

The `xint` packages source code

JEAN-FRANÇOIS BURNOL

jfbu (at) free (dot) fr

Package version: 1.4b (2020/02/25); documentation date: 2020/02/25.

From source file `xint.dtx`. Time-stamp: <25-02-2020 at 23:06:44 CET>.

Contents

1	Timeline (in brief)	2
2	Package <code>xintkernel</code> implementation	4
3	Package <code>xinttools</code> implementation	20
4	Package <code>xintcore</code> implementation	62
5	Package <code>xint</code> implementation	120
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	277
10	Package <code>xintcfrac</code> implementation	286
11	Package <code>xintexpr</code> implementation	309
12	Package <code>xinttrig</code> implementation	421
13	Package <code>xintlog</code> implementation	438
14	Cumulative line count	443

1 Timeline (in brief)

This is 1.4b of 2020/02/25.

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

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

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

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

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

Warning: pay attention when looking at the code to the catcode configuration as found in `\XINT_setcatcodes`. Additional temporary configuration is used at some locations. For example `!` is of catcode letter in `xintexpr` and there are locations with funny catcodes e.g. using some letters with the math shift catcode.

2 Package [xintkernel](#) implementation

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

This package provides the common minimal code base for loading management and catcode control and also a few programming utilities. With 1.2 a few more helper macros and all `\chardef`'s have been moved here. The package is loaded by both `xintcore.sty` and `xinttools.sty` hence by all other packages.

1.1. separated package.

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

1.3b. `\xintUniformDeviate`.

1.4 (2020/01/11). `\xintReplicate`, `\xintGobble`, `\xintLastOne`, `\xintFirstOne`.

2.1 Catcodes, ε -TeX and reload detection

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

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

```

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

```

```

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

```

2.1.1 \XINT_setcatcodes, \XINT_storecatcodes, \XINT_restorecatcodes_endinput

```

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

```

```

72 \edef\XINT_restorecatcodes_endinput
73 {%
74 \XINT_restorecatcodes\noexpand\endinput %
75 }%
76 \def\XINT_setcatcodes
77 {%
78 \catcode61=12 % =
79 \catcode32=10 % space
80 \catcode13=5 % ^^M
81 \endlinechar=13 %
82 \catcode123=1 % {
83 \catcode125=2 % }
84 \catcode95=11 % _ LETTER
85 \catcode35=6 % #
86 \catcode44=12 % ,
87 \catcode45=12 % -
88 \catcode46=12 % .
89 \catcode58=11 % : LETTER
90 \catcode60=12 % <
91 \catcode62=12 % >
92 \catcode43=12 % +
93 \catcode42=12 % *
94 \catcode40=12 % (
95 \catcode41=12 % )
96 \catcode47=12 % /
97 \catcode96=12 % `
98 \catcode94=11 % ^ LETTER
99 \catcode36=3 % $
100 \catcode91=12 % [
101 \catcode93=12 % ]
102 \catcode33=12 % ! (xintexpr.sty will use catcode 11)
103 \catcode64=11 % @ LETTER
104 \catcode38=7 % & for \romannumeral`&&@ trick.
105 \catcode124=12 % |
106 \catcode63=11 % ? LETTER
107 \catcode34=12 % "
108 \catcode39=12 % '
109 \catcode126=3 % ~ MATH
110 \catcode59=12 % ;
111 \catcode0=12 % for \romannumeral`&&@ trick
112 \catcode1=3 % for ultra-safe séparateur &&A
113 }%
114 \XINT_setcatcodes
115 }%
116 \PrepareCatcodes

```

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

```

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

```

123 }%

2.2 Package identification

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

1.09c uses e-TeX `\ifdefined`.

```
124 \ifdefined\ProvidesPackage
125   \let\XINT_providespackage\relax
126 \else
127   \def\XINT_providespackage #1#2[#3]%
128     {\immediate\write-1{Package: #2 #3}%
129     \expandafter\xdef\csname ver@#2.sty\endcsname{#3}}%
130 \fi
131 \XINT_providespackage
132 \ProvidesPackage {xintkernel}%
133 [2020/02/25 v1.4b Paraphernalia for the xint packages (JFB)]%
```

2.3 Constants

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

2.4 (WIP) `\xint_texuniformdeviate` and needed counts

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

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

2.5 Token management utilities

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

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



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

2.6 “gob til” macros and UD style fork

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

2.7 \xint_afterfi

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

2.8 \xint_bye, \xint_Bye

1.09. \xint_bye

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

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

2.9 \xintdothis, \xintorthat

1.1.

1.2. names without underscores.

To be used this way:

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

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

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

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

2.10 `\xint_zapspaces`

1.1.

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

1.2i. made `\long`.

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

```
247 \long\def\xint_zapspaces #1 #2{#1#2\xint_zapspaces }% 1.1
248 \long\def\xint_zapspaces_o #1{\expandafter\xint_zapspaces#1 \xint_gobble_i}%
```

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

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

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

2.12 `\xintReverseOrder`

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

Attention: removes brace pairs (and swallows spaces).

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

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

```
258 \def\xintReverseOrder {\romannumeral0\xintreverseorder }%
259 \long\def\xintreverseorder #1%
260 {%
261   \XINT_rord_main {}#1%
262   \xint:
263     \xint_bye\xint_bye\xint_bye\xint_bye
264     \xint_bye\xint_bye\xint_bye\xint_bye
265   \xint:
266 }%
267 \long\def\XINT_rord_main #1#2#3#4#5#6#7#8#9%
268 {%
269   \xint_bye #9\XINT_rord_cleanup\xint_bye
270   \XINT_rord_main {#9#8#7#6#5#4#3#2#1}%
271 }%
272 \def\XINT_rord_cleanup #1{%
273 \long\def\XINT_rord_cleanup\xint_bye\XINT_rord_main ##1##2\xint:
274 {%
275   \expandafter#1\xint_gob_til_xint: ##1%
276 }}\XINT_rord_cleanup { }%
```

2.13 `\xintLength`

1.0. does not expand its argument. See `\xintNthElt{0}` from `xinttools` which f-expands its argument.

1.2g. added `\xintCSVLength` to `xinttools`.

1.2i. rewrote this venerable macro. New code about 40% faster across all lengths. Syntax with `\romannumeral0` adds some slight (negligible) overhead; it is done to fit some general principles of structure of the xint package macros but maybe at some point I should drop it. And in fact it is often called directly via the `\numexpr` access point. (bad coding...)

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

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

2.14 \xintLastItem

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

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

2.15 \xintFirstItem

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

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

2.16 \xintLastOne

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

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

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

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

```

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

```

2.17 `\xintFirstOne`

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

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

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

```

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

```

2.18 `\xintLengthUpTo`

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

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

1.2j. changed ending and interface to core loop.

```

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

```

```

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

```

2.19 \xintreplicate, \xintReplicate

1.2i.

This is cloned from LaTeX3's `\prg_replicate:nn`, see Joseph's post at <http://tex.stackexchange.com/questions/16189/repeat-command-n-times>

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

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

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

`\romannumeral\XINT_rep N\endcsname {foo}`

to skip the `\numexpr`.

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

```

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

```

```

388             #1\XINT_rep_neg
389             -\XINT_rep
390             \krof #1}%
391 \long\def\XINT_rep_neg #1\endcsname #2{\xint_c_}%
392 \def\XINT_rep #1{\csname XINT_rep_f#1\XINT_rep_a}%
393 \def\XINT_rep_a #1{\csname XINT_rep_#1\XINT_rep_a}%
394 \def\XINT_rep_\XINT_rep_a{\endcsname}%
395 \long\expandafter\def\csname XINT_rep_0\endcsname #1%
396     {\endcsname{#1#1#1#1#1#1#1#1#1#1}}%
397 \long\expandafter\def\csname XINT_rep_1\endcsname #1%
398     {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1}%
399 \long\expandafter\def\csname XINT_rep_2\endcsname #1%
400     {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1}%
401 \long\expandafter\def\csname XINT_rep_3\endcsname #1%
402     {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1}%
403 \long\expandafter\def\csname XINT_rep_4\endcsname #1%
404     {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1}%
405 \long\expandafter\def\csname XINT_rep_5\endcsname #1%
406     {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1#1}%
407 \long\expandafter\def\csname XINT_rep_6\endcsname #1%
408     {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1#1#1}%
409 \long\expandafter\def\csname XINT_rep_7\endcsname #1%
410     {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1#1#1#1}%
411 \long\expandafter\def\csname XINT_rep_8\endcsname #1%
412     {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1#1#1#1#1}%
413 \long\expandafter\def\csname XINT_rep_9\endcsname #1%
414     {\endcsname{#1#1#1#1#1#1#1#1#1#1}#1#1#1#1#1#1#1#1#1}%
415 \long\expandafter\def\csname XINT_rep_f0\endcsname #1%
416     {\xint_c_}%
417 \long\expandafter\def\csname XINT_rep_f1\endcsname #1%
418     {\xint_c_ #1}%
419 \long\expandafter\def\csname XINT_rep_f2\endcsname #1%
420     {\xint_c_ #1#1}%
421 \long\expandafter\def\csname XINT_rep_f3\endcsname #1%
422     {\xint_c_ #1#1#1}%
423 \long\expandafter\def\csname XINT_rep_f4\endcsname #1%
424     {\xint_c_ #1#1#1#1}%
425 \long\expandafter\def\csname XINT_rep_f5\endcsname #1%
426     {\xint_c_ #1#1#1#1#1}%
427 \long\expandafter\def\csname XINT_rep_f6\endcsname #1%
428     {\xint_c_ #1#1#1#1#1#1}%
429 \long\expandafter\def\csname XINT_rep_f7\endcsname #1%
430     {\xint_c_ #1#1#1#1#1#1#1}%
431 \long\expandafter\def\csname XINT_rep_f8\endcsname #1%
432     {\xint_c_ #1#1#1#1#1#1#1#1}%
433 \long\expandafter\def\csname XINT_rep_f9\endcsname #1%
434     {\xint_c_ #1#1#1#1#1#1#1#1#1}%

```

2.20 \xintgobble, \xintGobble

1.2i.

I hesitated about allowing as many as $9^6-1=531440$ tokens to gobble, but $9^5-1=59058$ is too low

for playing with long decimal expansions.

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

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

```

435 \def\xintGobble{\romannumeral\xintgobble}%
436 \def\xintgobble #1%
437   {\csname xint_c_\expandafter\XINT_gobble_a\the\numexpr#1.0}%
438 \def\XINT_gobble #1.{\csname xint_c_\XINT_gobble_a #1.0}%
439 \def\XINT_gobble_a #1{\xint_gob_til_zero#1\XINT_gobble_d0\XINT_gobble_b#1}%
440 \def\XINT_gobble_b #1.#2%
441   {\expandafter\XINT_gobble_c
442     \the\numexpr (#1+\xint_c_v)/\xint_c_ix-\xint_c_i\expandafter.%
443     \the\numexpr #2+\xint_c_i.#1.}%
444 \def\XINT_gobble_c #1.#2.#3.%
445   {\csname XINT_g2\the\numexpr#3-\xint_c_ix*#1\relax\XINT_gobble_a #1.#2}%
446 \def\XINT_gobble_d0\XINT_gobble_b0.#1{\endcsname}%
447 \expandafter\let\csname XINT_g10\endcsname\endcsname
448 \long\expandafter\def\csname XINT_g11\endcsname#1{\endcsname}%
449 \long\expandafter\def\csname XINT_g12\endcsname#1#2{\endcsname}%
450 \long\expandafter\def\csname XINT_g13\endcsname#1#2#3{\endcsname}%
451 \long\expandafter\def\csname XINT_g14\endcsname#1#2#3#4{\endcsname}%
452 \long\expandafter\def\csname XINT_g15\endcsname#1#2#3#4#5{\endcsname}%
453 \long\expandafter\def\csname XINT_g16\endcsname#1#2#3#4#5#6{\endcsname}%
454 \long\expandafter\def\csname XINT_g17\endcsname#1#2#3#4#5#6#7{\endcsname}%
455 \long\expandafter\def\csname XINT_g18\endcsname#1#2#3#4#5#6#7#8{\endcsname}%
456 \expandafter\let\csname XINT_g20\endcsname\endcsname
457 \long\expandafter\def\csname XINT_g21\endcsname #1#2#3#4#5#6#7#8#9%
458   {\endcsname}%
459 \long\expandafter\edef\csname XINT_g22\endcsname #1#2#3#4#5#6#7#8#9%
460   {\expandafter\noexpand\csname XINT_g21\endcsname}%
461 \long\expandafter\edef\csname XINT_g23\endcsname #1#2#3#4#5#6#7#8#9%
462   {\expandafter\noexpand\csname XINT_g22\endcsname}%
463 \long\expandafter\edef\csname XINT_g24\endcsname #1#2#3#4#5#6#7#8#9%
464   {\expandafter\noexpand\csname XINT_g23\endcsname}%
465 \long\expandafter\edef\csname XINT_g25\endcsname #1#2#3#4#5#6#7#8#9%
466   {\expandafter\noexpand\csname XINT_g24\endcsname}%
467 \long\expandafter\edef\csname XINT_g26\endcsname #1#2#3#4#5#6#7#8#9%
468   {\expandafter\noexpand\csname XINT_g25\endcsname}%
469 \long\expandafter\edef\csname XINT_g27\endcsname #1#2#3#4#5#6#7#8#9%
470   {\expandafter\noexpand\csname XINT_g26\endcsname}%
471 \long\expandafter\edef\csname XINT_g28\endcsname #1#2#3#4#5#6#7#8#9%
472   {\expandafter\noexpand\csname XINT_g27\endcsname}%
473 \expandafter\let\csname XINT_g30\endcsname\endcsname
474 \long\expandafter\edef\csname XINT_g31\endcsname #1#2#3#4#5#6#7#8#9%
475   {\expandafter\noexpand\csname XINT_g28\endcsname}%
476 \long\expandafter\edef\csname XINT_g32\endcsname #1#2#3#4#5#6#7#8#9%
477   {\noexpand\csname XINT_g31\expandafter\noexpand\csname XINT_g28\endcsname}%
478 \long\expandafter\edef\csname XINT_g33\endcsname #1#2#3#4#5#6#7#8#9%
479   {\noexpand\csname XINT_g32\expandafter\noexpand\csname XINT_g28\endcsname}%
480 \long\expandafter\edef\csname XINT_g34\endcsname #1#2#3#4#5#6#7#8#9%
481   {\noexpand\csname XINT_g33\expandafter\noexpand\csname XINT_g28\endcsname}%
482 \long\expandafter\edef\csname XINT_g35\endcsname #1#2#3#4#5#6#7#8#9%
483   {\noexpand\csname XINT_g34\expandafter\noexpand\csname XINT_g28\endcsname}%

```



```

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

```

```

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

```

2.21 (WIP) \xintUniformDeviate

1.3b. See user manual for related information.

```

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

```

2.22 \xintMessage, \ifxintverbose

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

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

1.3e. set the `\newlinechar`.

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

2.23 `\ifxintglobaldefs`, `\XINT_global`

1.3c.

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

2.24 (WIP) Expandable error message

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

This is copied over from l3kernel code. I am using `\!/` control sequence though, which must be left undefined. `\xintError:` would be 6 letters more already.

