\XINT:NEhook:f:one:from:one

```

```

2871     {\romannumeral`&&\xintAbs#3}}%
2872 }%
2873 \let\xINT_flexpr_func_abs\xINT_expr_func_abs
2874 \def\xINT_iiexpr_func_abs #1#2#3%
2875 {%
2876     \expandafter #1\expandafter #2\expandafter{%
2877     \romannumeral`&&\XINT:NEhook:f:one:from:one
2878     {\romannumeral`&&\xintiAbs#3}}%
2879 }%
2880 \def\xINT_expr_func_sgn #1#2#3%
2881 {%
2882     \expandafter #1\expandafter #2\expandafter{%
2883     \romannumeral`&&\XINT:NEhook:f:one:from:one
2884     {\romannumeral`&&\xintSgn#3}}%
2885 }%
2886 \let\xINT_flexpr_func_sgn\xINT_expr_func_sgn
2887 \def\xINT_iiexpr_func_sgn #1#2#3%
2888 {%
2889     \expandafter #1\expandafter #2\expandafter{%
2890     \romannumeral`&&\XINT:NEhook:f:one:from:one
2891     {\romannumeral`&&\xintiSgn#3}}%
2892 }%
2893 \def\xINT_expr_func_frac #1#2#3%
2894 {%
2895     \expandafter #1\expandafter #2\expandafter{%
2896     \romannumeral`&&\XINT:NEhook:f:one:from:one
2897     {\romannumeral`&&\xintTFrac#3}}%
2898 }%
2899 \def\xINT_flexpr_func_frac #1#2#3%
2900 {%
2901     \expandafter #1\expandafter #2\expandafter{%
2902     \romannumeral`&&\XINT:NEhook:f:one:from:one
2903     {\romannumeral`&&\XINTinFloatFrac#3}}%
2904 }%

```

no \XINT_iiexpr_func_frac

```

2905 \def\xINT_expr_func_floor #1#2#3%
2906 {%
2907     \expandafter #1\expandafter #2\expandafter{%
2908     \romannumeral`&&\XINT:NEhook:f:one:from:one
2909     {\romannumeral`&&\xintFloor#3}}%
2910 }%
2911 \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.

```

2912 \def\xINT_iiexpr_func_floor #1#2#3%
2913 {%
2914     \expandafter #1\expandafter #2\expandafter{%
2915     \romannumeral`&&\XINT:NEhook:f:one:from:one
2916     {\romannumeral`&&\xintiFloor#3}}%
2917 }%
2918 \def\xINT_expr_func_ceil #1#2#3%

```

```

2919 {%
2920   \expandafter #1\expandafter #2\expandafter{%
2921   \romannumeral`&&\XINT:NEhook:f:one:from:one
2922   {\romannumeral`&&\xintCeil#3}}%
2923 }%
2924 \let\XINT_flexpr_func_ceil\XINT_expr_func_ceil
2925 \def\XINT_iiexpr_func_ceil #1#2#3%
2926 {%
2927   \expandafter #1\expandafter #2\expandafter{%
2928   \romannumeral`&&\XINT:NEhook:f:one:from:one
2929   {\romannumeral`&&\xintiCeil#3}}%
2930 }%
2931 \def\XINT_expr_func_sqr #1#2#3%
2932 {%
2933   \expandafter #1\expandafter #2\expandafter{%
2934   \romannumeral`&&\XINT:NEhook:f:one:from:one
2935   {\romannumeral`&&\xintSqr#3}}%
2936 }%
2937 \def\XINT_flexpr_func_sqr #1#2#3%
2938 {%
2939   \expandafter #1\expandafter #2\expandafter{%
2940   \romannumeral`&&\XINT:NEhook:f:one:from:one
2941   {\romannumeral`&&\XINTinFloatSqr#3}}%
2942 }%
2943 \def\XINT_iiexpr_func_sqr #1#2#3%
2944 {%
2945   \expandafter #1\expandafter #2\expandafter{%
2946   \romannumeral`&&\XINT:NEhook:f:one:from:one
2947   {\romannumeral`&&\xintiiSqr#3}}%
2948 }%
2949 \def\XINT_expr_func_? #1#2#3%
2950 {%
2951   \expandafter #1\expandafter #2\expandafter{%
2952   \romannumeral`&&\XINT:NEhook:f:one:from:one
2953   {\romannumeral`&&\xintiiIsNotZero#3}}%
2954 }%
2955 \let\XINT_flexpr_func_? \XINT_expr_func_?
2956 \let\XINT_iiexpr_func_? \XINT_expr_func_?
2957 \def\XINT_expr_func_! #1#2#3%
2958 {%
2959   \expandafter #1\expandafter #2\expandafter{%
2960   \romannumeral`&&\XINT:NEhook:f:one:from:one
2961   {\romannumeral`&&\xintiiIsZero#3}}%
2962 }%
2963 \let\XINT_flexpr_func_! \XINT_expr_func_!
2964 \let\XINT_iiexpr_func_! \XINT_expr_func_!
2965 \def\XINT_expr_func_not #1#2#3%
2966 {%
2967   \expandafter #1\expandafter #2\expandafter{%
2968   \romannumeral`&&\XINT:NEhook:f:one:from:one
2969   {\romannumeral`&&\xintiiIsZero#3}}%
2970 }%

```

```

2971 \let\XINT_flexpr_func_not \XINT_expr_func_not
2972 \let\XINT_iiexpr_func_not \XINT_expr_func_not
2973 \def\XINT_expr_func_odd #1#2#3%
2974 {%
2975     \expandafter #1\expandafter #2\expandafter{%
2976     \romannumeral`&&\XINT:NEhook:f:one:from:one
2977     {\romannumeral`&&\xintOdd#3}}%
2978 }%
2979 \let\XINT_flexpr_func_odd\XINT_expr_func_odd
2980 \def\XINT_iiexpr_func_odd #1#2#3%
2981 {%
2982     \expandafter #1\expandafter #2\expandafter{%
2983     \romannumeral`&&\XINT:NEhook:f:one:from:one
2984     {\romannumeral`&&\xintiiOdd#3}}%
2985 }%
2986 \def\XINT_expr_func_even #1#2#3%
2987 {%
2988     \expandafter #1\expandafter #2\expandafter{%
2989     \romannumeral`&&\XINT:NEhook:f:one:from:one
2990     {\romannumeral`&&\xintEven#3}}%
2991 }%
2992 \let\XINT_flexpr_func_even\XINT_expr_func_even
2993 \def\XINT_iiexpr_func_even #1#2#3%
2994 {%
2995     \expandafter #1\expandafter #2\expandafter{%
2996     \romannumeral`&&\XINT:NEhook:f:one:from:one
2997     {\romannumeral`&&\xintiiEven#3}}%
2998 }%
2999 \def\XINT_expr_func_isint #1#2#3%
3000 {%
3001     \expandafter #1\expandafter #2\expandafter{%
3002     \romannumeral`&&\XINT:NEhook:f:one:from:one
3003     {\romannumeral`&&\xintIsInt#3}}%
3004 }%
3005 \def\XINT_flexpr_func_isint #1#2#3%
3006 {%
3007     \expandafter #1\expandafter #2\expandafter{%
3008     \romannumeral`&&\XINT:NEhook:f:one:from:one
3009     {\romannumeral`&&\xintFloatIsInt#3}}%
3010 }%
3011 \let\XINT_iiexpr_func_isint\XINT_expr_func_isint % ? perhaps rather always 1
3012 \def\XINT_expr_func_isone #1#2#3%
3013 {%
3014     \expandafter #1\expandafter #2\expandafter{%
3015     \romannumeral`&&\XINT:NEhook:f:one:from:one
3016     {\romannumeral`&&\xintIsOne#3}}%
3017 }%
3018 \let\XINT_flexpr_func_isone\XINT_expr_func_isone
3019 \def\XINT_iiexpr_func_isone #1#2#3%
3020 {%
3021     \expandafter #1\expandafter #2\expandafter{%
3022     \romannumeral`&&\XINT:NEhook:f:one:from:one

```



```

3023     {\romannumeral`&&\xintiIsOne#3}}%
3024 }%
3025 \def\XINT_expr_func_factorial #1#2#3%
3026 {%
3027     \expandafter #1\expandafter #2\expandafter{\expandafter{%
3028     \romannumeral`&&\XINT:NEhook:f:one:and:opt:direct
3029     \XINT:expr:f:one:and:opt #3,! \xintFac\XINTinFloatFac
3030     }}%
3031 }%
3032 \def\XINT_flexpr_func_factorial #1#2#3%
3033 {%
3034     \expandafter #1\expandafter #2\expandafter{\expandafter{%
3035     \romannumeral`&&\XINT:NEhook:f:one:and:opt:direct
3036     \XINT:expr:f:one:and:opt#3,! \XINTinFloatFacdigits\XINTinFloatFac
3037     }}%
3038 }%
3039 \def\XINT_iiexpr_func_factorial #1#2#3%
3040 {%
3041     \expandafter #1\expandafter #2\expandafter{%
3042     \romannumeral`&&\XINT:NEhook:f:one:from:one
3043     {\romannumeral`&&\xintiiFac#3}}%
3044 }%
3045 \def\XINT_expr_func_sqrt #1#2#3%
3046 {%
3047     \expandafter #1\expandafter #2\expandafter{\expandafter{%
3048     \romannumeral`&&\XINT:NEhook:f:one:and:opt:direct
3049     \XINT:expr:f:one:and:opt #3,! \XINTinFloatSqrtdigits\XINTinFloatSqrt
3050     }}%
3051 }%
3052 \let\XINT_flexpr_func_sqrt\XINT_expr_func_sqrt
3053 \def\XINT_iiexpr_func_sqrt #1#2#3%
3054 {%
3055     \expandafter #1\expandafter #2\expandafter{%
3056     \romannumeral`&&\XINT:NEhook:f:one:from:one
3057     {\romannumeral`&&\xintiiSqrt#3}}%
3058 }%
3059 \def\XINT_iiexpr_func_sqrtr #1#2#3%
3060 {%
3061     \expandafter #1\expandafter #2\expandafter{%
3062     \romannumeral`&&\XINT:NEhook:f:one:from:one
3063     {\romannumeral`&&\xintiiSqrtR#3}}%
3064 }%
3065 \def\XINT_expr_func_inv #1#2#3%
3066 {%
3067     \expandafter #1\expandafter #2\expandafter{%
3068     \romannumeral`&&\XINT:NEhook:f:one:from:one
3069     {\romannumeral`&&\xintInv#3}}%
3070 }%
3071 \def\XINT_flexpr_func_inv #1#2#3%
3072 {%
3073     \expandafter #1\expandafter #2\expandafter{%
3074     \romannumeral`&&\XINT:NEhook:f:one:from:one

```

```

3075     {\romannumeral`&&\XINTinFloatInv#3}}}%
3076 }%
3077 \def\XINT_expr_func_round #1#2#3%
3078 {%
3079     \expandafter #1\expandafter #2\expandafter{\expandafter{%
3080     \romannumeral`&&\XINT:NEhook:f:tacitzeroifone:direct
3081     \XINT:expr:f:tacitzeroifone #3,! \xintiRound\xintRound
3082     }}%
3083 }%
3084 \let\XINT_flexpr_func_round\XINT_expr_func_round
3085 \def\XINT_iiexpr_func_round #1#2#3%
3086 {%
3087     \expandafter #1\expandafter #2\expandafter{\expandafter{%
3088     \romannumeral`&&\XINT:NEhook:f:iitacitzeroifone:direct
3089     \XINT:expr:f:iitacitzeroifone #3,! \xintiRound
3090     }}%
3091 }%
3092 \def\XINT_expr_func_trunc #1#2#3%
3093 {%
3094     \expandafter #1\expandafter #2\expandafter{\expandafter{%
3095     \romannumeral`&&\XINT:NEhook:f:tacitzeroifone:direct
3096     \XINT:expr:f:tacitzeroifone #3,! \xintiTrunc\xintTrunc
3097     }}%
3098 }%
3099 \let\XINT_flexpr_func_trunc\XINT_expr_func_trunc
3100 \def\XINT_iiexpr_func_trunc #1#2#3%
3101 {%
3102     \expandafter #1\expandafter #2\expandafter{\expandafter{%
3103     \romannumeral`&&\XINT:NEhook:f:iitacitzeroifone:direct
3104     \XINT:expr:f:iitacitzeroifone #3,! \xintiTrunc
3105     }}%
3106 }%

```

Hesitation at 1.3e about using \XINTinFloatSdigits and \XINTinFloatS. Finally I add a sfloat() function. It helps for xinttrig.sty.

```

3107 \def\XINT_expr_func_float #1#2#3%
3108 {%
3109     \expandafter #1\expandafter #2\expandafter{\expandafter{%
3110     \romannumeral`&&\XINT:NEhook:f:one:and:opt:direct
3111     \XINT:expr:f:one:and:opt #3,! \XINTinFloatdigits\XINTinFloat
3112     }}%
3113 }%
3114 \let\XINT_flexpr_func_float\XINT_expr_func_float

```

float_() was added at 1.4, as a shortcut alias to float() skipping the check for an optional second argument. This is useful to transfer function definitions between \xintexpr and \xint-floatexpr contexts.

No need for a similar shortcut for sfloat() as currently used in xinttrig.sty to go from float to expr: as it is used there as sfloat(x) with dummy x, it sees there is no optional argument, contrarily to for example float(\xintexpr...\relax) which has to allow for the inner expression to expand to an ople with two items, so does not know in which branch it is at time of definiion.

After some hesitation at 1.4e regarding guard digits mechanism the float_() got renamed to float_dgt(), but then renamed back to float_() to avoid a breaking change and having to document it.

Nevertheless the documentation of 1.4e mentioned float_dgt()... but it was still float_()... now changed into float_dgt() for real at 1.4f.

1.4f also adds private float_dgtormax and sfloat_dgtormax for matters of xinttrig.

```

3115 \def\XINT_expr_func_float_dgt #1#2#3%
3116 {%
3117   \expandafter #1\expandafter #2\expandafter{%
3118     \romannumeral`&&\XINT:NEhook:f:one:from:one
3119     {\romannumeral`&&\XINTinFloatdigits#3}}%
3120 }%
3121 \let\XINT_flexpr_func_float_dgt\XINT_expr_func_float_dgt
3122 % no \XINT_iiexpr_func_float_dgt
3123 \def\XINT_expr_func_float_dgtormax #1#2#3%
3124 {%
3125   \expandafter #1\expandafter #2\expandafter{%
3126     \romannumeral`&&\XINT:NEhook:f:one:from:one
3127     {\romannumeral`&&\XINTinFloatdigitsormax#3}}%
3128 }%
3129 \let\XINT_flexpr_func_float_dgtormax\XINT_expr_func_float_dgtormax
3130 \def\XINT_expr_func_sfloat #1#2#3%
3131 {%
3132   \expandafter #1\expandafter #2\expandafter{\expandafter{%
3133     \romannumeral`&&\XINT:NEhook:f:one:and:opt:direct
3134     \XINT:expr:f:one:and:opt #3,! \XINTinFloatSdigits\XINTinFloatS
3135     }}%
3136 }%
3137 \let\XINT_flexpr_func_sfloat\XINT_expr_func_sfloat
3138 % no \XINT_iiexpr_func_sfloat
3139 \def\XINT_expr_func_sfloat_dgtormax #1#2#3%
3140 {%
3141   \expandafter #1\expandafter #2\expandafter{%
3142     \romannumeral`&&\XINT:NEhook:f:one:from:one
3143     {\romannumeral`&&\XINTinFloatSdigitsormax#3}}%
3144 }%
3145 \let\XINT_flexpr_func_sfloat_dgtormax\XINT_expr_func_sfloat_dgtormax
3146 \expandafter\def\csname XINT_expr_func_ilog10\endcsname #1#2#3%
3147 {%
3148   \expandafter #1\expandafter #2\expandafter{\expandafter{%
3149     \romannumeral`&&\XINT:NEhook:f:one:and:opt:direct
3150     \XINT:expr:f:one:and:opt #3,! \xintiLogTen\XINTFloatiLogTen
3151     }}%
3152 }%
3153 \expandafter\def\csname XINT_flexpr_func_ilog10\endcsname #1#2#3%
3154 {%
3155   \expandafter #1\expandafter #2\expandafter{\expandafter{%
3156     \romannumeral`&&\XINT:NEhook:f:one:and:opt:direct
3157     \XINT:expr:f:one:and:opt #3,! \XINTFloatiLogTendigits\XINTFloatiLogTen
3158     }}%
3159 }%
3160 \expandafter\def\csname XINT_iiexpr_func_ilog10\endcsname #1#2#3%
3161 {%
3162   \expandafter #1\expandafter #2\expandafter{%
3163     \romannumeral`&&\XINT:NEhook:f:one:from:one

```

```

3164     {\romannumeral`&&\xintiilogTen#3}}%
3165 }%
3166 \def\XINT_expr_func_divmod #1#2#3%
3167 {%
3168     \expandafter #1\expandafter #2\expandafter{\romannumeral`&&%
3169     \XINT:NEhook:f:one:from:two
3170     {\romannumeral`&&\xintDivMod #3}}%
3171 }%
3172 \def\XINT_flexpr_func_divmod #1#2#3%
3173 {%
3174     \expandafter #1\expandafter #2\expandafter{\romannumeral`&&%
3175     \XINT:NEhook:f:one:from:two
3176     {\romannumeral`&&\XINTinFloatDivMod #3}}%
3177 }%
3178 \def\XINT_iiexpr_func_divmod #1#2#3%
3179 {%
3180     \expandafter #1\expandafter #2\expandafter{\romannumeral`&&%
3181     \XINT:NEhook:f:one:from:two
3182     {\romannumeral`&&\xintiiDivMod #3}}%
3183 }%
3184 \def\XINT_expr_func_mod #1#2#3%
3185 {%
3186     \expandafter #1\expandafter #2\expandafter{\romannumeral`&&%
3187     \XINT:NEhook:f:one:from:two
3188     {\romannumeral`&&\xintMod#3}}%
3189 }%
3190 \def\XINT_flexpr_func_mod #1#2#3%
3191 {%
3192     \expandafter #1\expandafter #2\expandafter{\romannumeral`&&%
3193     \XINT:NEhook:f:one:from:two
3194     {\romannumeral`&&\XINTinFloatMod#3}}%
3195 }%
3196 \def\XINT_iiexpr_func_mod #1#2#3%
3197 {%
3198     \expandafter #1\expandafter #2\expandafter{\romannumeral`&&%
3199     \XINT:NEhook:f:one:from:two
3200     {\romannumeral`&&\xintiiMod#3}}%
3201 }%
3202 \def\XINT_expr_func_binomial #1#2#3%
3203 {%
3204     \expandafter #1\expandafter #2\expandafter{\romannumeral`&&%
3205     \XINT:NEhook:f:one:from:two
3206     {\romannumeral`&&\xintBinomial #3}}%
3207 }%
3208 \def\XINT_flexpr_func_binomial #1#2#3%
3209 {%
3210     \expandafter #1\expandafter #2\expandafter{\romannumeral`&&%
3211     \XINT:NEhook:f:one:from:two
3212     {\romannumeral`&&\XINTinFloatBinomial #3}}%
3213 }%
3214 \def\XINT_iiexpr_func_binomial #1#2#3%
3215 {%

```

```

3216 \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
3217 \XINT:NEhook:f:one:from:two
3218 {\romannumeral`&&\xintiiBinomial #3}}%
3219 }%
3220 \def\XINT_expr_func_pfactorial #1#2#3%
3221 {%
3222 \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
3223 \XINT:NEhook:f:one:from:two
3224 {\romannumeral`&&\xintPFactorial #3}}%
3225 }%
3226 \def\XINT_flexpr_func_pfactorial #1#2#3%
3227 {%
3228 \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
3229 \XINT:NEhook:f:one:from:two
3230 {\romannumeral`&&\XINTinFloatPFactorial #3}}%
3231 }%
3232 \def\XINT_iiexpr_func_pfactorial #1#2#3%
3233 {%
3234 \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
3235 \XINT:NEhook:f:one:from:two
3236 {\romannumeral`&&\xintiiPFactorial #3}}%
3237 }%
3238 \def\XINT_expr_func_randrange #1#2#3%
3239 {%
3240 \expandafter #1\expandafter #2\expanded{{{
3241 \XINT:expr:randrange #3,!%
3242 }}}%
3243 }%
3244 \let\XINT_flexpr_func_randrange\XINT_expr_func_randrange
3245 \def\XINT_iiexpr_func_randrange #1#2#3%
3246 {%
3247 \expandafter #1\expandafter #2\expanded{{{
3248 \XINT:iiexpr:randrange #3,!%
3249 }}}%
3250 }%
3251 \def\XINT:expr:randrange #1#2#3!%
3252 {%
3253 \if\relax#3\relax\expandafter\xint_firstoftwo\else
3254 \expandafter\xint_secondoftwo\fi
3255 {\xintiiRandRange{\XINT:NEhook:f:one:from:one:direct\xintNum{#1}}}%
3256 {\xintiiRandRangeAtoB{\XINT:NEhook:f:one:from:one:direct\xintNum{#1}}}%
3257 {\XINT:NEhook:f:one:from:one:direct\xintNum{#2}}}%
3258 }%
3259 }%
3260 \def\XINT:iiexpr:randrange #1#2#3!%
3261 {%
3262 \if\relax#3\relax\expandafter\xint_firstoftwo\else
3263 \expandafter\xint_secondoftwo\fi
3264 {\xintiiRandRange{#1}}%
3265 {\xintiiRandRangeAtoB{#1}{#2}}%
3266 }%
3267 \def\XINT_iiexpr_func_iquo #1#2#3%

```

```

3268 {%
3269     \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
3270     \XINT:NEhook:f:one:from:two
3271     {\romannumeral`&&@\xintiQuo #3}}%
3272 }%
3273 \def\XINT_iiexpr_func_irem #1#2#3%
3274 {%
3275     \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
3276     \XINT:NEhook:f:one:from:two
3277     {\romannumeral`&&@\xintiiRem #3}}%
3278 }%
3279 \def\XINT_expr_func_gcd #1#2#3%
3280 {%
3281     \expandafter #1\expandafter #2\expandafter{\expandafter
3282     {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\XINT_GCDof#3^}}%
3283 }%
3284 \let\XINT_flexpr_func_gcd\XINT_expr_func_gcd
3285 \def\XINT_iiexpr_func_gcd #1#2#3%
3286 {%
3287     \expandafter #1\expandafter #2\expandafter{\expandafter
3288     {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\XINT_iiGCDof#3^}}%
3289 }%
3290 \def\XINT_expr_func_lcm #1#2#3%
3291 {%
3292     \expandafter #1\expandafter #2\expandafter{\expandafter
3293     {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\XINT_LCMof#3^}}%
3294 }%
3295 \let\XINT_flexpr_func_lcm\XINT_expr_func_lcm
3296 \def\XINT_iiexpr_func_lcm #1#2#3%
3297 {%
3298     \expandafter #1\expandafter #2\expandafter{\expandafter
3299     {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\XINT_iiLCMof#3^}}%
3300 }%
3301 \def\XINT_expr_func_max #1#2#3%
3302 {%
3303     \expandafter #1\expandafter #2\expandafter{\expandafter
3304     {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\XINT_Maxof#3^}}%
3305 }%
3306 \def\XINT_iiexpr_func_max #1#2#3%
3307 {%
3308     \expandafter #1\expandafter #2\expandafter{\expandafter
3309     {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\XINT_iiMaxof#3^}}%
3310 }%
3311 \def\XINT_flexpr_func_max #1#2#3%
3312 {%
3313     \expandafter #1\expandafter #2\expandafter{\expandafter
3314     {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\XINTinFloatMaxof#3^}}%
3315 }%
3316 \def\XINT_expr_func_min #1#2#3%
3317 {%
3318     \expandafter #1\expandafter #2\expandafter{\expandafter
3319     {\romannumeral`&&@\XINT:NEhook:f:from:delim:u\XINT_Minof#3^}}%

```

```

3320 }%
3321 \def\XINT_iiexpr_func_min #1#2#3%
3322 {%
3323     \expandafter #1\expandafter #2\expandafter{\expandafter
3324     {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_iiMinof#3^}}%
3325 }%
3326 \def\XINT_flexpr_func_min #1#2#3%
3327 {%
3328     \expandafter #1\expandafter #2\expandafter{\expandafter
3329     {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINTinFloatMinof#3^}}%
3330 }%
3331 \expandafter
3332 \def\csname XINT_expr_func_+\endcsname #1#2#3%
3333 {%
3334     \expandafter #1\expandafter #2\expandafter{\expandafter
3335     {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_Sum#3^}}%
3336 }%
3337 \expandafter
3338 \def\csname XINT_flexpr_func_+\endcsname #1#2#3%
3339 {%
3340     \expandafter #1\expandafter #2\expandafter{\expandafter
3341     {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINTinFloatSum#3^}}%
3342 }%
3343 \expandafter
3344 \def\csname XINT_iiexpr_func_+\endcsname #1#2#3%
3345 {%
3346     \expandafter #1\expandafter #2\expandafter{\expandafter
3347     {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_iiSum#3^}}%
3348 }%
3349 \expandafter
3350 \def\csname XINT_expr_func_*\endcsname #1#2#3%
3351 {%
3352     \expandafter #1\expandafter #2\expandafter{\expandafter
3353     {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_Prd#3^}}%
3354 }%
3355 \expandafter
3356 \def\csname XINT_flexpr_func_*\endcsname #1#2#3%
3357 {%
3358     \expandafter #1\expandafter #2\expandafter{\expandafter
3359     {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINTinFloatPrd#3^}}%
3360 }%
3361 \expandafter
3362 \def\csname XINT_iiexpr_func_*\endcsname #1#2#3%
3363 {%
3364     \expandafter #1\expandafter #2\expandafter{\expandafter
3365     {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_iiPrd#3^}}%
3366 }%
3367 \def\XINT_expr_func_all #1#2#3%
3368 {%
3369     \expandafter #1\expandafter #2\expandafter{\expandafter
3370     {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_ANDof#3^}}%
3371 }%

```

```

3372 \let\XINT_flexpr_func_all\XINT_expr_func_all
3373 \let\XINT_iiexpr_func_all\XINT_expr_func_all
3374 \def\XINT_expr_func_any #1#2#3%
3375 {%
3376     \expandafter #1\expandafter #2\expandafter{\expandafter
3377         {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_ORof#3^}}}%
3378 }%
3379 \let\XINT_flexpr_func_any\XINT_expr_func_any
3380 \let\XINT_iiexpr_func_any\XINT_expr_func_any
3381 \def\XINT_expr_func_xor #1#2#3%
3382 {%
3383     \expandafter #1\expandafter #2\expandafter{\expandafter
3384         {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_XORof#3^}}}%
3385 }%
3386 \let\XINT_flexpr_func_xor\XINT_expr_func_xor
3387 \let\XINT_iiexpr_func_xor\XINT_expr_func_xor
3388 \def\XINT_expr_func_len #1#2#3%
3389 {%
3390     \expandafter#1\expandafter#2\expandafter{\expandafter{%
3391         \romannumeral`&&\XINT:NEhook:f:LFL\xintLength
3392         {\romannumeral\XINT:NEhook:r:check#3^}}}%
3393     }%
3394 }%
3395 \let\XINT_flexpr_func_len \XINT_expr_func_len
3396 \let\XINT_iiexpr_func_len \XINT_expr_func_len
3397 \def\XINT_expr_func_first #1#2#3%
3398 {%
3399     \expandafter #1\expandafter #2\expandafter{%
3400         \romannumeral`&&\XINT:NEhook:f:LFL\xintFirstOne
3401         {\romannumeral\XINT:NEhook:r:check#3^}}}%
3402     }%
3403 }%
3404 \let\XINT_flexpr_func_first\XINT_expr_func_first
3405 \let\XINT_iiexpr_func_first\XINT_expr_func_first
3406 \def\XINT_expr_func_last #1#2#3%
3407 {%
3408     \expandafter #1\expandafter #2\expandafter{%
3409         \romannumeral`&&\XINT:NEhook:f:LFL\xintLastOne
3410         {\romannumeral\XINT:NEhook:r:check#3^}}}%
3411     }%
3412 }%
3413 \let\XINT_flexpr_func_last\XINT_expr_func_last
3414 \let\XINT_iiexpr_func_last\XINT_expr_func_last
3415 \def\XINT_expr_func_reversed #1#2#3%
3416 {%
3417     \expandafter #1\expandafter #2\expandafter{%
3418         \romannumeral`&&\XINT:NEhook:f:reverse\XINT_expr_reverse
3419         #3^^#3\xint:\xint:\xint:\xint:
3420         \xint:\xint:\xint:\xint:\xint_bye
3421     }%
3422 }%
3423 \def\XINT_expr_reverse #1#2%

```



```

3424 {%
3425   \if ^\noexpand#2%
3426     \expandafter\XINT_expr_reverse:_one_or_none\string#1.%
3427   \else
3428     \expandafter\XINT_expr_reverse:_at_least_two
3429   \fi
3430}%
3431 \def\XINT_expr_reverse:_at_least_two #1^^{\XINT_revwbr_loop {}}%
3432 \def\XINT_expr_reverse:_one_or_none #1%
3433 {%
3434   \if #1\bgroup\xint_dothis\XINT_expr_reverse:_nutple\fi
3435   \if #1^\xint_dothis\XINT_expr_reverse:_nil\fi
3436   \xint_orthat\XINT_expr_reverse:_leaf
3437}%
3438 \edef\XINT_expr_reverse:_nil #1\xint_bye{\noexpand\fi\space}%
3439 \def\XINT_expr_reverse:_leaf#1\fi #2\xint:#3\xint_bye{\fi\xint_gob_andstop_i#2}%
3440 \def\XINT_expr_reverse:_nutple%
3441 {%
3442   \expandafter\XINT_expr_reverse:_nutple_a\expandafter{\string}%
3443}%
3444 \def\XINT_expr_reverse:_nutple_a #1^#2\xint:#3\xint_bye
3445 {%
3446   \fi\expandafter
3447   {\romannumeral0\XINT_revwbr_loop{}}#2\xint:#3\xint_bye}%
3448}%
3449 \let\XINT_flexpr_func_reversed\XINT_expr_func_reversed
3450 \let\XINT_iiexpr_func_reversed\XINT_expr_func_reversed
3451 \def\XINT_expr_func_if #1#2#3%
3452 {%
3453   \expandafter #1\expandafter #2\expandafter{%
3454     \romannumeral`&&\XINT:NEhook:branch{\romannumeral`&&\xintiiifNotZero #3}}%
3455}%
3456 \let\XINT_flexpr_func_if\XINT_expr_func_if
3457 \let\XINT_iiexpr_func_if\XINT_expr_func_if
3458 \def\XINT_expr_func_ifint #1#2#3%
3459 {%
3460   \expandafter #1\expandafter #2\expandafter{%
3461     \romannumeral`&&\XINT:NEhook:branch{\romannumeral`&&\xintifInt #3}}%
3462}%
3463 \let\XINT_iiexpr_func_ifint\XINT_expr_func_ifint
3464 \def\XINT_flexpr_func_ifint #1#2#3%
3465 {%
3466   \expandafter #1\expandafter #2\expandafter{%
3467     \romannumeral`&&\XINT:NEhook:branch{\romannumeral`&&\xintifFloatInt #3}}%
3468}%
3469 \def\XINT_expr_func_ifone #1#2#3%
3470 {%
3471   \expandafter #1\expandafter #2\expandafter{%
3472     \romannumeral`&&\XINT:NEhook:branch{\romannumeral`&&\xintifOne #3}}%
3473}%
3474 \let\XINT_flexpr_func_ifone\XINT_expr_func_ifone
3475 \def\XINT_iiexpr_func_ifone #1#2#3%

```

```

3476 {%
3477   \expandafter #1\expandafter #2\expandafter{%
3478   \romannumeral`&&\XINT:NEhook:branch{\romannumeral`&&\xintiiifOne #3}}%
3479 }%
3480 \def\XINT_expr_func_ifsgn #1#2#3%
3481 {%
3482   \expandafter #1\expandafter #2\expandafter{%
3483   \romannumeral`&&\XINT:NEhook:branch{\romannumeral`&&\xintiiifSgn #3}}%
3484 }%
3485 \let\XINT_flexpr_func_ifsgn\XINT_expr_func_ifsgn
3486 \let\XINT_iiexpr_func_ifsgn\XINT_expr_func_ifsgn
3487 \def\XINT_expr_func_nuple #1#2#3{#1#2{{#3}}}%
3488 \let\XINT_flexpr_func_nuple\XINT_expr_func_nuple
3489 \let\XINT_iiexpr_func_nuple\XINT_expr_func_nuple
3490 \def\XINT_expr_func_unpack #1#2#3%
3491   {\expandafter#1\expandafter#2\romannumeral0\XINT:NEhook:unpack}%
3492 \let\XINT_flexpr_func_unpack\XINT_expr_func_unpack
3493 \let\XINT_iiexpr_func_unpack\XINT_expr_func_unpack
3494 \def\XINT_expr_func_flat #1#2#3%
3495 {%
3496   \expandafter#1\expandafter#2\expanded
3497   \XINT:NEhook:x:flatten\XINT:expr:flatten
3498 }%
3499 \let\XINT_flexpr_func_flat\XINT_expr_func_flat
3500 \let\XINT_iiexpr_func_flat\XINT_expr_func_flat
3501 \let\XINT:NEhook:x:flatten\empty
3502 \def\XINT_expr_func_zip #1#2#3%
3503 {%
3504   \expandafter#1\expandafter#2\romannumeral`&&%
3505   \XINT:NEhook:x:zip\XINT:expr:zip
3506 }%
3507 \let\XINT_flexpr_func_zip\XINT_expr_func_zip
3508 \let\XINT_iiexpr_func_zip\XINT_expr_func_zip
3509 \let\XINT:NEhook:x:zip\empty
3510 \def\XINT:expr:zip#1{\expandafter{\expanded\XINT_zip_A#1\xint_bye\xint_bye}}%

```

11.30 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.