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

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

3 Package [xinttools](#) implementation

.1	Catcodes, ε - $\text{T}_{\text{E}}\text{X}$ and reload detection . . .	20		
.2	Package identification	21		
.3	<code>\xintgodef</code> , <code>\xintgoodef</code> , <code>\xintgfdef</code> . . .	21		
.4	<code>\xintRevWithBraces</code>	21		
.5	<code>\xintZapFirstSpaces</code>	22		
.6	<code>\xintZapLastSpaces</code>	23		
.7	<code>\xintZapSpaces</code>	24		
.8	<code>\xintZapSpacesB</code>	24		
.9	<code>\xintCSVtoList</code> , <code>\xintCSVtoListNon-</code> <code>Stripped</code>	24		
.10	<code>\xintListWithSep</code>	26		
.11	<code>\xintNthElt</code>	27		
.12	<code>\xintNthOnePy</code>	28		
.13	<code>\xintKeep</code>	29		
.14	<code>\xintKeepUnbraced</code>	30		
.15	<code>\xintTrim</code>	31		
.16	<code>\xintTrimUnbraced</code>	33		
.17	<code>\xintApply</code>	34		
.18	<code>\xintApply:x</code> (WIP, commented-out) . . .	34		
.19	<code>\xintApplyUnbraced</code>	35		
.20	<code>\xintApplyUnbraced:x</code> (WIP, commented- out)	36		
.21	<code>\xintZip</code> (WIP, not public)	37		
.22	<code>\xintSeq</code>	39		
.23	<code>\xintloop</code> , <code>\xintbreakloop</code> , <code>\xintbreak-</code> <code>looppando</code> , <code>\xintloopskiptonext</code>	41		
.24	<code>\xintiloop</code> , <code>\xintiloopindex</code> , <code>\xint-</code> <code>bracediloopindex</code> , <code>\xintouteriloopindex</code> ,			
	<code>\xintbracedouteriloopindex</code> , <code>\xintbreak-</code> <code>iloop</code> , <code>\xintbreakilooppando</code> , <code>\xintiloop-</code> <code>skiptonext</code> , <code>\xintiloopskipandredo</code> . . .	42		
.25	<code>\XINT_xflet</code>	42		
.26	<code>\xintApplyInline</code>	43		
.27	<code>\xintFor</code> , <code>\xintFor*</code> , <code>\xintBreakFor</code> , <code>\xintBreakForAndDo</code>	44		
.28	<code>\XINT_forever</code> , <code>\xintintegers</code> , <code>\xintdi-</code> <code>mensions</code> , <code>\xintrationals</code>	46		
.29	<code>\xintForpair</code> , <code>\xintForthree</code> , <code>\xintFor-</code> <code>four</code>	48		
.30	<code>\xintAssign</code> , <code>\xintAssignArray</code> , <code>\xint-</code> <code>DigitsOf</code>	49		
.31	CSV (non user documented) variants of Length, Keep, Trim, NthElt, Reverse . . .	52		
.31.1	<code>\xintLength:f:csv</code>	53		
.31.2	<code>\xintLengthUpTo:f:csv</code>	54		
.31.3	<code>\xintKeep:f:csv</code>	55		
.31.4	<code>\xintTrim:f:csv</code>	56		
.31.5	<code>\xintNthEltPy:f:csv</code>	58		
.31.6	<code>\xintReverse:f:csv</code>	59		
.31.7	<code>\xintFirstItem:f:csv</code>	60		
.31.8	<code>\xintLastItem:f:csv</code>	60		
.31.9	<code>\xintKeep:x:csv</code>	60		
.31.10	Public names for the undocumented csv macros: <code>\xintCSVLength</code> , <code>\xintCSVKeep</code> , <code>\xintCSVKeepx</code> , <code>\xintCSVTrim</code> , <code>\xintCSVNthEltPy</code> , <code>\xintCSVRe-</code> <code>verse</code> , <code>\xintCSVFirstItem</code> , <code>\xintCSVLastItem</code>	61		

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 - $\text{T}_{\text{E}}\text{X}$ and reload detection

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

```

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

```

```

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

```

3.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xinttools}%
46 [2020/02/25 v1.4b Expandable and non-expandable utilities (JFB)]%

```

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

```

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

```

3.3 \xintgodef, \xintgoodef, \xintgfdef

1.09i. For use in \xintAssign.

```

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

```

3.4 \xintRevWithBraces

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

```

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

```

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

```

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

```

3.5 \xintZapFirstSpaces

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

```

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

```

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

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

3.6 \xintZapLastSpaces

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

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

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

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

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

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

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

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

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

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

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

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

3.7 \xintZapSpaces

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

```

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

```

3.8 \xintZapSpacesB

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

```

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

```

3.9 \xintCSVtoList, \xintCSVtoListNonStripped

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

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

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

```

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

```



```

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

```

1.1c revisits this old code and improves upon the earlier endings. But as the `_d..` macros have already nine parameters, I needed the `\expandafter` and `\xint_gob_til_Z` in `finish_b` (compare `\XINT_keep_endb`, or also `\XINT_RQ_end_b`).

```

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

```

```

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

```

3.10 \xintListWithSep

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

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

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

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

```

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

```

```

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

```

3.13 \xintKeep

First included in release 1.09m.

`\xintKeep{i}{L}` f-expands its second argument L. It then grabs the first i items from L and discards the rest.

ATTENTION: **each such kept item is returned inside a brace pair** Use `\xintKeepUnbraced` to avoid that.

For i equal or larger to the number N of items in (expanded) L, the full L is returned (with braced items). For i=0, the macro returns an empty output. For i<0, the macro discards the first N-|i| items. No brace pairs added to the remaining items. For i is less or equal to -N, the full L is returned (with no braces added.)

`\xintKeepNoExpand` does not expand the L argument.

Prior to 1.2i the code proceeded along a loop with no pre-computation of the length of L, for the i>0 case. The faster 1.2i version takes advantage of novel `\xintLengthUpTo` from `xintkernel.sty`.

```

328 \def\xintKeep          {\romannumeral0\xintkeep }%
329 \def\xintKeepNoExpand {\romannumeral0\xintkeepnoexpand }%
330 \long\def\xintkeep #1#2{\expandafter\XINT_keep_a\the\numexpr #1\expandafter.%
331 \expandafter{\romannumeral`&&@#2}}%
332 \def\xintkeepnoexpand #1{\expandafter\XINT_keep_a\the\numexpr #1.}%
333 \def\XINT_keep_a #1%
334 {%
335 \xint_UDzerominusfork
336 #1-\XINT_keep_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:\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{#{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}}%
375 \def\XINT_keep_loop_end-\expandafter\XINT_keep_loop
376     \the\numexpr-#1-\xint_c_viii\expandafter.\XINT_keep_loop_pickeight
377     {\csname XINT_keep_end#1\endcsname}%
378 \long\expandafter\def\csname XINT_keep_end1\endcsname
379     #1#2#3#4#5#6#7#8#9\xint_bye { #1{#2}{#3}{#4}{#5}{#6}{#7}{#8}}%
380 \long\expandafter\def\csname XINT_keep_end2\endcsname
381     #1#2#3#4#5#6#7#8\xint_bye { #1{#2}{#3}{#4}{#5}{#6}{#7}}%
382 \long\expandafter\def\csname XINT_keep_end3\endcsname
383     #1#2#3#4#5#6#7\xint_bye { #1{#2}{#3}{#4}{#5}{#6}}%
384 \long\expandafter\def\csname XINT_keep_end4\endcsname
385     #1#2#3#4#5#6\xint_bye { #1{#2}{#3}{#4}{#5}}%
386 \long\expandafter\def\csname XINT_keep_end5\endcsname
387     #1#2#3#4#5\xint_bye { #1{#2}{#3}{#4}}%
388 \long\expandafter\def\csname XINT_keep_end6\endcsname
389     #1#2#3#4\xint_bye { #1{#2}{#3}}%
390 \long\expandafter\def\csname XINT_keep_end7\endcsname
391     #1#2#3\xint_bye { #1{#2}}%
392 \long\expandafter\def\csname XINT_keep_end8\endcsname
393     #1#2\xint_bye { #1}%

```

3.14 \xintKeepUnbraced

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

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

Modified for 1.2i like `\xintKeep`.

```

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

```

```

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

```

3.15 \xintTrim

First included in release 1.09m.

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

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

`\xintTrimNoExpand` does not expand the L argument.

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

```

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

```

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

```

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

```



```

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

```

3.16 \xintTrimUnbraced

1.2a. Modified in 1.2i like \xintTrim

```

504 \def\xintTrimUnbraced      {\romannumeral0\xinttrimunbraced }%
505 \def\xintTrimUnbracedNoExpand {\romannumeral0\xinttrimunbracednoexpand }%
506 \long\def\xinttrimunbraced #1#2%
507     {\expandafter\XINT_trimunbr_a\the\numexpr #1\expandafter.%
508         \expandafter{\romannumeral`&&@#2}}%
509 \def\xinttrimunbracednoexpand #1%
510     {\expandafter\XINT_trimunbr_a\the\numexpr #1.}%
511 \def\XINT_trimunbr_a #1%
512 {%
513     \xint_UDzerominusfork
514         #1-\XINT_trim_trimnone
515         0#1\XINT_trimunbr_neg
516         0-{\XINT_trim_pos #1}%
517     \krof
518 }%
519 \long\def\XINT_trimunbr_neg #1.#2%
520 {%
521     \expandafter\XINT_trimunbr_neg_a\the\numexpr
522     #1-\numexpr\XINT_length_loop
523     #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
524     \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
525     \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
526     .}{#2\xint_bye
527 }%
528 \def\XINT_trimunbr_neg_a #1%
529 {%
530     \xint_UDsignfork
531     #1{\expandafter\XINT_keeponbr_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\empty` 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}%
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\empty}%
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\xintcsvtlist {#1}Z%
923 }%
924 \def\XINT_to_forever\fi #1\xintcsvtlist #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_xflet\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, \xinrationals

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 \xinrationals 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_secondoftwo
1065     \let\xintifForLast\xint_secondoftwo
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_secondoftwo
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_secondoftwo
1085     \let\xintifForLast\xint_secondoftwo
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}\toA\B...Z` resp. `\xintAssignArray {a}{b}..{z}\toU`.
`\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 \xintloop [\csname\xint_arrayname 0\endcsname+-1]

```

```

1183     \global
1184     \expandafter\let\csname\xint_arrayname\xintiloopindex\endcsname\relax
1185     \ifnum \xintiloopindex > \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 negative: 0 > ##1} }%
1247         \else
1248             \xint_afterfi {%
1249                 \ifnum ##1>#2
1250                     \xint_afterfi
1251                         {\XINT_expandableerror{Array index 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» ob-

ject 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\xintlengthupto:f:csv}%
1282 \long\def\xintlengthupto:f:csv #1#2%
1283 {%
1284   \expandafter\XINT_lengthupto: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 }%
```

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

```
1291 \long\def\XINT_lengthupto:f:csv_a #1.#2%
1292 {%
1293   \xint_gob_til_xint: #2\XINT_lengthupto:f:csv_empty\xint:%
1294   \XINT_lengthupto:f:csv_loop_b #1.#2%
1295 }%
1296 \def\XINT_lengthupto:f:csv_empty\xint:%
1297   \XINT_lengthupto:f:csv_loop_b #1.#2\xint_bye.{ #1}%
1298 \def\XINT_lengthupto:f:csv_loop_a #1%
1299 {%
1300   \xint_UDsignfork
1301   #1\XINT_lengthupto:f:csv_gt
1302   -\XINT_lengthupto:f:csv_loop_b
1303   \krof #1%
1304 }%
1305 \long\def\XINT_lengthupto:f:csv_gt #1\xint_bye.{-0}%
1306 \long\def\XINT_lengthupto:f:csv_loop_b #1.#2,#3,#4,#5,#6,#7,#8,#9,%
1307 {%
1308   \xint_gob_til_xint: #9\XINT_lengthupto:f:csv_finish_a\xint:%
1309   \expandafter\XINT_lengthupto:f:csv_loop_a\the\numexpr #1-\xint_c_viii.%
1310 }%
1311 \def\XINT_lengthupto:f:csv_finish_a\xint:
1312   \expandafter\XINT_lengthupto:f:csv_loop_a
1313   \the\numexpr #1-\xint_c_viii.#2,#3,#4,#5,#6,#7,#8,#9,%
1314 {%
1315   \expandafter\XINT_lengthupto:f:csv_finish_b\the\numexpr #1-#9\xint_bye
1316 }%
1317 \def\XINT_lengthupto: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_trimline
1366 }%
1367 \long\def\XINT_keep:f:csv_trimloop_trimline #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_trimmine
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\xintlastitem:f:csv}%
1564 \long\def\xintlastitem: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 \XINT_restorecatcodes_endinput%

```

4 Package [xintcore](#) implementation

.1	Catcodes, ε -TeX and reload detection . . .	62	.25	\XINT_zeroes_forviii	75	
.2	Package identification	63	.26	\XINT_sepbyviii_Z	75	
.3	(WIP!) Error conditions and exceptions . . .	63	.27	\XINT_sepbyviii_andcount	76	
.4	Counts for holding needed constants . . .	65	.28	\XINT_rsepbyviii	76	
Routines handling integers as lists of token digits			66	.29	\XINT_sepandrev	77
.5	\XINT_cuz_small	66	.30	\XINT_sepandrev_andcount	77	
.6	\xintNum, \xintiNum	66	.31	\XINT_rev_nounsep	78	
.7	\xintiiSgn	67	.32	\XINT_unrevbyviii	78	
.8	\xintiiOpp	68	Core arithmetic			78
.9	\xintiiAbs	68	.33	\xintiiAdd	79	
.10	\xintFDg	68	.34	\xintiiCmp	82	
.11	\xintLDg	69	.35	\xintiiSub	84	
.12	\xintDouble	69	.36	\xintiiMul	89	
.13	\xintHalf	70	.37	\xintiiDivision	93	
.14	\xintInc	70	Derived arithmetic			109
.15	\xintDec	71	.38	\xintiiQuo, \xintiiRem	109	
.16	\xintDSL	71	.39	\xintiiDivRound	109	
.17	\xintDSR	71	.40	\xintiiDivTrunc	110	
.18	\xintDSRr	72	.41	\xintiiModTrunc	110	
Blocks of eight digits			72	.42	\xintiiDivMod	111
.19	\XINT_cuz	72	.43	\xintiiDivFloor	112	
.20	\XINT_cuz_byviii	73	.44	\xintiiMod	112	
.21	\XINT_unsep_loop	73	.45	\xintiiSqr	112	
.22	\XINT_unsep_cuzsmall	74	.46	\xintiiPow	113	
.23	\XINT_div_unsepQ	74	.47	\xintiiFac	116	
.24	\XINT_div_unsepR	75	.48	\XINT_useiimessage	119	

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

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

The commenting continues (2020/02/25) 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 -TeX and reload detection

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

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

```

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

```

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

```

4.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintcore}%
46 [2020/02/25 v1.4b Expandable arithmetic on big integers (JFB)]%

```

4.3 (WIP!) Error conditions and exceptions

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

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

and the Python3 implementation in its Decimal module.

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

X3.274 rajoute LostDigits

Python rajoute FloatOperation (et n'inclut pas InsufficientStorage)

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

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

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

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

Various conditions can raise the same signal.

Only signals, not conditions, raise Flags.

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

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

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

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

Only four conditions are currently possibly encountered:

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

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

```

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

```



```

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

```

4.4 Counts for holding needed constants

```

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

```

```

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

```

Routines handling integers as lists of token digits

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

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

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

4.5 \XINT_cuz_small

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

```

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

```

4.6 \xintNum, \xintiNum

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

Very old routine got completely rewritten at 1.21.

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

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

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

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

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

```

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

```

```

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

```

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

```

150 \expandafter\XINT_num_loop
151 \else

```

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

```

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

```

4.7 `\xintiiSgn`

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

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

```

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

```

4.8 `\xintiiOpp`

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

```

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

```

4.9 `\xintiiAbs`

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

```

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

```

4.10 `\xintFDg`

FIRST DIGIT.

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

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

```

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

```



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

4.13 \xintHalf

Attention \xintHalf non robust against non terminated input.

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

4.14 \xintInc

1.2i much delayed complete rewrite in 1.2 style.

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

Attention \xintInc non robust against non terminated input.

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

4.15 `\xintDec`

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

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

```

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

```

4.16 `\xintDSL`

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

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

```

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

```

4.17 `\xintDSR`

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

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

```

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

```

```

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

```

4.18 \xintDSRr

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

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

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

```

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

```

Blocks of eight digits

The lingua of release 1.2.

4.19 \XINT_cuz

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

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

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

```

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

```

4.20 `\XINT_cuz_byviii`

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

```

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

```

4.21 `\XINT_unsep_loop`

This is used as

```

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

```

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

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

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

```

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

```

```

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

```

4.22 \XINT_unsep_cuzsmall

This is used as

```

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

```

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

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

```

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

```

4.23 \XINT_div_unsepQ

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

Rewritten for 1.2l in 1.2i style.