11.30.1 `\xintdeffunc`, `\xintdefiifunc`, `\xintdeffloatfunc` 411

11.30.2	<code>\xintdefufunc</code> , <code>\xintdefiifunc</code> , <code>\xintdeffloatufunc</code>	414
11.30.3	<code>\xintunassignexprfunc</code> , <code>\xintunassigniexprfunc</code> , <code>\xintunassignfloatexprfunc</code>	415
11.30.4	<code>\xintNewFunction</code>	416
11.30.5	Mysterious stuff	417
11.30.6	<code>\XINT_expr_redefinemacros</code>	429
11.30.7	<code>\xintNewExpr</code> , <code>\xintNewIExpr</code> , <code>\xintNewFloatExpr</code> , <code>\xintNewIIExpr</code>	430
11.30.8	<code>\ifxintexprsafecatcodes</code> , <code>\xintexprSafeCatcodes</code> , <code>\xintexprRestoreCatcodes</code>	432

11.30.1 `\xintdeffunc`, `\xintdefiifunc`, `\xintdeffloatfunc`

1.2c (2015/11/16) [commented 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 (2015/11/16) [commented 2015/11/13].

Function names first expanded then detokenized and cleaned of spaces.

1.2e (2015/11/22) [commented 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 (2016/03/12) [commented 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 (2016/03/12) [commented 2016/03/08].

Comma separated expressions allowed (formerly this required using parenthesis `\xintdeffunc foo(x,...):=(., ., .)`;

1.3c (2018/06/17) [commented 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 (2019/04/05).

`\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 (2020/01/31) [commented 2020/01/10].

Multi-letter variables can be used (with no prior declaration)

1.4 (2020/01/31) [commented 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`, `\xintdefiifunc`, `\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.

1.4 (2020/01/31) [commented 2020/01/19].

Addition de la syntaxe déclarative `\xintdeffunc foo(a,b,...,*z) = ...;`

```

3511 \def\XINT_tmpa #1#2#3#4#5%
3512 {%
3513   \def #1##1(##2)##3={%
3514     \edef\XINT_deffunc_tmpa {##1}%

```

```

3515 \edef\XINT_deffunc_tmpa {\xint_zapspaces_o \XINT_deffunc_tmpa}%
3516 \def\XINT_deffunc_tmpb {0}%
3517 \edef\XINT_deffunc_tmpd {##2}%
3518 \edef\XINT_deffunc_tmpd {\xint_zapspaces_o\XINT_deffunc_tmpd}%
3519 \def\XINT_deffunc_tmpe {0}%
3520 \expandafter#5\romannumeral\XINT_expr_fetch_to_semicolon
3521 }% end of \xintdeffunc_a definition
3522 \def#5##1{%
3523 \def\XINT_deffunc_tmpe{##1}%
3524 \ifnum\xintLength:f:csv{\XINT_deffunc_tmpd}>\xint_c_
3525 \xintFor #####1 in {\XINT_deffunc_tmpd}\do
3526 {%
3527 \xintifForFirst{\let\XINT_deffunc_tmpd\empty}\{%
3528 \def\XINT_deffunc_tmpe{#####1}%
3529 \if*\xintFirstItem{#####1}%
3530 \xintifForLast
3531 {%
3532 \def\XINT_deffunc_tmpe{1}%
3533 \edef\XINT_deffunc_tmpe{\xintTrim{1}{#####1}}%
3534 }%
3535 {%
3536 \edef\XINT_deffunc_tmpe{\xintTrim{1}{#####1}}%
3537 \xintMessage{xintexpr}{Error}
3538 {Only the last positional argument can be variadic. Trimmed #####1 to
3539 \XINT_deffunc_tmpe}%
3540 }%
3541 \fi
3542 \XINT_expr_makedummy{\XINT_deffunc_tmpe}%
3543 \edef\XINT_deffunc_tmpd{\XINT_deffunc_tmpd\XINT_deffunc_tmpe}%
3544 \edef\XINT_deffunc_tmpe {\the\numexpr\XINT_deffunc_tmpe+\xint_c_i}%
3545 \edef\XINT_deffunc_tmpe {subs(\unexpanded\expandafter{\XINT_deffunc_tmpe},%
3546 \XINT_deffunc_tmpe=#####\XINT_deffunc_tmpe)}%
3547 }%
3548 \fi

```

Place holder for comments. Logic at 1.4 is simplified here compared to earlier releases.

```

3549 \ifcase\XINT_deffunc_tmpe\space
3550 \expandafter\XINT_expr_defuserfunc_none\csname
3551 \else
3552 \expandafter\XINT_expr_defuserfunc\csname
3553 \fi
3554 XINT_#2_func_\XINT_deffunc_tmpe\expandafter\endcsname
3555 \csname XINT_#2_userfunc_\XINT_deffunc_tmpe\expandafter\endcsname
3556 \expandafter{\XINT_deffunc_tmpe}{#2}%
3557 \expandafter#3\csname XINT_#2_userfunc_\XINT_deffunc_tmpe\endcsname
3558 [\XINT_deffunc_tmpe]{\XINT_deffunc_tmpe}%
3559 \ifxintverbose\xintMessage {xintexpr}{Info}
3560 {Function \XINT_deffunc_tmpe\space for \string\xint #4 parser
3561 associated to \string\XINT_#2_userfunc_\XINT_deffunc_tmpe\space
3562 with \ifxintglobaldefs global \fi meaning \expandafter\meaning
3563 \csname XINT_#2_userfunc_\XINT_deffunc_tmpe\endcsname}%
3564 \fi
3565 \xintFor* #####1 in {\XINT_deffunc_tmpe}:{\xintrestorevariablesilently{#####1}}%

```

```

3566 \xintexprRestoreCatcodes
3567 }% end of \xintdeffunc_b definition
3568 }%
3569 \def\xintdeffunc      {\xintexprSafeCatcodes\xintdeffunc_a}%
3570 \def\xintdefiifunc    {\xintexprSafeCatcodes\xintdefiifunc_a}%
3571 \def\xintdeffloatfunc {\xintexprSafeCatcodes\xintdeffloatfunc_a}%
3572 \XINT_tmpa\xintdeffunc_a {expr} \XINT_NewFunc {expr}\xintdeffunc_b
3573 \XINT_tmpa\xintdefiifunc_a {iiexpr}\XINT_NewIIFunc {iiexpr}\xintdefiifunc_b
3574 \XINT_tmpa\xintdeffloatfunc_a{floatexpr}\XINT_NewFloatFunc{floatexpr}\xintdeffloatfunc_b
3575 \def\XINT_expr_defuserfunc_none #1#2#3#4%
3576 {%
3577 \XINT_global
3578 \def #1##1##2##3%
3579 {%
3580 \expandafter##1\expandafter##2\expanded{%
3581 {\XINT:NEhook:usernoargfunc\csname XINT_#4_userfunc_#3\endcsname}%
3582 }%
3583 }%
3584 }%
3585 \let\XINT:NEhook:usernoargfunc \empty
3586 \def\XINT_expr_defuserfunc #1#2#3#4%
3587 {%
3588 \if0\XINT_deffunc_tmpe
3589 \XINT_global
3590 \def #1##1##2##3%
3591 {%
3592 \expandafter ##1\expandafter##2\expanded\bgroup{\iffalse}\fi
3593 \XINT:NEhook:userfunc{XINT_#4_userfunc_#3}#2##3%
3594 }%
3595 \else

```

Last argument in the call signature is variadic (was prefixed by *).

```

3596 \def #1##1{%
3597 \XINT_global\def #1####1####2####3%
3598 {%
3599 \expandafter ####1\expandafter####2\expanded\bgroup{\iffalse}\fi
3600 \XINT:NEhook:userfunc:argv{##1}{XINT_#4_userfunc_#3}#2####3%
3601 }}\expandafter#1\expandafter{\the\numexpr\XINT_deffunc_tmpe-1}%
3602 \fi
3603 }%

```

Deliberate brace stripping of #3 to reveal the elements of the ople, which may be atoms i.e. numeric data such as {1}, or again oples, which means that the corresponding item was a nutple, for example it came from input syntax such as foo(1, 2, [1, 2], 3), so (up to details of raw encoding) {1}{2}{{1}{2}}{3}, which gives 4 braced arguments to macro #2.

```

3604 \def\XINT:NEhook:userfunc #1#2#3{#2#3\iffalse{\fi}}%

```

Here #1 indicates the number k-1 of standard positional arguments of the call signature, the kth and last one having been declared of variadic type. The braces around \xintTrim{#1}{#4} have the effect to gather all these remaining elements to provide a single one to the TeX macro.

For example input was foo(1,2,3,4,5) and call signature was foo(a,b,*z). Then #4 will fetch {{1}{2}{3}{4}{5}}, with one level of brace removal. We will have \xintKeep{2}{{1}{2}{3}{4}{5}} which produces {1}{2}. Then {\xintTrim{2}{{1}{2}{3}{4}{5}}} which produces {{3}{4}{5}}. So the macro will be used as \macro{1}{2}{{3}{4}{5}} having been declared as a macro with 3 arguments.

The above comments were added in June 2021 but the code was done on January 19, 2020 for 1.4.

Note on June 10, 2021: at core level `\XINT_NewFunc` is used which is derived from `\XINT_NewExpr` which has always prepared TeX macros with non-delimited parameters. A refactoring could add a final delimiter, for example `\relax`. The macro with 3 arguments would be defined as `\def\macro#1#2#3\relax{...}` for example. Then we could transfer to TeX core processing what is achieved here via `\xintKeep/\xintTrim`, of course adding efficiency, via insertion of the delimiter. In the case of `foo(1,2,3,4,5)` we would have the #3 of delimited `\macro fetch {3}{4}{5}`, no brace removal, which is equivalent to current situation fetching `{{3}{4}{5}}` with brace removal. But let's see in case of `foo(1,2,3)` then. This would lead to delimited `\macro{1}{2}{3}\relax` and #3 will fetch `{3}`, removing one brace pair. Whereas current non-delimited `\macro` is used as `\macro{1}{2}{{3}}` from the `Keep/Trim`, then #3 fetches `{{3}}`, removing one brace pair. Not the same thing. So it seems there is a stumbling-block here to adopt such an alternative method, in relation with brace removal. Rather relieved in fact, as my head starts spinning in ople world. Seems better to stop thinking about doing something like that, and what it would imply as consequences for user declarative interface also. Oples are dangerous to mental health, let's stick with one-ples: « named arguments in function body declaration must stand for one-ples », even the last one, although a priori it could be envisioned if `foo` has been declared with call signature `(x,y,z)` and is used with more items that `z` is mapped to the ople of extra elements beyond the first two ones. For my sanity I stick with my January 2020 concept of `(x,y,*z)` which makes `z` stand for a nutple always and having to be used as such in the function body (possibly unpacked there using `*z`).

```
3605 \def\XINT:NEhook:userfunc:argv #1#2#3#4%
3606   {\expandafter#3\expanded{\xintKeep{#1}{#4}{\xintTrim{#1}{#4}}}\iffalse{{\fi}}}%
3607 \let\xintdefefunc\xintdeffunc
3608 \let\xintdefiifunc\xintdefiifunc
3609 \let\xintdeffloatefunc\xintdeffloatfunc
```

11.30.2 `\xintdefufunc`, `\xintdefiifunc`, `\xintdeffloatufunc`

1.4

```
3610 \def\XINT_tmpa #1#2#3#4#5#6%
3611 {%
3612   \def #1##1(##2)##3={%
3613     \edef\XINT_defufunc_tmpa {##1}%
3614     \edef\XINT_defufunc_tmpa {\xint_zapspaces_o \XINT_defufunc_tmpa}%
3615     \edef\XINT_defufunc_tmpd {##2}%
3616     \edef\XINT_defufunc_tmpd {\xint_zapspaces_o\XINT_defufunc_tmpd}%
3617     \expandafter#5\romannumeral\XINT_expr_fetch_to_semicolon
3618   }% end of \xint_defufunc_a
3619   \def#5##1{%
3620     \def\XINT_defufunc_tmpe{##1}%
3621     \ifnum\xintLength:f:csv{\XINT_defufunc_tmpe}=\xint_c_i
3622       \expandafter#6%
3623     \else
3624       \xintMessage {xintexpr}{ERROR}
3625       {Universal functions must be functions of one argument only,
3626        but the declaration of \XINT_defufunc_tmpe\space
3627        has \xintLength:f:csv{\XINT_defufunc_tmpe} of them. Cancelled.}%
3628     \xintexprRestoreCatcodes
3629     \fi
3630   }% end of \xint_defufunc_b
3631 \def #6{%
3632   \XINT_expr_makedummy{\XINT_defufunc_tmpe}%
```

```

3633 \edef\XINT_defufunc_tmpc {subs(\unexpanded\expandafter{\XINT_defufunc_tmpc},%
3634 \XINT_defufunc_tmpd=#####1)}%
3635 \expandafter\XINT_expr_defuserufunc
3636 \csname XINT_#2_func_\XINT_defufunc_tmpa\expandafter\endcsname
3637 \csname XINT_#2_userufunc_\XINT_defufunc_tmpa\expandafter\endcsname
3638 \expandafter{\XINT_defufunc_tmpa}{#2}%
3639 \expandafter#3\csname XINT_#2_userufunc_\XINT_defufunc_tmpa\endcsname
3640 [1]{\XINT_defufunc_tmpc}%
3641 \ifxintverbose\xintMessage {xintexpr}{Info}
3642 {Universal function \XINT_defufunc_tmpa\space for \string\xint #4 parser
3643 associated to \string\XINT_#2_userufunc_\XINT_defufunc_tmpa\space
3644 with \ifxintglobaldefs global \fi meaning \expandafter\meaning
3645 \csname XINT_#2_userufunc_\XINT_defufunc_tmpa\endcsname}%
3646 \fi
3647 }% end of \xint_defufunc_c
3648 }%
3649 \def\xintdefufunc {\xintexprSafeCatcodes\xintdefufunc_a}%
3650 \def\xintdefiiufunc {\xintexprSafeCatcodes\xintdefiiufunc_a}%
3651 \def\xintdeffloatufunc {\xintexprSafeCatcodes\xintdeffloatufunc_a}%
3652 \XINT_tmpa\xintdefufunc_a {expr} \XINT_NewFunc {expr}%
3653 \xintdefufunc_b\xintdefufunc_c
3654 \XINT_tmpa\xintdefiiufunc_a {iiexpr}\XINT_NewIIFunc {iiexpr}%
3655 \xintdefiiufunc_b\xintdefiiufunc_c
3656 \XINT_tmpa\xintdeffloatufunc_a{flexpr}\XINT_NewFloatFunc{floatexpr}%
3657 \xintdeffloatufunc_b\xintdeffloatufunc_c
3658 \def\XINT_expr_defuserufunc #1#2#3#4%
3659 {%
3660 \XINT_global
3661 \def #1##1##2%##3%
3662 {%
3663 \expandafter ##1\expandafter##2\expanded
3664 \XINT:NEhook:userufunc{XINT_#4_userufunc_#3}##2%##3%
3665 }%
3666 }%
3667 \def\XINT:NEhook:userufunc #1{\XINT:expr:mapwithin}%

```

11.30.3 \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.

```

3668 \def\XINT_tmpa #1{\expandafter\def\csname xintunassign#1func\endcsname ##1{%
3669 \edef\XINT_unfunc_tmpa{##1}%
3670 \edef\XINT_unfunc_tmpa {\xint_zapspaces_o\XINT_unfunc_tmpa}%
3671 \XINT_global\expandafter
3672 \let\csname XINT_#1_func_\XINT_unfunc_tmpa\endcsname\xint_undefined
3673 \XINT_global\expandafter
3674 \let\csname XINT_#1_userufunc_\XINT_unfunc_tmpa\endcsname\xint_undefined
3675 \XINT_global\expandafter
3676 \let\csname XINT_#1_userufunc_\XINT_unfunc_tmpa\endcsname\xint_undefined
3677 \ifxintverbose\xintMessage {xintexpr}{Info}
3678 {Function \XINT_unfunc_tmpa\space for \string\xint #1 parser now
3679 \ifxintglobaldefs globally \fi undefined.}%

```



```
3680 \fi}}%
3681 \XINT_tmpa{expr}\XINT_tmpa{iiexpr}\XINT_tmpa{floatexpr}%
```

11.30.4 \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 parsing 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.

```
3682 \def\xintNewFunction #1#2[#3]#4%
3683 {%
3684 \edef\XINT_newfunc_tmpa {#1}%
3685 \edef\XINT_newfunc_tmpa {\xint_zapspaces_o \XINT_newfunc_tmpa}%
3686 \def\XINT_newfunc_tmpb ##1##2##3##4##5##6##7##8##9{#4}%
3687 \begingroup
3688 \ifcase #3\relax
3689 \toks0{}}%
3690 \or \toks0{##1}%
3691 \or \toks0{##1##2}%
3692 \or \toks0{##1##2##3}%
3693 \or \toks0{##1##2##3##4}%
3694 \or \toks0{##1##2##3##4##5}%
3695 \or \toks0{##1##2##3##4##5##6}%
3696 \or \toks0{##1##2##3##4##5##6##7}%
3697 \or \toks0{##1##2##3##4##5##6##7##8}%
3698 \else \toks0{##1##2##3##4##5##6##7##8##9}%
3699 \fi
3700 \expandafter
3701 \endgroup\expandafter
3702 \XINT_global\expandafter
3703 \def\csname XINT_expr_macrofunc_\XINT_newfunc_tmpa\expandafter\endcsname
3704 \the\toks0\expandafter{\XINT_newfunc_tmpb
3705 {\XINTfstop.{##1}}{\XINTfstop.{##2}}{\XINTfstop.{##3}}}%
3706 {\XINTfstop.{##4}}{\XINTfstop.{##5}}{\XINTfstop.{##6}}}%
3707 {\XINTfstop.{##7}}{\XINTfstop.{##8}}{\XINTfstop.{##9}}}%
3708 \expandafter\XINT_expr_newfunction
3709 \csname XINT_expr_func_\XINT_newfunc_tmpa\expandafter\endcsname
3710 \expandafter{\XINT_newfunc_tmpa}\xintbareeval
3711 \expandafter\XINT_expr_newfunction
3712 \csname XINT_iiexpr_func_\XINT_newfunc_tmpa\expandafter\endcsname
3713 \expandafter{\XINT_newfunc_tmpa}\xintbareiieval
3714 \expandafter\XINT_expr_newfunction
3715 \csname XINT_flexpr_func_\XINT_newfunc_tmpa\expandafter\endcsname
3716 \expandafter{\XINT_newfunc_tmpa}\xintbarefloateval
3717 \ifxintverbose
3718 \xintMessage {xintexpr}{Info}
3719 {Function \XINT_newfunc_tmpa\space for the expression parsers is
3720 associated to \string\XINT_expr_macrofunc_\XINT_newfunc_tmpa\space
```



```

3721         with \ifxintglobaldefs global \fi meaning \expandafter\meaning
3722         \csname XINT_expr_macrofunc_\XINT_newfunc_tmpa\endcsname}%
3723     \fi
3724 }%
3725 \def\XINT_expr_newfunction #1#2#3%
3726 {%
3727     \XINT_global
3728     \def#1##1##2##3%
3729         {\expandafter ##1\expandafter ##2%
3730         \romannumeral0\XINT:NEhook:macrofunc
3731         #3{\csname XINT_expr_macrofunc_#2\endcsname##3}\relax
3732         }%
3733 }%
3734 \let\XINT:NEhook:macrofunc\empty

```

11.30.5 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.

```

3735 \catcode`~ 12
3736 \def\XINT:NE:hashtilde#1~#2#3\relax{\unless\if !#21\fi}%
3737 \def\XINT:NE:hashash#1{%
3738 \def\XINT:NE:hashash##1#1##2##3\relax{\unless\if !##21\fi}%
3739 }\expandafter\XINT:NE:hashash\string#%
3740 \def\XINT:NE:unpack #1{%
3741 \def\XINT:NE:unpack ##1%
3742 {%
3743     \if0\XINT:NE:hashtilde ##1~!\relax
3744     \XINT:NE:hashash ##1#1!\relax 0\else
3745     \expandafter\XINT:NE:unpack:p\fi
3746     \xint_stop_atfirstofone{##1}%
3747 }}\expandafter\XINT:NE:unpack\string#%
3748 \def\XINT:NE:unpack:p#1#2%
3749     {{~romannumeral0~expandafter~xint_stop_atfirstofone~expanded{#2}}}%

```

```

3750 \def\XINT:NE:f:one:from:one #1{%
3751 \def\XINT:NE:f:one:from:one ##1%
3752 {%
3753   \if0\XINT:NE:hasilde ##1~!\relax
3754     \XINT:NE:hashash ##1#1!\relax 0\else
3755     \xint_dothis\XINT:NE:f:one:from:one_a\fi
3756     \xint_orthat\XINT:NE:f:one:from:one_b
3757     ##1&&A%
3758 }}\expandafter\XINT:NE:f:one:from:one\string#%
3759 \def\XINT:NE:f:one:from:one_a\romannumeral`&&@#1#2&&A%
3760 {%
3761   \expandafter{\detokenize{\expandafter#1}#2}%
3762 }%
3763 \def\XINT:NE:f:one:from:one_b#1{%
3764 \def\XINT:NE:f:one:from:one_b\romannumeral`&&@##1##2&&A%
3765 {%
3766   \expandafter{\romannumeral`&&@%
3767     \if0\XINT:NE:hasilde ##2~!\relax
3768       \XINT:NE:hashash ##2#1!\relax 0\else
3769       \expandafter\string\fi
3770     ##1{##2}}}%
3771 }}\expandafter\XINT:NE:f:one:from:one_b\string#%
3772 \def\XINT:NE:f:one:from:one:direct #1#2{\XINT:NE:f:one:from:one:direct_a #2&&A{#1}}%
3773 \def\XINT:NE:f:one:from:one:direct_a #1#2&&A#3%
3774 {%
3775   \if ##1\xint_dothis {\detokenize{#3}}\fi
3776   \if ~#1\xint_dothis {\detokenize{#3}}\fi
3777   \xint_orthat {#3}{#1#2}%
3778 }%
3779 \def\XINT:NE:f:one:from:two #1{%
3780 \def\XINT:NE:f:one:from:two ##1%
3781 {%
3782   \if0\XINT:NE:hasilde ##1~!\relax
3783     \XINT:NE:hashash ##1#1!\relax 0\else
3784     \xint_dothis\XINT:NE:f:one:from:two_a\fi
3785     \xint_orthat\XINT:NE:f:one:from:two_b ##1&&A%
3786 }}\expandafter\XINT:NE:f:one:from:two\string#%
3787 \def\XINT:NE:f:one:from:two_a\romannumeral`&&@#1#2&&A%
3788 {%
3789   \expandafter{\detokenize{\expandafter#1\expanded}{#2}}}%
3790 }%
3791 \def\XINT:NE:f:one:from:two_b#1{%
3792 \def\XINT:NE:f:one:from:two_b\romannumeral`&&@##1##2##3&&A%
3793 {%
3794   \expandafter{\romannumeral`&&@%
3795     \if0\XINT:NE:hasilde ##2##3~!\relax
3796       \XINT:NE:hashash ##2##3#1!\relax 0\else
3797       \expandafter\string\fi
3798     ##1{##2}{##3}}}%
3799 }}\expandafter\XINT:NE:f:one:from:two_b\string#%
3800 \def\XINT:NE:f:one:from:two:direct #1#2#3{\XINT:NE:two_fork #2&&A#3&&A#1{#2}{#3}}%
3801 \def\XINT:NE:two_fork #1#2&&A#3#4&&A{\XINT:NE:two_fork_nn#1#3}%

```

```

3802 \def\XINT:NE:two_fork_nm #1#2%
3803 {%
3804     \if #1##\xint_dothis\string\fi
3805     \if #1~\xint_dothis\string\fi
3806     \if #2##\xint_dothis\string\fi
3807     \if #2~\xint_dothis\string\fi
3808     \xint_orthat{}}%
3809}%
3810 \def\XINT:NE:f:one:and:opt:direct#1{%
3811 \def\XINT:NE:f:one:and:opt:direct##1!%
3812 {%
3813     \if0\XINT:NE:hasilde ##1~!\relax
3814         \XINT:NE:hashash ##1#1!\relax 0\else
3815         \xint_dothis\XINT:NE:f:one:and:opt_a\fi
3816         \xint_orthat\XINT:NE:f:one:and:opt_b ##1&&A%
3817 }}\expandafter\XINT:NE:f:one:and:opt:direct\string#%
3818 \def\XINT:NE:f:one:and:opt_a #1#2&&A#3#4%
3819 {%
3820     \detokenize{\romannumeral-`0\expandafter#1\expanded{#2}$XINT_expr_exclam#3#4}%$
3821}%
3822 \def\XINT:NE:f:one:and:opt_b\XINT:expr:f:one:and:opt #1#2#3&&A#4#5%
3823 {%
3824     \if\relax#3\relax\expandafter\xint_firstoftwo\else
3825         \expandafter\xint_secondoftwo\fi
3826     {\XINT:NE:f:one:from:one:direct#4}%
3827     {\expandafter\XINT:NE:f:one:withopttoone\expandafter#5%
3828         \expanded{{\XINT:NE:f:one:from:one:direct\xintNum{#2}}}}}%
3829     {#1}%
3830}%
3831 \def\XINT:NE:f:one:withopttoone#1#2#3{\XINT:NE:two_fork #2&&A#3&&A#1[#2]{#3}}%
3832 \def\XINT:NE:f:tacitzeroifone:direct#1{%
3833 \def\XINT:NE:f:tacitzeroifone:direct##1!%
3834 {%
3835     \if0\XINT:NE:hasilde ##1~!\relax
3836         \XINT:NE:hashash ##1#1!\relax 0\else
3837         \xint_dothis\XINT:NE:f:one:and:opt_a\fi
3838         \xint_orthat\XINT:NE:f:tacitzeroifone_b ##1&&A%
3839 }}\expandafter\XINT:NE:f:tacitzeroifone:direct\string#%
3840 \def\XINT:NE:f:tacitzeroifone_b\XINT:expr:f:tacitzeroifone #1#2#3&&A#4#5%
3841 {%
3842     \if\relax#3\relax\expandafter\xint_firstoftwo\else
3843         \expandafter\xint_secondoftwo\fi
3844     {\XINT:NE:f:one:from:two:direct#4{0}}%
3845     {\expandafter\XINT:NE:f:one:from:two:direct\expandafter#5%
3846         \expanded{{\XINT:NE:f:one:from:one:direct\xintNum{#2}}}}}%
3847     {#1}%
3848}%
3849 \def\XINT:NE:f:iitacitzeroifone:direct#1{%
3850 \def\XINT:NE:f:iitacitzeroifone:direct##1!%
3851 {%
3852     \if0\XINT:NE:hasilde ##1~!\relax
3853         \XINT:NE:hashash ##1#1!\relax 0\else

```

```

3854      \xint_dothis\XINT:NE:f:iitacitzeroifone_a\fi
3855      \xint_orthat\XINT:NE:f:iitacitzeroifone_b ##1&&A%
3856 }}\expandafter\XINT:NE:f:iitacitzeroifone:direct\string#%
3857 \def\XINT:NE:f:iitacitzeroifone_a #1#2&&A#3%
3858 {%
3859      \detokenize{\romannumeral`$XINT_expr_null\expandafter#1\expanded{#2}$XINT_expr_exclam#3}%
3860 }%
3861 \def\XINT:NE:f:iitacitzeroifone_b\XINT:expr:f:iitacitzeroifone #1#2#3&&A#4%
3862 {%
3863      \if\relax#3\relax\expandafter\xint_firstoftwo\else
3864              \expandafter\xint_secondoftwo\fi
3865      {\XINT:NE:f:one:from:two:direct#4{0}}}%
3866      {\XINT:NE:f:one:from:two:direct#4{#2}}}%
3867      {#1}%
3868 }%
3869 \def\XINT:NE:x:one:from:two #1#2#3{\XINT:NE:x:one:from:two_fork #2&&A#3&&A#1{#2}{#3}}%
3870 \def\XINT:NE:x:one:from:two_fork #1{%
3871 \def\XINT:NE:x:one:from:two_fork ##1##2&&A##3##4&&A%
3872 {%
3873      \if0\XINT:NE:hashtilde ##1##3~!\relax\XINT:NE:hashash ##1##3#1!\relax 0%
3874      \else
3875              \expandafter\XINT:NE:x:one:from:two:p
3876      \fi
3877 }}\expandafter\XINT:NE:x:one:from:two_fork\string#%
3878 \def\XINT:NE:x:one:from:two:p #1#2#3%
3879      {\~expanded{\detokenize{\expandafter#1}\~expanded{{#2}{#3}}}}}%
3880 \def\XINT:NE:x:listscl #1{%
3881 \def\XINT:NE:x:listscl ##1##2&%
3882 {%
3883      \if0\expandafter\XINT:NE:hashtilde\detokenize{##2}~!\relax
3884              \expandafter\XINT:NE:hashash\detokenize{##2}#1!\relax 0%
3885      \else
3886              \expandafter\XINT:NE:x:listscl:p
3887      \fi
3888      ##1##2&%
3889 }}\expandafter\XINT:NE:x:listscl\string#%
3890 \def\XINT:NE:x:listscl:p #1#2_#3&(#4%
3891 {%
3892      \detokenize{\expanded\XINT:expr:ListSel{{#3}{#4}}}}}%
3893 }%
3894 \def\XINT:expr:ListSel{\expandafter\XINT:expr:ListSel_i\expanded}%
3895 \def\XINT:expr:ListSel_i #1#2{\XINT_ListSel_top #2_#1&({#2}}}%
3896 \def\XINT:NE:f:reverse #1{%
3897 \def\XINT:NE:f:reverse ##1^%
3898 {%
3899      \if0\expandafter\XINT:NE:hashtilde\detokenize\expandafter{\xint_gobble_i##1}~!\relax
3900              \expandafter\XINT:NE:hashash\detokenize{##1}#1!\relax 0%
3901      \else
3902              \expandafter\XINT:NE:f:reverse:p
3903      \fi
3904      ##1^%
3905 }}\expandafter\XINT:NE:f:reverse\string#%

```

```

3906 \def\XINT:NE:f:reverse:p #1^#2\xint_bye
3907 {%
3908     \expandafter\XINT:NE:f:reverse:p_i\expandafter{\xint_gobble_i#1}%
3909 }%
3910 \def\XINT:NE:f:reverse:p_i #1%
3911 {%
3912     \detokenize{\romannumeral0\XINT:expr:f:reverse{#1}}}%
3913 }%
3914 \def\XINT:expr:f:reverse{\expandafter\XINT:expr:f:reverse_i\expanded}%
3915 \def\XINT:expr:f:reverse_i #1%
3916 {%
3917     \XINT_expr_reverse #1^^#1\xint:\xint:\xint:\xint:
3918         \xint:\xint:\xint:\xint:\xint_bye
3919 }%
3920 \def\XINT:NE:f:from:delim:u #1{%
3921 \def\XINT:NE:f:from:delim:u ##1##2^%
3922 {%
3923     \if0\expandafter\XINT:NE:hashtilde\detokenize{##2}~!\relax
3924         \expandafter\XINT:NE:hashash\detokenize{##2}#1!\relax 0%
3925         \xint_afterfi{\expandafter\XINT_foof_checkifnumber\expandafter##1\string}%
3926     \else
3927         \xint_afterfi{\XINT:NE:f:from:delim:u:p##1\empty}%
3928     \fi
3929     ##2^%
3930 }%\expandafter\XINT:NE:f:from:delim:u\string%%
3931 \def\XINT:NE:f:from:delim:u:p #1#2^%
3932 {%
3933     \detokenize{\expandafter\XINT:foof:checkifnumber\expandafter#1}~expanded{#2}$XINT_expr_caret%$
3934 }%
3935 \def\XINT:foof:checkifnumber#1{\expandafter\XINT_foof_checkifnumber\expandafter#1\string}%
3936 \def\XINT:NE:f:LFL#1#2{\expandafter\XINT:NE:f:LFL_a\expandafter#1#2\XINT:NE:f:LFL_a}%
3937 \def\XINT:NE:f:LFL_a#1#2%
3938 {%
3939     \if#2i\else\expandafter\XINT:NE:f:LFL_p
3940     \fi #1%
3941 }%
3942 \def\XINT:NE:r:check#1{%
3943 \def\XINT:NE:r:check##1\XINT:NE:f:LFL_a
3944 {%
3945     \if0\expandafter\XINT:NE:hashtilde\detokenize{##1}~!\relax%
3946         \expandafter\XINT:NE:hashash\detokenize{##1}#1!\relax 0%
3947     \else
3948         \expandafter\XINT:NE:r:check:p
3949     \fi
3950     1\expandafter{\romannumeral\XINT:NEsaved:r:check##1}%
3951 }%\expandafter\XINT:NE:r:check\string%%
3952 \def\XINT:NE:r:check:p 1\expandafter#1{\XINT:NE:r:check:p_i#1}%
3953 \def\XINT:NE:r:check:p_i\romannumeral\XINT:NEsaved:r:check{\XINT:NE:r:check:p_ii\empty}%
3954 \def\XINT:NE:r:check:p_ii#1^%
3955 {%
3956     5~expanded{{~romannumeral~XINT:NEsaved:r:check#1$XINT_expr_caret}}}%$
3957 }%

```

```

3958 \def\XINT:NE:f:LFL_p#1%
3959 {%
3960   \detokenize{\romannumeral`$XINT_expr_null\expandafter#1}%$
3961 }%
3962 \catcode`- 11
3963 \def\XINT:NE:exec_? #1#2%
3964 {%
3965   \XINT:NE:exec_?_b #2&&A#1{#2}%
3966 }%
3967 \def\XINT:NE:exec_?_b #1{%
3968 \def\XINT:NE:exec_?_b ##1&&A%
3969 {%
3970   \if0\XINT:NE:hasilde ##1~!\relax
3971     \XINT:NE:hashash ##1#1!\relax 0%
3972   \xint_dothis\XINT:NE:exec_?:x\fi
3973   \xint_orthat\XINT:NE:exec_?:p
3974 }}\expandafter\XINT:NE:exec_?_b\string%%
3975 \def\XINT:NE:exec_?:x #1#2#3%
3976 {%
3977   \expandafter\XINT_expr_check-_after?\expandafter#1%
3978   \romannumeral`&&@\expandafter\XINT_expr_getnext\romannumeral0\xintiiifnotzero#3%
3979 }%
3980 \def\XINT:NE:exec_?:p #1#2#3#4#5%
3981 {%
3982   \csname XINT_expr_func_*If\expandafter\endcsname
3983   \romannumeral`&&@#2\XINTfstop.{#3},{#4},{#5})%
3984 }%
3985 \expandafter\def\csname XINT_expr_func_*If\endcsname #1#2#3%
3986 {%
3987   #1#2{~expanded{~xintiiifNotZero#3}}%
3988 }%
3989 \def\XINT:NE:exec_?? #1#2#3%
3990 {%
3991   \XINT:NE:exec_??_b #2&&A#1{#2}%
3992 }%
3993 \def\XINT:NE:exec_??_b #1{%
3994 \def\XINT:NE:exec_??_b ##1&&A%
3995 {%
3996   \if0\XINT:NE:hasilde ##1~!\relax
3997     \XINT:NE:hashash ##1#1!\relax 0%
3998   \xint_dothis\XINT:NE:exec_??:x\fi
3999   \xint_orthat\XINT:NE:exec_??:p
4000 }}\expandafter\XINT:NE:exec_??_b\string%%
4001 \def\XINT:NE:exec_??:x #1#2#3%
4002 {%
4003   \expandafter\XINT_expr_check-_after?\expandafter#1%
4004   \romannumeral`&&@\expandafter\XINT_expr_getnext\romannumeral0\xintiiifsgn#3%
4005 }%
4006 \def\XINT:NE:exec_??:p #1#2#3#4#5#6%
4007 {%
4008   \csname XINT_expr_func_*IfSgn\expandafter\endcsname
4009   \romannumeral`&&@#2\XINTfstop.{#3},{#4},{#5},{#6})%

```

```

4010 }%
4011 \expandafter\def\csname XINT_expr_func_IfSgn\endcsname #1#2#3%
4012 {%
4013     #1#2{\expanded{\xintiifSgn#3}}%
4014 }%
4015 \catcode`- 12
4016 \def\XINT:NE:branch #1%
4017 {%
4018     \if0\XINT:NE:hasilde #1~!\relax 0\else
4019         \xint_dothis\XINT:NE:branch_a\fi
4020     \xint_orthat\XINT:NE:branch_b #1&&A%
4021 }%
4022 \def\XINT:NE:branch_a\romannumeral`&&@#1#2&&A%
4023 {%
4024     \expandafter{\detokenize{\expandafter#1\expanded}{#2}}%
4025 }%
4026 \def\XINT:NE:branch_b#1{%
4027 \def\XINT:NE:branch_b\romannumeral`&&@##1##2##3&&A%
4028 {%
4029     \expandafter{\romannumeral`&&@%
4030         \if0\XINT:NE:hasilde ##2~!\relax
4031             \XINT:NE:hashash ##2#1!\relax 0\else
4032             \expandafter\string\fi
4033         ##1{##2}##3}%
4034 }}\expandafter\XINT:NE:branch_b\string#%
4035 \def\XINT:NE:seqx#1{%
4036 \def\XINT:NE:seqx\XINT_allexpr_seqx##1##2%
4037 {%
4038     \if 0\expandafter\XINT:NE:hasilde\detokenize{##2}~!\relax
4039         \expandafter\XINT:NE:hashash \detokenize{##2}#1!\relax 0%
4040     \else
4041         \expandafter\XINT:NE:seqx:p
4042     \fi \XINT_allexpr_seqx{##1}{##2}%
4043 }}\expandafter\XINT:NE:seqx\string#%
4044 \def\XINT:NE:seqx:p\XINT_allexpr_seqx #1#2#3#4%
4045 {%
4046     \expandafter\XINT_expr_put_op_first
4047     \expanded {%
4048     {%
4049         \detokenize
4050         {%
4051             \expanded\bgroup
4052             \expanded
4053             {\unexpanded{\XINT_expr_seq:_b{#1#4\relax $XINT_expr_exclam #3}}}%
4054             #2$XINT_expr_caret}%
4055         }%
4056     }%
4057     \expandafter}\romannumeral`&&\XINT_expr_getop
4058 }%
4059 \def\XINT:NE:opx#1{%
4060 \def\XINT:NE:opx\XINT_allexpr_opx ##1##2##3##4##5##6##7##8%
4061 {%

```

```

4062 \if 0\expandafter\XINT:NE:hashtilde\detokenize{##4}~!\relax
4063 \expandafter\XINT:NE:hashash \detokenize{##4}#1!\relax 0%
4064 \else
4065 \expandafter\XINT:NE:opx:p
4066 \fi \XINT_allexpr_opx ##1{##2}{##3}{##4}% en fait ##2 = \xint_c_, ##3 = \relax
4067 }}\expandafter\XINT:NE:opx:string#%
4068 \def\XINT:NE:opx:p\XINT_allexpr_opx #1#2#3#4#5#6#7#8%
4069 {%
4070 \expandafter\XINT_expr_put_op_first
4071 \expanded {%
4072 {%
4073 \detokenize
4074 {%
4075 \expanded\bgroup
4076 \expanded{\unexpanded{\XINT_expr_iter:_b
4077 {#1\expandafter\XINT_allexpr_opx_ifnotomitted
4078 \romannumeral0#1#6\relax#7@\relax $XINT_expr_exclam #5}}}%
4079 #4$XINT_expr_caret$XINT_expr_tilde{{#8}}}%$
4080 }%
4081 }%
4082 \expandafter}\romannumeral`&&\XINT_expr_getop
4083 }%
4084 \def\XINT:NE:iter{\expandafter\XINT:NE:itery\expandafter}%
4085 \def\XINT:NE:itery#1{%
4086 \def\XINT:NE:itery\XINT_expr_itery##1##2%
4087 {%
4088 \if 0\expandafter\XINT:NE:hashtilde\detokenize{##1##2}~!\relax
4089 \expandafter\XINT:NE:hashash \detokenize{##1##2}#1!\relax 0%
4090 \else
4091 \expandafter\XINT:NE:itery:p
4092 \fi \XINT_expr_itery{##1}{##2}%
4093 }}\expandafter\XINT:NE:itery:string#%
4094 \def\XINT:NE:itery:p\XINT_expr_itery #1#2#3#4#5%
4095 {%
4096 \expandafter\XINT_expr_put_op_first
4097 \expanded {%
4098 {%
4099 \detokenize
4100 {%
4101 \expanded\bgroup
4102 \expanded{\unexpanded{\XINT_expr_iter:_b {#5#4\relax $XINT_expr_exclam #3}}}%
4103 #1$XINT_expr_caret$XINT_expr_tilde{#2}}}%$
4104 }%
4105 }%
4106 \expandafter}\romannumeral`&&\XINT_expr_getop
4107 }%
4108 \def\XINT:NE:rseq{\expandafter\XINT:NE:rseqy\expandafter}%
4109 \def\XINT:NE:rseqy#1{%
4110 \def\XINT:NE:rseqy\XINT_expr_rseqy##1##2%
4111 {%
4112 \if 0\expandafter\XINT:NE:hashtilde\detokenize{##1##2}~!\relax
4113 \expandafter\XINT:NE:hashash \detokenize{##1##2}#1!\relax 0%

```



```

4114 \else
4115 \expandafter\XINT:NE:rseqy:p
4116 \fi \XINT_expr_rseqy{##1}{##2}%
4117 }}\expandafter\XINT:NE:rseqy:string#%
4118 \def\XINT:NE:rseqy:p\XINT_expr_rseqy #1#2#3#4#5%
4119 {%
4120 \expandafter\XINT_expr_put_op_first
4121 \expanded {%
4122 {%
4123 \detokenize
4124 {%
4125 \expanded\bgroup
4126 \expanded{#2\unexpanded{\XINT_expr_rseq:_b {#5#4\relax $XINT_expr_exclam #3}}%
4127 #1$XINT_expr_caret$XINT_expr_tilde{#2}}}%$
4128 }%
4129 }%
4130 \expandafter}\romannumeral`&&\XINT_expr_getop
4131 }%
4132 \def\XINT:NE:iterr{\expandafter\XINT:NE:iterr\expandafter}%
4133 \def\XINT:NE:iterr#1{%
4134 \def\XINT:NE:iterr\XINT_expr_iterr##1#2%
4135 {%
4136 \if 0\expandafter\XINT:NE:hasilde\detokenize{##1##2}~!\relax
4137 \expandafter\XINT:NE:hashash \detokenize{##1##2}#1!\relax 0%
4138 \else
4139 \expandafter\XINT:NE:iterr:p
4140 \fi \XINT_expr_iterr{##1}{##2}%
4141 }}\expandafter\XINT:NE:iterr:string#%
4142 \def\XINT:NE:iterr:p\XINT_expr_iterr #1#2#3#4#5%
4143 {%
4144 \expandafter\XINT_expr_put_op_first
4145 \expanded {%
4146 {%
4147 \detokenize
4148 {%
4149 \expanded\bgroup
4150 \expanded{\unexpanded{\XINT_expr_iterr:_b {#5#4\relax $XINT_expr_exclam #3}}%
4151 #1$XINT_expr_caret$XINT_expr_tilde #20$XINT_expr_qmark}%
4152 }%
4153 }%
4154 \expandafter}\romannumeral`&&\XINT_expr_getop
4155 }%
4156 \def\XINT:NE:rrseq{\expandafter\XINT:NE:rrseq\expandafter}%
4157 \def\XINT:NE:rrseq#1{%
4158 \def\XINT:NE:rrseq\XINT_expr_rrseq##1#2%
4159 {%
4160 \if 0\expandafter\XINT:NE:hasilde\detokenize{##1##2}~!\relax
4161 \expandafter\XINT:NE:hashash \detokenize{##1##2}#1!\relax 0%
4162 \else
4163 \expandafter\XINT:NE:rrseqy:p
4164 \fi \XINT_expr_rrseqy{##1}{##2}%
4165 }}\expandafter\XINT:NE:rrseq:string#%

```

```

4166 \def\XINT:NE:rrseqy:p\XINT_expr_rrseqy #1#2#3#4#5#6%
4167 {%
4168     \expandafter\XINT_expr_put_op_first
4169     \expanded {%
4170         {%
4171             \detokenize
4172             {%
4173                 \expanded\bgroup
4174                 \expanded{#2\unexpanded{\XINT_expr_rrseq:_b {#6#5\relax $XINT_expr_exclam #4}}}%
4175                 #1$XINT_expr_caret$XINT_expr_tilde #30$XINT_expr_qmark}%
4176             }%
4177         }%
4178     \expandafter}\romannumeral`&&\XINT_expr_getop
4179 }%
4180 \def\XINT:NE:x:toblist#1{%
4181 \def\XINT:NE:x:toblist\XINT:expr:toblistwith##1##2%
4182 {%
4183     \if 0\expandafter\XINT:NE:hasilde\detokenize{##2}~!\relax
4184         \expandafter\XINT:NE:hashash \detokenize{##2}#1!\relax 0%
4185     \else
4186         \expandafter\XINT:NE:x:toblist:p
4187     \fi \XINT:expr:toblistwith{##1}{##2}%
4188 }}\expandafter\XINT:NE:x:toblist\string#%
4189 \def\XINT:NE:x:toblist:p\XINT:expr:toblistwith #1#2{{\XINTfstop.{#2}}}%
4190 \def\XINT:NE:x:flatten#1{%
4191 \def\XINT:NE:x:flatten\XINT:expr:flatten##1%
4192 {%
4193     \if 0\expandafter\XINT:NE:hasilde\detokenize{##1}~!\relax
4194         \expandafter\XINT:NE:hashash \detokenize{##1}#1!\relax 0%
4195     \else
4196         \expandafter\XINT:NE:x:flatten:p
4197     \fi \XINT:expr:flatten{##1}%
4198 }}\expandafter\XINT:NE:x:flatten\string#%
4199 \def\XINT:NE:x:flatten:p\XINT:expr:flatten #1%
4200 {%
4201     {%
4202         \detokenize
4203         {%
4204             \expandafter\XINT:expr:flatten_checkempty
4205             \detokenize\expandafter{\expanded{#1}}$XINT_expr_caret%$
4206         }%
4207     }%
4208 }%
4209 \def\XINT:NE:x:zip#1{%
4210 \def\XINT:NE:x:zip\XINT:expr:zip##1%
4211 {%
4212     \if 0\expandafter\XINT:NE:hasilde\detokenize{##1}~!\relax
4213         \expandafter\XINT:NE:hashash \detokenize{##1}#1!\relax 0%
4214     \else
4215         \expandafter\XINT:NE:x:zip:p
4216     \fi \XINT:expr:zip{##1}%
4217 }}\expandafter\XINT:NE:x:zip\string#%

```

```

4218 \def\XINT:NE:x:zip:p\XINT:expr:zip #1%
4219 {%
4220     \expandafter{%
4221         \detokenize
4222         {%
4223             \expanded\expandafter\XINT_zip_A\expanded{#1}\xint_bye\xint_bye
4224         }%
4225     }%
4226 }%
4227 \def\XINT:NE:x:mapwithin#1{%
4228 \def\XINT:NE:x:mapwithin\XINT:expr:mapwithin ##1##2%
4229 {%
4230     \if 0\expandafter\XINT:NE:hasilde\detokenize{##2}~!\relax
4231         \expandafter\XINT:NE:hashash \detokenize{##2}#1!\relax 0%
4232     \else
4233         \expandafter\XINT:NE:x:mapwithin:p
4234     \fi \XINT:expr:mapwithin {##1}{##2}%
4235 }%\expandafter\XINT:NE:x:mapwithin\string#%
4236 \def\XINT:NE:x:mapwithin:p \XINT:expr:mapwithin #1#2%
4237 {%
4238     {%
4239         \detokenize
4240         {%
4241             \expanded
4242             {%
4243                 \expandafter\XINT:expr:mapwithin_checkempty
4244                 \expanded{\noexpand#1$XINT_expr_exclam\expandafter}%$
4245                 \detokenize\expandafter{\expanded{#2}}$XINT_expr_caret%$
4246             }%
4247         }%
4248     }%
4249 }%
4250 \def\XINT:NE:x:ndmapx#1{%
4251 \def\XINT:NE:x:ndmapx\XINT_allexpr_ndmapx_a ##1##2^%
4252 {%
4253     \if 0\expandafter\XINT:NE:hasilde\detokenize{##2}~!\relax
4254         \expandafter\XINT:NE:hashash \detokenize{##2}#1!\relax 0%
4255     \else
4256         \expandafter\XINT:NE:x:ndmapx:p
4257     \fi \XINT_allexpr_ndmapx_a ##1##2^%
4258 }%\expandafter\XINT:NE:x:ndmapx\string#%
4259 \def\XINT:NE:x:ndmapx:p #1#2#3^~\relax
4260 {%
4261     \detokenize
4262     {%
4263         \expanded{%
4264             \expandafter#1\expandafter#2\expanded{#3}$XINT_expr_caret\relax %$
4265         }%
4266     }%
4267 }%

```

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 \xintdefunc never expands when used in another definition, so it can implement recursive definitions.

\XINT:NE:userefunc et al. added at 1.3e.

I added at \xintdefefunc, \xintdefiiefunc, \xintdeffloatefunc 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.

```

4268 \def\xINTusenoargfunc #1%
4269 {%
4270     0\csname #1\endcsname
4271 }%
4272 \def\xINT:NE:usenoargfunc\csname #1\endcsname
4273 {%
4274     ~romannumeral~XINTusenoargfunc{#1}%
4275 }%
4276 \def\xINTusefunc #1%
4277 {%
4278     0\csname #1\expandafter\endcsname\expanded
4279 }%
4280 \def\xINT:NE:usefunc #1#2#3%
4281 {%
4282     ~romannumeral~XINTusefunc{#1}{#3}\iffalse{\fi}%
4283 }%
4284 \def\xINTuseufunc #1%
4285 {%
4286     \expanded\expandafter\xINT:expr:mapwithin\csname #1\expandafter\endcsname\expanded
4287 }%
4288 \def\xINT:NE:useufunc #1#2#3%
4289 {%
4290     {{~expanded~XINTuseufunc{#1}{#3}}}%
4291 }%
4292 \def\xINT:NE:userfunc #1{%
4293 \def\xINT:NE:userfunc ##1##2##3%
4294 {%
4295     \if0\expandafter\xINT:NE:hasilde\detokenize{##3}~!\relax
4296         \expandafter\xINT:NE:hashash\detokenize{##3}#1!\relax 0%
4297         \expandafter\xINT:NE:userfunc_x
4298     \else
4299         \expandafter\xINT:NE:usefunc
4300     \fi {##1}{##2}{##3}%
4301 }}\expandafter\xINT:NE:userfunc\string#%
4302 \def\xINT:NE:userfunc_x #1#2#3{#2#3\iffalse{\fi}}%
4303 \def\xINT:NE:userufunc #1{%
4304 \def\xINT:NE:userufunc ##1##2##3%
4305 {%
4306     \if0\expandafter\xINT:NE:hasilde\detokenize{##3}~!\relax
4307         \expandafter\xINT:NE:hashash\detokenize{##3}#1!\relax 0%
4308         \expandafter\xINT:NE:userufunc_x
4309     \else
4310         \expandafter\xINT:NE:useufunc
4311     \fi {##1}{##2}{##3}%
4312 }}\expandafter\xINT:NE:userufunc\string#%
```

```

4313 \def\XINT:NE:userufunc_x #1{\XINT:expr:mapwithin}%
4314 \def\XINT:NE:macrofunc #1#2%
4315   {\expandafter\XINT:NE:macrofunc:a\string#1#2\empty&}%
4316 \def\XINT:NE:macrofunc:a#1\csname #2\endcsname#3&%
4317   {\XINTusemacrofunc{#1}{#2}{#3}}}%
4318 \def\XINTusemacrofunc #1#2#3%
4319 {%
4320   \romannumeral0\expandafter\xint_stop_atfirstofone
4321   \romannumeral0#1\csname #2\endcsname#3\relax
4322 }%

```

11.30.6 \XINT_expr_redefinemacros

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.

```

4323 \catcode`- 11
4324 \def\XINT_expr_redefinemacros {%
4325   \let\XINT:NEhook:unpack          \XINT:NE:unpack
4326   \let\XINT:NEhook:f:one:from:one  \XINT:NE:f:one:from:one
4327   \let\XINT:NEhook:f:one:from:one:direct \XINT:NE:f:one:from:one:direct
4328   \let\XINT:NEhook:f:one:from:two   \XINT:NE:f:one:from:two
4329   \let\XINT:NEhook:f:one:from:two:direct \XINT:NE:f:one:from:two:direct
4330   \let\XINT:NEhook:x:one:from:two   \XINT:NE:x:one:from:two
4331   \let\XINT:NEhook:f:one:and:opt:direct \XINT:NE:f:one:and:opt:direct
4332   \let\XINT:NEhook:f:tacitzeroifone:direct \XINT:NE:f:tacitzeroifone:direct
4333   \let\XINT:NEhook:f:iitacitzeroifone:direct \XINT:NE:f:iitacitzeroifone:direct
4334   \let\XINT:NEhook:x:listselsel    \XINT:NE:x:listselsel
4335   \let\XINT:NEhook:f:reverse       \XINT:NE:f:reverse
4336   \let\XINT:NEhook:f:from:delim:u  \XINT:NE:f:from:delim:u
4337   \let\XINT:NEhook:f:LFL           \XINT:NE:f:LFL
4338   \let\XINT:NEhook:r:check         \XINT:NE:r:check
4339   \let\XINT:NEhook:branch          \XINT:NE:branch
4340   \let\XINT:NEhook:seqx            \XINT:NE:seqx
4341   \let\XINT:NEhook:opx             \XINT:NE:opx
4342   \let\XINT:NEhook:rseq            \XINT:NE:rseq
4343   \let\XINT:NEhook:iter            \XINT:NE:iter
4344   \let\XINT:NEhook:rrseq           \XINT:NE:rrseq
4345   \let\XINT:NEhook:iterr           \XINT:NE:iterr
4346   \let\XINT:NEhook:x:toblist       \XINT:NE:x:toblist
4347   \let\XINT:NEhook:x:flatten       \XINT:NE:x:flatten
4348   \let\XINT:NEhook:x:zip           \XINT:NE:x:zip
4349   \let\XINT:NEhook:x:mapwithin     \XINT:NE:x:mapwithin
4350   \let\XINT:NEhook:x:ndmapx        \XINT:NE:x:ndmapx
4351   \let\XINT:NEhook:userfunc        \XINT:NE:userfunc
4352   \let\XINT:NEhook:userufunc       \XINT:NE:userufunc
4353   \let\XINT:NEhook:usernoargfunc    \XINT:NE:usernoargfunc
4354   \let\XINT:NEhook:macrofunc       \XINT:NE:macrofunc
4355   \def\XINTinRandomFloatSdigits{\XINTinRandomFloatSdigits }%
4356   \def\XINTinRandomFloatSixteen{\XINTinRandomFloatSixteen }%
4357   \def\xintiiRandRange{\xintiiRandRange }%

```

```

4358 \def\xintiiRandRangeAtoB{~xintiiRandRangeAtoB }%
4359 \def\xintRandBit{~xintRandBit }%
4360 \let\xINT_expr_exec_? \XINT:NE:exec_?
4361 \let\xINT_expr_exec_?? \XINT:NE:exec_??
4362 \def\xINT_expr_op_? {\XINT_expr_op_?{\XINT_expr_op_-xii\xINT_expr_oparen}}%
4363 \def\xINT_flexpr_op_?{\XINT_expr_op_?{\XINT_flexpr_op_-xii\xINT_flexpr_oparen}}%
4364 \def\xINT_iiexpr_op_?{\XINT_expr_op_?{\XINT_iiexpr_op_-xii\xINT_iiexpr_oparen}}%
4365 }%
4366 \catcode`- 12

```

11.30.7 \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).

```

4367 \def\xintNewExpr      {\XINT_NewExpr\xint_firstofone\xintexpr      \XINT_newexpr_clean}%
4368 \def\xintNewFloatExpr{\XINT_NewExpr\xint_firstofone\xintfloatexpr\XINT_newexpr_clean}%
4369 \def\xintNewIExpr     {\XINT_NewExpr\xint_firstofone\xintiexpr     \XINT_newexpr_clean}%
4370 \def\xintNewIIExpr    {\XINT_NewExpr\xint_firstofone\xintiiexpr    \XINT_newexpr_clean}%
4371 \def\xintNewBoolExpr {\XINT_NewExpr\xint_firstofone\xintboolexpr \XINT_newexpr_clean}%
4372 \def\xINT_newexpr_clean #1>#2#3{\noexpand\expanded\noexpand\xintNEprinthook}%
4373 \def\xintNEprinthook#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

```

4374 \def\xINT_NewFunc      {\XINT_NewExpr\xint_gobble_i\xintthebareeval\xINT_newfunc_clean}%
4375 \def\xINT_NewFloatFunc{\XINT_NewExpr\xint_gobble_i\xintthebarefloateval\xINT_newfunc_clean}%
4376 \def\xINT_NewIIFunc    {\XINT_NewExpr\xint_gobble_i\xintthebareiieval\xINT_newfunc_clean}%
4377 \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.

```

4378 \def\xINT_NewExpr #1#2#3#4#5[#6]%
4379 {%
4380 \begingroup
4381 \ifcase #6\relax
4382 \toks0 {\endgroup\xINT_global\def#4}%
4383 \or \toks0 {\endgroup\xINT_global\def#4##1}%
4384 \or \toks0 {\endgroup\xINT_global\def#4##1##2}%
4385 \or \toks0 {\endgroup\xINT_global\def#4##1##2##3}%
4386 \or \toks0 {\endgroup\xINT_global\def#4##1##2##3##4}%

```

```

4387 \or \toks0 {\endgroup\XINT_global\def#4##1##2##3##4##5}%
4388 \or \toks0 {\endgroup\XINT_global\def#4##1##2##3##4##5##6}%
4389 \or \toks0 {\endgroup\XINT_global\def#4##1##2##3##4##5##6##7}%
4390 \or \toks0 {\endgroup\XINT_global\def#4##1##2##3##4##5##6##7##8}%
4391 \or \toks0 {\endgroup\XINT_global\def#4##1##2##3##4##5##6##7##8##9}%
4392 \fi
4393 #1\xintexprSafeCatcodes
4394 \XINT_expr_redefinmacros
4395 \XINT_NewExpr_a #1#2#3#4%
4396 }%

```

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.

```

4397 \catcode`~ 3 \catcode`? 3
4398 \def\XINT_expr_tilde{~}\def\XINT_expr_qmark{?}% catcode 3
4399 \def\XINT_expr_caret{^}\def\XINT_expr_exclam{!}% catcode 11
4400 \def\XINT_expr_tab{&}% catcode 7
4401 \def\XINT_expr_null{&&@}%
4402 \catcode`~ 13 \catcode`@ 14 \catcode`\% 6 \catcode`# 12 \catcode`$ 11 @ $
4403 \def\XINT_NewExpr_a %1%2%3%4%5@
4404 {@
4405 \def\XINT_tmpa %1%2%3%4%5%6%7%8%9{%5}@
4406 \def~{$noexpand$}@
4407 \catcode`: 11 \catcode`_ 11 \catcode`\@ 11
4408 \catcode`# 12 \catcode`~ 13 \escapechar 126
4409 \endlinechar -1 \everyeof {\noexpand }@
4410 \edef\XINT_tmpb
4411 {\scantokens\expandafter{\romannumeral`&&\expandafter
4412 %2\XINT_tmpa{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}\relax}@
4413 }@

```

```

4414 \escapechar 92 \catcode`# 6 \catcode`$ 0 @ $
4415 \edef\XINT_tmpa %%1%%2%%3%%4%%5%%6%%7%%8%%9@
4416 { \scantokens\expandafter{\expandafter%3\meaning\XINT_tmpb}}@
4417 \the\toks0\expandafter
4418 {\XINT_tmpa{%%1}{%%2}{%%3}{%%4}{%%5}{%%6}{%%7}{%%8}{%%9}}@
4419 %1{\ifxintverbose
4420 \xintMessage{xintexpr}{Info}@
4421 {\string%4\space now with @
4422 \ifxintglobaldefs global \fi meaning \meaning%4}@
4423 \fi}@
4424 }@
4425 \catcode`% 14
4426 \XINTsetcatcodes % clean up to avoid surprises if something changes

```

11.30.8 \ifxintexprsafecatcodes, \xintexprSafeCatcodes, \xintexprRestoreCatcodes

1.3c (2018/06/17) [commented 2018/06/17].

Added \ifxintexprsafecatcodes to allow nesting

```

4427 \newif\ifxintexprsafecatcodes
4428 \let\xintexprRestoreCatcodes\empty
4429 \def\xintexprSafeCatcodes
4430 {%
4431 \unless\ifxintexprsafecatcodes
4432 \edef\xintexprRestoreCatcodes {%
4433 \endlinechar=\the\endlinechar
4434 \catcode59=\the\catcode59 % ;
4435 \catcode34=\the\catcode34 % "
4436 \catcode63=\the\catcode63 % ?
4437 \catcode124=\the\catcode124 % |
4438 \catcode38=\the\catcode38 % &
4439 \catcode33=\the\catcode33 % !
4440 \catcode93=\the\catcode93 % ]
4441 \catcode91=\the\catcode91 % [
4442 \catcode94=\the\catcode94 % ^
4443 \catcode95=\the\catcode95 % _
4444 \catcode47=\the\catcode47 % /
4445 \catcode41=\the\catcode41 % )
4446 \catcode40=\the\catcode40 % (
4447 \catcode42=\the\catcode42 % *
4448 \catcode43=\the\catcode43 % +
4449 \catcode62=\the\catcode62 % >
4450 \catcode60=\the\catcode60 % <
4451 \catcode58=\the\catcode58 % :
4452 \catcode46=\the\catcode46 % .
4453 \catcode45=\the\catcode45 % -
4454 \catcode44=\the\catcode44 % ,
4455 \catcode61=\the\catcode61 % =
4456 \catcode96=\the\catcode96 % `
4457 \catcode32=\the\catcode32\relax % space
4458 \noexpand\xintexprsafecatcodesfalse
4459 }%
4460 \fi

```



```

4461 \xintexprsafecatcodestrue
4462     \endlinechar=13 %
4463     \catcode59=12 % ;
4464     \catcode34=12 % "
4465     \catcode63=12 % ?
4466     \catcode124=12 % |
4467     \catcode38=4 % &
4468     \catcode33=12 % !
4469     \catcode93=12 % ]
4470     \catcode91=12 % [
4471     \catcode94=7 % ^
4472     \catcode95=8 % _
4473     \catcode47=12 % /
4474     \catcode41=12 % )
4475     \catcode40=12 % (
4476     \catcode42=12 % *
4477     \catcode43=12 % +
4478     \catcode62=12 % >
4479     \catcode60=12 % <
4480     \catcode58=12 % :
4481     \catcode46=12 % .
4482     \catcode45=12 % -
4483     \catcode44=12 % ,
4484     \catcode61=12 % =
4485     \catcode96=12 % `
4486     \catcode32=10 % space
4487 }%
4488 \let\XINT_tmpa\undefined \let\XINT_tmpb\undefined \let\XINT_tmpe\undefined
4489 \let\XINT_tmpe\undefined \let\XINT_tmpe\undefined
4490 \ifdefined\RequirePackage\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
4491 {\RequirePackage{xinttrig}}%
4492 \RequirePackage{xintlog}}%
4493 {\input xinttrig.sty
4494 \input xintlog.sty
4495 }%
4496 \XINTrestorecatcodesendinginput%

```

12 Package `xinttrig` implementation

Contents

12.1	Catcodes, ε -TeX and reload detection	435
12.2	Library identification	436
12.3	Ensure used letters are dummy letters	436
12.4	<code>\xintreloadxinttrig</code>	436
12.5	Auxiliary variables	436
12.5.1	<code>@twoPi</code> , <code>@threePiover2</code> , <code>@Pi</code> , <code>@Piover2</code>	437
12.5.2	<code>@oneDegree</code> , <code>@oneRadian</code>	437
12.6	Hack <code>\xintdeffloatfunc</code> for inserting usage of guard digits	437
12.7	The sine and cosine series	438
12.7.1	Support macros for the sine and cosine series	439
12.7.2	The poor man approximate but speedier approach for Digits at most 8	442
12.7.3	Declarations of the <code>@sin_aux()</code> and <code>@cos_aux()</code> functions	442
12.7.4	<code>@sin_series()</code> , <code>@cos_series()</code>	442
12.8	Range reduction for sine and cosine using degrees	443
12.8.1	Low level modulo 360 helper macro <code>\XINT_mod_ccclx_i</code>	443
12.8.2	<code>@sind_rr()</code> function and its support macro <code>\xintSind</code>	443
12.8.3	<code>@cosd_rr()</code> function and its support macro <code>\xintCosd</code>	445
12.9	<code>@sind()</code> , <code>@cosd()</code>	447
12.10	<code>@sin()</code> , <code>@cos()</code>	448
12.11	<code>@sinc()</code>	448
12.12	<code>@tan()</code> , <code>@tand()</code> , <code>@cot()</code> , <code>@cotd()</code>	448
12.13	<code>@sec()</code> , <code>@secd()</code> , <code>@csc()</code> , <code>@cscd()</code>	449
12.14	Core routine for inverse trigonometry	449
12.15	<code>@asin()</code> , <code>@asind()</code>	452
12.16	<code>@acos()</code> , <code>@acosd()</code>	452
12.17	<code>@atan()</code> , <code>@atand()</code>	453
12.18	<code>@Arg()</code> , <code>@atan2()</code> , <code>@Argd()</code> , <code>@atan2d()</code> , <code>@pArg()</code> , <code>@pArgd()</code>	453
12.19	Restore <code>\xintdeffloatfunc</code> to its normal state, with no extra digits	454
12.20	Let the functions be known to the <code>\xintexpr</code> parser	455
12.21	Synonyms: <code>@tg()</code> , <code>@cotg()</code>	455
12.22	Final clean-up	456

A preliminary implementation was done only late in the development of `\xintexpr`, as an example of the high level user interface, in January 2019. In March and April 2019 I improved the algorithm for the inverse trigonometrical functions and included the whole as a new `\xintexpr` module. But, as the high level interface provided no way to have intermediate steps executed with guard digits, the whole scheme could only target say P-2 digits where P is the prevailing precision, and only with a moderate requirement on what it means to have P-2 digits about correct.

Finally in April 2021, after having at long last added exponential and logarithm up to 62 digits and at a rather strong precision requirement (something like, say with inputs in normal ranges: targeting at most 0.505ulp distance to exact result), I revisited the code here.

We keep most of the high level usage of `\xintdeffloatfunc`, but hack into its process in order to let it map the 4 operations and some functions such as square-root to macros using 4 extra digits. This hack is enough to support the used syntax here, but is not usable generally. All functions and their auxiliaries defined during the time the hack applies are named with `@` as first letter.

Later the public functions, without the `@`, are defined as wrappers of the `@`-named ones, which float-round to P digits on output.

Apart from that the sine and cosine series were implemented at macro level, bypassing the `\xint-deffloatfunc` interface. This is done mainly for handling Digits at high value (24 or more) as it then becomes beneficial to float-round the variable to less and less digits, the deeper one goes into the series.

And regarding the arcsine I modified a bit my original idea in order to execute the first step in a single `\numexpr`. It turns out that that for 16 digits the algorithm then ``only'' needs one sine and one cosine evaluation (and a square-root), and there is no need for an arcsine series auxiliary then. I am aware this is by far not the ``best'' approach but the problem is that I am a bit enamored into the idea of the algorithm even though it is at least twice as costly than a sine evaluation! Actually, for many digits, it turns out the arcsine is less costly than two random sine evaluations, probably because the latter have the overhead of range reduction.

Speaking of this, the range reduction is rather naive and not extremely ambitious. I wrote it initially having only `sind()` and `cosd()` in mind, and in 2019 reduced degrees to radians in the most naive way possible. I have only slightly improved this for this 1.4e 2021 release, the announced precision for inputs less than say `1e6`, but at `1e8` and higher, one will start feeling the gradual loss of precision compared to the task of computing the exact mathematical result correctly rounded. Also, I do not worry here about what happens when the input is very near a big multiple of π , and one computes a sine for example. Maybe I will improve in future this aspect but I decided I was seriously running out of steam for the 1.4e release.