```

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

```

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

4.24 \XINT_div_unsepR

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

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

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

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

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

4.25 \XINT_zeroes_forviii

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

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

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

4.26 \XINT_sepbyviii_Z

This is used as

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

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

Prior to 1.2l it used `\Z` as terminator not the semi-colon (hence the name). The switch to `;` was done at a time I thought perhaps I would use an internal format maintaining such 8 digits blocks, and this has to be compatible with the `\csname...\endcsname` encapsulation in `\xintexpr` parsers.

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

4.27 \XINT_sepbyviii_andcount

This is used as

```
\the\numexpr\XINT_sepbyviii_andcount <8Ndigits>%
  \XINT_sepbyviii_end 2345678\relax
  \xint_c_vii!\xint_c_vi!\xint_c_v!\xint_c_iv!%
  \xint_c_iii!\xint_c_ii!\xint_c_i!\xint_c_\W
```

It will produce

```
1<8d>!1<8d>!...1<8d>!1\xint:<count of blocks>\xint:
```

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

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

4.28 \XINT_rsepbyviii

This is used as

```
\the\numexpr1\XINT_rsepbyviii <8Ndigits>%
  \XINT_rsepbyviii_end_A 2345678%
  \XINT_rsepbyviii_end_B 2345678\relax UV%
```

and will produce

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

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

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

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

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

4.29 \XINT_sepandrev

This is used typically as

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

and will produce

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

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

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

4.30 \XINT_sepandrev_andcount

This is used typically as

```
\romannumeral0\XINT_sepandrev_andcount <8Ndigits>%
    \XINT_rsepbyviii_end_A 2345678%
    \XINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
    \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
    \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_W
```

and will produce

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

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

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

```

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

```

4.31 \XINT_rev_nounsep

This is used as

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

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

```

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

```

4.32 \XINT_unrevbyviii

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

```
1;!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
```

The \romannumeral in unrevbyviii_a is for special effects (expand some token which was put as 1<token>! at the end of the original blocks). This mechanism is used by 1.2 subtraction (still true for 1.21).

```

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

```

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

```

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

```

Core arithmetic

The four operations have been rewritten entirely for release 1.2. The new routines works with separated blocks of eight digits. They all measure first the lengths of the arguments, even addition and subtraction (this was not the case with xintcore.sty 1.1 or earlier.)

The technique of chaining `\the\numexpr` induces a limitation on the maximal size depending on the size of the input save stack and the maximum expansion depth. For the current (TL2015) settings (5000, resp. 10000), the induced limit for addition of numbers is at 19968 and for multiplication it is observed to be 19959 (valid as of 2015/10/07).

Side remark: I tested that `\the\numexpr` was more efficient than `\number`. But it reduced the allowable numbers for addition from 19976 digits to 19968 digits.

4.33 `\xintiiAdd`

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

```

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

```

```

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

```

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

```

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

```

2 as first token of #1 stands for "no carry", 3 will mean a carry (we are adding 1<8digits> to 1<8digits>.) Version 1.2c has terminators of the shape 1;!, replacing the \Z! used in 1.2.

Call: \the\numexpr\XINT_add_a 2#11;!1;!1;!1;!1\W #21;!1;!1;!1;!1\W where #1 and #2 are blocks of 1<8d>!, and #1 is at most as long as #2. This last requirement is a bit annoying (if one wants to do recursive algorithms but not have to check lengths), and I will probably remove it at some point.

Output: blocks of 1<8d>! representing the addition, (least significant first), and a final 1;! . In recursive algorithm this 1;! terminator can thus conveniently be reused as part of input terminator (up to the length problem).

```

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

```



```

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

```

```

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

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

```

Here 2 stands for "carry", and 1 for "no carry" (we have been adding 1 to 1<8digits>.)

```

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

```

4.34 \xintiiCmp

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

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

1.20 deprecates `\xintCmp`, with `xintfrac` loaded it will get overwritten anyhow.

```

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

```

```

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

```

```

759 {%
760   \xint_gob_til_sc #1\XINT_cmp_equal ;%
761   \ifnum #1>#6 \XINT_cmp_gt\fi
762   \ifnum #1<#6 \XINT_cmp_lt\fi
763   \xint_gob_til_sc #2\XINT_cmp_equal ;%
764   \ifnum #2>#7 \XINT_cmp_gt\fi
765   \ifnum #2<#7 \XINT_cmp_lt\fi
766   \xint_gob_til_sc #3\XINT_cmp_equal ;%
767   \ifnum #3>#8 \XINT_cmp_gt\fi
768   \ifnum #3<#8 \XINT_cmp_lt\fi
769   \xint_gob_til_sc #4\XINT_cmp_equal ;%
770   \ifnum #4>#9 \XINT_cmp_gt\fi
771   \ifnum #4<#9 \XINT_cmp_lt\fi
772   \XINT_cmp_a #5\W
773 }%
774 \def\XINT_cmp_lt#1{\def\XINT_cmp_lt\fi ##1\W ##2\W {\fi#1-1}}\XINT_cmp_lt{ }%
775 \def\XINT_cmp_gt#1{\def\XINT_cmp_gt\fi ##1\W ##2\W {\fi#11}}\XINT_cmp_gt{ }%
776 \def\XINT_cmp_equal #1\W #2\W { 0}%

```

4.35 \xintiiSub

Entirely rewritten for 1.2.

Refactored at 1.2l. I was initially aiming at clinching some internal format of the type `1<8digits>!...!<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.

```

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

```

```

790 \krof
791 \xint_UDsignsfork
792 #1#2\XINT_sub_minusminus
793 #1-\XINT_sub_minusplus
794 #2-\XINT_sub_plusminus
795 --\XINT_sub_plusplus
796 \krof #1#2%
797 }%
798 \def\XINT_sub_firstiszero #1\krof 0#2#3\xint:#4\xint:{\XINT_opp #2#3}%
799 \def\XINT_sub_secondiszero #1\krof #20#3\xint:#4\xint:{ #2#4}%
800 \def\XINT_sub_plusminus #1#2{\XINT_add_pp_a #1{}}%
801 \def\XINT_sub_plusplus #1#2%
802 {\expandafter\XINT_opp\romannumeral0\XINT_sub_mm_a #1#2}%
803 \def\XINT_sub_minusplus #1#2%
804 {\expandafter-\romannumeral0\XINT_add_pp_a { }#2}%
805 \def\XINT_sub_minusminus #1#2{\XINT_sub_mm_a { }{}}%

806 \def\XINT_sub_mm_a #1#2#3\xint:
807 {%
808 \expandafter\XINT_sub_mm_b
809 \romannumeral0\expandafter\XINT_sepandrev_andcount
810 \romannumeral0\XINT_zeroes_forviii #2#3\R\R\R\R\R\R\R\R{10}0000001\W
811 #2#3\XINT_rsepyviii_end_A 2345678%
812 \XINT_rsepyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
813 \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
814 \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
815 \X #1%
816 }%
817 \def\XINT_sub_mm_b #1\xint:#2\X #3\xint:
818 {%
819 \expandafter\XINT_sub_checklengths
820 \the\numexpr #1\expandafter\xint:%
821 \romannumeral0\expandafter\XINT_sepandrev_andcount
822 \romannumeral0\XINT_zeroes_forviii #3\R\R\R\R\R\R\R\R{10}0000001\W
823 #3\XINT_rsepyviii_end_A 2345678%
824 \XINT_rsepyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
825 \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
826 \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
827 1;!1;!1;!1;!1\W
828 #21;!1;!1;!1;!1\W
829 1;!1\R!1\R!1\R!1\R!%
830 1\R!1\R!1\R!1\R!1\W
831 }%
832 \def\XINT_sub_checklengths #1\xint:#2\xint:%
833 {%
834 \ifnum #2>#1
835 \expandafter\XINT_sub_exchange
836 \else
837 \expandafter\XINT_sub_aa
838 \fi
839 }%
840 \def\XINT_sub_exchange #1\W #2\W
841 {%

```

```

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

```

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

```
848 \def\XINT_sub_out {\XINT_unrevbyviii{}}%
```

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

Call: `\the\numexpr`

```

\XINT_sub_a #1#1;!1;!1;!1;!1\W
          #2#1;!1;!1;!1;!1\W

```

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

1.2l uses `;! 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.

```

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

```

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

```

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

```

```

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

```

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

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

If A and B have the same length (in base 10⁸) then arbitrarily many zeros might have to be cleaned up, and if A<B, the whole result will have to be complemented first.

```

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

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

```

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

```

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

```



```

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

```

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

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

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

```

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

```

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

```

961 \def\XINT_sub_fix_none;{\XINT_cuz_small}%
962 \def\XINT_sub_fix_cuz ;{\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop}%

```

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

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

```

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

```

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

```

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

```

4.36 `\xintiiMul`

Completely rewritten for 1.2.

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

```

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

```

1.2 I have changed the fork, and it complicates matters elsewhere.
ATTENTION for example that 1.4 \xintiiPrd uses \XINT_mul_nfork now.

```

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

```


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

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

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

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

```
1072 \def\XINT_mul_a #1\W #2\W
1073 {%
1074   \expandafter\XINT_mul_b\the\numexpr
1075   \XINT_add_a \xint_c_ii #21;!1;!1;! \W #11;!1;!1;! \W\W
1076 }%
1077 \def\XINT_mul_b 1#1!{1#1\expandafter!\the\numexpr\XINT_mul_loop }%
1078 \def\XINT_mul_e;#1\W 1#2\W #3\W {1\relax #2}%
```

1.2 small and mini multiplication in base 10^8 with carry. Used by the main multiplication routines. But division, float factorial, etc.. have their own variants as they need output with specific constraints.

The `minimulwc` has $1\langle 8\text{digits carry}\rangle.\langle 4\text{ high digits}\rangle.\langle 4\text{ low digits}\rangle\langle 8\text{digits}\rangle.$

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

Multiplication by zero will produce blocks of zeros.

```
1079 \def\XINT_minimulwc_a 1#1\xint:#2\xint:#3!#4#5#6#7#8\xint:%
1080 {%
1081   \expandafter\XINT_minimulwc_b
1082   \the\numexpr \xint_c_x^ix+#1+#3*#8\xint:
1083               #3*#4#5#6#7+#2*#8\xint:
1084               #2*#4#5#6#7\xint:%
1085 }%
1086 \def\XINT_minimulwc_b 1#1#2#3#4#5#6\xint:#7\xint:%
1087 {%
1088   \expandafter\XINT_minimulwc_c
1089   \the\numexpr \xint_c_x^ix+#1#2#3#4#5+#7\xint:#6\xint:%
1090 }%
1091 \def\XINT_minimulwc_c 1#1#2#3#4#5#6\xint:#7\xint:#8\xint:%
1092 {%
1093   1#6#7\expandafter!%
1094   \the\numexpr\expandafter\XINT_smallmul_a
1095   \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#8\xint:%
1096 }%
1097 \def\XINT_smallmul 1#1#2#3#4#5!{\XINT_smallmul_a 100000000\xint:#1#2#3#4\xint:#5!}%
1098 \def\XINT_smallmul_a #1\xint:#2\xint:#3!1#4!%
1099 {%
1100   \xint_gob_til_sc #4\XINT_smallmul_e;%
```

```

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

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

```

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

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

```

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

```

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 10⁸,

not 10^4 and "drops" the quotient digits, rather than store them upfront as the earlier code, I may well have not correctly converted all such comments. At the last minute some previously #1 became stuff like #1#2#3#4, then of course the old comments describing what the macro parameters stand for are necessarily wrong.

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

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

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

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

On regarde les signes de A et de B.

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

Donc attention malgré son nom \XINT_div_prepare va jusqu'au bout. C'est donc en fait l'entrée principale (pour $B > 0$, $A > 0$) mais elle va regarder si B est $< 10^8$ et s'il vaut alors 1 ou 2, et si $A < 10^8$. Dans tous les cas le résultat est produit sous la forme $\{Q\}\{R\}$, avec Q et R sous leur forme final. On doit ensuite ajuster si le B ou le A initial était négatif. Je n'ai pas fait beaucoup d'efforts pour être un minimum efficace si A ou B n'est pas positif.

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

```

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

```

Le diviseur B va être étendu par des zéros pour que sa longueur soit multiple de huit. Les zéros seront mis du côté non significatif.

```

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

```

B a au plus huit chiffres. On se débarrasse des trucs superflus. Si B>0 n'est ni 1 ni 2, le point d'entrée est \XINT_div_small_a {B}{A} (avec un A positif).

```

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

```

B a au plus huit chiffres et est au moins 3. On va l'utiliser directement, sans d'abord le multiplier par une puissance de 10 pour qu'il ait 8 chiffres.

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

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

```
1226 \def\XINT_div_small_b #1!#2{#2#1!}%
```

On ajoute des zéros avant A, puis on le prépare sous la forme de blocs 1<8d>! Au passage on repère le cas d'un A<10^8.

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

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

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

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

```
1247 \def\XINT_div_dosmallldiv
1248   {{\expandafter\XINT_sdiv_out\the\numexpr\XINT_smalldivx_a}}%
```


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

```

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

```

attention qu'on calcule ici $x' = x + 1$ ($x =$ huit premiers chiffres du diviseur) et que si $x = 99999999$, x' aura donc 9 chiffres, pas compatible avec div_mini (avant 1.2, x avait 4 chiffres, et on faisait la division avec x' dans un \numexpr). Bon, facile à dire après avoir laissé passer ce bug dans 1.2. C'est le problème lorsqu'au lieu de tout refaire à partir de zéro on recycle d'anciennes routines qui avaient un contexte différent.

```

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

```

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

```

1290 \def\XINT_div_start_a #1#2%
1291 {%
1292     \ifnum #1 < #2
1293         \expandafter\XINT_div_zeroQ
1294     \else
1295         \expandafter\XINT_div_start_b
1296     \fi
1297     {#1}{#2}%
1298 }%
1299 \def\XINT_div_zeroQ #1#2#3#4#5#6#7%
1300 {%
1301     \expandafter\XINT_div_zeroQ_end
1302     \romannumeral0\XINT_unsep_cuzsmall
1303     #3\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax\xint:
1304 }%
1305 \def\XINT_div_zeroQ_end #1\xint:#2%
1306     {\expandafter{\expandafter0\expandafter}\XINT_div_cleanR #1#2\xint:}%

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

```

```

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

```

Ici ce sont routines de fin. Le reste déjà nettoyé. R.Q«c».

```

1317 \def\XINT_div_finish_b #1%
1318 {%
1319     \if0#1%
1320         \expandafter\XINT_div_finish_bRzero
1321     \else
1322         \expandafter\XINT_div_finish_bRpos
1323     \fi
1324     #1%
1325 }%
1326 \def\XINT_div_finish_bRzero 0\xint:#1#2{{#1}{0}}%
1327 \def\XINT_div_finish_bRpos #1\xint:#2#3%
1328 {%
1329     \expandafter\xint_exchangetwo_keepbraces\XINT_div_cleanR #1#3\xint:{#2}%
1330 }%
1331 \def\XINT_div_cleanR #1000000000\xint:{{#1}}%

```

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

```

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

```

#1=a, #2=alpha (de longueur K, à l'endroit).#3=reste de A.#4=x, #5={LK{x'y}x},#6=B,«c» -> a, x, alpha, B, {00000000}, L, K, {x'y},x, alpha'=reste de A, B«c».

```

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

```

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

```

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

```

On intercepte petit quotient nul: #1=a, x, alpha, B, #5=q0, L, K, {x'y}, x, alpha', B«c» -> on lâche un q puis {alpha} L, K, {x'y}, x, alpha', B«c».

```

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

```

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

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

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

attention très mauvaises notations avec _b et _db.

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

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

```
1400 \def\XINT_div_I_dc #1#2%
1401 {%
1402 \if-#2\expandafter\XINT_div_I_dd\else\expandafter\XINT_div_I_de\fi
1403 #1#2%
1404 }%
1405 \def\XINT_div_I_dd #1-\Z
1406 {%
1407 \if #11\expandafter\XINT_div_I_dz\fi
1408 \expandafter\XINT_div_I_dP\the\numexpr #1-\xint_c_i\xint: XX%
1409 }%
1410 \def\XINT_div_I_dz #1XX#2#3#4%
1411 {%
1412 1#4\XINT_div_I_g {#2}%
1413 }%
1414 \def\XINT_div_I_de #1#2\Z #3#4#5{1#5+#1\XINT_div_I_g {#2}}%
```

q.alpha, B, q0, L, K, {x'y},x, alpha'B«c» (q=0 has been intercepted) -> Inouveauq.nouvel alpha, L, K, {x'y}, x, alpha',B«c»

```

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

```

1#1=nouveau q. nouvel alpha, L, K, {x'y},x,alpha', BQ<<c>

#1=q,#2=nouvel alpha,#3=L, #4=K, #5={x'y}, #6=x, #7= alpha',#8=B, <<c> -> on laisse q puis {x'y}alpha.alpha'.{{x'y}xKL}B<<c>

```

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

```

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

```

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

```

ATTENTION DESCRIPTION OBSOLETE. #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>

```

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

```

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

```

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

```

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

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

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

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

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

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

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

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

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

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

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

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

1<a1>!1<a2>!, alpha (sur K+1 blocs de 8). x', y, B, q1, {{x'y},x,K,L}, alpha', B, «c».
Here also we are dividing with x' which could be 10^8 in the exceptional case x=99999999. Must intercept it before sending to \XINT_div_mini.

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

#1=q, #2=alpha (K+4), #3=B, #4=q1, {{x'y},x,K,L}, alpha', BQ«c» -> 1 puis nouveau q sur 8 chiffres. nouvel alpha sur K blocs, B, {{x'y},x,K,L}, alpha', B«c»

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

1 puis nouveau q sur 8 chiffres, #2=nouvel alpha sur K blocs, #3=B, #4={{x'y},x,K,L} avec L à ajuster, alpha', BQ«c» -> {x'y}x,K,L à diminuer de 1, {alpha}B{q}, alpha', BQ«c»

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

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

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

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

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

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

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

Special very small multiplication for division. We only need to cater for multiplicands from 1 to 9. The ending is different from standard `verysmallmul`, a zero carry is not suppressed. And no final `1;!` is added. If multiplicand is just 1 let's not forget to add the zero carry `10000000!` at the end.