As commented in [xintlog](#) regarding exponential and logarithms, even though we have instilled here some dose of lower level coding, the whole suffers from [xintfrac](#) not yet having made floating point numbers a native type. Thus inefficiencies accumulate...

At 8 digits, the gain was only about 40% compared to 16 digits. So at the last minute, on the day I was going to do the release I decided to implement a poorman way for sine and cosine, for "speed". I transferred the idea from the arcsine `\numexpr` to sine and cosine. Indeed there is an interesting speed again of about 4X compared to applying the same approach as for higher values of Digits. Correct rounding during random testing is still obtained reasonably often (at any rate more than 95% of cases near 45 degrees and always faithful rounding), although at less than the 99% reached for the main branch handling Digits up to 62. But the precision is more than enough for usage in plots for example. I am keeping the guard digits, as removing then would add a further speed gain of about 20% to 40% but the precision then would drop dramatically, and this is not acceptable at the time of our 2021 standards (not a period of enlightenment generally speaking, though).

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 \edef\XINTtrigendinput{\XINTrestorecatcodes\noexpand\endinput}\XINTsetcatcodes%
38 \catcode`? 12

```

12.2 Library identification

```

39 \ifcsname xintlibver@trig\endcsname
40     \expandafter\xint_firstoftwo
41 \else
42     \expandafter\xint_secondoftwo
43 \fi
44 {\immediate\write-1{Reloading xinttrig library using Digits=\xinttheDigits.}}%
45 {\expandafter\gdef\csname xintlibver@trig\endcsname{2021/06/11 v1.4i}%
46 \XINT_providespackage
47 \ProvidesPackage{xinttrig}%
48 [2021/06/11 v1.4i Trigonometrical functions for xintexpr (JFB)]%
49 }%

```

12.3 Ensure used letters are dummy letters

```

50 \xintFor* #1 in {iDTVtuwxyzX}\do{\xintensuredummy{#1}}%

```

12.4 \xintreloadxinttrig

Much simplified at 1.4e, from a modified catcode regime management.

```

51 \def\xintreloadxinttrig{\input xinttrig.sty }%

```

12.5 Auxiliary variables

The variables with private names have extra digits. Whether private or public, the variables can all be redefined without impacting the defined functions, whose meanings will contain already the variable values.

Formerly variables holding the $1/n!$ were defined, but this got removed at 1.4e.

12.5.1 @twoPi, @threePiover2, @Pi, @Piover2

At 1.4e we need more digits, also `\xintdeffloatvar` changed and always rounds to `P=Digits` precision so we use another path to store values with extra digits.

```
52 \xintdefvar @twoPi :=
53   float(
54 6.2831853071795864769252867665590057683943387987502116419498891846156328125724180
55   ,\XINTdigitsormax+4);%
56 \xintdefvar @threePiover2 :=
57   float(
58 4.7123889803846898576939650749192543262957540990626587314624168884617246094293135
59   ,\XINTdigitsormax+4);%
60 \xintdefvar @Pi :=
61   float(
62 3.1415926535897932384626433832795028841971693993751058209749445923078164062862090
63   ,\XINTdigitsormax+4);%
64 \xintdefvar @Piover2 :=
65   float(
66 1.5707963267948966192313216916397514420985846996875529104874722961539082031431045
67   ,\XINTdigitsormax+4);%
```

12.5.2 @oneDegree, @oneRadian

Those are needed for range reduction, particularly `@oneRadian`. We define it with 12 extra digits. But the whole process of range reduction in radians is very naive one.

```
68 \xintdefvar @oneDegree :=
69   float(
70 0.017453292519943295769236907684886127134428718885417254560971914401710091146034494
71   ,\XINTdigitsormax+4);%
72 \xintdefvar @oneRadian :=
73   float(
74 57.295779513082320876798154814105170332405472466564321549160243861202847148321553
75   ,\XINTdigitsormax+12);%
```

12.6 Hack `\xintdeffloatfunc` for inserting usage of guard digits

1.4e. This is not a general approach, but it sufficient for the limited use case done here of `\xintdeffloatfunc`. What it does is to let `\xintdeffloatfunc` hardcode usage of macros which will execute computations with an elevated number of digits. But for example if `5/3` is encountered in a float expression it will remain unevaluated so one would have to use alternate input syntax for efficiency (`\xintexpr float(5/3,\xinttheDigits+4)\relax` as a subexpression, for example).

```
76 \catcode`~ 12
77 \def\xINT_tmpa#1#2#3.#4.%
78 {%
79   \let #1#2%
80   \def #2##1##2##3##4{##2##3{{~expanded{~unexpanded{#4[#3]}~expandafter}~expanded{##1##4}}}}%
81 }%
82 \expandafter\xINT_tmpa
83   \csname XINT_flexpr_exec_+~\expandafter\endcsname
84   \csname XINT_flexpr_exec_+~\expandafter\endcsname
85   \the\numexpr\xINTdigitsormax+4.~XINTinFloatAdd_wopt.%
86 \expandafter\xINT_tmpa
```

```

87 \csname XINT_flexpr_exec_-_\expandafter\endcsname
88 \csname XINT_flexpr_exec_-_\expandafter\endcsname
89 \the\numexpr\XINTdigitsormax+4.\simXINTinFloatSub_wopt.%
90 \expandafter\XINT_tmpa
91 \csname XINT_flexpr_exec_*_\expandafter\endcsname
92 \csname XINT_flexpr_exec_*_\expandafter\endcsname
93 \the\numexpr\XINTdigitsormax+4.\simXINTinFloatMul_wopt.%
94 \expandafter\XINT_tmpa
95 \csname XINT_flexpr_exec_/_\expandafter\endcsname
96 \csname XINT_flexpr_exec_/_\expandafter\endcsname
97 \the\numexpr\XINTdigitsormax+4.\simXINTinFloatDiv_wopt.%
98 \def\XINT_tmpa#1#2#3.#4.%
99 {%
100 \let #1#2%
101 \def #2##1##2##3{##1##2{{\simexpanded{\simunexpanded{#4[#3]}\simexpandafter}}##3}}}%
102}%
103 \expandafter\XINT_tmpa
104 \csname XINT_flexpr_sqrfunc\expandafter\endcsname
105 \csname XINT_flexpr_func_sqr\expandafter\endcsname
106 \the\numexpr\XINTdigitsormax+4.\simXINTinFloatSqr_wopt.%
107 \expandafter\XINT_tmpa
108 \csname XINT_flexpr_sqrtfunc\expandafter\endcsname
109 \csname XINT_flexpr_func_sqrt\expandafter\endcsname
110 \the\numexpr\XINTdigitsormax+4.\simXINTinFloatSqrt.%
111 \expandafter\XINT_tmpa
112 \csname XINT_flexpr_invfunc\expandafter\endcsname
113 \csname XINT_flexpr_func_inv\expandafter\endcsname
114 \the\numexpr\XINTdigitsormax+4.\simXINTinFloatInv_wopt.%
115 \catcode`~ 3

```

12.7 The sine and cosine series

Old pending question: 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.

Update at last minute: this is actually exactly what I do if Digits is at most 8.

Small values of the variable are very badly handled here because a much shorter truncation of the series should be used.

At 1.4e the original `\xintdeffloatfunc` was converted into macros, whose principle can be seen also at work in `xintlog.sty`. We prepare the input variables with shorter and shorter mantissas for usage deep in the series.

This divided by about 3 the execution cost of the series for P about 60.

Originally, the thresholds were computed a priori with 0.79 as upper bound of the variable, but then for 1.4e I developed enough test files to try to adjust heuristically with a target of say 99,5% of correct rounding, and always at most 1ulp error. The numerical analysis is not easy due to the complications of the implementation...

Also, random testing never explores the weak spots...

The 0.79 (a bit more than $\pi/4$) upper bound induces a costly check of variable on input, if Digits is big. Much faster would be to check if input is less than 10 degrees or 1 radian as done in `xfp`. But using enough coefficients for allowing up to 1 radian, which is without pain for Digits=16 starts being annoying for higher values such as Digits=48.

But the main reason I don't do it now is that I spend too much time fine-tuning the table of thresholds... maybe in next release.

12.7.1 Support macros for the sine and cosine series

Computing the $1/n!$ from $n!$ then inverting would require costly divisions and significantly increase the loading time.

So a method is employed to simply divide by $2k(2k-1)$ or $(2k+1)(2k)$ step by step, with what we hope are enough 8 security digits, and reducing the sizes of the mantissas at each step.

This whole section is conditional on Digits being at least nine.

```

116 \ifnum\XINTdigits>8
117 \edef\XINT_tmpG % 1/3!
118 {1\xintReplicate{\XINTdigitsormax+2}{6}7[\the\numexpr-\XINTdigitsormax-4]]}%
119 \edef\XINT_tmpH % 1/5!
120 {8\xintReplicate{\XINTdigitsormax+1}{3}[\the\numexpr-\XINTdigitsormax-4]]}%
121 \edef\XINT_tmpd % 1/5!
122 {8\xintReplicate{\XINTdigitsormax+9}{3}[\the\numexpr-\XINTdigitsormax-12]]}%
123 \def\XINT_tmpe#1.#2.#3.#4.#5#6#7%
124 {%
125 \def#5##1\xint:
126 {%
127 \expandafter#6\romannumeral0\XINTinfloatS[#2]{##1}\xint:##1\xint:
128 }%
129 \def#6##1\xint:
130 {%
131 \expandafter#7\romannumeral0\xintsub{#4}{\XINTinfloat[#2]{\xintMul{#3}{##1}}}\xint:
132 }%
133 \def#7##1\xint:##2\xint:
134 {%
135 \xintSub{1/1[0]}{\XINTinfloat[#1]{\xintMul{##1}{##2}}}%
136 }%
137 }%
138 \expandafter\XINT_tmpe
139 \the\numexpr\XINTdigitsormax+4\expandafter.%
140 \the\numexpr\XINTdigitsormax+2\expandafter.\expanded{%
141 \XINT_tmpH.% 1/5!
142 \XINT_tmpG.% 1/3!
143 \expandafter}%
144 \csname XINT_SinAux_series_a_iii\expandafter\endcsname
145 \csname XINT_SinAux_series_b\expandafter\endcsname
146 \csname XINT_SinAux_series_c_i\endcsname
147 \def\XINT_tmpe #1 #2 #3 #4 #5 #6 #7 #8 %
148 {%
149 \def\XINT_tmpe ##1##2##3##4##5%
150 {%
151 \def\XINT_tmpe####1.####2.####3.####4.####5.%
152 {%
153 \def##1#####1\xint:
154 {%
155 \expandafter##2%
156 \romannumeral0\XINTinfloatS[####1]{#####1}\xint:#####1\xint:
157 }%

```

```

158 \def##2#####1\xint:
159 {%
160     \expandafter##3%
161     \romannumeral0\XINTinfloatS[####2]{#####1}\xint:#####1\xint:
162 }%
163 \def##3#####1\xint:
164 {%
165     \expandafter##4%
166     \romannumeral0\xintsub{####4}{\XINTinFloat[####2]{\xintMul{####3}{#####1}}}\xint:
167 }%
168 \def##4#####1\xint:#####2\xint:
169 {%
170     \expandafter##5%
171     \romannumeral0\xintsub{####5}{\XINTinFloat[####1]{\xintMul{#####1}{#####2}}}\xint:
172 }%
173 }%
174 }%
175 \expandafter\XINT_tmpb
176 \csname XINT_#8Aux_series_a_\romannumeral\numexpr#1-1\expandafter\endcsname
177 \csname XINT_#8Aux_series_a_\romannumeral\numexpr#1\expandafter\endcsname
178 \csname XINT_#8Aux_series_b_\expandafter\endcsname
179 \csname XINT_#8Aux_series_c_\romannumeral\numexpr#1-2\expandafter\endcsname
180 \csname XINT_#8Aux_series_c_\romannumeral\numexpr#1-3\endcsname
181 \edef\XINT_tmpd
182 {\XINTinFloat[\XINTdigitsormax-#2+8]{\xintDiv{\XINT_tmpd}{\the\numexpr#5*(#5-1)\relax}}}%
183 \let\XINT_tmpF\XINT_tmpG
184 \let\XINT_tmpG\XINT_tmpH
185 \edef\XINT_tmpH{\XINTinFloat[\XINTdigitsormax-#2]{\XINT_tmpd}}%
186 \expandafter\XINT_tmpe
187 \the\numexpr\XINTdigitsormax-#3\expandafter.%
188 \the\numexpr\XINTdigitsormax-#2\expandafter.\expanded{%
189 \XINT_tmpH.%
190 \XINT_tmpG.%
191 \XINT_tmpF.%
192 }%
193 }%
194 \XINT_tmpe 4 -1 -2 -4 7 5 3 Sin %
195 \ifnum\XINTdigits>3 \XINT_tmpe 5 1 -1 -2 9 7 5 Sin \fi
196 \ifnum\XINTdigits>5 \XINT_tmpe 6 3 1 -1 11 9 7 Sin \fi
197 \ifnum\XINTdigits>8 \XINT_tmpe 7 6 3 1 13 11 9 Sin \fi
198 \ifnum\XINTdigits>11 \XINT_tmpe 8 9 6 3 15 13 11 Sin \fi
199 \ifnum\XINTdigits>14 \XINT_tmpe 9 12 9 6 17 15 13 Sin \fi
200 \ifnum\XINTdigits>16 \XINT_tmpe 10 14 12 9 19 17 15 Sin \fi
201 \ifnum\XINTdigits>19 \XINT_tmpe 11 17 14 12 21 19 17 Sin \fi
202 \ifnum\XINTdigits>22 \XINT_tmpe 12 20 17 14 23 21 19 Sin \fi
203 \ifnum\XINTdigits>25 \XINT_tmpe 13 23 20 17 25 23 21 Sin \fi
204 \ifnum\XINTdigits>28 \XINT_tmpe 14 26 23 20 27 25 23 Sin \fi
205 \ifnum\XINTdigits>31 \XINT_tmpe 15 29 26 23 29 27 25 Sin \fi
206 \ifnum\XINTdigits>34 \XINT_tmpe 16 32 29 26 31 29 27 Sin \fi
207 \ifnum\XINTdigits>37 \XINT_tmpe 17 35 32 29 33 31 29 Sin \fi
208 \ifnum\XINTdigits>40 \XINT_tmpe 18 38 35 32 35 33 31 Sin \fi
209 \ifnum\XINTdigits>44 \XINT_tmpe 19 42 38 35 37 35 33 Sin \fi

```



```

210 \ifnum\XINTdigits>47 \XINT_tmpa 20 45 42 38 39 37 35 Sin \fi
211 \ifnum\XINTdigits>51 \XINT_tmpa 21 49 45 42 41 39 37 Sin \fi
212 \ifnum\XINTdigits>55 \XINT_tmpa 22 53 49 45 43 41 39 Sin \fi
213 \ifnum\XINTdigits>58 \XINT_tmpa 23 56 53 49 45 43 41 Sin \fi
214 \edef\XINT_tmpd % 1/4!
215 {41\xintReplicate{\XINTdigitsormax+8}{6}7[\the\numexpr-\XINTdigitsormax-12]]}%
216 \edef\XINT_tmph % 1/4!
217 {41\xintReplicate{\XINTdigitsormax}{6}7[\the\numexpr-\XINTdigitsormax-4]]}%
218 \def\XINT_tmpG{5[-1]]}% 1/2!
219 \expandafter\XINT_tmpe
220 \the\numexpr\XINTdigitsormax+4\expandafter.%
221 \the\numexpr\XINTdigitsormax+3\expandafter.\expanded{%
222 \XINT_tmph.%
223 \XINT_tmpG.%
224 \expandafter}%
225 \csname XINT_CosAux_series_a_iii\expandafter\endcsname
226 \csname XINT_CosAux_series_b\expandafter\endcsname
227 \csname XINT_CosAux_series_c_i\endcsname
228 \XINT_tmpa 4 -2 -3 -4 6 4 2 Cos %
229 \ifnum\XINTdigits>2 \XINT_tmpa 5 0 -2 -3 8 6 4 Cos \fi
230 \ifnum\XINTdigits>4 \XINT_tmpa 6 2 0 -2 10 8 6 Cos \fi
231 \ifnum\XINTdigits>7 \XINT_tmpa 7 5 2 0 12 10 8 Cos \fi
232 \ifnum\XINTdigits>9 \XINT_tmpa 8 7 5 2 14 12 10 Cos \fi
233 \ifnum\XINTdigits>12 \XINT_tmpa 9 10 7 5 16 14 12 Cos \fi
234 \ifnum\XINTdigits>15 \XINT_tmpa 10 13 10 7 18 16 14 Cos \fi
235 \ifnum\XINTdigits>18 \XINT_tmpa 11 16 13 10 20 18 16 Cos \fi
236 \ifnum\XINTdigits>20 \XINT_tmpa 12 18 16 13 22 20 18 Cos \fi
237 \ifnum\XINTdigits>24 \XINT_tmpa 13 22 18 16 24 22 20 Cos \fi
238 \ifnum\XINTdigits>27 \XINT_tmpa 14 25 22 18 26 24 22 Cos \fi
239 \ifnum\XINTdigits>30 \XINT_tmpa 15 28 25 22 28 26 24 Cos \fi
240 \ifnum\XINTdigits>33 \XINT_tmpa 16 31 28 25 30 28 26 Cos \fi
241 \ifnum\XINTdigits>36 \XINT_tmpa 17 34 31 28 32 30 28 Cos \fi
242 \ifnum\XINTdigits>39 \XINT_tmpa 18 37 34 31 34 32 30 Cos \fi
243 \ifnum\XINTdigits>42 \XINT_tmpa 19 40 37 34 36 34 32 Cos \fi
244 \ifnum\XINTdigits>45 \XINT_tmpa 20 43 40 37 38 36 34 Cos \fi
245 \ifnum\XINTdigits>49 \XINT_tmpa 21 47 43 40 40 38 36 Cos \fi
246 \ifnum\XINTdigits>53 \XINT_tmpa 22 51 47 43 42 40 38 Cos \fi
247 \ifnum\XINTdigits>57 \XINT_tmpa 23 55 51 47 44 42 40 Cos \fi
248 \ifnum\XINTdigits>60 \XINT_tmpa 24 58 55 51 46 44 42 Cos \fi
249 \let\XINT_tmph\xint_undefined\let\XINT_tmpG\xint_undefined\let\XINT_tmpF\xint_undefined
250 \let\XINT_tmpd\xint_undefined\let\XINT_tmpe\xint_undefined
251 \def\XINT_SinAux_series#1%
252 {%
253 \expandafter\XINT_SinAux_series_a_iii
254 \romannumeral0\XINTinfloatS[\XINTdigitsormax+4]{#1}\xint:
255 }%
256 \def\XINT_CosAux_series#1%
257 {%
258 \expandafter\XINT_CosAux_series_a_iii
259 \romannumeral0\XINTinfloatS[\XINTdigitsormax+4]{#1}\xint:
260 }%
261 \fi % end of \XINTdigits>8

```

12.7.2 The poor man approximate but speedier approach for Digits at most 8

```

262 \ifnum\XINTdigits<9
263 \def\XINT_SinAux_series#1%
264 {%
265     \the\numexpr\expandafter\XINT_SinAux_b\romannumeral0\xintiround9{#1}.[ -9]%
266 }%
267 \def\XINT_SinAux_b#1.%
268 {%
269     ((((((((((%(\xint_c_x^ix/-210)
270     -4761905*#1/\xint_c_x^ix+\xint_c_x^ix)/%
271     -156)*#1/\xint_c_x^ix+\xint_c_x^ix)/%
272     -110)*#1/\xint_c_x^ix+\xint_c_x^ix)/%
273     -72)*#1/\xint_c_x^ix+\xint_c_x^ix)/%
274     -42)*#1/\xint_c_x^ix+\xint_c_x^ix)/%
275     -20)*#1/\xint_c_x^ix+\xint_c_x^ix)/%
276     -6)*#1/\xint_c_x^ix+\xint_c_x^ix
277 }%
278 \def\XINT_CosAux_series#1%
279 {%
280     \the\numexpr\expandafter\XINT_CosAux_b\romannumeral0\xintiround9{#1}.[ -9]%
281 }%
282 \def\XINT_CosAux_b#1.%
283 {%
284     ((((((((((%(\xint_c_x^ix/-240)
285     -4166667*#1/\xint_c_x^ix+\xint_c_x^ix)/%
286     -182)*#1/\xint_c_x^ix+\xint_c_x^ix)/%
287     -132)*#1/\xint_c_x^ix+\xint_c_x^ix)/%
288     -90)*#1/\xint_c_x^ix+\xint_c_x^ix)/%
289     -56)*#1/\xint_c_x^ix+\xint_c_x^ix)/%
290     -30)*#1/\xint_c_x^ix+\xint_c_x^ix)/%
291     -12)*#1/\xint_c_x^ix+\xint_c_x^ix)/%
292     -2)*#1/\xint_c_x^ix+\xint_c_x^ix
293 }%
294 \fi

```

12.7.3 Declarations of the @sin_aux() and @cos_aux() functions

```

295 \def\XINT_flexpr_func_@sin_aux#1#2#3%
296 {%
297     \expandafter #1\expandafter #2\expandafter{%
298     \romannumeral`&&\XINT:NEhook:f:one:from:one
299     {\romannumeral`&&\XINT_SinAux_series#3}}%
300 }%
301 \def\XINT_flexpr_func_@cos_aux#1#2#3%
302 {%
303     \expandafter #1\expandafter #2\expandafter{%
304     \romannumeral`&&\XINT:NEhook:f:one:from:one
305     {\romannumeral`&&\XINT_CosAux_series#3}}%
306 }%

```

12.7.4 @sin_series(), @cos_series()

```

307 \xintdeffloatfunc @sin_series(x) := x * @sin_aux(sqr(x));%

```

```
308 \xintdeffloatfunc @cos_series(x) := @cos_aux(sqr(x));%
```

12.8 Range reduction for sine and cosine using degrees

As commented in the package introduction, Range reduction is a demanding domain and we handle it semi-satisfactorily. The main problem is that in January 2019 I had done only support for degrees, and when I added radians I used the most naive approach. But one can find worse: in 2019 I was surprised to observe important divergences with Maple's results 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 even though the approach here is still naive, it behaves much better.

The `@sind_rr()` and `@cosd_rr()` sine and cosine "doing range reduction" are coded directly at macro level via `\xintSind` and `\xintCosd` which will dispatch to usage of the sine or cosine series, depending on case.

Old note from 2019: attention that `\xintSind` and `\xintCosd` must be used with a positive argument.

We start with an auxiliary macro to reduce modulo 360 quickly.

12.8.1 Low level modulo 360 helper 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 that 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.

```
309 \def\XINT_mod_ccclx_i #1.%
310 {%
311     \expandafter\XINT_mod_ccclx_e\the\numexpr
312     \expandafter\XINT_mod_ccclx_j\the\numexpr1\ifcase#1 \or0\or00\else000\fi.%
313 }%
314 \def\XINT_mod_ccclx_j #1.#2.%
315 {%
316     (\XINT_mod_ccclx_ja {++}#2#1\XINT_mod_ccclx_jb 0000000\relax
317 }%
318 \def\XINT_mod_ccclx_ja #1#2#3#4#5#6#7#8#9%
319 {%
320     #9+#8+#7+#6+#5+#4+#3+#2\xint_firstoftwo{+\XINT_mod_ccclx_ja{+#9+#8+#7}}{#1}%
321 }%
322 \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 ...

```
323 \def\XINT_mod_ccclx_jc  +#1+#2+#3#4\relax{+80*(#3+#2+#1)+#3#2#1.}%
324 \def\XINT_mod_ccclx_e#1.{\expandafter\XINT_mod_ccclx_z\the\numexpr(#1+180)/360-1.#1.}%
325 \def\XINT_mod_ccclx_z#1.#2.{#2-360*#1.}%

```

12.8.2 `@sind_rr()` function and its support macro `\xintSind`

```
326 \def\XINT_flexpr_func@sind_rr #1#2#3%
327 {%
328     \expandafter #1\expandafter #2\expandafter{%
329     \romannumeral`&&\XINT:NEhook:f:one:from:one{\romannumeral`&&\xintSind#3}}%
330 }%

```

old comment: Must be f-expandable for nesting macros from \xintNewExpr

This is where the prize of using the same macros for two distinct use cases has serious disadvantages. The reason of Digits+12 is only to support an input which contains a multiplication by @oneRadian with its extended digits.

Then we do a somewhat strange truncation to a fixed point of fractional digits, which is ok in the "Degrees" case, but causes issues of its own in the "Radians" case. Please consider this whole thing as marked for future improvement, when times allows.

ATTENTION \xintSind ONLY FOR POSITIVE ARGUMENTS

```

331 \def\xINT_tmpa #1.{%
332 \def\xintSind##1%
333 {%
334     \romannumeral\&\&\expandafter\xintsind\romannumeral0\xINTinfloatS[#1]{##1}}%
335 }\expandafter\xINT_tmpa\the\numexpr\xINTdigitsormax+12.%
336 \def\xintsind #1[#2#3]%
337 {%
338     \xint_UDsignfork
339     #2\xINT_sind
340     -\xINT_sind_int
341     \krof#2#3.#1..%
342 }%
343 \def\xINT_tmpa #1.{%
344 \def\xINT_sind ##1.##2.%
345 {%
346     \expandafter\xINT_sind_a
347     \romannumeral0\xinttrunc{#1}{##2[##1]]}%
348 }%
349 }\expandafter\xINT_tmpa\the\numexpr\xINTdigitsormax+5.%
350 \def\xINT_sind_a{\expandafter\xINT_sind_i\the\numexpr\xINT_mod_ccclx_i0.}%
351 \def\xINT_sind_int
352 {%
353     \expandafter\xINT_sind_i\the\numexpr\expandafter\xINT_mod_ccclx_i
354 }%
355 \def\xINT_sind_i #1.%
356 {%
357     \ifcase\numexpr#1/90\relax
358     \expandafter\xINT_sind_A
359     \or\expandafter\xINT_sind_B\the\numexpr-90+%
360     \or\expandafter\xINT_sind_C\the\numexpr-180+%
361     \or\expandafter\xINT_sind_D\the\numexpr-270+%
362     \else\expandafter\xINT_sind_E\the\numexpr-360+%
363     \fi#1.%
364 }%
365 \def\xINT_tmpa #1.#2.{%
366 \def\xINT_sind_A##1.##2.%
367 {%
368     \XINT_expr_unlock\expandafter\xINT_flexpr_userfunc_@sin_series\expandafter
369     {\romannumeral0\xINTinfloat[#1]{\xintMul{##1.##2}#2}}%
370 }%
371 \def\xINT_sind_B_n-##1.##2.%
372 {%
373     \XINT_expr_unlock\expandafter\xINT_flexpr_userfunc_@cos_series\expandafter
374     {\romannumeral0\xINTinfloat[#1]{\xintMul{\xintSub{##1[0]}{.##2}}#2}}%

```

```

375 }%
376 \def\XINT_sind_B_p##1.##2.%
377 {%
378     \XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_@cos_series\expandafter
379     {\romannumeral0\XINTinfloat[#1]{\xintMul{##1.##2}#2}}%
380 }%
381 \def\XINT_sind_C_n-##1.##2.%
382 {%
383     \XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_@sin_series\expandafter
384     {\romannumeral0\XINTinfloat[#1]{\xintMul{\xintSub{##1[0]}{.##2}}#2}}%
385 }%
386 \def\XINT_sind_C_p##1.##2.%
387 {%
388     \xintiopp\XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_@sin_series\expandafter
389     {\romannumeral0\XINTinfloat[#1]{\xintMul{##1.##2}#2}}%
390 }%
391 \def\XINT_sind_D_n-##1.##2.%
392 {%
393     \xintiopp\XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_@cos_series\expandafter
394     {\romannumeral0\XINTinfloat[#1]{\xintMul{\xintSub{##1[0]}{.##2}}#2}}%
395 }%
396 \def\XINT_sind_D_p##1.##2.%
397 {%
398     \xintiopp\XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_@cos_series\expandafter
399     {\romannumeral0\XINTinfloat[#1]{\xintMul{##1.##2}#2}}%
400 }%
401 \def\XINT_sind_E-##1.##2.%
402 {%
403     \xintiopp\XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_@sin_series\expandafter
404     {\romannumeral0\XINTinfloat[#1]{\xintMul{\xintSub{##1[0]}{.##2}}#2}}%
405 }%
406 }\expandafter\XINT_tmpa
407     \the\numexpr\XINTdigitsormax+4\expandafter.%
408     \romannumeral`&&\xintbarefloateval @oneDegree\relax.%
409 \def\XINT_sind_B#1{\xint_UDsignfork#1\XINT_sind_B_n-\XINT_sind_B_p\krof #1}%
410 \def\XINT_sind_C#1{\xint_UDsignfork#1\XINT_sind_C_n-\XINT_sind_C_p\krof #1}%
411 \def\XINT_sind_D#1{\xint_UDsignfork#1\XINT_sind_D_n-\XINT_sind_D_p\krof #1}%

```

12.8.3 @cosd_rr() function and its support macro \xintCosd

```

412 \def\XINT_flexpr_func_@cosd_rr #1#2#3%
413 {%
414     \expandafter #1\expandafter #2\expandafter{%
415     \romannumeral`&&\XINT:NEhook:f:one:from:one{\romannumeral`&&\xintCosd#3}}%
416 }%

```

ATTENTION ONLY FOR POSITIVE ARGUMENTS

```

417 \def\XINT_tmpa #1.{%
418 \def\xintCosd##1%
419 {%
420     \romannumeral`&&\expandafter\xintcosd\romannumeral0\XINTinfloatS[#1]{##1}}%
421 }\expandafter\XINT_tmpa\the\numexpr\XINTdigitsormax+12.%
422 \def\xintcosd #1[#2#3]%

```

```

423 {%
424     \xint_UDsignfork
425     #2\XINT_cosd
426     -\XINT_cosd_int
427     \krof#2#3.#1..%
428 }%
429 \def\XINT_tmpa #1.{%
430 \def\XINT_cosd ##1.##2.%
431 {%
432     \expandafter\XINT_cosd_a
433     \romannumeral0\xinttrunc{#1}{##2[##1]}%
434 }%
435 }\expandafter\XINT_tmpa\the\numexpr\XINTdigitsormax+5.%
436 \def\XINT_cosd_a{\expandafter\XINT_cosd_i\the\numexpr\XINT_mod_ccclx_i0.}%
437 \def\XINT_cosd_int
438 {%
439     \expandafter\XINT_cosd_i\the\numexpr\expandafter\XINT_mod_ccclx_i
440 }%
441 \def\XINT_cosd_i #1.%
442 {%
443     \ifcase\numexpr#1/90\relax
444     \expandafter\XINT_cosd_A
445     \or\expandafter\XINT_cosd_B\the\numexpr-90+%
446     \or\expandafter\XINT_cosd_C\the\numexpr-180+%
447     \or\expandafter\XINT_cosd_D\the\numexpr-270+%
448     \else\expandafter\XINT_cosd_E\the\numexpr-360+%
449     \fi#1.%
450 }%

```

#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.

```

451 \def\XINT_tmpa#1.#2.{%
452 \def\XINT_cosd_A##1.##2.%
453 {%
454     \XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_@cos_series\expandafter
455     {\romannumeral0\XINTinfloat[#1]{\xintMul{##1.##2}#2}}%
456 }%
457 \def\XINT_cosd_B_n-##1.##2.%
458 {%
459     \XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_@sin_series\expandafter
460     {\romannumeral0\XINTinfloat[#1]{\xintMul{\xintSub{##1[0]}{.##2}}#2}}%
461 }%
462 \def\XINT_cosd_B_p##1.##2.%
463 {%
464     \xintiopp\XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_@sin_series\expandafter
465     {\romannumeral0\XINTinfloat[#1]{\xintMul{##1.##2}#2}}%
466 }%
467 \def\XINT_cosd_C_n-##1.##2.%
468 {%
469     \xintiopp\XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_@cos_series\expandafter
470     {\romannumeral0\XINTinfloat[#1]{\xintMul{\xintSub{##1[0]}{.##2}}#2}}%
471 }%
472 \def\XINT_cosd_C_p##1.##2.%

```

```

473 {%
474     \xintiopp\XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_@cos_series\expandafter
475         {\romannumeral0\XINTinfloat[#1]{\xintMul{##1.##2}#2}}}%
476 }%
477 \def\XINT_cosd_D_n-##1.##2.%
478 {%
479     \xintiopp\XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_@sin_series\expandafter
480         {\romannumeral0\XINTinfloat[#1]{\xintMul{\xintSub{##1[0]}.{##2}}#2}}}%
481 }%
482 \def\XINT_cosd_D_p##1.##2.%
483 {%
484     \XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_@sin_series\expandafter
485         {\romannumeral0\XINTinfloat[#1]{\xintMul{##1.##2}#2}}}%
486 }%
487 \def\XINT_cosd_E-##1.##2.%
488 {%
489     \XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_@cos_series\expandafter
490         {\romannumeral0\XINTinfloat[#1]{\xintMul{\xintSub{##1[0]}.{##2}}#2}}}%
491 }%
492 }\expandafter\XINT_tmpa
493     \the\numexpr\XINTdigitsormax+4\expandafter.%
494     \romannumeral`&&\xintbarefloateval @oneDegree\relax.%
495 \def\XINT_cosd_B#1{\xint_UDsignfork#1\XINT_cosd_B_n-\XINT_cosd_B_p\krof #1}%
496 \def\XINT_cosd_C#1{\xint_UDsignfork#1\XINT_cosd_C_n-\XINT_cosd_C_p\krof #1}%
497 \def\XINT_cosd_D#1{\xint_UDsignfork#1\XINT_cosd_D_n-\XINT_cosd_D_p\krof #1}%

```

12.9 @sind(), @cosd()

The -45 is stored internally as -45/1[0] from the action of the unary minus operator, which float macros then parse faster. The 45e0 is to let it become 45[0] and not simply 45.