```
1544 \def\XINT_div_verysmallmul #1%
1545     {\xint_gob_til_one #1\XINT_div_verysmallisone 1\XINT_div_verysmallmul_a 0\xint:#1}%
1546 \def\XINT_div_verysmallisone 1\XINT_div_verysmallmul_a 0\xint:1!1#11;!%
1547     {1\relax #1100000000!}%
1548 \def\XINT_div_verysmallmul_a #1\xint:#2!1#3!%
1549 {%
1550     \xint_gob_til_sc #3\XINT_div_verysmallmul_e;%
1551     \expandafter\XINT_div_verysmallmul_b
1552     \the\numexpr \xint_c_x^ix+#2*#3+#1\xint:#2!%
1553 }%
1554 \def\XINT_div_verysmallmul_b 1#1#2\xint:%
1555     {1#2\expandafter!\the\numexpr\XINT_div_verysmallmul_a #1\xint:}%
1556 \def\XINT_div_verysmallmul_e;#1;+#2#3!{1\relax 0000000#2!}%
```

Special subtraction for division purposes. If the subtracted thing turns out to be bigger, then just return a `-`. If not, then we must reverse the result, keeping the separators.


```

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

```

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

```

On boucle vers `\XINT_smallldiv_d`.

```

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

```

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

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

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

```

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

```

Note (2015/10/08). Attention à la différence dans l'ordre des arguments avec ce que je vois en dans `\XINT_smallldiv_f`. Je ne me souviens plus du tout s'il y a une raison quelconque.

```

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

```

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

Derived arithmetic

4.38 \xintiiQuo, \xintiiRem

```
1735 \def\xintiiQuo {\romannumeral0\xintiiquo }%
1736 \def\xintiiRem {\romannumeral0\xintiiirem }%
1737 \def\xintiiquo
1738   {\expandafter\xint_stop_atfirstoftwo\romannumeral0\xintiidivision }%
1739 \def\xintiiirem
1740   {\expandafter\xint_stop_atsecondoftwo\romannumeral0\xintiidivision }%
```

4.39 \xintiiDivRound

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

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

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

```

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

```

4.40 \xintiDivTrunc

1.21: \xintiDivTrunc made robust against non terminated input.

```

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

```

Attention to not move DivRound code beyond that point.

```

1787 \let\xINT_iDivTrunc_divbyzero\xINT_iDivRound_divbyzero
1788 \let\xINT_iDivTrunc_aiszero \XINT_iDivRound_aiszero
1789 \def\xINT_iDivTrunc_bpos #1%
1790 {%
1791 \xint_UDsignfork
1792 #1{\xintiOpp\xINT_iDivTrunc_pos {}}%
1793 -{\XINT_iDivTrunc_pos #1}%
1794 \krof
1795 }%
1796 \def\xINT_iDivTrunc_bneg #1%
1797 {%
1798 \xint_UDsignfork
1799 #1{\XINT_iDivTrunc_pos {}}%
1800 -{\xintiOpp\xINT_iDivTrunc_pos #1}%
1801 \krof
1802 }%
1803 \def\xINT_iDivTrunc_pos #1#2\xint:#3\xint:
1804 {\expandafter\xint_stop_atfirstoftwo
1805 \romannumeral0\xINT_div_prepare {#2}{#1#3}}%

```

4.41 \xintiModTrunc

Renamed from \xintiMod to \xintiModTrunc at 1.2p.

```

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

```

```

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

```

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

```

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

```

4.42 \xintiiDivMod

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

```

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

```

```

1856 \romannumeral0\xint_UDsignfork
1857     #1{\XINT_iidivmod_bpos {}}%
1858     -{\XINT_iidivmod_bpos {-#1}}%
1859 \krof
1860 }%
1861 \def\XINT_iidivmod_bneg_finish#1#2%
1862 {%
1863 \expandafter\xint_exchangetwo_keepbraces\expandafter
1864 {\romannumeral0\xintiiopt#2}{#1}%
1865 }%
1866 \def\XINT_iidivmod_bpos #1#2\xint:#3\xint: {\xintiidivision{#1#3}{#2}}%

```

4.43 \xintiiDivFloor

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

```

1867 \def\xintiiDivFloor {\romannumeral0\xintiidivfloor}%
1868 \def\xintiidivfloor {\expandafter\xint_stop_atfirstoftwo
1869 \romannumeral0\xintiidivmod}%

```

4.44 \xintiiMod

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

```

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

```

4.45 \xintiiSqr

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

```

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

```

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


```

1889 \def\XINT_sqr_a #1\xint:
1890 {%
1891     \ifnum #1=\xint_c_i \expandafter\XINT_sqr_small
1892         \else\expandafter\XINT_sqr_start\fi
1893 }%
1894 \def\XINT_sqr_small 1#1#2#3#4#5!\xint:
1895 {%
1896     \ifnum #1#2#3#4#5<46341 \expandafter\XINT_sqr_verysmall\fi
1897     \expandafter\XINT_sqr_small_out
1898     \the\numexpr\XINT_minimul_a #1#2#3#4\xint:#5!#1#2#3#4#5!%
1899 }%
1900 \def\XINT_sqr_verysmall#1{%
1901 \def\XINT_sqr_verysmall
1902     \expandafter\XINT_sqr_small_out\the\numexpr\XINT_minimul_a ##1!##2!%
1903     {\expandafter#1\the\numexpr ##2*##2\relax}%
1904 }\XINT_sqr_verysmall{ }%
1905 \def\XINT_sqr_small_out 1#1!1#2!%
1906 {%
1907     \XINT_cuz #2#1\R
1908 }%

```

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.

```

1909 \def\XINT_sqr_start #1\xint:
1910 {%
1911     \expandafter\XINT_mul_out
1912     \the\numexpr\XINT_mul_loop
1913         100000000!1;!W #11;!W #11;!%
1914     1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1915 }%

```

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.

```

1916 \def\xintiiPow {\romannumeral0\xintiipow }%
1917 \def\xintiipow #1#2%
1918 {%
1919     \expandafter\xint_pow\the\numexpr #2\expandafter
1920     .\romannumeral`&&@#1\xint:
1921 }%
1922 \def\xint_pow #1.#2%#3\xint:
1923 {%
1924     \xint_UDzerominusfork
1925     #2-\XINT_pow_AisZero
1926     0#2\XINT_pow_Aneg

```

```

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

```

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

```

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

```

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

```

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

```



```

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

```

```

2129 {%
2130   \XINT_fac_smallloop_loop 2.#1.100000001!1;!%
2131 }%
2132 \expandafter\def\csname XINT_fac_smallloop_-2\endcsname #1.%
2133 {%
2134   \XINT_fac_smallloop_loop 3.#1.100000002!1;!%
2135 }%
2136 \expandafter\def\csname XINT_fac_smallloop_-1\endcsname #1.%
2137 {%
2138   \XINT_fac_smallloop_loop 4.#1.100000006!1;!%
2139 }%
2140 \expandafter\def\csname XINT_fac_smallloop_0\endcsname #1.%
2141 {%
2142   \XINT_fac_smallloop_loop 5.#1.1000000024!1;!%
2143 }%
2144 \def\XINT_fac_smallloop_loop #1.#2.%
2145 {%
2146   \ifnum #1>#2 \expandafter\XINT_fac_loop_exit\fi
2147   \expandafter\XINT_fac_smallloop_loop
2148   \the\numexpr #1+\xint_c_iv\expandafter.%
2149   \the\numexpr #2\expandafter.\the\numexpr\XINT_fac_smallloop_mul #1!%
2150 }%
2151 \def\XINT_fac_smallloop_mul #1!%
2152 {%
2153   \expandafter\XINT_smallmul
2154   \the\numexpr
2155     \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2156 }%
2157 \def\XINT_fac_loop_exit #1!#2;!#3{#3#2;!}%

```

4.48 \XINT_useiimessage

1.2o

```

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

```

5 Package *xint* implementation

<p>.1 Package identification 121</p> <p>.2 More token management 121</p> <p>.3 (WIP) A constant needed by <code>\xintRandomDigits</code> et al. 121</p> <p>.4 <code>\xintLen</code>, <code>\xintiLen</code> 122</p> <p>.5 <code>\xintiiLogTen</code> 122</p> <p>.6 <code>\xintReverseDigits</code> 122</p> <p>.7 <code>\xintiiE</code> 123</p> <p>.8 <code>\xintDecSplit</code> 124</p> <p>.9 <code>\xintDecSplitL</code> 125</p> <p>.10 <code>\xintDecSplitR</code> 126</p> <p>.11 <code>\xintDSHr</code> 126</p> <p>.12 <code>\xintDSH</code> 127</p> <p>.13 <code>\xintDSx</code> 127</p> <p>.14 <code>\xintiiEq</code> 129</p> <p>.15 <code>\xintiiNotEq</code> 129</p> <p>.16 <code>\xintiiGeq</code> 129</p> <p>.17 <code>\xintiiGt</code> 130</p> <p>.18 <code>\xintiiLt</code> 130</p> <p>.19 <code>\xintiiGtorEq</code> 130</p> <p>.20 <code>\xintiiLtorEq</code> 130</p> <p>.21 <code>\xintiiIsZero</code> 130</p> <p>.22 <code>\xintiiIsNotZero</code> 130</p> <p>.23 <code>\xintiiIsOne</code> 130</p> <p>.24 <code>\xintiiOdd</code> 131</p> <p>.25 <code>\xintiiEven</code> 131</p> <p>.26 <code>\xintiiMON</code> 131</p> <p>.27 <code>\xintiiMMON</code> 131</p> <p>.28 <code>\xintSgnFork</code> 132</p> <p>.29 <code>\xintiiifSgn</code> 132</p> <p>.30 <code>\xintiiifCmp</code> 132</p> <p>.31 <code>\xintiiifEq</code> 133</p> <p>.32 <code>\xintiiifGt</code> 133</p> <p>.33 <code>\xintiiifLt</code> 133</p> <p>.34 <code>\xintiiifZero</code> 133</p>	<p>.35 <code>\xintiiifNotZero</code> 134</p> <p>.36 <code>\xintiiifOne</code> 134</p> <p>.37 <code>\xintiiifOdd</code> 134</p> <p>.38 <code>\xintifTrueAelseB</code>, <code>\xintifFalseAelseB</code> 134</p> <p>.39 <code>\xintIsTrue</code>, <code>\xintIsFalse</code> 135</p> <p>.40 <code>\xintNOT</code> 135</p> <p>.41 <code>\xintAND</code>, <code>\xintOR</code>, <code>\xintXOR</code> 135</p> <p>.42 <code>\xintANDof</code> 135</p> <p>.43 <code>\xintORof</code> 136</p> <p>.44 <code>\xintXORof</code> 136</p> <p>.45 <code>\xintiiMax</code> 136</p> <p>.46 <code>\xintiiMin</code> 137</p> <p>.47 <code>\xintiiMaxof</code> 138</p> <p>.48 <code>\xintiiMinof</code> 139</p> <p>.49 <code>\xintiiSum</code> 139</p> <p>.50 <code>\xintiiPrd</code> 140</p> <p>.51 <code>\xintiiSquareRoot</code> 141</p> <p>.52 <code>\xintiiSqrt</code>, <code>\xintiiSqrTR</code> 148</p> <p>.53 <code>\xintiiBinomial</code> 148</p> <p>.54 <code>\xintiiPFactorial</code> 154</p> <p>.55 <code>\xintBool</code>, <code>\xintToggle</code> 157</p> <p>.56 <code>\xintiiGCD</code> 157</p> <p>.57 <code>\xintiiLCM</code> 158</p> <p>.58 <code>\xintiiGCDof</code> 158</p> <p>.59 <code>\xintiiLCMof</code> 159</p> <p>.60 (WIP) <code>\xintRandomDigits</code> 159</p> <p>.61 (WIP) <code>\XINT_eightranddigits</code>, <code>\xintEightRandomDigits</code> 159</p> <p>.62 (WIP) <code>\xintRandBit</code> 160</p> <p>.63 (WIP) <code>\xintXRandomDigits</code> 160</p> <p>.64 (WIP) <code>\xintiiRandRangeAtoB</code> 160</p> <p>.65 (WIP) <code>\xintiiRandRange</code> 161</p> <p>.66 (WIP) Adjustments for engines without <code>uniformdeviate</code> primitive 162</p>
---	--

With release 1.1 the core arithmetic routines `\xintiiAdd`, `\xintiiSub`, `\xintiiMul`, `\xintiiQuo`, `\xintiiPow` were separated to be the main component of the then new *xintcore*.

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

1.3b adds randomness related macros.

```

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

```



```

11 \catcode58=12 % :
12 \let\z\endgroup
13 \expandafter\let\expandafter\x\csname ver@xint.sty\endcsname
14 \expandafter\let\expandafter\w\csname ver@xintcore.sty\endcsname
15 \expandafter
16 \ifx\csname PackageInfo\endcsname\relax
17 \def\y#1#2{\immediate\write-1{Package #1 Info: #2.}}%
18 \else
19 \def\y#1#2{\PackageInfo{#1}{#2}}%
20 \fi
21 \expandafter
22 \ifx\csname numexpr\endcsname\relax
23 \y{xint}{\numexpr not available, aborting input}%
24 \aftergroup\endinput
25 \else
26 \ifx\x\relax % plain-TeX, first loading of xintcore.sty
27 \ifx\w\relax % but xintkernel.sty not yet loaded.
28 \def\z{\endgroup\input xintcore.sty\relax}%
29 \fi
30 \else
31 \def\empty {}%
32 \ifx\x\empty % LaTeX, first loading,
33 % variable is initialized, but \ProvidesPackage not yet seen
34 \ifx\w\relax % xintcore.sty not yet loaded.
35 \def\z{\endgroup\RequirePackage{xintcore}}%
36 \fi
37 \else
38 \aftergroup\endinput % xint already loaded.
39 \fi
40 \fi
41 \fi
42 \z%
43 \XINTsetupcatcodes% defined in xintkernel.sty (loaded by xintcore.sty)

```

5.1 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xint}%
46 [2020/02/25 v1.4b 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 $\text{len}(A)+\text{len}(B)-1$ (somewhat arbitrary). It applies `\xintNum` to its argument. A minus sign is accepted and ignored.

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

`\xintLen` gets redefined by `xintfrac`.

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

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

5.5 `\xintiLogTen`

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

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

5.6 `\xintReverseDigits`

1.2.

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

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

This macro is currently not used elsewhere in `xint` code.

```

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

```

5.7 \xintiiE

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

```

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

```

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

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

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

5.8 \xintDecSplit

DECIMAL SPLIT

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

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

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

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

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

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

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

```

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

190 \def\XINT_split_fromleft_end_b #1\xint_bye#2\xint_bye.{.#1}% puis .
191 \def\XINT_split_fromright #1.#2\xint_bye
192 {%
193 \expandafter\XINT_split_fromright_a
194 \the\numexpr#1-\numexpr\XINT_length_loop
195 #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\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

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

```

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

```

5.25 \xintiiEven

```

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

```

5.26 \xintiiMON

MINUS ONE TO THE POWER N

```

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

```

5.27 \xintiiMMON

MINUS ONE TO THE POWER N-1

```

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

```

```

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

```

5.28 `\xintSgnFork`

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

```

425 \def\xintSgnFork {\romannumeral0\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\xintiiszero}%
```

5.41 `\xintAND`, `\xintOR`, `\xintXOR`

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

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

5.42 `\xintANDof`

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

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

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

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_zeroplus % 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 `\xintiiMaxof`

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

NOT compatible with empty list.

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

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

```

642 \def\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 `\xintiSum`

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

```

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

```

```

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

    bad coding as it depends on internal conventions of \XINT_add_nfork

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

    see previous comment!

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

```

5.50 \xintiiPrd

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

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

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

```

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

```

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

```

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

    bad coding as it depends on internal conventions of \XINT_mul_nfork

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

```

```

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

see previous comment!

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

```

5.51 \xintiiSquareRoot

First done with 1.08.