Here and below the \ifnum\XINTdigits>8 45\else60\fi will all be resolved at time of definition. This is the charm and power of expandable parsers!

```

498 \xintdeffloatfunc @sind(x) := (x)??
499                                     {(x>=-\ifnum\XINTdigits>8 45\else60\fi)?
500                                     {@sin_series(x*@oneDegree)}
501                                     {@sind_rr(-x)}
502                                     }
503                                     {0e0}
504                                     {(x<=-\ifnum\XINTdigits>8 45\else60\fi e0)?
505                                     {@sin_series(x*@oneDegree)}
506                                     {@sind_rr(x)}
507                                     }
508                                     ;%
509 \xintdeffloatfunc @cosd(x) := (x)??
510                                     {(x>=-\ifnum\XINTdigits>8 45\else60\fi)?
511                                     {@cos_series(x*@oneDegree)}
512                                     {@cosd_rr(-x)}
513                                     }
514                                     {1e0}
515                                     {(x<=-\ifnum\XINTdigits>8 45\else60\fi e0)?
516                                     {@cos_series(x*@oneDegree)}
517                                     {@cosd_rr(x)}

```

```
518             }
519             ;%
```

12.10 @sin(), @cos()

For some reason I did not define sin() and cos() in January 2019 ??

The sub \xintexpr x*@oneRadian\relax means that the multiplication will be done exactly @oneRadian having its 12 extra digits (and x its 4 extra digits), before being rounded in entrance of \xintSind, respectively \xintCosd, to P+12 mantissa.

The strange 79e-2 could be 0.79 which would give 79[-2] internally too.

```
520 \xintdeffloatfunc @sin(x):= (abs(x)<\ifnum\XINTdigits>8 79e-2\else1e0\fi)?
521     {@sin_series(x)}
522     {(x)}??
523     {-@sind_rr(-\xintexpr x*@oneRadian\relax)}
524     {0}
525     {@sind_rr(\xintexpr x*@oneRadian\relax)}
526     }
527     ;%
528 \xintdeffloatfunc @cos(x):= (abs(x)<\ifnum\XINTdigits>8 79e-2\else1e0\fi)?
529     {@cos_series(x)}
530     {@cosd_rr(abs(\xintexpr x*@oneRadian\relax))}
531     ;%
```

12.11 @sinc()

Should I also consider adding $(1-\cos(x))/(x^2/2)$? it is $\text{sinc}^2(x/2)$ but avoids a square.

```
532 \xintdeffloatfunc @sinc(x):= (abs(x)<\ifnum\XINTdigits>8 79e-2\else1e0\fi) ?
533     {@sin_aux(sqr(x))}
534     {@sind_rr(\xintexpr abs(x)*@oneRadian\relax)/abs(x)}
535     ;%
```

12.12 @tan(), @tand(), @cot(), @cotd()

The 0 in cot(x) is a dummy place holder. We don't have a notion of Inf yet.

```
536 \xintdeffloatfunc @tand(x):= @sind(x)/@cosd(x);%
537 \xintdeffloatfunc @cotd(x):= @cosd(x)/@sind(x);%
538 \xintdeffloatfunc @tan(x) := (x)?
539     {(x>-\ifnum\XINTdigits>8 79e-2\else1e0\fi)?
540     {@sin(x)/@cos(x)}
541     {-@cotd(\xintexpr 9e1+x*@oneRadian\relax)}
542     }
543     }
544     {0e0}
545     {(x<\ifnum\XINTdigits>8 79e-2\else1e0\fi)?
546     {@sin(x)/@cos(x)}
547     {@cotd(\xintexpr 9e1-x*@oneRadian\relax)}
548     }
549     ;%
550 \xintdeffloatfunc @cot(x) := (abs(x)<\ifnum\XINTdigits>8 79e-2\else1e0\fi)?
551     {@cos(x)/@sin(x)}
552     {(x)}??
```



```

553             {-@tand(\xintexpr9e1+x*@oneRadian\relax)}
554             {0}
555             {@tand(\xintexpr9e1-x*@oneRadian\relax)}
556         };%
```

12.13 @sec(), @secd(), @csc(), @cscd()

```

557 \xintdeffloatfunc @sec(x) := inv(@cos(x));%
558 \xintdeffloatfunc @csc(x) := inv(@sin(x));%
559 \xintdeffloatfunc @secd(x) := inv(@cosd(x));%
560 \xintdeffloatfunc @cscd(x) := inv(@sind(x));%
```

12.14 Core routine for inverse trigonometry

I always liked very much the general algorithm whose idea I found in 2019. But it costs a square root plus a sine plus a cosine all at target precision. For the arctangent the square root will be avoided by a trick. (memo: it is replaced by a division and I am not so sure now this is advantageous in fact)

And now I like it even more as I have re-done the first step entirely in a single \numexpr... Thus the inverse trigonometry got a serious improvement at 1.4e...

Here is the idea. We have $0 < t < \sqrt{2}/2$ and we want $a = \text{Arcsin } t$.

Imagine we have some very good approximation $b = a - h$. We know b , and don't know yet h . No problem h is $a - b$ so $\sin(h) = \sin(a)\cos(b) - \cos(a)\sin(b)$. And we know everything here: $\sin(a)$ is t , $\cos(a)$ is $u = \sqrt{1-t^2}$, and we can compute $\cos(b)$ and $\sin(b)$.

I said h was small so the computation of $\sin(a)\cos(b) - \cos(a)\sin(b)$ will involve a lot of cancellation, no problem with xint, as it knows how to compute exactly... and if we wanted to go very low level we could do $\cos(a)\sin(b)$ paying attention only on least significant digits.

Ok, so we have $\sin(h)$, but h is small, so the series of Arcsine can be used with few terms!

In fact h will be at most of the order of $1e-9$, so it is no problem to simply replace $\sin(h)$ with h if the target precision is 16 !

Ok, so how do we obtain b , the good approximation to $\text{Arcsin } t$? Simply by using its Taylor series, embedded in a single \numexpr working with nine digits numbers... I like this one! Notice that it reminisces with my questioning about how to best do Horner like for sine and cosine. Here in \numexpr we can only manipulate whole integers and simply can't do things such as $\dots *x + 5/112) *x + 3/40) *x + 1/6) *x + 1 \dots$. But I found another way, see the code, which uses extensively the "scaling" operations in \numexpr.

I have not proven rigorously that $b - a$ is always less or equal in absolute value than $1e-9$, but it is possible for example in Python to program it and go through all possible (less than) $1e9$ inputs and check what happens.

Very small inputs will give $b=0$ (first step is a fixed point rounding of t to nine fractional digits, so this rounding gives zero for input $< 0.5e-9$, others will give $b=t$, because the arcsine numexpr will end up with 1000000000 (last time I checked that was for t a bit less than $5e-5$, the latter gives 1000000001). All seems to work perfectly fine, in practice...

First we let the @sin_aux() and @cos_aux() functions be usable in exact \xintexpr context.

The @asin_II() function will be used only for Digits>16.

```

561 \expandafter\let\csname XINT_expr_func_@sin_aux\expandafter\endcsname
562       \csname XINT_flexpr_func_@sin_aux\endcsname
563 \expandafter\let\csname XINT_expr_func_@cos_aux\expandafter\endcsname
564       \csname XINT_flexpr_func_@cos_aux\endcsname
565 \ifnum\XINTdigits>16
566 \def\XINT_flexpr_func_@asin_II#1#2#3%
567 {%
```

```

568 \expandafter #1\expandafter #2\expandafter{%
569 \romannumeral`&&\XINT:NEhook:f:one:from:one
570 {\romannumeral`&&\XINT_Arcsin_II_a#3}}%
571 }%
572 \def\XINT_tmpc#1.%
573 {%
574 \def\XINT_Arcsin_II_a##1%
575 {%
576 \expandafter\XINT_Arcsin_II_c_i\romannumeral0\XINTinfloatS[#1]{##1}%
577 }%
578 \def\XINT_Arcsin_II_c_i##1[##2]%
579 {%
580 \xintAdd{1/1[0]}{##1/6[##2]}%
581 }%
582 }%
583 \expandafter\XINT_tmpc\the\numexpr\XINTdigitsormax-14.%
584 \fi
585 \ifnum\XINTdigits>34
586 \def\XINT_tmpc#1.#2.#3.#4.%
587 {%
588 \def\XINT_Arcsin_II_a##1%
589 {%
590 \expandafter\XINT_Arcsin_II_a_iii\romannumeral0\XINTinfloatS[#1]{##1}\xint:
591 }%
592 \def\XINT_Arcsin_II_a_iii##1\xint:
593 {%
594 \expandafter\XINT_Arcsin_II_b\romannumeral0\XINTinfloatS[#2]{##1}\xint:##1\xint:
595 }%
596 \def\XINT_Arcsin_II_b##1\xint:
597 {%
598 \expandafter\XINT_Arcsin_II_c_i\romannumeral0\xintadd{#4}{\XINTinFloat[#2]{\xintMul{#3}{##1}}}\xint:
599 }%
600 \def\XINT_Arcsin_II_c_i##1\xint:##2\xint:
601 {%
602 \xintAdd{1/1[0]}{\XINTinFloat[#1]{\xintMul{##1}{##2}}}%
603 }%
604 }%
605 \expandafter\XINT_tmpc
606 \the\numexpr\XINTdigitsormax-14\expandafter.%
607 \the\numexpr\XINTdigitsormax-32\expandafter.\expanded{%
608 \XINTinFloat[\XINTdigitsormax-32]{3/40[0]}.%
609 \XINTinFloat[\XINTdigitsormax-14]{1/6[0]}.%
610 }%
611 \fi
612 \ifnum\XINTdigits>52
613 \def\XINT_tmpc#1.#2.#3.#4.#5.%
614 {%
615 \def\XINT_Arcsin_II_a_iii##1\xint:
616 {%
617 \expandafter\XINT_Arcsin_II_a_iv\romannumeral0\XINTinfloatS[#1]{##1}\xint:##1\xint:
618 }%
619 \def\XINT_Arcsin_II_a_iv##1\xint:

```

```

620 {%
621   \expandafter\XINT_Arcsin_II_b\romannumeral0\XINTinfloatS[#2]{##1}\xint:##1\xint:
622 }%
623 \def\XINT_Arcsin_II_b##1\xint:
624 {%
625   \expandafter\XINT_Arcsin_II_c_ii
626   \romannumeral0\xintadd{#4}{\XINTinfloat[#2]{\xintMul{#3}{##1}}}\xint:
627 }%
628 \def\XINT_Arcsin_II_c_ii##1\xint:##2\xint:
629 {%
630   \expandafter\XINT_Arcsin_II_c_i
631   \romannumeral0\xintadd{#5}{\XINTinfloat[#1]{\xintMul{##1}{##2}}}\xint:
632 }%
633 }%
634 \expandafter\XINT_tmpc
635 \the\numexpr\XINTdigitsormax-32\expandafter.%
636 \the\numexpr\XINTdigitsormax-50\expandafter.\expanded{%
637 \XINTinfloat[\XINTdigitsormax-50]{5/112[0]}.%
638 \XINTinfloat[\XINTdigitsormax-32]{3/40[0]}.%
639 \XINTinfloat[\XINTdigitsormax-14]{1/6[0]}.%
640 }%
641 \fi
642 \def\XINT_flexpr_func_@asin_I#1#2#3%
643 {%
644   \expandafter #1\expandafter #2\expandafter{%
645     \romannumeral`&&\XINT:NEhook:f:one:from:one
646     {\romannumeral`&&\XINT_Arcsin_I#3}}%
647 }%
648 \def\XINT_Arcsin_I#1{\the\numexpr\expandafter\XINT_Arcsin_Ia\romannumeral0\xintiround9{#1}.}%
649 \def\XINT_Arcsin_Ia#1.%
650 {%
651   (\expandafter\XINT_Arcsin_Ib\the\numexpr#1*#1/\xint_c_x^ix.)%
652   #1/\xint_c_x^ix[-9]%
653 }%
654 \def\XINT_Arcsin_Ib#1.%
655 {%((((((((((((((((
656 % 3481/3660)*#1/\xint_c_x^ix+\xint_c_x^ix)%
657 % 3249/3422)*#1/\xint_c_x^ix+\xint_c_x^ix)%
658 % 3025/3192)*#1/\xint_c_x^ix+\xint_c_x^ix)%
659 % 2809/2970)*#1/\xint_c_x^ix+\xint_c_x^ix)%
660 % 2601/2756)*#1/\xint_c_x^ix+\xint_c_x^ix)%
661 % 2401/2550)*#1/\xint_c_x^ix+\xint_c_x^ix)%
662 % 2209/2352)*#1/\xint_c_x^ix+\xint_c_x^ix)%
663 % 2025/2162)*#1/\xint_c_x^ix+\xint_c_x^ix)%
664 (((((((((((((((((((((((((((((((((((((((((((((((((((((((
665 %(\xint_c_x^ix*1849/1980)*%
666 933838384*#1/\xint_c_x^ix+\xint_c_x^ix)%
667 1681/1806)*#1/\xint_c_x^ix+\xint_c_x^ix)%
668 1521/1640)*#1/\xint_c_x^ix+\xint_c_x^ix)%
669 1369/1482)*#1/\xint_c_x^ix+\xint_c_x^ix)%
670 1225/1332)*#1/\xint_c_x^ix+\xint_c_x^ix)%
671 1089/1190)*#1/\xint_c_x^ix+\xint_c_x^ix)%

```

```

672 961/1056)*#1/\xint_c_x^ix+\xint_c_x^ix)*%
673 841/930)*#1/\xint_c_x^ix+\xint_c_x^ix)*%
674 729/812)*#1/\xint_c_x^ix+\xint_c_x^ix)*%
675 625/702)*#1/\xint_c_x^ix+\xint_c_x^ix)*%
676 529/600)*#1/\xint_c_x^ix+\xint_c_x^ix)*%
677 441/506)*#1/\xint_c_x^ix+\xint_c_x^ix)*%
678 361/420)*#1/\xint_c_x^ix+\xint_c_x^ix)*%
679 289/342)*#1/\xint_c_x^ix+\xint_c_x^ix)*%
680 225/272)*#1/\xint_c_x^ix+\xint_c_x^ix)*%
681 169/210)*#1/\xint_c_x^ix+\xint_c_x^ix)*%
682 121/156)*#1/\xint_c_x^ix+\xint_c_x^ix)*%
683 81/110)*#1/\xint_c_x^ix+\xint_c_x^ix)*%
684 49/72)*#1/\xint_c_x^ix+\xint_c_x^ix)*%
685 25/42)*#1/\xint_c_x^ix+\xint_c_x^ix)*%
686 9/20)*#1/\xint_c_x^ix+\xint_c_x^ix)*%
687 1/6)*#1/\xint_c_x^ix+\xint_c_x^ix
688 }%
689 \ifnum\XINTdigits>16
690 \xintdeffloatfunc @asin_o(D, T) := T + D*@asin_II(sqr(D));%
691 \xintdeffloatfunc @asin_n(V, T, t, u) :=%
692     @asin_o(\xintexpr t*@cos_aux(V) - u*T*@sin_aux(V)\relax, T);%
693 \else
694 \xintdeffloatfunc @asin_n(V, T, t, u) :=%
695     \xintexpr t*@cos_aux(V) - u*T*@sin_aux(V)\relax + T;%
696 \fi
697 \xintdeffloatfunc @asin_m(T, t, u) := @asin_n(sqr(T), T, t, u);%
698 \xintdeffloatfunc @asin_l(t, u) := @asin_m(@asin_I(t), t, u);%

```

12.15 @asin(), @asind()

Only non-negative arguments t and u for $\text{asin}_a(t,u)$, and $\text{asind}_a(t,u)$.

```

699 \xintdeffloatfunc @asin_a(t, u) := (t<u)?
700     {@asin_l(t, u)}
701     {@Piover2 - @asin_l(u, t)}
702     ;%
703 \xintdeffloatfunc @asind_a(t, u):= (t<u)?
704     {@asin_l(t, u) * @oneRadian}
705     {9e1 - @asin_l(u, t) * @oneRadian}
706     ;%
707 \xintdeffloatfunc @asin(t) := (t)??
708     {-@asin_a(-t, sqrt(1e0-sqr(t)))}
709     {0e0}
710     {@asin_a(t, sqrt(1e0-sqr(t)))}
711     ;%
712 \xintdeffloatfunc @asind(t) := (t)??
713     {-@asind_a(-t, sqrt(1e0-sqr(t)))}
714     {0e0}
715     {@asind_a(t, sqrt(1e0-sqr(t)))}
716     ;%

```

12.16 @acos(), @acosd()

```
717 \xintdeffloatfunc @acos(t) := @Piover2 - @asin(t);%
718 \xintdeffloatfunc @acosd(t) := 9e1 - @asind(t);%
```

12.17 @atan(), @atand()

Uses same core routine `asin_l()` as for `asin()`, but avoiding a square-root extraction in preparing its arguments (to the cost of computing an inverse, rather).

radians

```
719 \xintdeffloatfunc @atan_b(t, w, z) := 5e-1 * (w < 0)?
720                                     {@Pi - @asin_a(2e0*z * t, -w*z)}
721                                     {@asin_a(2e0*z * t, w*z)}
722                                     ;%
723 \xintdeffloatfunc @atan_a(t, T) := @atan_b(t, 1e0-T, inv(1e0+T));%
724 \xintdeffloatfunc @atan(t) := (t)??
725                               {-@atan_a(-t, sqr(t))}
726                               {0}
727                               {@atan_a(t, sqr(t))}
728                               ;%
```

degrees

```
729 \xintdeffloatfunc @atand_b(t, w, z) := 5e-1 * (w < 0)?
730                                     {18e1 - @asind_a(2e0*z * t, -w*z)}
731                                     {@asind_a(2e0*z * t, w*z)}
732                                     ;%
733 \xintdeffloatfunc @atand_a(t, T) := @atand_b(t, 1e0-T, inv(1e0+T));%
734 \xintdeffloatfunc @atand(t) := (t)??
735                               {-@atand_a(-t, sqr(t))}
736                               {0}
737                               {@atand_a(t, sqr(t))}
738                               ;%
```

12.18 @Arg(), @atan2(), @Argd(), @atan2d(), @pArg(), @pArgd()

`Arg(x,y)` function from $-\pi$ (excluded) to $+\pi$ (included)

```
739 \xintdeffloatfunc @Arg(x, y) := (y > x)?
740                               {(y > -x)?
741                               {@Piover2 - @atan(x/y)}
742                               {(y < 0)?
743                               {-@Pi + @atan(y/x)}
744                               {@Pi + @atan(y/x)}
745                               }
746                               }
747                               {(y > -x)?
748                               {@atan(y/x)}
749                               {-@Piover2 + @atan(x/-y)}
750                               }
751                               ;%
```

`atan2(y,x) = Arg(x,y)` ... (some people have `atan2` with arguments reversed but the convention here seems the most often encountered)

```
752 \xintdeffloatfunc @atan2(y,x) := @Arg(x, y);%
```

Argd(x,y) function from -180 (excluded) to +180 (included)

```

753 \xintdeffloatfunc @Argd(x, y):= (y>x)?
754     {(y>-x)?
755         {9e1 - @atand(x/y)}
756         {(y<0)?
757             {-18e1 + @atand(y/x)}
758             {18e1 + @atand(y/x)}
759         }
760     }
761     {(y>-x)?
762         {@atand(y/x)}
763         {-9e1 + @atand(x/-y)}
764     }
765     ;%

```

atan2d(y,x) = Argd(x,y)

```

766 \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.

```

767 \xintdeffloatfunc @pArg(x, y):= (y>x)?
768     {(y>-x)?
769         {@Piover2 - @atan(x/y)}
770         {@Pi + @atan(y/x)}
771     }
772     {(y>-x)?
773         {(y<0)?
774             {@twoPi + @atan(y/x)}
775             {@atan(y/x)}
776         }
777         {@threePiover2 + @atan(x/-y)}
778     }
779     ;%

```

pArgd(x,y) function from 0 (included) to 360 (excluded)

```

780 \xintdeffloatfunc @pArgd(x, y):=(y>x)?
781     {(y>-x)?
782         {9e1 - @atan(x/y)*@oneRadian}
783         {18e1 + @atan(y/x)*@oneRadian}
784     }
785     {(y>-x)?
786         {(y<0e0)?
787             {36e1 + @atan(y/x)*@oneRadian}
788             {@atan(y/x)*@oneRadian}
789         }
790         {27e1 + @atan(x/-y)*@oneRadian}
791     }
792     ;%

```

12.19 Restore \xintdeffloatfunc to its normal state, with no extra digits

```

793 \expandafter\let
794     \csname XINT_flexpr_exec_+\expandafter\endcsname

```

```

795 \csname XINT_flexpr_exec_+\endcsname
796 \expandafter\let
797 \csname XINT_flexpr_exec_-\expandafter\endcsname
798 \csname XINT_flexpr_exec_-\endcsname
799 \expandafter\let
800 \csname XINT_flexpr_exec_+\expandafter\endcsname
801 \csname XINT_flexpr_exec_+\endcsname
802 \expandafter\let
803 \csname XINT_flexpr_exec_/\expandafter\endcsname
804 \csname XINT_flexpr_exec_/\endcsname
805 \expandafter\let
806 \csname XINT_flexpr_func_sqr\expandafter\endcsname
807 \csname XINT_flexpr_sqrfunc\endcsname
808 \expandafter\let
809 \csname XINT_flexpr_func_sqrt\expandafter\endcsname
810 \csname XINT_flexpr_sqrtfunc\endcsname
811 \expandafter\let
812 \csname XINT_flexpr_func_inv\expandafter\endcsname
813 \csname XINT_flexpr_invfunc\endcsname

```

12.20 Let the functions be known to the \xintexpr parser

We use here `float_dgtormax` which uses the smaller of `Digits` and 64.

```

814 \edef\XINTinFloatdigitsormax{\noexpand\XINTinFloat[\the\numexpr\XINTdigitsormax]}%
815 \edef\XINTinFloatSdigitsormax{\noexpand\XINTinFloatS[\the\numexpr\XINTdigitsormax]}%
816 \xintFor #1 in {sin, cos, tan, sec, csc, cot,
817               asin, acos, atan}\do
818 {%
819 \xintdeffloatfunc #1(x) := float_dgtormax(@#1(x));%
820 \xintdeffloatfunc #1d(x) := float_dgtormax(@#1d(x));%
821 \xintdeffunc #1(x) := float_dgtormax(\xintfloatexpr @#1(sfloat_dgtormax(x))\relax);%
822 \xintdeffunc #1d(x) := float_dgtormax(\xintfloatexpr @#1d(sfloat_dgtormax(x))\relax);%
823 }%
824 \xintFor #1 in {Arg, pArg, atan2}\do
825 {%
826 \xintdeffloatfunc #1(x, y) := float_dgtormax(@#1(x, y));%
827 \xintdeffloatfunc #1d(x, y) := float_dgtormax(@#1d(x, y));%
828 \xintdeffunc #1(x, y) := float_dgtormax(\xintfloatexpr @#1(sfloat_dgtormax(x), sfloat_dgtormax(y))\relax);%
829 \xintdeffunc #1d(x, y) := float_dgtormax(\xintfloatexpr @#1d(sfloat_dgtormax(x), sfloat_dgtormax(y))\relax);%
830 }%
831 \xintdeffloatfunc sinc(x) := float_dgtormax(@sinc(x));%
832 \xintdeffunc sinc(x) := float_dgtormax(\xintfloatexpr @sinc(sfloat_dgtormax(x))\relax);%

```

12.21 Synonyms: @tg(), @cotg()

These are my childhood notations and I am attached to them. In radians only, and for \xintfloateval only. We skip some overhead here by using a \let at core level.

```

833 \expandafter\let\csname XINT_flexpr_func_tg\expandafter\endcsname
834 \csname XINT_flexpr_func_tan\endcsname
835 \expandafter\let\csname XINT_flexpr_func_cotg\expandafter\endcsname
836 \csname XINT_flexpr_func_cot\endcsname

```

12.22 Final clean-up

Restore used dummy variables to their status prior to the package reloading. On first loading this is not needed, but I have not added a way to check here whether this a first loading or a re-loading.

```

837 \xintdefvar twoPi := float_dgtormax(@twoPi);%
838 \xintdefvar threePiover2 := float_dgtormax(@threePiover2);%
839 \xintdefvar Pi := float_dgtormax(@Pi);%
840 \xintdefvar Piover2 := float_dgtormax(@Piover2);%
841 \xintdefvar oneDegree := float_dgtormax(@oneDegree);%
842 \xintdefvar oneRadian := float_dgtormax(@oneRadian);%
843 \xintunassignvar{@twoPi}\xintunassignvar{@threePiover2}%
844 \xintunassignvar{@Pi}\xintunassignvar{@Piover2}%
845 \xintunassignvar{@oneRadian}\xintunassignvar{@oneDegree}%
846 \xintFor* #1 in {iDTVtuwxyzX}\do{\xintrestorevariable{#1}}%
847 \XINTtrigendinput%

```


13 Package **xintlog** implementation

Contents

13.1	Catcodes, ε -TeX and reload detection	458
13.2	Library identification	459
13.3	<code>\xintreloadxintlog</code>	459
13.4	Loading the <code>poormanlog</code> package	459
13.5	Macro layer on top of the <code>poormanlog</code> package	460
13.5.1	<code>\PoorManLogBaseTen</code> , <code>\PoorManLog</code>	460
13.5.2	<code>\PoorManPowerOfTen</code> , <code>\PoorManExp</code>	461
13.5.3	Removed: <code>\PoorManPower</code> , see <code>\XINTinFloatSciPow</code>	462
13.5.4	Made a no-op: <code>\poormanloghack</code>	462
13.6	Macro support for powers	462
13.6.1	<code>\XINTinFloatSciPow</code>	462
13.6.2	<code>\xintPow</code>	465
13.7	Macro support for <code>\xintexpr</code> and <code>\xintfloatexpr</code> syntax	466
13.7.1	The <code>log10()</code> and <code>pow10()</code> functions	466
13.7.2	The <code>log()</code> , <code>exp()</code> functions	467
13.7.3	The <code>pow()</code> function	468
13.8	End of package loading for low Digits	468
13.9	Stored constants	468
13.10	April 2021: at last, <code>\XINTinFloatPowTen</code> , <code>\XINTinFloatExp</code>	471
13.10.1	Exponential series	475
13.11	April 2021: at last <code>\XINTinFloagLogTen</code> , <code>\XINTinFloatLog</code>	477
13.11.1	Log series, case II	483
13.11.2	Log series, case III	488

In 2019, at 1.3e release I almost included extended precision for `log()` and `exp()` but the time I could devote to `xint` expired. Finally, at long last, (and I had procrastinated far more than the two years since 2019) the 1.4e release in April 2021 brings `log10()`, `pow10()`, `log()`, `pow()` to $P=\text{Digits}$ precision: up to 62 digits with at least (said roughly) 99% chances of correct rounding (the design is targeting less than about 0.005ulp distance to mathematical value, before rounding).

Implementation is EXPERIMENTAL.

For up to $\text{Digits}=8$, it is simply based upon the `poormanlog` package. The probability of correct rounding will be less than for $\text{Digits}>8$, especially in the cases of $\text{Digits}=8$ and to a lesser extent $\text{Digits}=7$. And, for all $\text{Digits}\leq 8$, there is a systematic loss of rounding precision in the floating point sense in the case of `log10(x)` for inputs close to 1:

Summary of limitations of `log10()` and `pow10()` in the case of $\text{Digits}\leq 8$:

- For `log10(x)` with x near 1, the precision of output as floating point will be mechanically reduced from the fact that this is based on a fixed point result, for example `log10(1.0011871)` is produced as 5.15245e-4, which stands for 0.000515145 having indeed 9 correct fractional digits, but only 6 correct digits in the floating point sense.

This feature affects the entire range $\text{Digits}\leq 8$.

- Even if limiting to inputs x with $1.26<x<10$ (1.26 is a bit more than $10^{0.1}$ hence its choice as lower bound), the `poormanlog` documentation mentions an absolute error possibly up to about $1e-9$. In practice a test of 10000 random inputs $1.26<x<10$ revealed 9490 correctly rounded `log10(x)` at 8 digits (and the 510 non-correctly rounded ones with an error of 1 in last digit compared to correct rounding). So correct rounding achieved only in about 95% of cases here.

At 7 digits the same 10000 random inputs are correctly rounded in 99.4% of cases, and at 6 digits it is 99.94% of cases.

Again with Digits=8, the $\log_{10}(i)$ for i in 1..1000 are all correctly rounded to 8 digits with two exceptions: $\log_{10}(3)$ and $\log_{10}(297)$ with a 1ulp error.

- Regarding the computation of 10^x , I obtained for $-1 < x < 1$ the following with 10000 random inputs: 518/10000 errors at 1ulp, 60/10000, and 8/10000, at respectively Digits = 8, 7, 6 so chances of correct rounding are respectively about 95%, 99.4% and more than 99.9%.

Despite its limitations the poormanlog based approach used for Digits up to 8 has the advantage of speed (at least 8X compared to working with 16 digits) and is largely precise enough for plots.

For 9 digits or more, the observed precision in some random tests appears to be at least of 99.9% chances of correct rounding, and the $\log_{10}(x)$ with x near 1 are correctly (if not really efficiently) handled in the floating point sense for the output. The poormanlog approximate $\log_{10}()$ is still used to boot-strap the process, generally. The $\text{pow}_{10}()$ at Digits=9 or more is done independently of poormanlog.

All of this is done on top of my 2013 structures for floating point computations which have always been marked as provisory and rudimentary and instills intrinsic non-efficiency:

- no internal data format for a ``floating point number at P digits'',
- mantissa lengths are again and again computed,
- digits are not pre-organized say in blocks of 4 by 4 or 8 by 8,
- floating point multiplication is done via an **exact** multiplication, then rounding to P digits!

This is legacy of the fact that the project was initially devoted to big integers only, but in the weeks that followed its inception in March 2013 I added more and more functionalities without a well laid out preliminary plan.

Anyway, for years I have felt a better foundation would help achieve at least something such as 2X gain (perhaps the last item by itself, if improved upon, would bring most of such 2X gain?)

I did not try to optimize for the default 16 digits, the goal being more of having a general scalable structure in place and there is no difficulty to go up to 100 digits precision if one stores extended pre-computed constants and increases the length of the ``series'' support.

Apart from $\log(10)$ and its inverse, no other logarithms are stored or pre-computed: the rest of the stored data is the same for $\text{pow}_{10}()$ and $\log_{10}()$ and consists of the fractional powers $10^{\pm 0.i}$, $10^{\pm 0.0i}$, ..., $10^{\pm 0.00000i}$ at P+5 and also at P+10 digits.

In order to reduce the loading time of the package the inverses are not computed internally (as this would require costly divisions) but simply hard-coded with enough digits to cover the allowed Digits range.

13.1 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:^^}%
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{xintlog}{numexpr not available, aborting input}%
26 \aftergroup\endinput
27 \else
28 \ifx\w\relax % xintexpr.sty not yet loaded.
29 \y{xintlog}%
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 \edef\XINTendxintloginput{\XINTrestorecatcodes\noexpand\endinput}\XINTsetcatcodes%

```

13.2 Library identification

```

38 \ifcsname xintlibver@log\endcsname
39 \expandafter\xint_firstoftwo
40 \else
41 \expandafter\xint_secondoftwo
42 \fi
43 {\immediate\write-1{Reloading xintlog library using Digits=\xinttheDigits.}}%
44 {\expandafter\gdef\csname xintlibver@log\endcsname{2021/06/11 v1.4i}%
45 \XINT_providespackage
46 \ProvidesPackage{xintlog}%
47 [2021/06/11 v1.4i Logarithms and exponentials for xintexpr (JFB)]%
48 }%

```

13.3 \xintreloadxintlog

Now needed at 1.4e.

```

49 \def\xintreloadxintlog{\input xintlog.sty }%

```

13.4 Loading the poormanlog package

Attention to the catcode regime when loading poormanlog.

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...

Also, for `xintlog.sty` to be multiple-times loadable, we need to avoid using LaTeX's `\RequirePackage` twice.

```

50 \xintexprSafeCatcodes\catcode`_ 11
51 \unless\ifdefined\XINTinFloatPowTen

```

```

52 \ifdefined\RequirePackage
53   \RequirePackage{poormanlog}%
54 \else
55   \input poormanlog.tex
56 \fi\fi
57 \xintexprRestoreCatcodes\XINTsetcatcodes

```

13.5 Macro layer on top of the poormanlog package

This was moved here with some macro renames from xintfrac on occasion of 1.4e release.

Breaking changes at 1.4e:

- \poormanloghack now a no-op,
- \xintLog was used for \xinteval and differed slightly from its counterpart used for \xintfloateval, the latter float-rounded to $P = \text{Digits}$, the former did not and kept completely meaningless digits in output. Both macros now replaced by a \PoorManLog which will always float round the output to $P = \text{Digits}$. Because xint does not really implement a fixed point interface anyhow.
- \xintExp (used in \xinteval) and another macro (used in \xintfloateval) did not use a sufficiently long approximation to $1/\log(10)$ to support precisely enough $\exp(x)$ if output of the order of 10^{10000} for example, (last two digits wrong then) and situation became worse for very high values such as $\exp(1e8)$ which had only 4 digits correct.

The new \PoorManExp which replaces them is more careful... and for example $\exp(12345678)$ obtains correct rounding ($\text{Digits}=8$).

- \XINTinFloatxintLog and \XINTinFloatxintExp were removed; they were used for $\log()$ and $\exp()$ in \xintfloateval, and differed from \xintLog and \xintExp a bit, now renamed to \PoorManLog and \PoorManExp.

- \PoorManPower has simply disappeared, see \XINTinFloatSciPow and \xintPow.

See the general xintlog introduction for some comments on the achieved precision and probabilities of correct rounding.

13.5.1 \PoorManLogBaseTen, \PoorManLog

1.3f. Code originally in poormanlog v0.04 got transferred here. It produces the logarithm in base 10 with an error (believed to be at most) of the order of 1 unit in the 9th (i.e. last, fixed point) fractional digit. Testing seems to indicate the error is never exceeding 2 units in the 9th place, in worst cases.

These macros will still be the support macros for \xintfloatexpr $\log_{10}()$, $\text{pow}_{10}()$, etc... up to $\text{Digits}=8$ and the poormanlog logarithm is used as starting point for higher precision if Digits is at least 9.

Notice that \PML@999999999. expands (in \numexpr) to 1000000000 (ten digits), which is the only case with the output having ten digits. But there is no need here to treat this case especially, it works fine in \PML@logbaseten.

Breaking change at 1.4e: for consistency with various considerations on floats, the output will be float rounded to $P=\text{Digits}$.

One could envision the \xinteval variant to keep 9 fractional digits (it is known the last one may very well be off by 1 unit). But this creates complications of principles.

All of this is very strange because the logarithm clearly shows the deficiencies of the whole idea of floating point arithmetic, logarithm goes from floating point to fixed point, and coercing it into pure floating point has moral costs. Anyway, I shall obide.

```

58 \def\PoorManLogBaseTen{\romannumeral0\poormanlogbaseten}%
59 \def\poormanlogbaseten #1%
60 {%
61   \XINTinfloat[\XINTdigits]%

```

```

62     {\romannumeral0\expandafter\PML@logbaseten\romannumeral0\XINTinfloat[9]{#1}}%
63 }%
64 \def\PoorManLogBaseTen_raw#1
65 {%
66     \romannumeral0\expandafter\PML@logbaseten\romannumeral0\XINTinfloat[9]{#1}%
67 }%
68 \def\PML@logbaseten#1[#2]%
69 {%
70     \xintiiadd{\xintDSx{-9}}{\the\numexpr#2+8\relax}}{\the\numexpr\PML@#1.}{-9}%
71 }%
72 \def\PoorManLog#1%
73 {%
74     \XINTinFloat[\XINTdigits]{\xintMul{\PoorManLogBaseTen_raw{#1}}{23025850923[-10]}}%
75 }%

```

13.5.2 \PoorManPowerOfTen, \PoorManExp

Originally in poormanlog v0.04, got transferred into xintfrac.sty at 1.3f, then here into xintlog.sty at 1.4e.

Produces 10^x with 9 digits of float precision, with an error (believed to be) at most 2 units in the last place, when $0 < x < 1$. Of course for this the input must be precise enough to have 9 fractional digits of **fixed point** precision.

Breaking change at 1.4e: output always float-rounded at $P = \text{Digits}$.

The 1.3f definition for \xintExp (now \PoorManExp) was not careful enough (see comments above) for very large exponents. This has been corrected at 1.4e. Formerly `exp(12345678)` produced shameful `6.3095734e5361659` where only the first digit (and exponent...) is correct! Now, with `\xintDigits:=8;`, `exp(12345678)` will produce `6.7725836e5361659` which is correct rounding to 8 digits. Sorry if your rover expedition to Mars ended in failure due to using my software. I was not expecting anyone to use it so I did back then in 2019 a bit too expeditively the \xintExp thing on top of 10^x .

The 1.4e \PoorManExp replaces and amends deceased \xintExp.

Before using \xintRound we screen out the case of zero as \xintRound in this case outputs no fractional digits.

```

76 \def\PoorManPowerOfTen{\romannumeral0\poormanpoweroften}%
77 \def\poormanpoweroften #1%
78 {%
79     \expandafter\PML@powoften@out
80     \the\numexpr\expandafter\PML@powoften\romannumeral0\xintra{#1}%
81 }%
82 \def\PML@powoften@out#1[#2]{\XINTinfloat[\XINTdigits]{#1[#2]}}%
83 \def\PML@powoften#1%
84 {%
85     \xint_UDzerominusfork
86     #1-\PML@powoften@zero
87     0#1\PML@powoften@neg
88     0-\PML@powoften@pos
89     \krof #1%
90 }%
91 \def\PML@powoften@zero 0/1[0]{1\relax/1[0]}%
92 \def\PML@powoften@pos#1[#2]%
93 {%
94     \expandafter\PML@powoften@pos@a\romannumeral0\xintround{9}{#1[#2]}.%

```

```

95 }%
96 \def\PML@powoften@pos@a#1.#2.{\PML@Pa#2.\expandafter[\the\numexpr-8+#1]}%
97 \def\PML@powoften@neg#1[#2]%
98 {%
99     \expandafter\PML@powoften@neg@a\romannumeral0\xintround{9}{#1[#2]}.%
100 }%
101 \def\PML@powoften@neg@a#1.#2.%
102 {%
103     \ifnum#2=\xint_c_ \xint_afterfi{1\relax/1[#1]}\else
104     \expandafter\expandafter\expandafter
105     \PML@Pa\expandafter\xint_gobble_i\the\numexpr2000000000-#2.%
106     \expandafter[\the\numexpr-9+#1\expandafter]\fi
107 }%
108 \def\PoorManExp#1{\PoorManPowerOfTen{\xintMul{#1}{43429448190325182765[-20]}}}%

```

13.5.3 Removed: \PoorManPower, see \XINTinFloatSciPow

Removed at 1.4e. See \XINTinFloatSciPow.

13.5.4 Made a no-op: \poormanloghack

Made a no-op at 1.4e.

```

109 \def\poormanloghack#1%
110 {%
111     \xintMessage{xintexpr}{Warning}%
112     {\string\poormanloghack\space is a no-op since 1.4e and will be removed at next major release}%
113 }%

```

13.6 Macro support for powers

13.6.1 \XINTinFloatSciPow

This is the new name and extension of \XINTinFloatPowerH which was a non user-documented macro used for a^b previously, and previously was located in [xintfrac](#).

A check is done whether the exponent is integer or half-integer, and if positive, the legacy \xintFloatPower/\xintFloatSqrt macros are used. The rationale is that:

- they give faster evaluations for integer exponent $b < 10000$ (and beyond)
- they operate at any value of Digits
- they keep accuracy even with gigantic exponents, whereas the $\text{pow}10()/\text{log}10()$ path starts losing accuracy for b about $1e8$. In fact at 1.4e it was even for b about 1000, as $\text{log}10(A)$ was not computed with enough fractional digits, except for $0.8 < A < 1.26$ (roughly), for this usage. At the 1.4f bugfix we compute $\text{log}10(A)$ with enough accuracy for A^b to be safe with b as large as $1e7$, and show visible degradation only for b about $1e9$.

The user documentation of \xintFloatPower mentions a 0.52 ulp(Z) error where Z is the computed result, which seems not as good as the kind of accuracy we target for $\text{pow}10()$ (for $-1 < x < 1$) and $\text{log}10()$ (for $1 < x < 10$) which is more like about 0.505ulp. Perhaps in future I will examine if I need to increase a bit the theoretical accuracy of \xintFloatPower but at time of 1.4e/1.4f release I have left it standing as is.

The check whether exponent is integer or half-integer is not on the value but on the representation. Even in \xintfloatexpr, input such $10^{\text{xintexpr}4/2}\text{relax}$ is possible, and 4/2 will not be recognized as integer to avoid costly overhead. 3/2 will not be recognized as half-integer. But 2.0 will be recognized as integer, $25e-1$ as half-integer.

In the computation of A^b , A will be float-rounded to Digits, but the exponent b will be handled "as is" until last minute. Recall that the `\xintfloatexpr` parser does not automatically float round isolated inputs, this happens only once involved in computations.

In the `Digits<=8` branch we do the same as for `Digits>8` since 1.4f. At 1.4e I had strangely chosen (for "speed", but that was anyhow questionable for integer exponents less than 10 for example) to always use `log10()/pow10()`... But with only 9 fractional digits for the logarithms, exponents such as 1000 naturally led to last 2 or 3 digits being wrong and let's not even mention when the exponent was of the order or $1e6$... now A^{1000} and $A^{1000.5}$ are accurately computed and one can handle $a^{1000.1}$ as $a^{1000} \cdot a^{0.1}$.

I wrote the code during 1.4e to 1.4f transition for doing this split of exponent automatically, but it induced a very significant time penalty down the line for fractional exponents, whereas currently a^b is computed at `Digits=8` with perfectly acceptable accuracy for fractional $\text{abs}(b)<10$, and at high speed, and accuracy for big exponents can be obtained by manually splitting as above (although the above has no user interface for keeping each contribution with its extra digits; a single one for a^h , $-1<h<1$).

```

114 \def\xINTinFloatSciPow{\romannumeral0\xINTinfloatscipow}%
115 \def\xINTinfloatscipow#1#2%
116 {%
117     \expandafter\xINT_scipow_a\romannumeral0\xintrez{#2}\XINT_scipow_int{#1}%
118 }%
119 \def\xINT_scipow_a #1%
120 {%
121     \xint_gob_til_zero#1\xINT_scipow_Biszero0\xINT_scipow_b#1%
122 }%
123 \def\xINT_scipow_Biszero#1#2#3{ 1[0]}%
124 \def\xINT_scipow_b #1#2/#3[#4]#5%
125 {%
126     \unless\if1\xINT_is_One#3XY\xint_dothis\xINT_scipow_c\fi
127     \ifnum#4<\xint_c_mone\xint_dothis\xINT_scipow_c\fi
128     \ifnum#4=\xint_c_mone
129         \if5\xintLDg{#1#2} %
130         \xint_afterfi{\xint_dothis\xINT_scipow_halfint}\else
131         \xint_afterfi{\xint_dothis\xINT_scipow_c}%
132     \fi
133     \fi
134     \xint_orthat#5#1#2/#3[#4]%
135 }%
136 \def\xINT_scipow_int #1/1[#2]#3%
137 {%
138     \expandafter\xINT_flpower_checkB_a
139     \romannumeral0\xINT_dsx_addzeros{#2}#1;.\XINTdigits.{#3}\XINTinfloatS[\XINTdigits]}%
140 }%

```

The `\XINT_flpowerh_finish` is the sole remnant of `\XINTinFloatPowerH` which was formerly stitched to `\xintFloatPower` and checked for half-integer exponent.

```

141 \def\xINT_scipow_halfint#1/1[#2]#3%
142 {%
143     \expandafter\xINT_flpower_checkB_a
144     \romannumeral0\xintdsr{\xintDouble{#1}}.\XINTdigits.{#3}\XINT_flpowerh_finish
145 }%
146 \def\xINT_flpowerh_finish #1%
147 {%
148     \XINTinfloatS[\XINTdigits]{\XINTinFloatSqrt[\XINTdigits+\xint_c_iii]{#1}}%

```



```

149 }%
150 \def\XINT_tmpa#1.{%
151 \def\XINT_scipow_c ##1[##2]##3%
152 {%
153     \expandafter\XINT_scipow_d\romannumeral0\XINTinfloatS[#1]{##3}\xint:##1[##2]\xint:
154 }%
155 }\expandafter\XINT_tmpa\the\numexpr\XINTdigits.%
156 \def\XINT_scipow_d #1%
157 {%
158     \xint_UDzerominusfork
159     #1-\XINT_scipow_Aiszero
160     0#1\XINT_scipow_Aisneg
161     0-\XINT_scipow_Aispos
162     \krof #1%
163 }%
164 \def\XINT_scipow_Aiszero #1\xint:#2#3\xint:
165 {%

```

Missing NaN and Infinity causes problems. Inserting something like 1["7FFF8000] is risky as certain macros convert [N] into N zeros... so the run can appear to stall and will crash possibly badly if we do that. There is some usage in relation to `ilog10` in `xint.sty` and `xintfrac.sty` of "7FFF8000 but here I will stay prudent and insert the usual 0 value (changed at 1.4g)

```

166     \if-#2\xint_dothis
167     {\XINT_signalcondition{InvalidOperation}{0 raised to power #2#3.}{\ 0[0]}}\fi
168     \xint_orthat{ 0[0]}%
169 }%
170 \def\XINT_scipow_Aispos #1\xint:#2\xint:
171 {%
172     \XINTinfloatpowten{\xintMul{#2}{\XINTinFloatLogTen_xdgout#1}}%
173 }%

```

If a^b with $a < 0$, we arrive here only if b was not considered to be an integer exponent. So let's raise an error.

```

174 \def\XINT_scipow_Aisneg #1#2\xint:#3\xint:
175 {%
176     \XINT_signalcondition{InvalidOperation}%
177     {Fractional power #3 of negative #1#2.}{\ 0[0]}%
178 }%
179 \ifnum\XINTdigits<9

```

At 1.4f we only need for Digits up to 8 to insert usage of `poormanlog` for non integer, non half-integer exponents. At 1.4e the code was more complicated because I had strangely opted for using always the `log10()` path. However we have to be careful to use `\PML@logbaseten` with 9 digits always.

As the legacy macros used for integer and half-integer exponents float-round the input to Digits digits, we must do the same here for coherence. Which induces some small complications here.

```

180 \def\XINT_tmpa#1.#2.#3.{%
181 \def\XINT_scipow_c ##1[##2]##3%
182 {%
183     \expandafter\XINT_scipow_d
184     \romannumeral0\expandafter\XINT_scipow_c_i
185     \romannumeral0\XINTinfloat[#1]{##3}\xint:##1[##2]\xint:
186 }%
187 \def\XINT_scipow_c_i##1[##2]{ ##1#3[##2-#2]}%
188 }\expandafter\XINT_tmpa\the\numexpr\XINTdigits\expandafter.%

```



```

189 \the\numexpr9-\XINTdigits\expandafter.%
190 \romannumeral\xintreplicate{9-\XINTdigits}0.%
191 \def\XINT_scipow_Aispos #1\xint:#2\xint:
192 {%
193 \poormanpoweroften{\xintMul{#2}{\romannumeral0\expandafter\PML@logbaseten#1}}%
194 }%
195 \fi

```

13.6.2 \xintPow

Support macro for a^b in \xinteval. This overloads the original [xintfrac](#) macro, keeping its original meaning only for integer exponents, which are not too big: for exact evaluation of A^b , we want the output to not have more than about 10000 digits (separately for numerator and denominator). For this we limit b depending on the length of A , simply we want b to be smaller than the rounded value of 10000 divided by the length of A . For one-digit A , this would give 10000 as maximal exponent but due to organization of code related to avoid arithmetic overflow (we can't immediately operate in \numexpr with b as it is authorized to be beyond TeX bound), the maximal exponent is 9999.

The criterion, which guarantees output (numerator and denominator separately) does not exceed by much 10000 digits if at all is that the exponent should be less than the (rounded in the sense of \numexpr) quotient of 10000 by the number of digits of a (considering separately numerator and denominator).

The decision whether to compute A^b exactly depends on the length of internal representation of A . So 9^{9999} is evaluated exactly (in \xinteval) but for 9.0 it is 9.0^{5000} the maximal power. This may change in future.

1.4e had the following bug (for $\text{Digits} > 8$): big integer exponents used the $\log_{10}()/\text{pow}_{10}()$ based approach rather than the legacy macro path which goes via \xintFloatPower, as done by \xintfloat-eval! As a result powers very large integer exponents were more precise in \xintfloateval than in \xinteval!

1.4f fixes this. Also, it handles $\text{Digits} \leq 8$ as $\text{Digits} > 8$, bringing much simplification here.

```

196 \def\xintPow{\romannumeral0\xintpow}%
197 \def\xintpow#1#2%
198 {%
199 \expandafter\XINT_scipow_a\romannumeral0\xintrez{#2}\XINT_pow_int{#1}%
200 }%

```

In case of half-integer exponent the \XINT_scipow_a will have triggered usage of the (new incarnation) of \XINTinFloatPowerH which combines \xintFloatPower and square root extraction. So we only have to handle here the case of integer exponents which will trigger execution of this \XINT_pow_int macro passed as parameter to \xintpow.

```

201 \def\XINT_pow_int #1/1[#2]%
202 {%
203 \expandafter\XINT_pow_int_a\romannumeral0\XINT_dsx_addzeros{#2}#1;.%
204 }%

```

1.4e had a bug here for integer exponents ≥ 10000 : they triggered going back to the floating point routine but at a late location where the $\log_{10}()/\text{pow}_{10}()$ approach is used.

```

205 \def\XINT_pow_int_a #1#2.%
206 {%
207 \ifnum\if-#1\xintLength{#2}\else\xintLength{#1#2}\fi>\xint_c_iv
208 \expandafter\XINT_pow_bigint
209 \else\expandafter\XINT_pow_int_b
210 \fi #1#2.%

```

211 }%

At 1.4f we correctly jump to the appropriate entry point into the `\xintFloatPower` routine of `xintfrac`, in case of a big integer exponent.

212 \def\xINT_pow_bigint #1.#2%

213 {%

214 \XINT_flpower_checkB_a#1.\XINTdigits.{#2}{\XINTinfloatS[\XINTdigits]]%

215 }%

216 \def\xINT_pow_int_b #1.#2%

217 {%

We now check if the output will not be too bulky. We use here (on the a of a^b) `\xintraw`, not `\xintrez`, on purpose so that for example 9.0^{9999} is computed in floating point sense but 9^{9999} is computed exactly. However 9.0^{5000} will be computed exactly. And if I used `\xintrez` here `\xinteval{100^2}` would print 10000.0 and `\xinteval{100^3}` would print 1.0e6. Thus situation is complex.

By the way I am happy to see that 9.0×9.0 in `\xinteval` does print 81.0 but the truth is that internally it does have the more bulky $8100/1[-2]$ maybe I should make some revision of this, i.e. use rather systematically `\xintREZ` on input rather than `\xintRaw` (note taken on 2021/05/08 at time of doing 1.4f bugfix release).

218 \expandafter\xINT_pow_int_c\romannumeral0\xintraw{#2}\xint:#1\xint:

219 }%

The `\XINT_fpow_fork` is (quasi top level) entry point we have found into the legacy `\xintPow` routine of `xintfrac`. Its interface is a bit weird, but let's not worry about this now.

220 \def\xINT_pow_int_c#1#2/#3[#4]\xint:#5\xint:

221 {%

222 \if0\ifnum\numexpr\xint_c_x^iv/%

223 (\xintLength{#1#2}\if-#1-\xint_c_i\fi)<\XINT_Abs#5 %

224 1\else

225 \ifnum\numexpr\xint_c_x^iv/\xintLength{#3}<\XINT_Abs#5 %

226 1\else

227 0\fi\fi

228 \expandafter\xINT_fpow_fork\else\expandafter\xINT_pow_bigint_i

229 \fi

230 #5\Z{#4}{#1#2}{#3}%

231 }%

`\XINT_pow_bigint_i` is like `\XINT_pow_bigint` but has its parameters organized differently.

232 \def\xINT_pow_bigint_i#1\Z#2#3#4%

233 {%

234 \XINT_flpower_checkB_a#1.\XINTdigits.{#3/#4[#2]}{\XINTinfloatS[\XINTdigits]]%

235 }%

13.7 Macro support for `\xintexpr` and `\xintfloatexpr` syntax

13.7.1 The `log10()` and `pow10()` functions

Up to 8 digits included we use the `poormanlog` based ones.

236 \ifnum\xINTdigits<9

237 \expandafter\def\csname XINT_expr_func_log10\endcsname#1#2#3%

238 {%

239 \expandafter #1\expandafter #2\expandafter{%

240 \romannumeral`&&\XINT:NEhook:f:one:from:one

241 {\romannumeral`&&\PoorManLogBaseTen#3}}%

```

242 }%
243 \expandafter\def\csname XINT_expr_func_pow10\endcsname#1#2#3%
244 {%
245     \expandafter #1\expandafter #2\expandafter{%
246     \romannumeral`&&\XINT:NEhook:f:one:from:one
247     {\romannumeral`&&\PoorManPowerOfTen#3}}}%
248 }%
249 \else
250 \expandafter\def\csname XINT_expr_func_log10\endcsname#1#2#3%
251 {%
252     \expandafter #1\expandafter #2\expandafter{%
253     \romannumeral`&&\XINT:NEhook:f:one:from:one
254     {\romannumeral`&&\XINTinFloatLogTen#3}}}%
255 }%
256 \expandafter\def\csname XINT_expr_func_pow10\endcsname#1#2#3%
257 {%
258     \expandafter #1\expandafter #2\expandafter{%
259     \romannumeral`&&\XINT:NEhook:f:one:from:one
260     {\romannumeral`&&\XINTinFloatPowTen#3}}}%
261 }%
262 \fi
263 \expandafter\let\csname XINT_flexpr_func_log10\expandafter\endcsname
264     \csname XINT_expr_func_log10\endcsname
265 \expandafter\let\csname XINT_flexpr_func_pow10\expandafter\endcsname
266     \csname XINT_expr_func_pow10\endcsname

```

13.7.2 The log(), exp() functions

```

267 \ifnum\XINTdigits<9
268 \def\XINT_expr_func_log #1#2#3%
269 {%
270     \expandafter #1\expandafter #2\expandafter{%
271     \romannumeral`&&\XINT:NEhook:f:one:from:one
272     {\romannumeral`&&\PoorManLog#3}}}%
273 }%
274 \def\XINT_expr_func_exp #1#2#3%
275 {%
276     \expandafter #1\expandafter #2\expandafter{%
277     \romannumeral`&&\XINT:NEhook:f:one:from:one
278     {\romannumeral`&&\PoorManExp#3}}}%
279 }%
280 \let\XINT_flexpr_func_log\XINT_expr_func_log
281 \let\XINT_flexpr_func_exp\XINT_expr_func_exp
282 \else
283 \def\XINT_expr_func_log #1#2#3%
284 {%
285     \expandafter #1\expandafter #2\expandafter{%
286     \romannumeral`&&\XINT:NEhook:f:one:from:one
287     {\romannumeral`&&\XINTinFloatLog#3}}}%
288 }%
289 \def\XINT_expr_func_exp #1#2#3%
290 {%
291     \expandafter #1\expandafter #2\expandafter{%

```

```

292 \romannumeral`&&\XINT:NEhook:f:one:from:one
293 {\romannumeral`&&\XINTinFloatExp#3}}%
294 }%
295 \let\XINT_flexpr_func_log\XINT_expr_func_log
296 \let\XINT_flexpr_func_exp\XINT_expr_func_exp
297 \fi

```

13.7.3 The pow() function

The mapping of `**` and `^` to `\XINTinFloatSciPow` (in `\xintfloatexpr` context) and `\xintPow` (in `\xintexpr` context), is done in `xintexpr`.

```

298 \def\XINT_expr_func_pow #1#2#3%
299 {%
300 \expandafter #1\expandafter #2\expandafter{%
301 \romannumeral`&&\XINT:NEhook:f:one:from:two
302 {\romannumeral`&&\xintPow#3}}%
303 }%
304 \def\XINT_flexpr_func_pow #1#2#3%
305 {%
306 \expandafter #1\expandafter #2\expandafter{%
307 \romannumeral`&&\XINT:NEhook:f:one:from:two
308 {\romannumeral`&&\XINTinFloatSciPow#3}}%
309 }%

```

13.8 End of package loading for low Digits

```

310 \ifnum\XINTdigits<9 \expandafter\XINTendxintloginput\fi%

```

13.9 Stored constants

The constants were obtained from Maple at 80 digits: fractional power of 10, but only one logarithm `log(10)`.

Currently the code whether for exponential or logarithm will not screen out 0 digits and even will do silly multiplication by $10^0 = 1$ in that case, and we need to store such silly values.

We add the data for the $10^{-0.i}$ etc... because pre-computing them on the fly significantly adds overhead to the package loading.

The fractional powers of ten with D+5 digits are used to compute `pow10()` function, those with D+10 digits are used to compute `log10()` function. This is done with an elevated precision for two reasons: (- handling of inputs near 1, :- in order for $a^b = \text{pow10}(b \cdot \log_{10}(a))$ to keep accuracy even with large exponents, say in absolute value up to $1e7$, degradation beginning to show-up at $1e8$.)

```

311 \def\XINT_tmpa{1[0]}%
312 \expandafter\let\csname XINT_c_1_0\endcsname\XINT_tmpa
313 \expandafter\let\csname XINT_c_2_0\endcsname\XINT_tmpa
314 \expandafter\let\csname XINT_c_3_0\endcsname\XINT_tmpa
315 \expandafter\let\csname XINT_c_4_0\endcsname\XINT_tmpa
316 \expandafter\let\csname XINT_c_5_0\endcsname\XINT_tmpa
317 \expandafter\let\csname XINT_c_6_0\endcsname\XINT_tmpa
318 \expandafter\let\csname XINT_c_1_0_x\endcsname\XINT_tmpa
319 \expandafter\let\csname XINT_c_2_0_x\endcsname\XINT_tmpa
320 \expandafter\let\csname XINT_c_3_0_x\endcsname\XINT_tmpa
321 \expandafter\let\csname XINT_c_4_0_x\endcsname\XINT_tmpa

```

```

322 \expandafter\let\csname XINT_c_5_0_x\endcsname\XINT_tmpa
323 \expandafter\let\csname XINT_c_6_0_x\endcsname\XINT_tmpa
324 \expandafter\let\csname XINT_c_1_0_inv\endcsname\XINT_tmpa
325 \expandafter\let\csname XINT_c_2_0_inv\endcsname\XINT_tmpa
326 \expandafter\let\csname XINT_c_3_0_inv\endcsname\XINT_tmpa
327 \expandafter\let\csname XINT_c_4_0_inv\endcsname\XINT_tmpa
328 \expandafter\let\csname XINT_c_5_0_inv\endcsname\XINT_tmpa
329 \expandafter\let\csname XINT_c_6_0_inv\endcsname\XINT_tmpa
330 \expandafter\let\csname XINT_c_1_0_inv_x\endcsname\XINT_tmpa
331 \expandafter\let\csname XINT_c_2_0_inv_x\endcsname\XINT_tmpa
332 \expandafter\let\csname XINT_c_3_0_inv_x\endcsname\XINT_tmpa
333 \expandafter\let\csname XINT_c_4_0_inv_x\endcsname\XINT_tmpa
334 \expandafter\let\csname XINT_c_5_0_inv_x\endcsname\XINT_tmpa
335 \expandafter\let\csname XINT_c_6_0_inv_x\endcsname\XINT_tmpa
336 \def\XINT_tmpa#1#2#3#4;%
337   {\expandafter\edef\csname XINT_c_#1_#2\endcsname{\XINTinFloat[\XINTdigitsormax+5]{#3#4[-79]}};%
338   \expandafter\edef\csname XINT_c_#1_#2_x\endcsname{\XINTinFloat[\XINTdigitsormax+10]{#3#4[-79]}};%
339   }%
340 % 10^0.i
341 \XINT_tmpa 1 1 12589254117941672104239541063958006060936174094669310691079230195266476157825020;%
342 \XINT_tmpa 1 2 15848931924611134852021013733915070132694421338250390683162968123166568636684540;%
343 \XINT_tmpa 1 3 1995262314968879601352455396739535579862743154053460992299136670049309106980490;%
344 \XINT_tmpa 1 4 25118864315095801110850320677993273941585181007824754286798884209082432477235613;%
345 \XINT_tmpa 1 5 31622776601683793319988935444327185337195551393252168268575048527925944386392382;%
346 \XINT_tmpa 1 6 39810717055349725077025230508775204348767703729738044686528414806022485386945804;%
347 \XINT_tmpa 1 7 50118723362727228500155418688494576806047198983281926392969745588901125568883069;%
348 \XINT_tmpa 1 8 63095734448019324943436013662234386467294525718822872452772952883349494329768681;%
349 \XINT_tmpa 1 9 79432823472428150206591828283638793258896063175548433209232392931695569719148754;%
350 % 10^0.0i
351 \XINT_tmpa 2 1 10232929922807541309662751748198778273411640572379813085994255856738296458625172;%
352 \XINT_tmpa 2 2 10471285480508995334645020315281400790567914715039292120056525299012577641023719;%
353 \XINT_tmpa 2 3 10715193052376064174083022246945087339158659633422172707894501914136771607653870;%
354 \XINT_tmpa 2 4 10964781961431850131437136061411270464271158762483023169080841607885740984711300;%
355 \XINT_tmpa 2 5 11220184543019634355910389464779057367223085073605529624450744481701033026862244;%
356 \XINT_tmpa 2 6 11481536214968827515462246116628360182562102373996119340874991068894793593040890;%
357 \XINT_tmpa 2 7 11748975549395295417220677651268442278134317971793124791953875805007912852226246;%
358 \XINT_tmpa 2 8 12022644346174129058326127151935204486942664354881189151104892745683155052368222;%
359 \XINT_tmpa 2 9 12302687708123815342415404364750907389955639574572144413097319170011637639124482;%
360 % 10^0.00i
361 \XINT_tmpa 3 1 10023052380778996719154048893281105540536684535421606464116348523047431367720401;%
362 \XINT_tmpa 3 2 10046157902783951424046519858132787392010166060319618489538315083825599423438638;%
363 \XINT_tmpa 3 3 10069316688518041699296607872661381368099438247964820601930206419324524707606686;%
364 \XINT_tmpa 3 4 10092528860766844119155277641202580844111492027373621434478800545314309618714957;%
365 \XINT_tmpa 3 5 10115794542598985244409323144543146957419235215102899054703546688078254946034250;%
366 \XINT_tmpa 3 6 10139113857366794119988279023017296985954042032867436525450889437280417044987125;%
367 \XINT_tmpa 3 7 10162486928706956276733661150135543062420167220622552197768982666050994284378619;%
368 \XINT_tmpa 3 8 10185913880541169240797988673338257820431768224957171297560936579346433061037662;%
369 \XINT_tmpa 3 9 10209394837076799554149033101487543990018213667630072574873723356334069913329713;%
370 % 10^0.000i
371 \XINT_tmpa 4 1 10002302850208247526835942556719413318678216124626534526963475845228205382579041;%
372 \XINT_tmpa 4 2 10004606230728403216239656646745503559081482371024284871882409614422496765669196;%
373 \XINT_tmpa 4 3 10006910141682589957025973521996241909035914023642264228577379693841345823180462;%

```

```

374 \XINT_tmpa 4 4 10009214583192958761081718336761022426385537997384755843291864010938378093197023;%
375 \XINT_tmpa 4 5 10011519555381688769842032367472488618040778885656970999331288116685029387850446;%
376 \XINT_tmpa 4 6 10013825058370987260768186632475607982636715641432550952229573271596547716373358;%
377 \XINT_tmpa 4 7 10016131092283089653826887255241073941084503769368844606021481400409002185558343;%
378 \XINT_tmpa 4 8 10018437657240259517971072914549205297136779497498835020699531587537662833033174;%
379 \XINT_tmpa 4 9 10020744753364788577622204725249622301332888222801030351604197113557132455165040;%
380 % 10^0.0000i
381 \XINT_tmpa 5 1 10000230261160268806710649793464495797824846841503180050673957122443571394978721;%
382 \XINT_tmpa 5 2 10000460527622557806255008596155855743730116854295068547616656160734125748005947;%
383 \XINT_tmpa 5 3 10000690799386989083565213461287219981856579552059660369243804541364501659468630;%
384 \XINT_tmpa 5 4 10000921076453684726384543254593368743049141124080210677706489564626675960578367;%
385 \XINT_tmpa 5 5 10001151358822766825267483384008265483772370538793312970508590203623535763866465;%
386 \XINT_tmpa 5 6 10001381646494357473579790530833073090516914490540536234536867917078761046656260;%
387 \XINT_tmpa 5 7 10001611939468578767498557382394677469502542123237272447312733350028467607076918;%
388 \XINT_tmpa 5 8 10001842237745552806012277366194752842273812293689190856411757410911882303011468;%
389 \XINT_tmpa 5 9 10002072541325401690920909385549403068574626162727745910217443397959031898734024;%
390 % 10^0.00000i
391 \XINT_tmpa 6 1 10000023025877439451356029805459000097926504781151663770980171880313737943886754;%
392 \XINT_tmpa 6 2 10000046051807898005897723104514851394069452605882077809669546315010724085277647;%
393 \XINT_tmpa 6 3 10000069077791375785706217087438809625967243923218032821061587553353589726808164;%
394 \XINT_tmpa 6 4 10000092103827872912862930047032391734439796534302560512742030066798473305401477;%
395 \XINT_tmpa 6 5 10000115129917389509449561379274639104559958866285946533811801963402821672829477;%
396 \XINT_tmpa 6 6 10000138156059925697548091583969382297005329013199894805417325991907389143667949;%
397 \XINT_tmpa 6 7 10000161182255481599240782265392507269793911275470978276390154932321984777772469;%
398 \XINT_tmpa 6 8 10000184208504057336610176132939223090407041937631374389422968832433217547184883;%
399 \XINT_tmpa 6 9 10000207234805653031739097001771331138303016031686764989867510425362339583809842;%
400 \def\XINT_tmpa#1#2#3#4;%
401   {\expandafter\edef\csname XINT_c_#1_#2_inv\endcsname{\XINTinFloat[\XINTdigitsormax+5]{#3#4[-80]}}}%
402   \expandafter\edef\csname XINT_c_#1_#2_inv_x\endcsname{\XINTinFloat[\XINTdigitsormax+10]{#3#4[-80]}}}%
403   }%
404 % 10^-0.i
405 \XINT_tmpa 1 1 79432823472428150206591828283638793258896063175548433209232392931695569719148754;%
406 \XINT_tmpa 1 2 63095734448019324943436013662234386467294525718822872452772952883349494329768681;%
407 \XINT_tmpa 1 3 50118723362727228500155418688494576806047198983281926392969745588901125568883069;%
408 \XINT_tmpa 1 4 39810717055349725077025230508775204348767703729738044686528414806022485386945804;%
409 \XINT_tmpa 1 5 31622776601683793319988935444327185337195551393252168268575048527925944386392382;%
410 \XINT_tmpa 1 6 25118864315095801110850320677993273941585181007824754286798884209082432477235613;%
411 \XINT_tmpa 1 7 19952623149688796013524553967395355579862743154053460992299136670049309106980490;%
412 \XINT_tmpa 1 8 15848931924611134852021013733915070132694421338250390683162968123166568636684540;%
413 \XINT_tmpa 1 9 12589254117941672104239541063958006060936174094669310691079230195266476157825020;%
414 % 10^-0.0i
415 \XINT_tmpa 2 1 97723722095581068269707600696156123863427170069897801526639004097175507042084888;%
416 \XINT_tmpa 2 2 95499258602143594972395937950148401513087269708053320302465127242741421479104601;%
417 \XINT_tmpa 2 3 93325430079699104353209661168364840720225485199736026149257155811788093771138272;%
418 \XINT_tmpa 2 4 91201083935590974212095940791872333509323858755696109214760361851771695487999100;%
419 \XINT_tmpa 2 5 89125093813374552995310868107829696398587478293004836994794349506746891059190135;%
420 \XINT_tmpa 2 6 87096358995608063751082742520877054774747128501284704090761796673224328569285177;%
421 \XINT_tmpa 2 7 85113803820237646781712631859248682794521725442067093899553745086385146367436049;%
422 \XINT_tmpa 2 8 83176377110267100616669140273840405263880767161887438462740286611379995442629360;%
423 \XINT_tmpa 2 9 81283051616409924654127879773132980187568851100062454636602325121954484722491710;%
424 % 10^-0.00i
425 \XINT_tmpa 3 1 99770006382255331719442194285376231055211861394573154624878230890945476532432225;%