1.1 added \xintiiSquareRoot.

1.1a added \xintiiSqrtR.

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

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

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

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

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

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

```

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

```

```

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

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

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

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

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

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

```

```

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

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

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

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

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

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

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

```

```

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

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

834 \def\xINT_sqrt_big_eE_a #1.#2;#3%
835 {%
836 \expandafter\xINT_sqrt_bigormed_f
837 \romannumeral0\xINT_sqrt_small_e #2000.#3.#1;%
838 }%

839 \def\xINT_sqrt_big_e0 #1;#2#3#4#5#6#7#8#9%
840 {%
841 \XINT_sqrt_big_e0_a #1;{#2#3#4#5#6#7#8#9}%
842 }%
843 \def\xINT_sqrt_big_e0_a #1.#2;#3#4%
844 {%
845 \expandafter\xINT_sqrt_bigormed_f
846 \romannumeral0\xINT_sqrt_small_e #20000.#3#4.#1;%
847 }%

848 \def\xINT_sqrt_bigormed_f #1#2#3;%
849 {%
850 \ifnum#3<\xint_c_ix
851 \xint_dothis {\csname XINT_sqrt_med_f\romannumeral#3\endcsname}%
852 \fi
853 \xint_orthat\xINT_sqrt_big_f #1.#2.#3;%
854 }%
855 \def\xINT_sqrt_med_fv {\XINT_sqrt_med_fa .}%
856 \def\xINT_sqrt_med_fvi {\XINT_sqrt_med_fa 0.}%
857 \def\xINT_sqrt_med_fvii {\XINT_sqrt_med_fa 00.}%
858 \def\xINT_sqrt_med_fviii{\XINT_sqrt_med_fa 000.}%

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

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

```



```

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

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

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

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

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

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

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

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

```

```

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

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

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

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

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

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

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

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

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

```

```

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

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

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

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

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

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

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

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

5.52 \xintiiSqrt, \xintiiSqrtR

```
1006 \def\xintiiSqrt {\romannumeral0\xintiisqrt }%
1007 \def\xintiisqrt {\expandafter\XINT_sqrt_post\romannumeral0\xintiisquareroot }%
1008 \def\XINT_sqrt_post #1#2{\XINT_dec #1\XINT_dec_bye234567890\xint_bye}%
1009 \def\xintiiSqrtR {\romannumeral0\xintiisqrtr }%
1010 \def\xintiisqrtr {\expandafter\XINT_sqrtr_post\romannumeral0\xintiisquareroot }%
```

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

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

5.53 \xintiiBinomial

2015/11/28-29 for 1.2f.

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

I really lack some kind of infinity or NaN value.

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

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

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

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

```

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

```

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

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

```

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

```

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

```

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

```

y.j.k. Au départ on avait $x - k + 1$. 1 . k . Ensuite on a des blocs $1 < 8d >!$ donnant le résultat intermédiaire, dans l'ordre, et à la fin on a $1! 1! ; ! 0!$. Dans *smallloop* on peut prendre 4 par 4.

```

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

```

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

```

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

```

```

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

```

Ici on prend trois par trois.

```

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

```

Ici on prend deux par deux.

```

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

```

```

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

```

Et finalement un par un.

```

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

```

y.j.k. La partie very small. y est au plus 26 (non 29 mais retesté dans \XINT_binom_vsmallloop_a), et tous les binomial(29,n) sont $<10^8$. On peut donc faire $y(y+1)(y+2)(y+3)$ et aussi il y a le fait que etex fait $a*b/c$ en double precision. Pour ne pas bifurquer à la fin sur smallloop, si $n=27, 27$, ou 29 on procède un peu différemment des autres boucles. Si je testais aussi #1 après #3-#2 pour les autres il faudrait des terminaisons différentes.

```

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

```

```

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

```

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

```

1180 \def\XINT_binom_vsmallmuldiv #1!#2!1#3!\{\xint_c_x^viii+#2*#3/#1!}%

```

On a des terminaisons communes aux trois situations small, med, big, et on est sûr de pouvoir faire les multiplications dans \numexpr, car on vient ici *après* avoir comparé à 9999 ou 463 ou 98.

```

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

```



```

1198 {%
1199   \expandafter\XINT_binom_finish
1200   \the\numexpr\expandafter\XINT_binom_div
1201     \the\numexpr #2*(#2+\xint_c_i)\expandafter
1202     !\romannumeral0\expandafter\XINT_binom_mul
1203     \the\numexpr #1*(#1+\xint_c_i)!%
1204 }%
1205 \def\XINT_binom_end_ #1.#2.#3.%
1206 {%
1207   \expandafter\XINT_binom_finish
1208   \the\numexpr\expandafter\XINT_binom_div\the\numexpr #2\expandafter
1209   !\romannumeral0\XINT_binom_mul #1!%
1210 }%
1211 \def\XINT_binom_finish #1;!0!%
1212   {\XINT_unsep_cuzsmall #1\xint_bye!2!3!4!5!6!7!8!9!\xint_bye\xint_c_i\relax}%

```

Duplication de code seulement pour la boucle avec très petits coeffs, mais en plus on fait au maximum des possibilités. (on pourrait tester plus le résultat déjà obtenu).

```

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

```

```

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

```

5.54 \xintiPFactorial

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.

```

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

```

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

```

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

```

```

1285 \xint_orthat
1286 {\ifodd\numexpr#2-#1\relax\xint_afterfi{\expandafter-\romannumeral`&&@\}\fi
1287 \expandafter\XINT_pfac_a }%
1288 \the\numexpr #2-\xint_c_i\expandafter.\the\numexpr#1-\xint_c_i.%
1289 }%
1290 \def\XINT_pfac_a #1.#2.%
1291 {%
1292 \expandafter\XINT_pfac_b\the\numexpr \xint_c_i+#1.#2.1000000011;!%
1293 1\R!1\R!1\R!1\R!1\R!1\R!1\R!1\R!\W
1294 }%
1295 \def\XINT_pfac_b #1.%
1296 {%
1297 \ifnum #1>9999 \xint_dothis\XINT_pfac_vbigloop \fi
1298 \ifnum #1>463 \xint_dothis\XINT_pfac_bigloop \fi
1299 \ifnum #1>98 \xint_dothis\XINT_pfac_medloop \fi
1300 \xint_orthat\XINT_pfac_smallloop #1.%
1301 }%
1302 \def\XINT_pfac_smallloop #1.#2.%
1303 {%
1304 \ifcase\numexpr #2-#1\relax
1305 \expandafter\XINT_pfac_end_
1306 \or \expandafter\XINT_pfac_end_i
1307 \or \expandafter\XINT_pfac_end_ii
1308 \or \expandafter\XINT_pfac_end_iii
1309 \else\expandafter\XINT_pfac_smallloop_a
1310 \fi #1.#2.%
1311 }%
1312 \def\XINT_pfac_smallloop_a #1.#2.%
1313 {%
1314 \expandafter\XINT_pfac_smallloop_b
1315 \the\numexpr #1+\xint_c_iv\expandafter.%
1316 \the\numexpr #2\expandafter.%
1317 \the\numexpr\expandafter\XINT_smallmul
1318 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
1319 }%
1320 \def\XINT_pfac_smallloop_b #1.%
1321 {%
1322 \ifnum #1>98 \expandafter\XINT_pfac_medloop \else
1323 \expandafter\XINT_pfac_smallloop \fi #1.%
1324 }%
1325 \def\XINT_pfac_medloop #1.#2.%
1326 {%
1327 \ifcase\numexpr #2-#1\relax
1328 \expandafter\XINT_pfac_end_
1329 \or \expandafter\XINT_pfac_end_i
1330 \or \expandafter\XINT_pfac_end_ii
1331 \else\expandafter\XINT_pfac_medloop_a
1332 \fi #1.#2.%
1333 }%
1334 \def\XINT_pfac_medloop_a #1.#2.%
1335 {%
1336 \expandafter\XINT_pfac_medloop_b

```

```

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

```

```

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

```

5.55 \xintBool, \xintToggle

1.09c

```

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

```

5.56 \xintiigcd

Copied over `\xintiigcd` code from `xintgcd` at 1.3d in order to support `gcd()` function in `\xintexpr`.
 At 1.4 original code removed from `xintgcd` as the latter now requires `xint`.

```

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

```

```

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

```

5.57 \xintiilCM

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

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

```

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

```

5.58 \xintiGCDof

New with 1.09a (`xintgcd.sty`).

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

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

```

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

```

5.59 \xintiilCMof

See comments of \xintiigCDof

```

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

```

5.60 (WIP) \xintRandomDigits

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

```

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

```

5.61 (WIP) \XINT_eightrandomdigits, \xintEightRandomDigits

1.3b. 1.4 adds some public alias...

```

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

```

5.62 (WIP) \xintRandBit

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

```

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

```

5.63 (WIP) \xintXRandomDigits

1.3b.

```

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

```

5.64 (WIP) \xintiiRandRangeAtoB

1.3b. Support for randrange() function.

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

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

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

```

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

```



```

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

```

5.65 (WIP) \xintiiRandRange

1.3b. Support for randrange().

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

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

```

5.66 (WIP) Adjustments for engines without uniformdeviate primitive

1.3b.

```

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

```

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

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

6 Package *xintbinhex* implementation

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

The commenting is currently (2020/02/25) 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 ΗΕΙΚΟ ΟΒΕΡΔΙΕΚ's packages, then modified.

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

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2  \catcode13=5    % ^^M
3  \endlinechar=13 %
4  \catcode123=1  % {
5  \catcode125=2  % }
6  \catcode64=11  % @
7  \catcode35=6   % #
8  \catcode44=12  % ,
9  \catcode45=12  % -
10 \catcode46=12  % .
11 \catcode58=12  % :
12 \let\z\endgroup
13 \expandafter\let\expandafter\x\csname ver@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 [2020/02/25 v1.4b Expandable binary and hexadecimal conversions (JFB)]%

```

6.3 Constants, etc...

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

```

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

```

```

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

```

6.4 Helper macros

6.4.1 \XINT_zeroes_foriv

```

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

```

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

```

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

```

6.5 \xintDecToHex

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

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

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

```

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

```

```

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

```

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

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

```

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

```

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

```

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

```

```

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

```

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

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


```

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

```

6.6 \xintDecToBin

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

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

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

Most of the code canvas is shared with \xintDecToHex.

```

211 \def\xintDecToBin {\romannumeral0\xintdectobin }%
212 \def\xintdectobin #1%
213 {%
214   \expandafter\XINT_dtb_checkin\romannumeral`&&@#1\xint:
215 }%
216 \def\XINT_dtb_checkin #1%
217 {%
218   \xint_UDsignfork
219     #1\XINT_dtb_neg
220     -{\XINT_dtb_main #1}%
221   \krof
222 }%
223 \def\XINT_dtb_neg {\expandafter-\romannumeral0\XINT_dtb_main}%
224 \def\XINT_dtb_main #1\xint:
225 {%
226   \expandafter\XINT_dtb_finish
227   \romannumeral`&&\expandafter\XINT_dtb_start
228   \romannumeral0\XINT_zeroes_foriv
229   #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
230   #1\xint_bye\XINT_dtb_tobin
231 }%
232 \def\XINT_dtb_tobin
233 {%
234   \expandafter\expandafter\expandafter\XINT_dtb_tobin_a\cename\XINT_tosixteenbits
235 }%
236 \def\XINT_dtb_tobin_a\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\xintbintohehex }%
400 \def\xintbintohehex #1%
401 {%
402     \expandafter\XINT_bth_checkin\romannumeral`&&@#1\xint:
403 }%
404 \def\XINT_bth_checkin #1%
405 {%
406     \xint_UDsignfork
407     #1\XINT_bth_N
408     -{\XINT_bth_main #1}%
409     \krof
410 }%
411 \def\XINT_bth_N {\expandafter-\romannumeral0\XINT_bth_main }%
412 \def\XINT_bth_main #1\xint:
413 {%
414     \csname space\csname\expandafter\XINT_bth_loop
415     \romannumeral0\XINT_zeroes_foriv
416     #1\R{0\R}{00\R}{000\R}\R{0\R}{00\R}{000\R}\R\W
417     #1\xint_bye2345678\xint_bye none\endcsname
418 }%
419 \def\XINT_bth_loop #1#2#3#4#5#6#7#8%
420 {%
421     XINT_csbth_#1#2#3#4%

```

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

6.10 \xintHexToBin

Completely rewritten for 1.2m.

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

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

Rewritten again at 1.2n for \csname governed expansion.

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

6.11 \xintCHexToBin

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

Rewritten again at 1.2n for \csname governed expansion.

```
457 \def\xintCHexToBin {\romannumeral0\xintchextobin }%
```

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

7 Package [xintgcd](#) implementation

.1	Catcodes, 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 (2020/02/25) very sparse.

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

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 $\mathbb{H}\mathbb{E}\mathbb{I}\mathbb{K}\mathbb{O}$ $\mathbb{O}\mathbb{B}\mathbb{E}\mathbb{R}\mathbb{D}\mathbb{I}\mathbb{E}\mathbb{K}$'s packages, then modified. The method for catcodes was also initially directly inspired by these packages.

```

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

```



```

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

```

7.2 Package identification

```

51 \XINT_providespackage
52 \ProvidesPackage{xintgcd}%
53 [2020/02/25 v1.4b Euclide 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\xintiopp{#2}}{\romannumeral0\xintiopp{#1}}%
103 }%
104 \def\XINT_bezout_mm_postb #1#2{\expandafter{#2}{#1}}%

```

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

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

plusminus A < 0, B > 0

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

```

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

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

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

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

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

```

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

```

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

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

```

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

```

```

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

```

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

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

```

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

```

7.4 \xintEuclideanAlgorithm

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

$u<2n> = u<2n+3>u<2n+2> + u<2n+4>$ à la n ième étape.

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

```

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

```

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

```

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

```

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

```

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

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

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

```

```

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

222 \def\XINT_euc_b #1.#2#3#4%
223 {%
224     \XINT_euc_c #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\xintiiabs{\xintNum{#1}}}%
246 }%
247 \def\XINT_bezalg #1#2%
248 {%
249 \expandafter\XINT_bezalg_fork\romannumeral0\xintiiabs{\xintNum{#2}}\Z #1\Z
250 }%

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

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

    pour préparer l'étape n+1 il faut {n}{r(n)}{r(n-1)}{alpha(n)}{beta(n)}{alpha(n-1)}{beta(n-1)}
    {{q(n)}{r(n)}{alpha(n)}{beta(n)}}... division de #3 par #2

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

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

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

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

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

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

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

```

$r(n+1)\backslash Z$ $\{n+1\}\{r(n+1)\}\{r(n)\}\{\alpha(n+1)\}\{\beta(n+1)\}$
 $\{\alpha(n)\}\{\beta(n)\}\{q,r,\alpha,\beta(n+1)\}$
 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.
 $\{n+1\}\{r(n+1)\}\{r(n)\}\{\alpha(n+1)\}\{\beta(n+1)\}\{\alpha(n)\}\{\beta(n)\}$
 $\{q,r,\alpha,\beta(n+1)\} \dots \{\{A\}\{B\}\}\backslash Z$
 On veut renvoyer
 $\{N\}\{A\}\{0\}\{1\}\{D=r(n)\}\{B\}\{1\}\{0\}\{q_1\}\{r_1\}\{\alpha_1=q_1\}\{\beta_1=1\}$
 $\{q_2\}\{r_2\}\{\alpha_2\}\{\beta_2\} \dots \{q_N\}\{r_N=0\}\{\alpha_N=A/D\}\{\beta_N=B/D\}$

```
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 }%
```

$\{N\}\{D\}\{A\}\{B\}\{q_1\}\{r_1\}\{\alpha_1=q_1\}\{\beta_1=1\}\{q_2\}\{r_2\}\{\alpha_2\}\{\beta_2\}$
 $\dots\{q_N\}\{r_N=0\}\{\alpha_N=A/D\}\{\beta_N=B/D\}$
 On veut renvoyer
 $\{N\}\{A\}\{0\}\{1\}\{D=r(n)\}\{B\}\{1\}\{0\}\{q_1\}\{r_1\}\{\alpha_1=q_1\}\{\beta_1=1\}$
 $\{q_2\}\{r_2\}\{\alpha_2\}\{\beta_2\} \dots \{q_N\}\{r_N=0\}\{\alpha_N=A/D\}\{\beta_N=B/D\}$

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

7.6 \xintTypesetEuclideanAlgorithm

TYPESETTING

Organisation:

$\{N\}\{A\}\{D\}\{B\}\{q_1\}\{r_1\}\{q_2\}\{r_2\}\{q_3\}\{r_3\} \dots \{q_N\}\{r_N=0\}$
 $\backslash U1 = N = \text{nombre d'étapes}, \backslash U3 = \text{PGCD}, \backslash U2 = A, \backslash U4=B$ $q_1 = \backslash U5, q_2 = \backslash U7 \rightarrow q_n = \backslash U<2n+3>, r_n =$
 $\backslash U<2n+4> b_n = r_n. B = r_0. A=r(-1)$
 $r(n-2) = q(n)r(n-1)+r(n)$ (n e étape)
 $\backslash U\{2n\} = \backslash U\{2n+3\} \times \backslash U\{2n+2\} + \backslash U\{2n+4\}$, n e étape. (avec n entre 1 et N)
 1.09h uses \xintloop, and \par rather than \endgraf; and \par rather than \hfill\break

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

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

$\alpha(n) = U\{4n+7\}$, n au moins -1

$\beta(n) = U\{4n+8\}$, n au moins -1

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

```

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     $\U\times\A - \V\times\B = \D$%
364     \fi
365     \par
366     \endgroup
367 }%
368 \XINT_restorecatcodes_endinput%
```

8 Package **xintfrac** implementation

.1	Catcodes, ε -TeX and reload detection	188	.46	<code>\xintPrd</code>	224
.2	Package identification	189	.47	<code>\xintDiv</code>	224
.3	<code>\XINT_cntSgnFork</code>	189	.48	<code>\xintDivFloor</code>	225
.4	<code>\xintLen</code>	189	.49	<code>\xintDivTrunc</code>	225
.5	<code>\XINT_outfrac</code>	189	.50	<code>\xintDivRound</code>	225
.6	<code>\XINT_inFrac</code>	190	.51	<code>\xintModTrunc</code>	225
.7	<code>\XINT_frac_gen</code>	192	.52	<code>\xintDivMod</code>	226
.8	<code>\XINT_factortens</code>	194	.53	<code>\xintMod</code>	227
.9	<code>\xintEq</code> , <code>\xintNotEq</code> , <code>\xintGt</code> , <code>\xintLt</code> , <code>\xintGtorEq</code> , <code>\xintLtorEq</code> , <code>\xintIsZero</code> , <code>\xintIsNotZero</code> , <code>\xintOdd</code> , <code>\xintEven</code> , <code>\xintifSgn</code> , <code>\xintifCmp</code> , <code>\xintifEq</code> , <code>\xintifGt</code> , <code>\xintifLt</code> , <code>\xintifZero</code> , <code>\xintifNotZero</code> , <code>\xintifOne</code> , <code>\xintifOdd</code>	195	.54	<code>\xintIsOne</code>	228
.10	<code>\xintRaw</code>	197	.55	<code>\xintGeq</code>	228
.11	<code>\xintiLogTen</code>	197	.56	<code>\xintMax</code>	229
.12	<code>\xintPRaw</code>	198	.57	<code>\xintMaxof</code>	230
.13	<code>\xintSPRaw</code>	198	.58	<code>\xintMin</code>	231
.14	<code>\xintFracToSci</code> , <code>\xintFracToSciE</code>	199	.59	<code>\xintMinof</code>	231
.15	<code>\xintRawWithZeros</code>	200	.60	<code>\xintCmp</code>	232
.16	<code>\xintDecToString</code>	201	.61	<code>\xintAbs</code>	233
.17	<code>\xintFloor</code> , <code>\xintiFloor</code>	201	.62	<code>\xintOpp</code>	233
.18	<code>\xintCeil</code> , <code>\xintiCeil</code>	201	.63	<code>\xintInv</code>	233
.19	<code>\xintNumerator</code>	202	.64	<code>\xintSgn</code>	234
.20	<code>\xintDenominator</code>	202	.65	<code>\xintGCD</code> , <code>\xintLCM</code>	234
.21	<code>\xintFrac</code>	202	.66	<code>\xintGCDof</code>	234
.22	<code>\xintSignedFrac</code>	203	.67	<code>\xintLCMof</code>	235
.23	<code>\xintFwOver</code>	203	.68	Floating point macros	236
.24	<code>\xintSignedFwOver</code>	204	.69	<code>\xintDigits</code> , <code>\xintSetDigits</code>	237
.25	<code>\xintREZ</code>	204	.70	<code>\xintFloat</code>	237
.26	<code>\xintE</code>	205	.71	<code>\XINTinFloat</code> , <code>\XINTinFloatS</code> , <code>\XINTiLogTen</code>	238
.27	<code>\xintIrr</code> , <code>\xintPIrr</code>	205	.72	<code>\xintPFloat</code> , <code>\xintPFloatE</code>	245
.28	<code>\xintifInt</code>	207	.73	<code>\XINTinFloatFracdigits</code>	247
.29	<code>\xintIsInt</code>	207	.74	<code>\xintFloatAdd</code> , <code>\XINTinFloatAdd</code>	247
.30	<code>\xintJrr</code>	207	.75	<code>\xintFloatSub</code> , <code>\XINTinFloatSub</code>	248
.31	<code>\xintTFrac</code>	209	.76	<code>\xintFloatMul</code> , <code>\XINTinFloatMul</code>	249
.32	<code>\xintTrunc</code> , <code>\xintiTrunc</code>	209	.77	<code>\XINTinFloatInv</code>	249
.33	<code>\xintTTrunc</code>	212	.78	<code>\xintFloatDiv</code> , <code>\XINTinFloatDiv</code>	250
.34	<code>\xintNum</code>	212	.79	<code>\xintFloatPow</code> , <code>\XINTinFloatPow</code>	251
.35	<code>\xintRound</code> , <code>\xintiRound</code>	212	.80	<code>\xintFloatPower</code> , <code>\XINTinFloatPower</code>	254
.36	<code>\xintXTrunc</code>	213	.81	<code>\xintFloatFac</code> , <code>\XINTFloatFac</code>	258
.37	<code>\xintAdd</code>	219	.82	<code>\xintFloatPFactorial</code> , <code>\XINTinFloatPFactorial</code>	263
.38	<code>\xintSub</code>	221	.83	<code>\xintFloatBinomial</code> , <code>\XINTinFloatBinomial</code>	267
.39	<code>\xintSum</code>	221	.84	<code>\xintFloatSqrt</code> , <code>\XINTinFloatSqrt</code>	268
.40	<code>\xintMul</code>	221	.85	<code>\xintFloatE</code> , <code>\XINTinFloatE</code>	271
.41	<code>\xintSqr</code>	222	.86	<code>\XINTinFloatMod</code>	272
.42	<code>\xintPow</code>	222	.87	<code>\XINTinFloatDivFloor</code>	272
.43	<code>\xintFac</code>	223	.88	<code>\XINTinFloatDivMod</code>	272
.44	<code>\xintBinomial</code>	223	.89	<code>\xintifFloatInt</code>	273
.45	<code>\xintPFactorial</code>	223	.90	<code>\xintFloatIsInt</code>	273
			.91	<code>\XINTinFloatdigits</code> , <code>\XINTinFloatSqrt-digits</code> , <code>\XINTinFloatFacdigits</code> , <code>\XINTinFloatFacdigits</code>	

<p>TiLogTendigits 273</p> <p>.92 (WIP) \XINTinRandomFloatS, \XINTinRandomFloatSdigits 273</p> <p>.93 (WIP) \XINTinRandomFloatSixteen 274</p> <p>.94 \PoorManLogBaseTen 274</p>	<p>.95</p> <p>.96</p> <p>.97</p>	<p>\PoorManPowerOfTen 275</p> <p>\PoorManPower 275</p> <p>Support macros for natural logarithm and exponential <i>xintexpr</i> functions 276</p>
--	----------------------------------	--

The commenting is currently (2020/02/25) very sparse.

8.1 Catcodes, ϵ -TeX and reload detection

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

```

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

```

```
41 \fi
42 \z%
43 \XINTsetupcatcodes% defined in xintkernel.sty
```

8.2 Package identification

```
44 \XINT_providespackage
45 \ProvidesPackage{xintfrac}%
46 [2020/02/25 v1.4b Expandable operations on fractions (JFB)]%
```

8.3 \XINT_cntSgnFork

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

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

8.4 \xintLen

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

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

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

8.5 \XINT_outfrac

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

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

```

69 {%
70   \ifcase\XINT_cntSgn #3\xint:
71     \expandafter \XINT_outfrac_divisionbyzero
72   \or
73     \expandafter \XINT_outfrac_P
74   \else
75     \expandafter \XINT_outfrac_N
76   \fi
77   {#2}{#3}[#1]%
78 }%
79 \def\XINT_outfrac_divisionbyzero #1#2%
80 {%
81   \XINT_signalcondition{DivisionByZero}{Division of #1 by #2}{0/1[0]}%
82 }%
83 \def\XINT_outfrac_P#1{%
84 \def\XINT_outfrac_P ##1##2%
85   {\if0\XINT_Sgn ##1\xint:\expandafter\XINT_outfrac_Zero\fi##1/##2}%
86 }\XINT_outfrac_P{ }%
87 \def\XINT_outfrac_Zero #1[#2]{ 0/1[0]}%
88 \def\XINT_outfrac_N #1#2%
89 {%
90   \expandafter\XINT_outfrac_N_a\expandafter
91   {\romannumeral0\XINT_opp #2}{\romannumeral0\XINT_opp #1}%
92 }%
93 \def\XINT_outfrac_N_a #1#2%
94 {%
95   \expandafter\XINT_outfrac_P\expandafter {#2}{#1}%
96 }%

```

8.6 \XINT_inFrac

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

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

2015/10/09: this venerable macro from the very early days (1.03, 2013/04/14) has gotten a 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:\xint:
162 \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
163 \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
164 ~#2#1!%
165 }%
166 \def\XINT_frac_gen_C #1!#2.#3%
167 {%
168 \xint_UDXINTWfork
169 #3\XINT_frac_gen_Ca
170 \XINT_W\XINT_frac_gen_Cb
171 \krof
172 #1!#2.#3%
173 }%
174 \def\XINT_frac_gen_Ca #1~#2!#3e#4e#5\XINT_T
175 {%
176 \expandafter\XINT_frac_gen_F\the\numexpr #4-#1\expandafter
177 ~\romannumeral0\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
178 #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z~#3~%
179 }%
180 \def\XINT_frac_gen_Cb #1.#2e%
181 {%
182 \expandafter\XINT_frac_gen_Cc\romannumeral`&&@#2.#1e%
183 }%
184 \def\XINT_frac_gen_Cc #1.#2~#3!#4e#5e#6\XINT_T
185 {%
186 \expandafter\XINT_frac_gen_F\the\numexpr #5-#2-%

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

191 \relax\expandafter~%
192 \romannumeral0\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
193 #3\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z
194 ~#4#1~%
195 }%
196 \def\XINT_frac_gen_F #1~#2%
197 {%
198 \xint_UDzerominusfork
199 #2-\XINT_frac_gen_Gdivbyzero
200 0#2{\XINT_frac_gen_G -{}}%
201 0-{\XINT_frac_gen_G }#2}%
202 \krof #1~%
203 }%
204 \def\XINT_frac_gen_Gdivbyzero #1~~#2~%
205 {%
206 \expandafter\XINT_frac_gen_Gdivbyzero_a
207 \romannumeral0\expandafter\XINT_num_cleanup\the\numexpr\XINT_num_loop
208 #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:\Z~#1~%

```



```

251 \def\XINT_factortens_f #1#2\xint_c_i#3.#4.#5.%
252   {\expandafter\XINT_factortens_g\the\numexpr#1+#5.#3.}%
253 \def\XINT_factortens_g #1.#2.{.\XINT_factortens_y#2.#1.}%
254 \def\XINT_factortens_e #1..#2.%
255   {\expandafter.\expandafter\XINT_factortens_c
256     \the\numexpr\xint_c_ix+#2.}%

```

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

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

```

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

```

```

295         \else\expandafter\xint_stop_atfirstofthree
296     \fi
297 }%
298 \def\xintifCmp{\romannumeral0\xintifcmp }%
299 \def\xintifcmp #1#2%
300 {%
301     \ifcase\xintCmp {#1}{#2}
302         \expandafter\xint_stop_atsecondofthree
303         \or\expandafter\xint_stop_atthirdofthree
304         \else\expandafter\xint_stop_atfirstofthree
305     \fi
306 }%
307 \def\xintifEq {\romannumeral0\xintifeq }%
308 \def\xintifeq #1#2%
309 {%
310     \if0\xintCmp{#1}{#2}%
311         \expandafter\xint_stop_atfirstoftwo
312         \else\expandafter\xint_stop_atsecondoftwo
313     \fi
314 }%
315 \def\xintifGt {\romannumeral0\xintifgt }%
316 \def\xintifgt #1#2%
317 {%
318     \if1\xintCmp{#1}{#2}%
319         \expandafter\xint_stop_atfirstoftwo
320         \else\expandafter\xint_stop_atsecondoftwo
321     \fi
322 }%
323 \def\xintifLt {\romannumeral0\xintiflt }%
324 \def\xintiflt #1#2%
325 {%
326     \ifnum\xintCmp{#1}{#2}<\xint_c_
327         \expandafter\xint_stop_atfirstoftwo
328     \else \expandafter\xint_stop_atsecondoftwo
329     \fi
330 }%
331 \def\xintifZero {\romannumeral0\xintifzero }%
332 \def\xintifzero #1%
333 {%
334     \if0\xintSgn{#1}%
335         \expandafter\xint_stop_atfirstoftwo
336     \else
337         \expandafter\xint_stop_atsecondoftwo
338     \fi
339 }%
340 \def\xintifNotZero{\romannumeral0\xintifnotzero }%
341 \def\xintifnotzero #1%
342 {%
343     \if0\xintSgn{#1}%
344         \expandafter\xint_stop_atsecondoftwo
345     \else
346         \expandafter\xint_stop_atfirstoftwo

```

```

347   \fi
348 }%
349 \def\xintifOne {\romannumeral0\xintifone }%
350 \def\xintifone #1%
351 {%
352   \if1\xintIsOne{#1}%
353   \expandafter\xint_stop_atfirstoftwo
354   \else
355   \expandafter\xint_stop_atsecondoftwo
356   \fi
357 }%
358 \def\xintifOdd {\romannumeral0\xintifodd }%
359 \def\xintifodd #1%
360 {%
361   \if\xintOdd{#1}1%
362   \expandafter\xint_stop_atfirstoftwo
363   \else
364   \expandafter\xint_stop_atsecondoftwo
365   \fi
366 }%

```

8.10 \xintRaw

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

```

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

```

8.11 \xintiLogTen

New at 1.3e

```

373 \def\xintiLogTen {\the\numexpr\xintilogten}%
374 \def\xintilogten
375 {%
376   \expandafter\XINT_ilogten\romannumeral0\xintraW
377 }%
378 \def\XINT_ilogten #1%
379 {%
380   \xint_UDzerominusfork
381   0#1\XINT_ilogten_p
382   #1-\XINT_ilogten_z
383   0-\{XINT_ilogten_p#1}%
384   \krof
385 }%
386 \def\XINT_ilogten_z #1[#2]{-7FFF8000\relax}%
387 \def\XINT_ilogten_p #1/#2[#3]%
388 {%

```

```

389 #3+\expandafter\XINT_ilogten_a
390 \the\numexpr\xintLength{#1}\expandafter.\the\numexpr\xintLength{#2}.#1.#2.%
391 }%
392 \def\XINT_ilogten_a #1.#2.%
393 {%
394 #1-#2\ifnum#1>#2
395 \expandafter\XINT_ilogten_aa
396 \else
397 \expandafter\XINT_ilogten_ab
398 \fi #1.#2.%
399 }%
400 \def\XINT_ilogten_aa #1.#2.#3.#4.%
401 {%
402 \xintiiiflt{#3}{\XINT_dsx_addzerosnofuss{#1-#2}#4;}{-1}{}\relax
403 }%
404 \def\XINT_ilogten_ab #1.#2.#3.#4.%
405 {%
406 \xintiiiflt{\XINT_dsx_addzerosnofuss{#2-#1}#3;}{#4}{-1}{}\relax
407 }%

```

8.12 \xintPRaw

1.09b

```

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

```

8.13 \xintSPRaw

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

Attention that `\xintSPRaw` assumed that if the number has no `[N]` part it did not have a fraction part `/B` either. Indeed this was the case always with 1.3f: parsing of an integer by `\xintexpr` does not add the `[0]` as the code is shared with `\xintiexpr`, and when there was `/B` `\xintexpr` always

added postfix [0]; even `qfrac()` parses via `\xintRaw`; and `reduce()` internally used `\xintIrr` which outputs A/B but it added [0].