```



```

426 \XINT_tmpa 3 2 99540541735152696244806147089510943107144177264574823668081299845609359857038344;%
427 \XINT_tmpa 3 3 99311604842093377157642607688515474663519162181123336122073822476734517364853150;%
428 \XINT_tmpa 3 4 99083194489276757440828314388392035249938006860819409201135652190410238171119287;%
429 \XINT_tmpa 3 5 98855309465693884028524792978202683686410726723055209558576898759166522286083202;%
430 \XINT_tmpa 3 6 98627948563121047157261523093421290951784086730437722805070296627452491731402556;%
431 \XINT_tmpa 3 7 98401110576113374484101831088824192144756194053451911515003663381199842081528019;%
432 \XINT_tmpa 3 8 98174794301998439937928161622872240632362817134775142288598128693131032909278350;%
433 \XINT_tmpa 3 9 97948998540869887269961493687844910565420716785032030061251916654655049965062649;%
434 % 10^-0.000i
435 \XINT_tmpa 4 1 99976976799815658635141604638981297541396466984477711459083930684685186989697929;%
436 \XINT_tmpa 4 2 99953958900308784552845777251512089759003230012954649234748668826546533498169555;%
437 \XINT_tmpa 4 3 99930946300258992168693777702512591351888960684418033717545524043693899420866954;%
438 \XINT_tmpa 4 4 99907938998446176870082987427724649318531547584410414997787083472394558389284098;%
439 \XINT_tmpa 4 5 99884936993650514951538205746462968844845952521633937925370747725933629958238429;%
440 \XINT_tmpa 4 6 99861940284652463550037839584112909891259691850983307437097305856727153967481065;%
441 \XINT_tmpa 4 7 99838948870232760580354983175435314251655958968480344701699631967048474751069525;%
442 \XINT_tmpa 4 8 99815962749172424670413384320528274471550942114263604264788586703624513163664479;%
443 \XINT_tmpa 4 9 9979298192025275509665829376608502587039285410603746599001121635652334125368417;%
444 % 10^-0.0000i
445 \XINT_tmpa 5 1 99997697441416293040019992468837639003787989306240470048763511538639048400765328;%
446 \XINT_tmpa 5 2 99995394935850346394065999228750187791584034668237852053859761641089829514536011;%
447 \XINT_tmpa 5 3 99993092483300939297147020491645017932348508508297743745039515152378182676736684;%
448 \XINT_tmpa 5 4 99990790083766851012380885556584619169980753943113396677545915245611923361705686;%
449 \XINT_tmpa 5 5 99988487737246860830993605587529673614422529030613405900998412734419982883669223;%
450 \XINT_tmpa 5 6 99986185443739748072318726405984801565268578044798475766025647187221659622450651;%
451 \XINT_tmpa 5 7 99983883203244292083796681298546635825139453823571398432959235283529730820181019;%
452 \XINT_tmpa 5 8 99981581015759272240974143839353881367972777961073357987943600347058023396510672;%
453 \XINT_tmpa 5 9 99979278881283467947503380727439017235290006415950636109257677645557027950744160;%
454 % 10^-0.00000i
455 \XINT_tmpa 6 1 99999769741755795297487775997495948154386159348543852707438213487494386559762090;%
456 \XINT_tmpa 6 2 99999539484041779185217876175552674518572114763104546143049036309870762496098218;%
457 \XINT_tmpa 6 3 99999309226857950442387361668529812394860404492721699528707852590634886516924591;%
458 \XINT_tmpa 6 4 99999078970204307848196104610199226516866442484686906173860803560254163287393673;%
459 \XINT_tmpa 6 5 99998848714080850181846788127272455158309917012010320554498356105168896062430977;%
460 \XINT_tmpa 6 6 99998618458487576222544906332928167145404344730731751204389698696345970645201375;%
461 \XINT_tmpa 6 7 99998388203424484749498764320339633772810463403640242228131015918494067456365331;%
462 \XINT_tmpa 6 8 99998157948891574541919478156202215623119146605983303201215215949834619332550929;%
463 \XINT_tmpa 6 9 9999792769488844379020974874260864289829523807763942234420930258187873904191138;%
464 % log(10)
465 \edef\XINT_c_logten
466 {\XINTinFloat[\XINTdigitsormax+4]
467 {23025850929940456840179914546843642076011014886287729760333279009675726096773525[-79]}}%
468 \edef\XINT_c_oneoverlogten
469 {\XINTinFloat[\XINTdigitsormax+4]
470 {43429448190325182765112891891660508229439700580366656611445378316586464920887077[-80]}}%
471 \edef\XINT_c_oneoverlogten_xx
472 {\XINTinFloat[\XINTdigitsormax+14]
473 {43429448190325182765112891891660508229439700580366656611445378316586464920887077[-80]}}%

```

13.10 April 2021: at last, \XINTinFloatPowTen, \XINTinFloatExp

Done April 2021. I have procrastinated (or did not have time to devote to this) at least 5 years, even more.

Speed improvements will have to wait to long delayed refactoring of core floating point support which is still in the 2013 primitive state !

I did not try to optimize for say 16 digits, as I was more focused on reaching 60 digits in a reasonably efficient manner (trigonometric functions achieved this since 2019) in the same coding framework. Finally, up to 62 digits.

The stored constants are $\log(10)$ at P+4 digits and the powers $10^{0.d}$, $10^{0.0d}$, etc, up to $10^{0.00000d}$ for $d=1..9$, as well as their inverses, at P+5 and P+10 digits. The constants were obtained from Maple at 80 digits.

Initially I constructed the exponential series $\exp(h)$ as one big unique nested macro. It contained pre-rounded values of the $1/i!$ but would float-round h to various numbers of digits, with always the full initial h as input.

After having experimented with the logarithm, I redid $\exp(h) = 1 + h(1 + h(1/2 + \dots))$ with many macros in order to have more readable code, and to dynamically cut-off more and more digits from h the deeper it is used. See the logarithm code for (perhaps) more comments.

The thresholds have been obtained from considerations including an h_{\max} (a bit more than $0.5 \log(10) 10^{-6}$). Here is the table:

- maximal value of P: 8, 15, 21, 28, 35, 42, 48, 55, 62
- last included term: $/1$, $/2$, $/6$, $/4!$, $/5!$, $/6!$, $/7!$, $/8!$, $/9!$

Computations are done morally targeting P+4 fractional fixed point digits, with a stopping criteria at say about $5e(-P-4)$, which was used for the table above using only the worst case. As the used macros are a mix of exact operations and floating point reductions this is in practice a bit different. The h will be initially float rounded to P-1 digits. It is cut-off more and more, the deeper nested it is used.

The code for this evaluation of 10^x is very poor with x very near zero: it does silly multiplication by 1, and uses more terms of exponential series than would then be necessary.

For the computation of $\exp(x)$ as $10^{(c*x)}$ with $c=\log(10)^{-1}$, we need more precise c the larger $\text{abs}(x)$ is. For $\text{abs}(x)<1$ (or 2), the c with P+4 fractional digits is sufficient. But decimal exponents are more or less allowed to be near the TeX maximum $2^{31}-1$, which means that $\text{abs}(x)$ could be as big as $0.5e10$, and we then need c with P+14 digits to cover that range.

I am hesitating whether to first examine integral part of $\text{abs}(x)$ and for example to use c with either P+4, P+9 or P+14 digits, and also take this opportunity to inject an error message if x is too big before TeX arithmetic overflow happens later on. For time being I will use overhead of `oneoverlogten` having ample enough digits...

The exponent received as input is float rounded to P + 14 digits. In practice the input will be already a P-digits float. The motivation here is for low Digits situation: but this done so that for example with Digits=4, we want $\exp(12345)$ not to be evaluated as $\exp(12350)$ which would have no meaning at all. The +14 is because we have prepared $1/\log(10)$ with that many significant digits. This conundrum is due to the inadequation of the world of floating point numbers with $\exp()$ and $\log()$: clearly $\exp()$ goes from fixed point to floating point and $\log()$ goes from floating point to fixed point, and coercing them to work inside the sole floating point domain is not mathematically natural. Although admittedly it does create interesting mathematical questions! A similar situation applies to functions such as $\cos()$ and $\sin()$, what sense is there in the expression $\cos(\exp(50))$ for example with 16 digits precision? My opinion is that it does not make ANY sense. Anyway, I shall obide.

As `\XINTinFloatS` will not add unnecessarily trailing zeros, the `\XINTdigits+14` is not really an enormous overhead for integer exponents, such as in the example above the 12345, or more realistically small integer exponents, and if the input is already float rounded to P digits, the overhead is also not enormous (float-rounding is costly when the input is a fraction).

`\XINTinfloatpowten` will receive an input with at least P+14 and up to $2P+28$ digits... fortunately with no fraction part and will start rounding it in the fixed point sense of its input to P+4

digits after decimal point, which is not enormously costly.

Of course all these things pile up...

```

474 \def\XINTinFloatExp{\romannumeral0\XINTinfloatexp}%
475 \def\XINT_tmpa#1.{%
476 \def\XINTinfloatexp##1%
477 {%
478   \XINTinfloatpowten
479   {\xintMul{\XINT_c_oneoverlogten_xx}{\XINTinFloatS[#1]{##1}}}%
480 }%
481 }\expandafter\XINT_tmpa\the\numexpr\XINTdigitsormax+14.%

```

Here is how the reduction to computations of an exp(h) via series is done.

Starting from x, after initial argument normalization, it is fixed-point rounded to 6 fractional digits giving x' = ±n.d₁...d₆ (which may be 0).

I have to resist temptation using very low level routines here and wisely will employ the available user-level stuff. One computes then the difference x-x' which gives some eta, and the h will be log(10).eta. The subtraction and multiplication are done exactly then float rounded to P-1 digits to obtain the h.

Then exp(h) is computed. And to finish it is multiplied with the stored 10^{±0.d₁}, 10^{±0.d₂}, etc..., constants and its decimal exponent is increased by ±n. These operations are done at P+5 floating point digits. The final result is then float-rounded to the target P digits.

Currently I may use nested macros for some operations but will perhaps revise in future (it makes tracing very complicated if one does not have intermediate macros). The exponential series itself was initially only one single macro, but as commented above I have now modified it.

```

482 \def\XINTinFloatPowTen{\romannumeral0\XINTinfloatpowten}%
483 \def\XINT_tmpa#1.{%
484 \def\XINTinfloatpowten##1%
485 {%
486   \expandafter\XINT_powten_fork
487   \romannumeral0\xintiround{#1}{##1}[-#1]%
488 }%
489 }\expandafter\XINT_tmpa\the\numexpr\XINTdigitsormax+4.%
490 \def\XINT_powten_fork#1%
491 {%
492   \xint_UDzerominusfork
493   #1-\XINT_powten_zero
494   0#1\XINT_powten_neg
495   0-\XINT_powten_pos
496   \krof #1%
497 }%
498 \def\XINT_powten_zero #1[#2]{ 1[0]}%

```

This rounding may produce 0.000000 but will always have 6 exactly fractional digits, because the special case of a zero input was filtered out preventively.

```

499 \def\XINT_powten_pos#1[#2]%
500 {%
501   \expandafter\XINT_powten_pos_a\romannumeral0\xintround{6}{#1[#2]}#1[#2]%
502 }%
503 \def\XINT_tmpa #1.#2.#3.{%
504 \def\XINT_powten_pos_a ##1.##2##3##4##5##6##7##8[##9]%
505 {%
506   \expandafter\XINT_infloate
507   \romannumeral0\XINTinfloat[#3]}%

```

```

508 \xintMul{\csname XINT_c_1_##2\endcsname}{%
509 \XINTinFloat[#1]{%
510 \xintMul{\csname XINT_c_2_##3\endcsname}{%
511 \XINTinFloat[#1]{%
512 \xintMul{\csname XINT_c_3_##4\endcsname}{%
513 \XINTinFloat[#1]{%
514 \xintMul{\csname XINT_c_4_##5\endcsname}{%
515 \XINTinFloat[#1]{%
516 \xintMul{\csname XINT_c_5_##6\endcsname}{%
517 \XINTinFloat[#1]{%
518 \xintMul{\csname XINT_c_6_##7\endcsname}{%
519 \xintAdd{1[0]}{%
520 \expandafter\XINT_Exp_series_a_ii
521 \romannumeral0\XINTinfloat[#2]{%
522 \xintMul{\XINT_c_logten}%
523 {\xintAdd{##1.##2##3##4##5##6##7}{##8[##9]}}%
524 }%
525 \xint:
526 }%
527 }}}} }}}} }%
528 }}\expandafter\XINT_tmpa
529 \the\numexpr\XINTdigitsormax+5\expandafter.%
530 \the\numexpr\XINTdigitsormax-1\expandafter.%
531 \the\numexpr\XINTdigitsormax.%

```

This rounding may produce -0.000000 but will always have 6 exactly fractional digits and a leading minus sign.

```

532 \def\XINT_powten_neg#1[#2]%
533 {%
534 \expandafter\XINT_powten_neg_a\romannumeral0\xintround{6}{#1[#2]}#1[#2]%
535 }%
536 \def\XINT_tmpa #1.#2.#3.{%
537 \def\XINT_powten_neg_a -##1.##2##3##4##5##6##7##8[##9]%
538 {%
539 \expandafter\XINT_infloate
540 \romannumeral0\XINTinfloat[#3]{%
541 \xintMul{\csname XINT_c_1_##2_inv\endcsname}{%
542 \XINTinFloat[#1]{%
543 \xintMul{\csname XINT_c_2_##3_inv\endcsname}{%
544 \XINTinFloat[#1]{%
545 \xintMul{\csname XINT_c_3_##4_inv\endcsname}{%
546 \XINTinFloat[#1]{%
547 \xintMul{\csname XINT_c_4_##5_inv\endcsname}{%
548 \XINTinFloat[#1]{%
549 \xintMul{\csname XINT_c_5_##6_inv\endcsname}{%
550 \XINTinFloat[#1]{%
551 \xintMul{\csname XINT_c_6_##7_inv\endcsname}{%
552 \xintAdd{1[0]}{%
553 \expandafter\XINT_Exp_series_a_ii
554 \romannumeral0\XINTinfloat[#2]{%
555 \xintMul{\XINT_c_logten}%
556 {\xintAdd{##1.##2##3##4##5##6##7}{##8[##9]}}%
557 }%

```

```

558             \xint:
559             }%
560             {}{}{}{}{}{}{}{}{}{}{-##1}%
561 }}\expandafter\xINT_tmpa
562   \the\numexpr\xINTdigitsormax+5\expandafter.%
563   \the\numexpr\xINTdigitsormax-1\expandafter.%
564   \the\numexpr\xINTdigitsormax.%

```

13.10.1 Exponential series

Or rather here $h(1 + h(1/2 + h(1/6 + \dots)))$. Upto at most $h^9/9!$ term.

The used initial h has been float rounded to $P-1$ digits.

```

565 \def\XINT_tmpa#1.#2.{%
566 \def\XINT_Exp_series_a_ii##1\xint:
567 {%
568     \expandafter\XINT_Exp_series_b
569     \romannumeral0\XINTinfloatS[#1]{##1}\xint:##1\xint:
570 }%
571 \def\XINT_Exp_series_b##1[##2]\xint:
572 {%
573     \expandafter\XINT_Exp_series_c_
574     \romannumeral0\xintadd{1}{\xintHalf{##10}[##2-1]}\xint:
575 }%
576 \def\XINT_Exp_series_c_##1\xint:##2\xint:
577 {%
578     \XINTinFloat[#2]{\xintMul{##1}{##2}}%
579 }%
580 }%
581 \expandafter\XINT_tmpa
582     \the\numexpr\XINTdigitsormax-6\expandafter.%
583     \the\numexpr\XINTdigitsormax-1.%
584 \ifnum\XINTdigits>15
585 \def\XINT_tmpa#1.#2.#3.#4.{%
586 \def\XINT_Exp_series_a_ii##1\xint:
587 {%
588     \expandafter\XINT_Exp_series_a_iii
589     \romannumeral0\XINTinfloatS[#2]{##1}\xint:##1\xint:
590 }%
591 \def\XINT_Exp_series_a_iii##1\xint:
592 {%
593     \expandafter\XINT_Exp_series_b
594     \romannumeral0\XINTinfloatS[#1]{##1}\xint:##1\xint:
595 }%
596 \def\XINT_Exp_series_b##1[##2]\xint:
597 {%
598     \expandafter\XINT_Exp_series_c_i
599     \romannumeral0\xintadd{#3}{##1/6[##2]}\xint:
600 }%
601 \def\XINT_Exp_series_c_i##1\xint:##2\xint:
602 {%
603     \expandafter\XINT_Exp_series_c_
604     \romannumeral0\xintadd{#4}{\XINTinFloat[#2]{\xintMul{##1}{##2}}}\xint:

```

```

605 }%
606 }\expandafter\XINT_tmpa
607 \the\numexpr\XINTdigitsormax-13\expandafter.%
608 \the\numexpr\XINTdigitsormax-6.%
609 {5[-1]}.%
610 {1[0]}.%
611 \fi
612 \ifnum\XINTdigits>21
613 \def\XINT_tmpa#1.#2.#3.#4.{%
614 \def\XINT_Exp_series_a_iii##1\xint:
615 {%
616     \expandafter\XINT_Exp_series_a_iv
617     \romannumeral0\XINTinfloatS[#2]{##1}\xint:##1\xint:
618 }%
619 \def\XINT_Exp_series_a_iv##1\xint:
620 {%
621     \expandafter\XINT_Exp_series_b
622     \romannumeral0\XINTinfloatS[#1]{##1}\xint:##1\xint:
623 }%
624 \def\XINT_Exp_series_b##1[##2]\xint:
625 {%
626     \expandafter\XINT_Exp_series_c_ii
627     \romannumeral0\xintadd{#3}{##1/24[##2]}\xint:
628 }%
629 \def\XINT_Exp_series_c_ii##1\xint:##2\xint:
630 {%
631     \expandafter\XINT_Exp_series_c_i
632     \romannumeral0\xintadd{#4}{\XINTinfloat[#2]{\xintMul{##1}{##2}}}\xint:
633 }%
634 }\expandafter\XINT_tmpa
635 \the\numexpr\XINTdigitsormax-19\expandafter.%
636 \the\numexpr\XINTdigitsormax-13\expandafter.%
637 \romannumeral0\XINTinfloat[\XINTdigitsormax-13]{1/6[0]}.%
638 {5[-1]}.%
639 \fi
640 \ifnum\XINTdigits>28
641 \def\XINT_tmpa #1 #2 #3 #4 #5 #6 #7 %
642 {%
643 \def\XINT_tmpb ##1##2##3##4%
644 {%
645 \def\XINT_tmpc####1.####2.####3.####4.%
646 {%
647 \def##2#####1\xint:
648 {%
649     \expandafter##1%
650     \romannumeral0\XINTinfloatS[####2]{#####1}\xint:#####1\xint:
651 }%
652 \def##1#####1\xint:
653 {%
654     \expandafter\XINT_Exp_series_b
655     \romannumeral0\XINTinfloatS[####1]{#####1}\xint:#####1\xint:
656 }%

```

```

657 \def\XINT_Exp_series_b#####1[#####2]\xint:
658 {%
659     \expandafter##3%
660     \romannumeral0\xintadd{####3}{#####1/#5[#####2]}\xint:
661 }%
662 \def##3#####1\xint:#####2\xint:
663 {%
664     \expandafter##4%
665     \romannumeral0\xintadd{####4}{\XINTinFloat[###2]{\xintMul{#####1}{#####2}}}\xint:
666 }%
667 }%
668 }%
669 \expandafter\XINT_tmpb
670 \csname XINT_Exp_series_a_\romannumeral\numexpr#1\expandafter\endcsname
671 \csname XINT_Exp_series_a_\romannumeral\numexpr#1-1\expandafter\endcsname
672 \csname XINT_Exp_series_c_\romannumeral\numexpr#1-2\expandafter\endcsname
673 \csname XINT_Exp_series_c_\romannumeral\numexpr#1-3\endcsname
674 \expandafter\XINT_tmpc
675 \the\numexpr\XINTdigitsormax-#2\expandafter.%
676 \the\numexpr\XINTdigitsormax-#3\expandafter.\expanded{%
677 \XINTinFloat[\XINTdigitsormax-#3]{1/#6[0]}.%
678 \XINTinFloat[\XINTdigitsormax-#4]{1/#7[0]}.%
679 }%
680 }%
681 \XINT_tmpa 5 26 19 13 120 24 6 %<-- keep space
682 \ifnum\XINTdigits>35 \XINT_tmpa 6 33 26 19 720 120 24 \fi
683 \ifnum\XINTdigits>42 \XINT_tmpa 7 40 33 26 5040 720 120 \fi
684 \ifnum\XINTdigits>48 \XINT_tmpa 8 46 40 33 40320 5040 720 \fi
685 \ifnum\XINTdigits>55 \XINT_tmpa 9 53 46 40 362880 40320 5040 \fi
686 \fi

```

13.11 April 2021: at last \XINTinFloagLogTen, \XINTinFloatLog

Attention that this is not supposed to be used with \XINTdigits at 8 or less, it will crash if that is the case. The log10() and log() functions in case \XINTdigits is at most 8 are mapped to \PoormanLogBaseTen respectively \PoormanLog macros.

In the explanations here I use the function names rather than the macro names.

Both log(x) and log10(x) are on top of an underlying macro which will produce z and h such that x is about $10^z e^h$ (with h being small is obtained via a log series). Then log(x) computes $\log(10)z+h$ whereas log10(x) computes as $z+h/\log(10)$.

There will be three branches [NO FINALLY ONLY TWO BRANCHES SINCE 1.4f] according to situation of x relative to 1. Let y be the math value log10(x) that we want to approximate to target precision P digits. P is assumed at least 9.

I will describe the algorithm roughly, but skip its underlying support analysis; at some point I mention "fixed point calculations", but in practice it is not done exactly that way, but describing it would be complicated so look at the code which is very readable (by the author, at the present time).

First we compute $z = \pm n.d_{1d_2}...d_6$ as the rounded to 6 fractional digits approximation of $y=\log_{10}(x)$ obtained by first using the poormanlog macros on x (float rounded to 9 digits) then rounding as above.

Warning: this description is not in sync with the code, now the case where $d_{1d_2}...d_6$ is 000000 is filtered out and one jumps directly either to case I if $n \neq 0$ or to case III if $n=0$. The case when

rounding produces a z equal to zero is also handled especially.

WARNING: at 1.4f, the CASE I was REMOVED. Everything is handled as CASE II or exceptionally case III. Indeed this removal was observed to simply cost about 10% extra time at D=16 digits, which was deemed an acceptable cost. The cost is certainly higher at D=9 but also relatively lower at high D's. It means that logarithms are always computed with 9, not 4, safety ****fractional**** digits, and this allows to compute powers accurately with exponents say up to $1e7$, degradation starting to show at $1e8$ and for sure at $1e9$. However for integer and half-integer exponents the old routine `\xintFloatPower` will still be used, and perhaps it will need some increased precision update as the documented 0.52ulp error bound is higher than our more stringent standards of 2021.

CASE I: [removed at 1.4f!] either n is NOT zero or `d_1d_2...d_6` is at least 100001. Then we compute $X = 10^{(-z)} * x$ which is near 1, by using the table of powers of 10, using P+5 digits significands. Then we compute (exactly) $\eta = X - 1$, (which is in absolute value less than 0.0000012) and obtain y as $z + \log(10)^{(-1)}$ times $\log(1+\eta)$ where $\log(1+\eta) = \eta - \eta^2/2 + \eta^3/3 - \dots$ is "computed with P+4 fractional fixed point digits" [1]_ according to the following table:

- maximal value of P: 9, 15, 21, 27, 33, 39, 45, 51, 57, 63
- last included term: /1, /2, /3, /4, /5, /6, /7, /8, /9, /10

.. [1] this "P+4" includes leading fractional zeroes so in practice it will rather be done as $\eta(1 - \eta(1/2 + \eta(1/3 - \dots)))$, and the inner sums will be done in various precisions, the top level (external) η probably at P-1 digits, the first inner η at P-7 digits, the next at P-13, something in this style. The heuristics is simple: at P=9 we don't need the first inner η , so let's use there P-9 or rather P-7 digits by security. Similarly at P=3 we would not need at all the η , so let's use the top level one rounded at $P-3+2 = P-1$ digits. And there is a shift by 6 less digits at each inner level. RÉFLÉCHIR SI C'EST PAS PLUTÔT P-2 ICI, suffisant au regard de la précision par ailleurs pour la réduction près de 1.

The sequence of maximal P's is simply an arithmetic progression.

The addition of z will trigger the final rounding to P digits. The inverse of $\log(10)$ is precomputed with P+4 digits.

This case I essentially handles x such as $\max(x, 1/x) > 10^{0.1} = 1.2589\dots$

CASE II: n is zero and `d_1d_2...d_6` is not zero. We operate as in CASE I, up to the following differences:

- the table of fractional powers of 10 is used with P+10 significands.
- the X is also computed with P+10 digits, i.e. $\eta = X - 1$ (which obeys the given estimate) is estimated with P+9 [2]_ fractional fixed points digits and the log series will be evaluated in this sense.

- the constant $\log(10)^{(-1)}$ is still used with only P+4 digits

The log series is terminated according to the following table:

- maximal value of P: 4, 10, 16, 22, 28, 34, 40, 46, 52, 58, 64
- last included term: /1, /2, /3, /4, /5, /6, /7, /8, /9, /10

Again the P's are in arithmetic progression, the same as before shifted by 5.

.. [2] same remark as above. The top level η in $\eta(1 - \eta(1/2 - \eta(\dots)))$ will use P+4 significant digits, but the first inner η will be used with only P-2 digits, the next inner one with P-8 digits etc...

This case II handles the x which are near 1, but not as close as $10^{\pm 0.000001}$.

CASE III: z=0. In this case $X = x = 1 + \eta$ and we use the log series in this sense : $\log(10)^{(-1)} * \eta * (1 - \eta/2 + \eta^2/3 - \dots)$ where again $\log(10)^{(-1)}$ has been precomputed with P+4 digits and morally the series uses P+4 fractional digits (P+3 would probably be enough for the precision I want, need to check my notes) and the thresholds table is:

- maximal value of P: 3, 9, 15, 21, 27, 33, 39, 45, 51, 57, 63
- last included term: /1, /2, /3, /4, /5, /6, /7, /8, /9, /10, /11

This is same progression but shifted by one.

To summarize some relevant aspects:

- this algorithm uses only $\log(10)^{(-1)}$ as precomputed logarithm

- in particular the logarithms of small integers 2, 3, 5, ... are not pre-computed. Added note: I have now tested at 16, 32, 48 and 62 digits that all of the $\log_{10}(n)$, for $n = 1..1000$, are computed with correct rounding. In fact, generally speaking, random testing of a about 20000 inputs has failed to reveal a single non-correct rounding. Naturally, randomly testing is not the way to corner the software into its weak points...

- it uses two tables of fractional powers of ten: one with P+5 digits and another one with extended precision at P+10 digits.

- it needs three distinct implementations of the log series.

- it does not use the well-known trick reducing to using only odd powers in the log series (somehow I have come to dread divisions, even though here as is well-known it could be replaced with some product, my impression was that what is gained on one side is lost on the other, for the range of P I am targeting, i.e. P up to about 60.)

- all of this is experimental (in particular the previous item was not done perhaps out of sheer laziness)

Absolutely no error check is done whether the input x is really positive. As seen above the maximal target precision is 63 (not 64).

Update for 1.4f: when the logarithm is computed via case I, i.e. basically always except roughly for $0.8 < a < 1.26$, its fractional part has only about 4 safety digits. This is barely enough for a^b with b near 1000 and certainly not enough for a^b with b of the order 10000.

I hesitated with the option to always handle b as N+h with N integer for which we can use old `\xintFloatPower` (which perhaps I will have to update to ensure better than the 0.52ulp it mentions in its documentation). But in the end, I decided to simply add a variant where case I is handled as case II, i.e. with 9 not 4 safety fractional digits for the logarithm. This variant will be the one used by the power function for fractional exponents (non integer, non half-integer).

```

687 \def\xINT_tmpa#1.{%
688 \def\xINTinFloatLog{\romannumeral0\xINTinfloatlog}%
689 \def\xINTinfloatlog
690 {%
691     \expandafter\xINT_log_out
692     \romannumeral0\expandafter\xINT_logtenxdg_a
693     \romannumeral0\xINTinfloat[#1]{##1}
694 }%
695 \def\xINT_log_out ##1\xint:##2\xint:
696 {%
697     \XINTinfloat[#1]%
698     {\xintAdd{\xintMul{\XINT_c_logten}{##1}}{##2}}%
699 }%
700 \def\xINTinFloatLogTen{\romannumeral0\xINTinfloatlogten}%
701 \def\xINTinfloatlogten
702 {%
703     \expandafter\xINT_logten_out
704     \romannumeral0\expandafter\xINT_logtenxdg_a
705     \romannumeral0\xINTinfloat[#1]{##1}
706 }%
707 \def\xINT_logten_out ##1\xint:##2\xint:
708 {%
709     \XINTinfloat[#1]%
710     {\xintAdd{##1}{\xintMul{\XINT_c_oneoverlogten}{##2}}}%
711 }%
712 }\expandafter\xINT_tmpa\the\numexpr\xINTdigitsormax.%
713 \def\xINTinFloatLogTen_xdgout%#1[#2]
714 {%

```

```

715 \romannumeral0\expandafter\XINT_logten_xdgout\romannumeral0\XINT_logtenxdg_a
716 }%
717 \def\XINT_logten_xdgout #1\xint:#2\xint:
718 {%
719 \xintadd{#1}{\xintMul{\XINT_c_oneoverlogten_xx}{#2}}%
720 }%

```

No check is done whether input is negative or vanishes. We apply `\XINTinfloat[9]` which if input is not zero always produces 9 digits (and perhaps a minus sign) the first digit is non-zero. This is the expected input to `\numexpr\PML@<digits><dot>.\relax`

The variants `xdg_a`, `xdg_b`, `xdg_c`, `xdg_d` were added at 1.4f to always go via II or III, ensuring more fractional digits to the logarithm for accuracy of fractional powers with big exponents. "Old" 1.4e routines were removed.

```

721 \def\XINT_logtenxdg_a#1[#2]%
722 {%
723 \expandafter\XINT_logtenxdg_b
724 \romannumeral0\XINTinfloat[9]{#1[#2]}#1[#2]%
725 }%
726 \def\XINT_logtenxdg_b#1[#2]%
727 {%
728 \expandafter\XINT_logtenxdg_c
729 \romannumeral0\xintround{6}%
730 {\xintiiAdd{\xintDSx{-9}}{\the\numexpr#2+8\relax}}%
731 {\the\numexpr\PML@#1.\relax}%
732 [-9]]%
733 \xint:
734 }%

```

If we were either in 100000000[0] or 999999999[-1] for the `#1[#2]` `\XINT_logten_b` input, and only in those cases, the `\xintRound{6}` produced "0". We are very near 1 and will treat this as case III, but this is sub-optimal.

```

735 \def\XINT_logtenxdg_c #1#2%
736 {%
737 \xint_gob_til_xint:#2\XINT_logten_IV\xint:
738 \XINT_logtenxdg_d #1#2%
739 }%
740 \def\XINT_logten_IV\xint:\XINT_logtenxdg_d0{\XINT_logten_f_III}%

```

Here we are certain that `\xintRound{6}` produced a decimal point and 6 fractional digit tokens #2, but they can be zeros and also -0.000000 is possible.

If #1 vanishes and #2>100000 we are in case I.

If #1 vanishes and 100000>=#2>0 we are in case II.

If #1 and #2 vanish we are in case III.

If #1 does not vanish we are in case I with a direct quicker access if #2 vanishes.

Attention to the sign of #1, it is checked later on.

At 1.4f, we handle the case I with as many digits as case II (and exceptionnally case III).

```

741 \def\XINT_logtenxdg_d #1.#2\xint:
742 {%
743 \ifcase
744 \ifnum#1=\xint_c_
745 \ifnum #2=\xint_c_ \xint_c_iii\else \xint_c_ii\fi
746 \else
747 \ifnum#2>\xint_c_ \xint_c_ii\else \xint_c_\fi
748 \fi

```



```

749     \expandafter\XINT_logten_f_Isp
750     \or% never
751     \or\expandafter\XINT_logten_f_IorII
752     \else\expandafter\XINT_logten_f_III
753     \fi
754     #1.#2\xint:
755 }%
756 \def\XINT_logten_f_IorII#1%
757 {%
758     \xint_UDsignfork
759     #1\XINT_logten_f_IorII_neg
760     -\XINT_logten_f_IorII_pos
761     \krof #1%
762 }%

```

We are here only with a non-zero ##1, so no risk of a -0[0] which would be illegal usage of A[N] raw format. A negative ##1 is no trouble in ##3-##1.

```

763 \def\XINT_tmpa#1.{%
764 \def\XINT_logten_f_Isp##1.000000\xint:##2[##3]%
765 {%
766     {##1[0]}\xint:
767     {\expandafter\XINT_LogTen_serII_a_ii
768         \romannumeral0\XINTinfloatS[#1]{\xintAdd{##2[##3-##1]}\{-1[0]}}}%
769     \xint:
770     }\xint:
771 }%
772 }\expandafter\XINT_tmpa\the\numexpr\XINTdigitsormax.%
773 \def\XINT_tmpa#1.{%
774 \def\XINT_logten_f_III##1\xint:##2[##3]%
775 {%
776     {0[0]}\xint:
777     {\expandafter\XINT_LogTen_serIII_a_ii
778         \romannumeral0\XINTinfloatS[#1]{\xintAdd{##2[##3]}\{-1[0]}}}%
779     \xint:
780     }\xint:
781 }}\expandafter\XINT_tmpa\the\numexpr\XINTdigitsormax+4.%
782 \def\XINT_tmpa#1.#2.{%
783 \def\XINT_logten_f_IorII_pos##1.##2##3##4##5##6##7\xint:##8[##9]%
784 {%
785     {\the\numexpr##1##2##3##4##5##6##7[-6]}\xint:
786     {\expandafter\XINT_LogTen_serII_a_ii
787         \romannumeral0\XINTinfloat[#2]%
788         {\xintAdd{-1[0]}}%
789         {\xintMul{\csname XINT_c_1_##2_inv_x\endcsname}{%
790             \XINTinFloat[#1]{%
791                 \xintMul{\csname XINT_c_2_##3_inv_x\endcsname}{%
792                     \XINTinFloat[#1]{%
793                         \xintMul{\csname XINT_c_3_##4_inv_x\endcsname}{%
794                             \XINTinFloat[#1]{%
795                                 \xintMul{\csname XINT_c_4_##5_inv_x\endcsname}{%
796                                     \XINTinFloat[#1]{%
797                                         \xintMul{\csname XINT_c_5_##6_inv_x\endcsname}{%
798                                             \XINTinFloat[#1]{%

```

```

799          \xintMul{\csname XINT_c_6_##7_inv_x\endcsname}
800              {##8[##9-##1]}%
801          }%
802      }%
803      }\xint:
804      }\xint:
805 }%
806 \def\XINT_logten_f_IorII_neg##1.##2##3##4##5##6##7\xint:##8[##9]%
807 {%
808     {\the\numexpr##1##2##3##4##5##6##7[-6]}\xint:
809     {\expandafter\XINT_LogTen_serII_a_ii
810       \romannumeral0\XINTinfloat[#2]}%
811     {\xintAdd{-1[0]}%
812       {\xintMul{\csname XINT_c_1_##2_x\endcsname}{%
813         \XINTinFloat[#1]}%
814         \xintMul{\csname XINT_c_2_##3_x\endcsname}{%
815           \XINTinFloat[#1]}%
816           \xintMul{\csname XINT_c_3_##4_x\endcsname}{%
817             \XINTinFloat[#1]}%
818             \xintMul{\csname XINT_c_4_##5_x\endcsname}{%
819               \XINTinFloat[#1]}%
820               \xintMul{\csname XINT_c_5_##6_x\endcsname}{%
821                 \XINTinFloat[#1]}%
822                 \xintMul{\csname XINT_c_6_##7_x\endcsname}
823                   {##8[##9-##1]}%
824                 }%
825             }%
826           }\xint:
827         }\xint:
828 }%
829 }\expandafter\XINT_tmpa
830 \the\numexpr\XINTdigitsormax+10\expandafter.\the\numexpr\XINTdigitsormax+4.%

```

Initially all of this was done in a single big nested macro but the float-rounding of argument to less digits worked again each time from initial long input; the advantage on the other hand was that the 1/i constants were all pre-computed and rounded.

Pre-coding the successive rounding to six digits less at each stage could be done via a single loop which would then walk back up inserting coeffs like 1/#1 having no special optimizing tricks. Pre-computing the 1/#1 too is possible but then one would have to copy the full set of such constants (which would be pre-computed depending on P), and this will add grabbing overhead in the loop expansion. Or one defines macros to hold the pre-rounded constants.

Finally I do define macros, not only to hold the constants but to hold the whole build-up. Sacrificing brevity of code to benefit of expansion "speed".

Firts one prepares eta, with P+4 digits for mantissa, and then hands it over to the log series. This will proceed via first preparing eta\xint: eta\xint: ... eta\xint:, the leftmost ones being more and more reduced in number of digits. Finally one goes back up to the right, the hard-coded number of steps depending on value of P=\XINTdigits at time of reloading of package. This number of steps is hard-coded in the number of macros which get defined.

Descending (leftwards) chain: _a, Turning point: _b, Ascending: _c.

As it is very easy to make silly typing mistakes in the numerous macros I have refactored a number of times the set-up to make manual verification straightforward. Automatization is possible but the _b macros complicate things, each one is its own special case. In the end the set-up will define then redefine some _a and the (finally unique) _b macro, this allows easier to read code, with no

nesting of conditionals or else branches.

Actually series III and series II differ by only a shift by and we could use always the slightly more costly series III in place of series II. But that would add one un-needed term and a bit overhead to the default P which is 16...

(1.4f: hesitation on 2021/05/09 after removal or case I log series should I not follow the simplifying logic and use always the slightly more costly III?)

13.11.1 Log series, case II

```

831 \def\XINT_tmpa#1.#2.{%
832 \def\XINT_LogTen_serII_a_ii##1\xint:
833 {%
834     \expandafter\XINT_LogTen_serII_b
835     \romannumeral0\XINTinfloatS[#1]{##1}\xint:##1\xint:
836 }%
837 \def\XINT_LogTen_serII_b#1[#2]\xint:
838 {%
839     \expandafter\XINT_LogTen_serII_c_
840     \romannumeral0\xintadd{1}{\xintiiOpp\xintHalf{#10}[#2-1]}\xint:
841 }%
842 \def\XINT_LogTen_serII_c_##1\xint:##2\xint:
843 {%
844     \XINTinFloat[#2]{\xintMul{##1}{##2}}%
845 }%
846 }%
847 \expandafter\XINT_tmpa
848     \the\numexpr\XINTdigitsormax-2\expandafter.%
849     \the\numexpr\XINTdigitsormax+4.%
850 \ifnum\XINTdigits>10
851 \def\XINT_tmpa#1.#2.#3.#4.{%
852 \def\XINT_LogTen_serII_a_ii##1\xint:
853 {%
854     \expandafter\XINT_LogTen_serII_a_iii
855     \romannumeral0\XINTinfloatS[#2]{##1}\xint:##1\xint:
856 }%
857 \def\XINT_LogTen_serII_a_iii##1\xint:
858 {%
859     \expandafter\XINT_LogTen_serII_b
860     \romannumeral0\XINTinfloatS[#1]{##1}\xint:##1\xint:
861 }%
862 \def\XINT_LogTen_serII_b##1[##2]\xint:
863 {%
864     \expandafter\XINT_LogTen_serII_c_i
865     \romannumeral0\xintadd{#3}{##1/3[##2]}\xint:
866 }%
867 \def\XINT_LogTen_serII_c_i##1\xint:##2\xint:
868 {%
869     \expandafter\XINT_LogTen_serII_c_
870     \romannumeral0\xintadd{#4}{\XINTinFloat[#2]{\xintMul{##1}{##2}}}\xint:
871 }%
872 }\expandafter\XINT_tmpa
873     \the\numexpr\XINTdigitsormax-8\expandafter.%
874     \the\numexpr\XINTdigitsormax-2.%

```

```

875 {-5[-1]}.%
876 {1[0]}.%
877 \fi
878 \ifnum\XINTdigits>16
879 \def\XINT_tmpa#1.#2.#3.#4.{%
880 \def\XINT_LogTen_serII_a_iii##1\xint:
881 {%
882     \expandafter\XINT_LogTen_serII_a_iv
883     \romannumeral0\XINTinfloatS[#2]{##1}\xint:##1\xint:
884 }%
885 \def\XINT_LogTen_serII_a_iv##1\xint:
886 {%
887     \expandafter\XINT_LogTen_serII_b
888     \romannumeral0\XINTinfloatS[#1]{##1}\xint:##1\xint:
889 }%
890 \def\XINT_LogTen_serII_b##1[##2]\xint:
891 {%
892     \expandafter\XINT_LogTen_serII_c_ii
893     \romannumeral0\xintadd{#3}{\xintiMul{-25}{##1}[##2-2]}\xint:
894 }%
895 \def\XINT_LogTen_serII_c_ii##1\xint:##2\xint:
896 {%
897     \expandafter\XINT_LogTen_serII_c_i
898     \romannumeral0\xintadd{#4}{\XINTinFloat[#2]{\xintMul{##1}{##2}}}\xint:
899 }%
900 }\expandafter\XINT_tmpa
901 \the\numexpr\XINTdigitsormax-14\expandafter.%
902 \the\numexpr\XINTdigitsormax-8\expandafter.%
903 \romannumeral0\XINTinfloat[\XINTdigitsormax-8]{1/3[0]}.%
904 {-5[-1]}.%
905 \fi
906 \ifnum\XINTdigits>22
907 \def\XINT_tmpa#1.#2.#3.#4.{%
908 \def\XINT_LogTen_serII_a_iv##1\xint:
909 {%
910     \expandafter\XINT_LogTen_serII_a_v
911     \romannumeral0\XINTinfloatS[#2]{##1}\xint:##1\xint:
912 }%
913 \def\XINT_LogTen_serII_a_v##1\xint:
914 {%
915     \expandafter\XINT_LogTen_serII_b
916     \romannumeral0\XINTinfloatS[#1]{##1}\xint:##1\xint:
917 }%
918 \def\XINT_LogTen_serII_b##1[##2]\xint:
919 {%
920     \expandafter\XINT_LogTen_serII_c_iii
921     \romannumeral0\xintadd{#3}{\xintDouble{##1}[##2-1]}\xint:
922 }%
923 \def\XINT_LogTen_serII_c_iii##1\xint:##2\xint:
924 {%
925     \expandafter\XINT_LogTen_serII_c_ii
926     \romannumeral0\xintadd{#4}{\XINTinFloat[#2]{\xintMul{##1}{##2}}}\xint:

```

```

927 }%
928 }\expandafter\XINT_tmpa
929 \the\numexpr\XINTdigitsormax-20\expandafter.%
930 \the\numexpr\XINTdigitsormax-14\expandafter.\expanded{%
931 {-25[-2]}.%
932 \XINTinFloat[\XINTdigitsormax-8]{1/3[0]}.%
933 }%
934 \fi
935 \ifnum\XINTdigits>28
936 \def\XINT_tmpa#1.#2.#3.#4.{%
937 \def\XINT_LogTen_serII_a_v##1\xint:
938 {%
939     \expandafter\XINT_LogTen_serII_a_vi
940     \romannumeral0\XINTinfloatS[#2]{##1}\xint:##1\xint:
941 }%
942 \def\XINT_LogTen_serII_a_vi##1\xint:
943 {%
944     \expandafter\XINT_LogTen_serII_b
945     \romannumeral0\XINTinfloatS[#1]{##1}\xint:##1\xint:
946 }%
947 \def\XINT_LogTen_serII_b##1[##2]\xint:
948 {%
949     \expandafter\XINT_LogTen_serII_c_iv
950     \romannumeral0\xintadd{#3}{\xintiiOpp##1/6[##2]}\xint:
951 }%
952 \def\XINT_LogTen_serII_c_iv##1\xint:##2\xint:
953 {%
954     \expandafter\XINT_LogTen_serII_c_iii
955     \romannumeral0\xintadd{#4}{\XINTinFloat[#2]{\xintMul{##1}{##2}}}\xint:
956 }%
957 }\expandafter\XINT_tmpa
958 \the\numexpr\XINTdigitsormax-26\expandafter.%
959 \the\numexpr\XINTdigitsormax-20.%
960 {2[-1]}.%
961 {-25[-2]}.%
962 \fi
963 \ifnum\XINTdigits>34
964 \def\XINT_tmpa#1.#2.#3.#4.{%
965 \def\XINT_LogTen_serII_a_vi##1\xint:
966 {%
967     \expandafter\XINT_LogTen_serII_a_vii
968     \romannumeral0\XINTinfloatS[#2]{##1}\xint:##1\xint:
969 }%
970 \def\XINT_LogTen_serII_a_vii##1\xint:
971 {%
972     \expandafter\XINT_LogTen_serII_b
973     \romannumeral0\XINTinfloatS[#1]{##1}\xint:##1\xint:
974 }%
975 \def\XINT_LogTen_serII_b##1[##2]\xint:
976 {%
977     \expandafter\XINT_LogTen_serII_c_v
978     \romannumeral0\xintadd{#3}{##1/7[##2]}\xint:

```

```

979 }%
980 \def\XINT_LogTen_serII_c_v##1\xint:##2\xint:
981 {%
982     \expandafter\XINT_LogTen_serII_c_iv
983     \romannumeral0\xintadd{#4}{\XINTinFloat[#2]{\xintMul{##1}{##2}}}\xint:
984 }%
985 }\expandafter\XINT_tmpa
986 \the\numexpr\XINTdigitsormax-32\expandafter.%
987 \the\numexpr\XINTdigitsormax-26\expandafter.%
988 \romannumeral0\XINTinfloatS[\XINTdigitsormax-26]{-1/6[0]}.%
989 {2[-1]}.%
990 \fi
991 \ifnum\XINTdigits>40
992 \def\XINT_tmpa#1.#2.#3.#4.{%
993 \def\XINT_LogTen_serII_a_vii##1\xint:
994 {%
995     \expandafter\XINT_LogTen_serII_a_viii
996     \romannumeral0\XINTinfloatS[#2]{##1}\xint:##1\xint:
997 }%
998 \def\XINT_LogTen_serII_a_viii##1\xint:
999 {%
1000     \expandafter\XINT_LogTen_serII_b
1001     \romannumeral0\XINTinfloatS[#1]{##1}\xint:##1\xint:
1002 }%
1003 \def\XINT_LogTen_serII_b##1[##2]\xint:
1004 {%
1005     \expandafter\XINT_LogTen_serII_c_vi
1006     \romannumeral0\xintadd{#3}{\xintiiMul{-125}{##1}[##2-3]}\xint:
1007 }%
1008 \def\XINT_LogTen_serII_c_vi##1\xint:##2\xint:
1009 {%
1010     \expandafter\XINT_LogTen_serII_c_v
1011     \romannumeral0\xintadd{#4}{\XINTinFloat[#2]{\xintMul{##1}{##2}}}\xint:
1012 }%
1013 }\expandafter\XINT_tmpa
1014 \the\numexpr\XINTdigitsormax-38\expandafter.%
1015 \the\numexpr\XINTdigitsormax-32\expandafter.\expanded{%
1016 \XINTinFloat[\XINTdigitsormax-32]{1/7[0]}.%
1017 \XINTinFloat[\XINTdigitsormax-26]{-1/6[0]}.%
1018 }%
1019 \fi
1020 \ifnum\XINTdigits>46
1021 \def\XINT_tmpa#1.#2.#3.#4.{%
1022 \def\XINT_LogTen_serII_a_viii##1\xint:
1023 {%
1024     \expandafter\XINT_LogTen_serII_a_ix
1025     \romannumeral0\XINTinfloatS[#2]{##1}\xint:##1\xint:
1026 }%
1027 \def\XINT_LogTen_serII_a_ix##1\xint:
1028 {%
1029     \expandafter\XINT_LogTen_serII_b
1030     \romannumeral0\XINTinfloatS[#1]{##1}\xint:##1\xint:

```

```

1031 }%
1032 \def\XINT_LogTen_serII_b##1[##2]\xint:
1033 {%
1034     \expandafter\XINT_LogTen_serII_c_vii
1035     \romannumeral0\xintadd{#3}{##1/9[##2]}\xint:
1036 }%
1037 \def\XINT_LogTen_serII_c_vii##1\xint:##2\xint:
1038 {%
1039     \expandafter\XINT_LogTen_serII_c_vii
1040     \romannumeral0\xintadd{#4}{\XINTinFloat[#2]{\xintMul{##1}{##2}}}\xint:
1041 }%
1042 }\expandafter\XINT_tmpa
1043 \the\numexpr\XINTdigitsormax-44\expandafter.%
1044 \the\numexpr\XINTdigitsormax-38\expandafter.\expanded{%
1045 {-125[-3]}.%
1046 \XINTinFloat[\XINTdigitsormax-32]{1/7[0]}.%
1047 }%
1048 \fi
1049 \ifnum\XINTdigits>52
1050 \def\XINT_tmpa#1.#2.#3.#4.{%
1051 \def\XINT_LogTen_serII_a_ix##1\xint:
1052 {%
1053     \expandafter\XINT_LogTen_serII_a_x
1054     \romannumeral0\XINTinfloatS[#2]{##1}\xint:##1\xint:
1055 }%
1056 \def\XINT_LogTen_serII_a_x##1\xint:
1057 {%
1058     \expandafter\XINT_LogTen_serII_b
1059     \romannumeral0\XINTinfloatS[#1]{##1}\xint:##1\xint:
1060 }%
1061 \def\XINT_LogTen_serII_b##1[##2]\xint:
1062 {%
1063     \expandafter\XINT_LogTen_serII_c_viii
1064     \romannumeral0\xintadd{#3}{\xintiiOpp##1[##2-1]}\xint:
1065 }%
1066 \def\XINT_LogTen_serII_c_viii##1\xint:##2\xint:
1067 {%
1068     \expandafter\XINT_LogTen_serII_c_vii
1069     \romannumeral0\xintadd{#4}{\XINTinFloat[#2]{\xintMul{##1}{##2}}}\xint:
1070 }%
1071 }\expandafter\XINT_tmpa
1072 \the\numexpr\XINTdigitsormax-50\expandafter.%
1073 \the\numexpr\XINTdigitsormax-44\expandafter.%
1074 \romannumeral0\XINTinfloat[\XINTdigitsormax-44]{1/9[0]}.%
1075 {-125[-3]}.%
1076 \fi
1077 \ifnum\XINTdigits>58
1078 \def\XINT_tmpa#1.#2.#3.#4.{%
1079 \def\XINT_LogTen_serII_a_x##1\xint:
1080 {%
1081     \expandafter\XINT_LogTen_serII_a_xi
1082     \romannumeral0\XINTinfloatS[#2]{##1}\xint:##1\xint:

```

```

1083 }%
1084 \def\XINT_LogTen_serII_a_xi##1\xint:
1085 {%
1086     \expandafter\XINT_LogTen_serII_b
1087     \romannumeral0\XINTinfloatS[#1]{##1}\xint:##1\xint:
1088 }%
1089 \def\XINT_LogTen_serII_b##1[##2]\xint:
1090 {%
1091     \expandafter\XINT_LogTen_serII_c_ix
1092     \romannumeral0\xintadd{#3}{##1/11[##2]}\xint:
1093 }%
1094 \def\XINT_LogTen_serII_c_ix##1\xint:##2\xint:
1095 {%
1096     \expandafter\XINT_LogTen_serII_c_viii
1097     \romannumeral0\xintadd{#4}{\XINTinFloat[#2]{\xintMul{##1}{##2}}}\xint:
1098 }%
1099 }\expandafter\XINT_tmpa
1100 \the\numexpr\XINTdigitsormax-56\expandafter.%
1101 \the\numexpr\XINTdigitsormax-50\expandafter.\expanded{%
1102 {-1[-1]}.%
1103 \XINTinFloat[\XINTdigitsormax-44]{1/9[0]}.%
1104 }%
1105 \fi

```

13.11.2 Log series, case III

```

1106 \def\XINT_tmpa#1.#2.{%
1107 \def\XINT_LogTen_serIII_a_ii##1\xint:
1108 {%
1109     \expandafter\XINT_LogTen_serIII_b
1110     \romannumeral0\XINTinfloatS[#1]{##1}\xint:##1\xint:
1111 }%
1112 \def\XINT_LogTen_serIII_b#1[##2]\xint:
1113 {%
1114     \expandafter\XINT_LogTen_serIII_c_
1115     \romannumeral0\xintadd{1}{\xintiiOpp\xintHalf{#10}[#2-1]}\xint:
1116 }%
1117 \def\XINT_LogTen_serIII_c_##1\xint:##2\xint:
1118 {%
1119     \XINTinFloat[#2]{\xintMul{##1}{##2}}%
1120 }%
1121 }%
1122 \expandafter\XINT_tmpa
1123     \the\numexpr\XINTdigitsormax-1\expandafter.%
1124     \the\numexpr\XINTdigitsormax+4.%
1125 \ifnum\XINTdigits>9
1126 \def\XINT_tmpa#1.#2.#3.#4.{%
1127 \def\XINT_LogTen_serIII_a_ii##1\xint:
1128 {%
1129     \expandafter\XINT_LogTen_serIII_a_iii
1130     \romannumeral0\XINTinfloatS[#2]{##1}\xint:##1\xint:
1131 }%
1132 \def\XINT_LogTen_serIII_a_iii##1\xint:

```



```

1133 {%
1134     \expandafter\XINT_LogTen_serIII_b
1135     \romannumeral0\XINTinfloatS[#1]{##1}\xint:##1\xint:
1136 }%
1137 \def\XINT_LogTen_serIII_b##1[##2]\xint:
1138 {%
1139     \expandafter\XINT_LogTen_serIII_c_i
1140     \romannumeral0\xintadd{#3}{##1/3[##2]}\xint:
1141 }%
1142 \def\XINT_LogTen_serIII_c_i##1\xint:##2\xint:
1143 {%
1144     \expandafter\XINT_LogTen_serIII_c_
1145     \romannumeral0\xintadd{#4}{\XINTinFloat[#2]{\xintMul{##1}{##2}}}\xint:
1146 }%
1147 }\expandafter\XINT_tmpa
1148 \the\numexpr\XINTdigitsormax-7\expandafter.%
1149 \the\numexpr\XINTdigitsormax-1.%
1150 {-5[-1]}.%
1151 {1[0]}.%
1152 \fi
1153 \ifnum\XINTdigits>15
1154 \def\XINT_tmpa#1.#2.#3.#4.{%
1155 \def\XINT_LogTen_serIII_a_iii##1\xint:
1156 {%
1157     \expandafter\XINT_LogTen_serIII_a_iv
1158     \romannumeral0\XINTinfloatS[#2]{##1}\xint:##1\xint:
1159 }%
1160 \def\XINT_LogTen_serIII_a_iv##1\xint:
1161 {%
1162     \expandafter\XINT_LogTen_serIII_b
1163     \romannumeral0\XINTinfloatS[#1]{##1}\xint:##1\xint:
1164 }%
1165 \def\XINT_LogTen_serIII_b##1[##2]\xint:
1166 {%
1167     \expandafter\XINT_LogTen_serIII_c_ii
1168     \romannumeral0\xintadd{#3}{\xintiiMul{-25}{##1}[##2-2]}\xint:
1169 }%
1170 \def\XINT_LogTen_serIII_c_ii##1\xint:##2\xint:
1171 {%
1172     \expandafter\XINT_LogTen_serIII_c_i
1173     \romannumeral0\xintadd{#4}{\XINTinFloat[#2]{\xintMul{##1}{##2}}}\xint:
1174 }%
1175 }\expandafter\XINT_tmpa
1176 \the\numexpr\XINTdigitsormax-13\expandafter.%
1177 \the\numexpr\XINTdigitsormax-7\expandafter.%
1178 \romannumeral0\XINTinfloat[\XINTdigitsormax-7]{1/3[0]}.%
1179 {-5[-1]}.%
1180 \fi
1181 \ifnum\XINTdigits>21
1182 \def\XINT_tmpa#1.#2.#3.#4.{%
1183 \def\XINT_LogTen_serIII_a_iv##1\xint:
1184 {%

```

```

1185 \expandafter\XINT_LogTen_serIII_a_v
1186 \romannumeral0\XINTinfloatS[#2]{##1}\xint:##1\xint:
1187 }%
1188 \def\XINT_LogTen_serIII_a_v##1\xint:
1189 {%
1190 \expandafter\XINT_LogTen_serIII_b
1191 \romannumeral0\XINTinfloatS[#1]{##1}\xint:##1\xint:
1192 }%
1193 \def\XINT_LogTen_serIII_b##1[##2]\xint:
1194 {%
1195 \expandafter\XINT_LogTen_serIII_c_iii
1196 \romannumeral0\xintadd{#3}{\xintDouble{##1}[##2-1]}\xint:
1197 }%
1198 \def\XINT_LogTen_serIII_c_iii##1\xint:##2\xint:
1199 {%
1200 \expandafter\XINT_LogTen_serIII_c_ii
1201 \romannumeral0\xintadd{#4}{\XINTinFloat[#2]{\xintMul{##1}{##2}}}\xint:
1202 }%
1203 }\expandafter\XINT_tmpa
1204 \the\numexpr\XINTdigitsormax-19\expandafter.%
1205 \the\numexpr\XINTdigitsormax-13\expandafter.\expanded{%
1206 {-25[-2]}.%
1207 \XINTinFloat[\XINTdigitsormax-7]{1/3[0]}.%
1208 }%
1209 \fi
1210 \ifnum\XINTdigits>27
1211 \def\XINT_tmpa#1.#2.#3.#4.{%
1212 \def\XINT_LogTen_serIII_a_v##1\xint:
1213 {%
1214 \expandafter\XINT_LogTen_serIII_a_vi
1215 \romannumeral0\XINTinfloatS[#2]{##1}\xint:##1\xint:
1216 }%
1217 \def\XINT_LogTen_serIII_a_vi##1\xint:
1218 {%
1219 \expandafter\XINT_LogTen_serIII_b
1220 \romannumeral0\XINTinfloatS[#1]{##1}\xint:##1\xint:
1221 }%
1222 \def\XINT_LogTen_serIII_b##1[##2]\xint:
1223 {%
1224 \expandafter\XINT_LogTen_serIII_c_iv
1225 \romannumeral0\xintadd{#3}{\xintiiOpp##1/6[##2]}\xint:
1226 }%
1227 \def\XINT_LogTen_serIII_c_iv##1\xint:##2\xint:
1228 {%
1229 \expandafter\XINT_LogTen_serIII_c_iii
1230 \romannumeral0\xintadd{#4}{\XINTinFloat[#2]{\xintMul{##1}{##2}}}\xint:
1231 }%
1232 }\expandafter\XINT_tmpa
1233 \the\numexpr\XINTdigitsormax-25\expandafter.%
1234 \the\numexpr\XINTdigitsormax-19.%
1235 {2[-1]}.%
1236 {-25[-2]}.%

```

```

1237 \fi
1238 \ifnum\XINTdigits>33
1239 \def\XINT_tmpa#1.#2.#3.#4.{%
1240 \def\XINT_LogTen_serIII_a_vii##1\xint:
1241 {%
1242     \expandafter\XINT_LogTen_serIII_a_vii
1243     \romannumeral0\XINTinfloatS[#2]{##1}\xint:##1\xint:
1244 }%
1245 \def\XINT_LogTen_serIII_a_vii##1\xint:
1246 {%
1247     \expandafter\XINT_LogTen_serIII_b
1248     \romannumeral0\XINTinfloatS[#1]{##1}\xint:##1\xint:
1249 }%
1250 \def\XINT_LogTen_serIII_b##1[##2]\xint:
1251 {%
1252     \expandafter\XINT_LogTen_serIII_c_v
1253     \romannumeral0\xintadd{#3}{##1/7[##2]}\xint:
1254 }%
1255 \def\XINT_LogTen_serIII_c_v##1\xint:##2\xint:
1256 {%
1257     \expandafter\XINT_LogTen_serIII_c_iv
1258     \romannumeral0\xintadd{#4}{\XINTinFloat[#2]{\xintMul{##1}{##2}}}\xint:
1259 }%
1260 }\expandafter\XINT_tmpa
1261 \the\numexpr\XINTdigitsormax-31\expandafter.%
1262 \the\numexpr\XINTdigitsormax-25\expandafter.%
1263 \romannumeral0\XINTinfloatS[\XINTdigitsormax-25]{-1/6[0]}.%
1264 {2[-1]}.%
1265 \fi
1266 \ifnum\XINTdigits>39
1267 \def\XINT_tmpa#1.#2.#3.#4.{%
1268 \def\XINT_LogTen_serIII_a_viii##1\xint:
1269 {%
1270     \expandafter\XINT_LogTen_serIII_a_viii
1271     \romannumeral0\XINTinfloatS[#2]{##1}\xint:##1\xint:
1272 }%
1273 \def\XINT_LogTen_serIII_a_viii##1\xint:
1274 {%
1275     \expandafter\XINT_LogTen_serIII_b
1276     \romannumeral0\XINTinfloatS[#1]{##1}\xint:##1\xint:
1277 }%
1278 \def\XINT_LogTen_serIII_b##1[##2]\xint:
1279 {%
1280     \expandafter\XINT_LogTen_serIII_c_vi
1281     \romannumeral0\xintadd{#3}{\xintiiMul{-125}{##1}[##2-3]}\xint:
1282 }%
1283 \def\XINT_LogTen_serIII_c_vi##1\xint:##2\xint:
1284 {%
1285     \expandafter\XINT_LogTen_serIII_c_v
1286     \romannumeral0\xintadd{#4}{\XINTinFloat[#2]{\xintMul{##1}{##2}}}\xint:
1287 }%
1288 }\expandafter\XINT_tmpa

```

```

1289 \the\numexpr\XINTdigitsormax-37\expandafter.%
1290 \the\numexpr\XINTdigitsormax-31\expandafter.\expanded{%
1291 \XINTinFloat[\XINTdigitsormax-31]{1/7[0]}.%
1292 \XINTinFloat[\XINTdigitsormax-25]{-1/6[0]}.%
1293 }%
1294 \fi
1295 \ifnum\XINTdigits>45
1296 \def\XINT_tmpa#1.#2.#3.#4.{%
1297 \def\XINT_LogTen_serIII_a_viii##1\xint:
1298 {%
1299 \expandafter\XINT_LogTen_serIII_a_ix
1300 \romannumeral0\XINTinfloatS[#2]{##1}\xint:##1\xint:
1301 }%
1302 \def\XINT_LogTen_serIII_a_ix##1\xint:
1303 {%
1304 \expandafter\XINT_LogTen_serIII_b
1305 \romannumeral0\XINTinfloatS[#1]{##1}\xint:##1\xint:
1306 }%
1307 \def\XINT_LogTen_serIII_b##1[##2]\xint:
1308 {%
1309 \expandafter\XINT_LogTen_serIII_c_vii
1310 \romannumeral0\xintadd{#3}{##1/9[##2]}\xint:
1311 }%
1312 \def\XINT_LogTen_serIII_c_vii##1\xint:##2\xint:
1313 {%
1314 \expandafter\XINT_LogTen_serIII_c_vi
1315 \romannumeral0\xintadd{#4}{\XINTinFloat[#2]{\xintMul{##1}{##2}}}\xint:
1316 }%
1317 }\expandafter\XINT_tmpa
1318 \the\numexpr\XINTdigitsormax-43\expandafter.%
1319 \the\numexpr\XINTdigitsormax-37\expandafter.\expanded{%
1320 {-125[-3]}.%
1321 \XINTinFloat[\XINTdigitsormax-31]{1/7[0]}.%
1322 }%
1323 \fi
1324 \ifnum\XINTdigits>51
1325 \def\XINT_tmpa#1.#2.#3.#4.{%
1326 \def\XINT_LogTen_serIII_a_ix##1\xint:
1327 {%
1328 \expandafter\XINT_LogTen_serIII_a_x
1329 \romannumeral0\XINTinfloatS[#2]{##1}\xint:##1\xint:
1330 }%
1331 \def\XINT_LogTen_serIII_a_x##1\xint:
1332 {%
1333 \expandafter\XINT_LogTen_serIII_b
1334 \romannumeral0\XINTinfloatS[#1]{##1}\xint:##1\xint:
1335 }%
1336 \def\XINT_LogTen_serIII_b##1[##2]\xint:
1337 {%
1338 \expandafter\XINT_LogTen_serIII_c_viii
1339 \romannumeral0\xintadd{#3}{\xintiiOpp##1[##2-1]}\xint:
1340 }%

```

```

1341 \def\XINT_LogTen_serIII_c_viii##1\xint:##2\xint:
1342 {%
1343     \expandafter\XINT_LogTen_serIII_c_vii
1344     \romannumeral0\xintadd{#4}{\XINTinFloat[#2]{\xintMul{##1}{##2}}}\xint:
1345 }%
1346 }\expandafter\XINT_tmpa
1347 \the\numexpr\XINTdigitsormax-49\expandafter.%
1348 \the\numexpr\XINTdigitsormax-43\expandafter.%
1349 \romannumeral0\XINTinfloat[\XINTdigitsormax-43]{1/9[0]}.%
1350 {-125[-3]}.%
1351 \fi
1352 \ifnum\XINTdigits>57
1353 \def\XINT_tmpa#1.#2.#3.#4.{%
1354 \def\XINT_LogTen_serIII_a_x##1\xint:
1355 {%
1356     \expandafter\XINT_LogTen_serIII_a_xi
1357     \romannumeral0\XINTinfloatS[#2]{##1}\xint:##1\xint:
1358 }%
1359 \def\XINT_LogTen_serIII_a_xi##1\xint:
1360 {%
1361     \expandafter\XINT_LogTen_serIII_b
1362     \romannumeral0\XINTinfloatS[#1]{##1}\xint:##1\xint:
1363 }%
1364 \def\XINT_LogTen_serIII_b##1[##2]\xint:
1365 {%
1366     \expandafter\XINT_LogTen_serIII_c_ix
1367     \romannumeral0\xintadd{#3}{##1/11[##2]}\xint:
1368 }%
1369 \def\XINT_LogTen_serIII_c_ix##1\xint:##2\xint:
1370 {%
1371     \expandafter\XINT_LogTen_serIII_c_viii
1372     \romannumeral0\xintadd{#4}{\XINTinFloat[#2]{\xintMul{##1}{##2}}}\xint:
1373 }%
1374 }\expandafter\XINT_tmpa
1375 \the\numexpr\XINTdigitsormax-55\expandafter.%
1376 \the\numexpr\XINTdigitsormax-49\expandafter.\expanded{%
1377 {-1[-1]}.%
1378 \XINTinFloat[\XINTdigitsormax-43]{1/9[0]}.%
1379 }%
1380 \fi
1381 \XINTendxintloginput%

```

14 Cumulative line count

`xintkernel`: 631. Total number of code lines: 18625. (but 4293 lines among them
`xinttools`:1629. start either with `{%` or with `%}`.)
`xintcore`:2172. Each package starts with circa 50 lines dealing with cat-
`xint`:1624. codes, package identification and reloading management,
`xintbinhex`: 472. also for Plain TeX. Version 1.4i of 2021/06/11.
`xintgcd`: 368.
`xintfrac`:3590.
`xintseries`: 386.
`xintcfrac`:1029.
`xintexpr`:4496.
`xinttrig`: 847.
`xintlog`:1381.