```
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`, `\xintFracToSciE`

1.4

This is the new macro used in place of `\xintSPRaw` (which basically was `\xintPRaw`) by `\xintexpr` typesetter. Attention that it is also used by `\xintiepr` with inputs having being already converted to decimal form, hence must understand this input form. This means it must for example not think 0.123 is 0 because it starts with 0.

And indeed the code here lets 0.123 go through as is. Identification of 0 as first digit is done only in case of A[N], A/B, and A/B[N] formats.

As `reduce()` does not anymore append the [0] at 1.4, `\xintFracToSci` has indeed to recognize A, A[N], A/B and A/B[N] but does not have to parse multiple plus or minus signs or scientific part etc like `\xintRaw` does (delegating to `\XINT_infrac` like all other `xintfrac` macros). It has to identify say 0/5 (although I don't think that can arise) and -0 is never occurring.

The difference with `\xintSPRaw` is that it outputs AeN/B. It will not print the /B if B=1 and eN if N is zero.

If input is empty `\xintFracToSci` output is also empty, whereas `\xintRaw` produces 0/1[0] out of empty. But `\XINTexprprint` anyhow has its own special routine for empty input.

1.4b extends the macro to intercept scientific notation and thus allow customizability of the «e» via `\xintFracToSciE`. Without this `\xintieval` with a negative optional argument uses «e» in output with no possibility to modify it.

The expansion context from `\xinttheexpr`, `\xinttheiexpr`, `\xinteval`, `\xintieval` is the scope of one `\expanded`.

Attention indeed that this macro is not f-expandable only x-expandable.

```
434 \edef\xintFracToSci #1%
435   {\unexpanded{\expandafter\XINT_FracToSci\romannumeral`&&@}#1\string e%
436   \unexpanded{Z/\W[\R]}}%
437 \def\XINT_FracToSci #1/#2#3[#4%
438 {%
439   \xint_gob_til_W #2\XINT_FracToSci_no\W
440   \xint_gob_til_R #4\XINT_FracToSci_yesno\R
441   \XINT_FracToSci_yesyes #1/#2#3[#4%
442 }%
443 \def\XINT_FracToSci_no #1\XINT_FracToSci_yesyes #2[#3%
444 {%
445   \xint_gob_til_R #3\XINT_FracToSci_nono\R
446   \XINT_FracToSci_noyes #2[#3%
447 }%
448 \edef\XINT_tmpa{##1\string e##2}%
449 \def\XINT_tmpb{\def\XINT_FracToSci_nono\R\XINT_FracToSci_noyes}%
450 \expandafter
451 \XINT_tmpb\XINT_tmpa
452 {%
453   #1\xint_gob_til_Z #2\XINT_FracToSci_nono\Z
```

```

454 \XINT_FracToSci_nonoyes #2%
455 }%
456 \edef\XINT_tmpa{##1\string e}%
457 \def\XINT_tmpb{\def\XINT_FracToSci_nonoyes}%
458 \expandafter
459 \XINT_tmpb\XINT_tmpa\Z/\W[\R]{\xintFracToSciE#1}%
460 \def\XINT_FracToSci_nonono\Z\XINT_FracToSci_nonoyes\Z/\W[\R]{}%
461 \edef\XINT_tmpa{##1##2[##3]\string e}%
462 \def\XINT_tmpb{\def\XINT_FracToSci_noyes}%
463 \expandafter
464 \XINT_tmpb\XINT_tmpa\Z/\W[\R]%
465 {%
466 #1\xint_gob_til_zero#1\expandafter\iffalse\xint_gobble_ii0\iftrue
467 #2\ifnum #3=\xint_c_\else\xintFracToSciE#3\fi\fi
468 }%
469 \edef\XINT_tmpa{##1##2/##3\string e}%
470 \def\XINT_tmpb{\def\XINT_FracToSci_yesno\R\XINT_FracToSci_yesyes}%
471 \expandafter
472 \XINT_tmpb\XINT_tmpa\Z/\W[\R]%
473 {%
474 #1\xint_gob_til_zero#1\expandafter\iffalse\xint_gobble_ii0\iftrue
475 #2\if\XINT_isOne{#3}1\else/#3\fi\fi
476 }%
477 \edef\XINT_tmpa{##1##2/##3[##4]\string e}%
478 \def\XINT_tmpb{\def\XINT_FracToSci_yesyes}%
479 \expandafter
480 \XINT_tmpb\XINT_tmpa\Z/\W[\R]%
481 {%
482 #1\xint_gob_til_zero#1\expandafter\iffalse\xint_gobble_ii0\iftrue
483 #2\ifnum #4=\xint_c_\else\xintFracToSciE#4\fi
484 \if\XINT_isOne{#3}1\else/#3\fi\fi
485 }%
486 \def\xintFracToSciE{e}%

```

8.15 \xintRawWithZeros

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

```

487 \def\xintRawWithZeros {\romannumeral0\xinrawwithzeros }%
488 \def\xinrawwithzeros
489 {%
490 \expandafter\XINT_rawz_fork\romannumeral0\XINT_infrac
491 }%

492 \def\XINT_rawz_fork #1%
493 {%
494 \ifnum#1<\xint_c_
495 \expandafter\XINT_rawz_Ba
496 \else
497 \expandafter\XINT_rawz_A
498 \fi
499 #1.%

```



```

500 }%
501 \def\XINT_rawz_A #1.#2#3{\XINT_dsx_addzeros{#1}#2;/#3}%
502 \def\XINT_rawz_Ba -#1.#2#3{\expandafter\XINT_rawz_Bb
503   \expandafter{\romannumeral0\XINT_dsx_addzeros{#1}#3;}{#2}}%
504 \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.

```

505 \def\xintDecToString{\romannumeral0\xintdectostring}%
506 \def\xintdectostring#1{\expandafter\XINT_dectostr\romannumeral0\xintraew{#1}}%
507 \def\XINT_dectostr #1/#2[#3]{\xintiifZero {#1}%
508   \XINT_dectostr_z
509   {\if1\XINT_isOne{#2}\expandafter\XINT_dectostr_a
510     \else\expandafter\XINT_dectostr_b
511     \fi}%
512   #1/#2[#3]}%
513 }%
514 \def\XINT_dectostr_z#1[#2]{ 0}%
515 \def\XINT_dectostr_a#1/#2[#3]{%
516   \ifnum#3<\xint_c_\xint_dothis{\xinttrunc{-#3}{#1[#3]}}\fi
517   \xint_orthat{\xintiie{#1}{#3}}%
518 }%
519 \def\XINT_dectostr_b#1/#2[#3]{% just to handle this somehow
520   \ifnum#3<\xint_c_\xint_dothis{\xinttrunc{-#3}{#1[#3]}/#2}\fi
521   \xint_orthat{\xintiie{#1}{#3}/#2}%
522 }%

```

8.17 \xintFloor, \xintiFloor

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

```

523 \def\xintFloor {\romannumeral0\xintfloor }%
524 \def\xintfloor #1% devrais-je faire \xintREZ?
525   {\expandafter\XINT_ifloor \romannumeral0\xintraewwithzeros {#1}./1[0]}%
526 \def\xintiFloor {\romannumeral0\xintifloor }%
527 \def\xintifloor #1%
528   {\expandafter\XINT_ifloor \romannumeral0\xintraewwithzeros {#1}.}%
529 \def\XINT_ifloor #1/#2.{\xintiiquo {#1}{#2}}%

```

8.18 \xintCeil, \xintiCeil

1.09a

```

530 \def\xintCeil {\romannumeral0\xintceil }%
531 \def\xintceil #1{\xintiioff {\xintFloor {\xintOpp{#1}}}}%
532 \def\xintiCeil {\romannumeral0\xinticeil }%
533 \def\xinticeil #1{\xintiioff {\xintiFloor {\xintOpp{#1}}}}%

```

8.19 `\xintNumerator`

```

534 \def\xintNumerator {\romannumeral0\xintnumerator }%
535 \def\xintnumerator
536 {%
537   \expandafter\XINT_numer\romannumeral0\XINT_infrac
538 }%
539 \def\XINT_numer #1%
540 {%
541   \ifcase\XINT_cntSgn #1\xint:
542     \expandafter\XINT_numer_B
543   \or
544     \expandafter\XINT_numer_A
545   \else
546     \expandafter\XINT_numer_B
547   \fi
548   {#1}%
549 }%
550 \def\XINT_numer_A #1#2#3{\XINT_dsx_addzeros{#1}#2;}%
551 \def\XINT_numer_B #1#2#3{ #2}%

```

8.20 `\xintDenominator`

```

552 \def\xintDenominator {\romannumeral0\xintdenominator }%
553 \def\xintdenominator
554 {%
555   \expandafter\XINT_denom_fork\romannumeral0\XINT_infrac
556 }%
557 \def\XINT_denom_fork #1%
558 {%
559   \ifnum#1<\xint_c_
560     \expandafter\XINT_denom_B
561   \else
562     \expandafter\XINT_denom_A
563   \fi
564   #1.%
565 }%
566 \def\XINT_denom_A #1.#2#3{ #3}%
567 \def\XINT_denom_B -#1.#2#3{\XINT_dsx_addzeros{#1}#3;}%

```

8.21 `\xintFrac`

Useless typesetting macro.

```

568 \def\xintFrac {\romannumeral0\xintfrac }%
569 \def\xintfrac #1%
570 {%
571   \expandafter\XINT_fracfrac_A\romannumeral0\XINT_infrac {#1}%
572 }%
573 \def\XINT_fracfrac_A #1{\XINT_fracfrac_B #1\Z }%
574 \catcode\^=7
575 \def\XINT_fracfrac_B #1#2\Z
576 {%
577   \xint_gob_til_zero #1\XINT_fracfrac_C 0\XINT_fracfrac_D {10^{#1#2}}%

```

```

578 }%
579 \def\XINT_fracfrac_C 0\XINT_fracfrac_D #1#2#3%
580 {%
581   \if1\XINT_isOne {#3}%
582     \xint_afterfi {\expandafter\xint_stop_atfirstoftwo\xint_gobble_ii }%
583   \fi
584   \space
585   \frac {#2}{#3}%
586 }%
587 \def\XINT_fracfrac_D #1#2#3%
588 {%
589   \if1\XINT_isOne {#3}\XINT_fracfrac_E\fi
590   \space
591   \frac {#2}{#3}#1%
592 }%
593 \def\XINT_fracfrac_E \fi\space\frac #1#2{\fi \space #1\cdot }%

```

8.22 \xintSignedFrac

```

594 \def\xintSignedFrac {\romannumeral0\xintsignedfrac }%
595 \def\xintsignedfrac #1%
596 {%
597   \expandafter\XINT_sgnfrac_a\romannumeral0\XINT_infrac {#1}%
598 }%
599 \def\XINT_sgnfrac_a #1#2%
600 {%
601   \XINT_sgnfrac_b #2\Z {#1}%
602 }%
603 \def\XINT_sgnfrac_b #1%
604 {%
605   \xint_UDsignfork
606   #1\XINT_sgnfrac_N
607   -{\XINT_sgnfrac_P #1}%
608   \krof
609 }%
610 \def\XINT_sgnfrac_P #1\Z #2%
611 {%
612   \XINT_fracfrac_A {#2}{#1}%
613 }%
614 \def\XINT_sgnfrac_N
615 {%
616   \expandafter-\romannumeral0\XINT_sgnfrac_P
617 }%

```

8.23 \xintFwOver

```

618 \def\xintFwOver {\romannumeral0\xintfwover }%
619 \def\xintfwover #1%
620 {%
621   \expandafter\XINT_fwover_A\romannumeral0\XINT_infrac {#1}%
622 }%
623 \def\XINT_fwover_A #1{\XINT_fwover_B #1\Z }%
624 \def\XINT_fwover_B #1#2\Z
625 {%

```

```

626 \xint_gob_til_zero #1\XINT_fwover_C 0\XINT_fwover_D {10^{#1#2}}%
627 }%
628 \catcode`^=11
629 \def\XINT_fwover_C #1#2#3#4#5%
630 {%
631 \if0\XINT_isOne {#5}\xint_afterfi { {#4\over #5}}%
632 \else\xint_afterfi { #4}%
633 \fi
634 }%
635 \def\XINT_fwover_D #1#2#3%
636 {%
637 \if0\XINT_isOne {#3}\xint_afterfi { {#2\over #3}}%
638 \else\xint_afterfi { #2\cdot }%
639 \fi
640 #1%
641 }%

```

8.24 \xintSignedFwOver

```

642 \def\xintSignedFwOver {\romannumeral0\xintsignedfwover }%
643 \def\xintsignedfwover #1%
644 {%
645 \expandafter\XINT_sgnfwover_a\romannumeral0\XINT_infrac {#1}%
646 }%
647 \def\XINT_sgnfwover_a #1#2%
648 {%
649 \XINT_sgnfwover_b #2\Z {#1}%
650 }%
651 \def\XINT_sgnfwover_b #1%
652 {%
653 \xint_UDsignfork
654 #1\XINT_sgnfwover_N
655 -{\XINT_sgnfwover_P #1}%
656 \krof
657 }%
658 \def\XINT_sgnfwover_P #1\Z #2%
659 {%
660 \XINT_fwover_A {#2}{#1}%
661 }%
662 \def\XINT_sgnfwover_N
663 {%
664 \expandafter-\romannumeral0\XINT_sgnfwover_P
665 }%

```

8.25 \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.

```

666 \def\xintREZ {\romannumeral0\xintrez }%
667 \def\xintrez
668 {%
669 \expandafter\XINT_rez_A\romannumeral0\XINT_infrac

```

```

670 }%
671 \def\XINT_rez_A #1#2%
672 {%
673   \XINT_rez_AB #2\Z {#1}%
674 }%
675 \def\XINT_rez_AB #1%
676 {%
677   \xint_UDzerominusfork
678   #1-\XINT_rez_zero
679   0#1\XINT_rez_neg
680   0-\XINT_rez_B #1}%
681 \krof
682 }%
683 \def\XINT_rez_zero #1\Z #2#3{ 0/1[0]}%
684 \def\XINT_rez_neg {\expandafter-\romannumeral0\XINT_rez_B }%
685 \def\XINT_rez_B #1\Z
686 {%
687   \expandafter\XINT_rez_C\romannumeral0\XINT_factortens {#1}%
688 }%
689 \def\XINT_rez_C #1.#2.#3#4%
690 {%
691   \expandafter\XINT_rez_D\romannumeral0\XINT_factortens {#4}#3+#2.#1.%
692 }%
693 \def\XINT_rez_D #1.#2.#3.%
694 {%
695   \expandafter\XINT_rez_E\the\numexpr #3-#2.#1.%
696 }%
697 \def\XINT_rez_E #1.#2.#3.{ #3/#2[#1]}%

```

8.26 \xintE

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

```

698 \def\xintE {\romannumeral0\xinte }%
699 \def\xinte #1%
700 {%
701   \expandafter\XINT_e \romannumeral0\XINT_infrac {#1}%
702 }%
703 \def\XINT_e #1#2#3#4%
704 {%
705   \expandafter\XINT_e_end\the\numexpr #1+#4.{#2}{#3}%
706 }%
707 \def\XINT_e_end #1.#2#3{ #2/#3[#1]}%

```

8.27 \xintIrr, \xintPIrr

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

```

708 \def\xintIrr {\romannumeral0\xintirr }%
709 \def\xintPIrr{\romannumeral0\xintpirr }%
710 \def\xintirr #1%
711 {%

```

```

712 \expandafter\XINT_irr_start\romannumeral0\xintraawwithzeros {#1}\Z
713 }%
714 \def\xintpirr #1%
715 {%
716 \expandafter\XINT_pirr_start\romannumeral0\xintraaw{#1}%
717 }%
718 \def\XINT_irr_start #1#2/#3\Z
719 {%
720 \if0\XINT_isOne {#3}%
721 \xint_afterfi
722 {\xint_UDsignfork
723 #1\XINT_irr_negative
724 -{\XINT_irr_nonneg #1}%
725 \krof}%
726 \else
727 \xint_afterfi{\XINT_irr_denomisone #1}%
728 \fi
729 #2\Z {#3}%
730 }%
731 \def\XINT_pirr_start #1#2/#3[%
732 {%
733 \if0\XINT_isOne {#3}%
734 \xint_afterfi
735 {\xint_UDsignfork
736 #1\XINT_irr_negative
737 -{\XINT_irr_nonneg #1}%
738 \krof}%
739 \else
740 \xint_afterfi{\XINT_irr_denomisone #1}%
741 \fi
742 #2\Z {#3}[%
743 }%
744 \def\XINT_irr_denomisone #1\Z #2{ #1/1}% changed in 1.08
745 \def\XINT_irr_negative #1\Z #2{\XINT_irr_D #1\Z #2\Z -}%
746 \def\XINT_irr_nonneg #1\Z #2{\XINT_irr_D #1\Z #2\Z \space}%
747 \def\XINT_irr_D #1#2\Z #3#4\Z
748 {%
749 \xint_UDzerosfork
750 #3#1\XINT_irr_indeterminate
751 #30\XINT_irr_divisionbyzero
752 #10\XINT_irr_zero
753 00\XINT_irr_loop_a
754 \krof
755 {#3#4}{#1#2}{#3#4}{#1#2}%
756 }%
757 \def\XINT_irr_indeterminate #1#2#3#4#5%
758 {%
759 \XINT_signalcondition{DivisionUndefined}{indeterminate: 0/0}{0/1}%
760 }%
761 \def\XINT_irr_divisionbyzero #1#2#3#4#5%
762 {%
763 \XINT_signalcondition{DivisionByZero}{vanishing denominator: #5#2/0}{0/1}%

```

```

764 }%
765 \def\XINT_irr_zero #1#2#3#4#5{ 0/1}% changed in 1.08
766 \def\XINT_irr_loop_a #1#2%
767 {%
768   \expandafter\XINT_irr_loop_d
769   \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
770 }%
771 \def\XINT_irr_loop_d #1#2%
772 {%
773   \XINT_irr_loop_e #2\Z
774 }%
775 \def\XINT_irr_loop_e #1#2\Z
776 {%
777   \xint_gob_til_zero #1\XINT_irr_loop_exit0\XINT_irr_loop_a {#1#2}%
778 }%
779 \def\XINT_irr_loop_exit0\XINT_irr_loop_a #1#2#3#4%
780 {%
781   \expandafter\XINT_irr_loop_exitb\expandafter
782   {\romannumeral0\xintiigo {#3}{#2}}%
783   {\romannumeral0\xintiigo {#4}{#2}}%
784 }%
785 \def\XINT_irr_loop_exitb #1#2%
786 {%
787   \expandafter\XINT_irr_finish\expandafter {#2}{#1}%
788 }%
789 \def\XINT_irr_finish #1#2#3{#3#1/#2}% changed in 1.08

```

8.28 \xintifInt

```

790 \def\xintifInt {\romannumeral0\xintifint }%
791 \def\xintifint #1{\expandafter\XINT_ifint\romannumeral0\xintrawwithzeros {#1}.}%
792 \def\XINT_ifint #1/#2.%
793 {%
794   \if 0\xintiiRem {#1}{#2}%
795   \expandafter\xint_stop_atfirstoftwo
796   \else
797   \expandafter\xint_stop_atsecondoftwo
798   \fi
799 }%

```

8.29 \xintIsInt

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

```

800 \def\xintIsInt {\romannumeral0\xintisint }%
801 \def\xintisint #1%
802   {\expandafter\XINT_ifint\romannumeral0\xintrawwithzeros {#1}.10}%

```

8.30 \xintJrr

```

803 \def\xintJrr {\romannumeral0\xintjrr }%
804 \def\xintjrr #1%
805 {%
806   \expandafter\XINT_jrr_start\romannumeral0\xintrawwithzeros {#1}\Z

```

```

807 }%
808 \def\XINT_jrr_start #1#2/#3\Z
809 {%
810   \if0\XINT_isOne {#3}\xint_afterfi
811     {\xint_UDsignfork
812       #1\XINT_jrr_negative
813       -{\XINT_jrr_nonneg #1}%
814     \krof}%
815   \else
816     \xint_afterfi{\XINT_jrr_denomisone #1}%
817   \fi
818   #2\Z {#3}%
819 }%
820 \def\XINT_jrr_denomisone #1\Z #2{ #1/1}% changed in 1.08
821 \def\XINT_jrr_negative #1\Z #2{\XINT_jrr_D #1\Z #2\Z -}%
822 \def\XINT_jrr_nonneg #1\Z #2{\XINT_jrr_D #1\Z #2\Z \space}%
823 \def\XINT_jrr_D #1#2\Z #3#4\Z
824 {%
825   \xint_UDzerosfork
826     #3#1\XINT_jrr_indeterminate
827     #30\XINT_jrr_divisionbyzero
828     #10\XINT_jrr_zero
829     00\XINT_jrr_loop_a
830   \krof
831   {#3#4}{#1#2}1001%
832 }%
833 \def\XINT_jrr_indeterminate #1#2#3#4#5#6#7%
834 {%
835   \XINT_signalcondition{DivisionUndefined}{indeterminate: 0/0}{0/1}%
836 }%
837 \def\XINT_jrr_divisionbyzero #1#2#3#4#5#6#7%
838 {%
839   \XINT_signalcondition{DivisionByZero}{Vanishing denominator: #7#2/0}{0/1}%
840 }%
841 \def\XINT_jrr_zero #1#2#3#4#5#6#7{ 0/1}% changed in 1.08
842 \def\XINT_jrr_loop_a #1#2%
843 {%
844   \expandafter\XINT_jrr_loop_b
845   \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
846 }%
847 \def\XINT_jrr_loop_b #1#2#3#4#5#6#7%
848 {%
849   \expandafter \XINT_jrr_loop_c \expandafter
850     {\romannumeral0\xintiiadd{\XINT_mul_fork #4\xint:#1\xint:}{#6}}%
851     {\romannumeral0\xintiiadd{\XINT_mul_fork #5\xint:#1\xint:}{#7}}%
852     {#2}{#3}{#4}{#5}%
853 }%
854 \def\XINT_jrr_loop_c #1#2%
855 {%
856   \expandafter \XINT_jrr_loop_d \expandafter{#2}{#1}%
857 }%
858 \def\XINT_jrr_loop_d #1#2#3#4%

```



```

859 {%
860   \XINT_jrr_loop_e #3\Z {#4}{#2}{#1}%
861 }%
862 \def\XINT_jrr_loop_e #1#2\Z
863 {%
864   \xint_gob_til_zero #1\XINT_jrr_loop_exit0\XINT_jrr_loop_a {#1#2}%
865 }%
866 \def\XINT_jrr_loop_exit0\XINT_jrr_loop_a #1#2#3#4#5#6%
867 {%
868   \XINT_irr_finish {#3}{#4}%
869 }%

```

8.31 \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.

```

870 \def\xintTFrac {\romannumeral0\xinttfrac }%
871 \def\xinttfrac #1{\expandafter\XINT_tfrac_fork\romannumeral0\xintraawithzeros {#1}\Z }%
872 \def\XINT_tfrac_fork #1%
873 {%
874   \xint_UDzerominusfork
875     #1-\XINT_tfrac_zero
876     0#1{\xintiopp\XINT_tfrac_P }%
877     0-{\XINT_tfrac_P #1}%
878   \krof
879 }%
880 \def\XINT_tfrac_zero #1\Z { 0/1[0]}%
881 \def\XINT_tfrac_P #1/#2\Z {\expandafter\XINT_rez_AB
882   \romannumeral0\xintiirem{#1}{#2}\Z {0}{#2}}%

```

8.32 \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.

```

883 \def\xintTrunc  {\romannumeral0\xinttrunc }%
884 \def\xintiTrunc {\romannumeral0\xintitrunc}%
885 \def\xinttrunc  #1{\expandafter\XINT_trunc\the\numexpr#1.\XINT_trunc_G}%
886 \def\xintitrunc #1{\expandafter\XINT_trunc\the\numexpr#1.\XINT_itrunc_G}%
887 \def\XINT_trunc #1.#2#3%
888 {%
889   \expandafter\XINT_trunc_a\romannumeral0\XINT_infrac{#3}#1.#2%
890 }%
891 \def\XINT_trunc_a #1#2#3#4.#5%
892 {%
893   \if0\XINT_Sgn#2\xint:\xint_dothis\XINT_trunc_zero\fi
894   \if1\XINT_is_One#3XY\xint_dothis\XINT_trunc_sp_b\fi
895   \xint_orthat\XINT_trunc_b #1+#4.{#2}{#3}#5#4.%
896 }%
897 \def\XINT_trunc_zero #1.#2.{ 0}%
898 \def\XINT_trunc_b     {\expandafter\XINT_trunc_B\the\numexpr}%
899 \def\XINT_trunc_sp_b  {\expandafter\XINT_trunc_sp_B\the\numexpr}%
900 \def\XINT_trunc_B #1%
901 {%
902   \xint_UDsignfork
903   #1\XINT_trunc_C
904   -\XINT_trunc_D
905   \krof #1%
906 }%
907 \def\XINT_trunc_sp_B #1%
908 {%
909   \xint_UDsignfork
910   #1\XINT_trunc_sp_C
911   -\XINT_trunc_sp_D
912   \krof #1%
913 }%
914 \def\XINT_trunc_C -#1.#2#3%
915 {%
916   \expandafter\XINT_trunc_CE
917   \romannumeral0\XINT_dsx_addzeros{#1}#3;.{#2}%
918 }%
919 \def\XINT_trunc_CE #1.#2{\XINT_trunc_E #2.{#1}}%

920 \def\XINT_trunc_sp_C -#1.#2#3{\XINT_trunc_sp_Ca #2.#1.%
921 \def\XINT_trunc_sp_Ca #1%
922 {%

```

```

923 \xint_UDsignfork
924 #1{\XINT_trunc_sp_Cb -}%
925 -{\XINT_trunc_sp_Cb \space#1}%
926 \krof
927 }%
928 \def\XINT_trunc_sp_Cb #1#2.#3.%
929 {%
930 \expandafter\XINT_trunc_sp_Cc
931 \romannumeral0\expandafter\XINT_split_fromright_a
932 \the\numexpr#3-\numexpr\XINT_length_loop
933 #2\xint:\xint:\xint:\xint:\xint:\xint:\xint:\xint:
934 \xint_c_viii\xint_c_vii\xint_c_vi\xint_c_v
935 \xint_c_iv\xint_c_iii\xint_c_ii\xint_c_i\xint_c_\xint_bye
936 .#2\xint_bye2345678\xint_bye..#1%
937 }%
938 \def\XINT_trunc_sp_Cc #1%
939 {%
940 \if.#1\xint_dothis{\XINT_trunc_sp_Cd 0.}\fi
941 \xint_orthat {\XINT_trunc_sp_Cd #1}%
942 }%
943 \def\XINT_trunc_sp_Cd #1.#2.#3%
944 {%
945 \XINT_trunc_sp_F #3#1.%
946 }%
947 \def\XINT_trunc_D #1.#2%
948 {%
949 \expandafter\XINT_trunc_E
950 \romannumeral0\XINT_dsx_addzeros {#1}#2;.%
951 }%
952 \def\XINT_trunc_sp_D #1.#2#3%
953 {%
954 \expandafter\XINT_trunc_sp_E
955 \romannumeral0\XINT_dsx_addzeros {#1}#2;.%
956 }%
957 \def\XINT_trunc_E #1%
958 {%
959 \xint_UDsignfork
960 #1{\XINT_trunc_F -}%
961 -{\XINT_trunc_F \space#1}%
962 \krof
963 }%
964 \def\XINT_trunc_sp_E #1%
965 {%
966 \xint_UDsignfork
967 #1{\XINT_trunc_sp_F -}%
968 -{\XINT_trunc_sp_F\space#1}%
969 \krof
970 }%
971 \def\XINT_trunc_F #1#2.#3#4%
972 {\expandafter#4\romannumeral`&&\expandafter\xint_firstoftwo
973 \romannumeral0\XINT_div_prepare {#3}{#2}.#1}%
974 \def\XINT_trunc_sp_F #1#2.#3{#3#2.#1}%

```

```

975 \def\XINT_itrunc_G #1#2.#3#4.%
976 {%
977   \if#10\xint_dothis{ 0}\fi
978   \xint_orthat{#3#1}#2%
979 }%
980 \def\XINT_trunc_G #1.#2#3#4.%
981 {%
982   \xint_gob_til_minus#3\XINT_trunc_Hc-%
983   \expandafter\XINT_trunc_H
984   \the\numexpr\romannumeral0\xintlength {#1}-#3#4.#3#4.{#1}#2%
985 }%
986 \def\XINT_trunc_Hc-\expandafter\XINT_trunc_H
987   \the\numexpr\romannumeral0\xintlength #1.-#2.#3#4{#4#3e#2}%
988 \def\XINT_trunc_H #1.#2.%
989 {%
990   \ifnum #1 > \xint_c_ \xint_dothis{\XINT_trunc_Ha {#2}}\fi
991   \xint_orthat {\XINT_trunc_Hb {-#1}}% -0,--1,--2, ....
992 }%
993 \def\XINT_trunc_Ha%
994 {%
995   \expandafter\XINT_trunc_Haa\romannumeral0\xintdecsplit
996 }%
997 \def\XINT_trunc_Haa #1#2#3{#3#1.#2}%
998 \def\XINT_trunc_Hb #1#2#3%
999 {%
1000   \expandafter #3\expandafter0\expandafter.%
1001   \romannumeral\xintreplicate{#1}0#2%
1002 }%

```

8.33 \xintTTrunc

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

```

1003 \def\xintTTrunc {\romannumeral0\xintttrunc }%
1004 \def\xintttrunc {\xintitrunc\xint_c_}%

```

8.34 \xintNum

```

1005 \let\xintnum \xintttrunc

```

8.35 \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.

```

1006 \def\xintRound {\romannumeral0\xintround }%
1007 \def\xintiRound {\romannumeral0\xintiround }%
1008 \def\xintround #1{\expandafter\XINT_round\the\numexpr #1.\XINT_round_A}%
1009 \def\xintiround #1{\expandafter\XINT_round\the\numexpr #1.\XINT_iround_A}%

```

```

1010 \def\XINT_round #1.{\expandafter\XINT_round_aa\the\numexpr #1+\xint_c_i.#1.}%
1011 \def\XINT_round_aa #1.#2.#3#4%
1012 {%
1013   \expandafter\XINT_round_a\romannumeral0\XINT_infrac{#4}#1.#3#2.%
1014 }%
1015 \def\XINT_round_a #1#2#3#4.%
1016 {%
1017   \if0\XINT_Sgn#2\xint:\xint_dothis\XINT_trunc_zero\fi
1018   \if1\XINT_is_One#3XY\xint_dothis\XINT_trunc_sp_b\fi
1019   \xint_orthat\XINT_trunc_b #1+#4.{#2}{#3}%
1020 }%
1021 \def\XINT_round_A{\expandafter\XINT_trunc_G\romannumeral0\XINT_round_B}%
1022 \def\XINT_iround_A{\expandafter\XINT_itrunc_G\romannumeral0\XINT_round_B}%
1023 \def\XINT_round_B #1.%
1024   {\XINT_dsrr #1\xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax.}%

```

8.36 \xintXTrunc

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

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

```

1025 \def\xintXTrunc #1#2%
1026 {%
1027   \expandafter\XINT_xtrunc_a
1028   \the\numexpr #1\expandafter.\romannumeral0\xintraw
1029 }%
1030 \def\XINT_xtrunc_a #1.% ?? faire autre chose
1031 {%
1032   \expandafter\XINT_xtrunc_b\the\numexpr\ifnum#1<\xint_c_i \xint_c_i-\fi #1.%
1033 }%

1034 \def\XINT_xtrunc_b #1.#2{\XINT_xtrunc_c #2{#1}}%

1035 \def\XINT_xtrunc_c #1%
1036 {%
1037   \xint_UDzerominusfork
1038     #1-\XINT_xtrunc_zero
1039     0#1{-\XINT_xtrunc_d {}}%
1040     0-{\XINT_xtrunc_d #1}%
1041   \krof
1042 }%[
1043 \def\XINT_xtrunc_zero #1#2]{0.\romannumeral\xintreplicate{#1}0}%

1044 \def\XINT_xtrunc_d #1#2#3/#4[#5]%
1045 {%
1046   \XINT_xtrunc_prepare_a#4\R\R\R\R\R\R\R {10}0000001\W

```

```

1047     !{#4};{#5}{#2}{#1#3}%
1048 }%
1049 \def\XINT_xtrunc_prepare_a #1#2#3#4#5#6#7#8#9%
1050 {%
1051     \xint_gob_til_R #9\XINT_xtrunc_prepare_small\R
1052     \XINT_xtrunc_prepare_b #9%
1053 }%
1054 \def\XINT_xtrunc_prepare_small\R #1!#2;%
1055 {%
1056     \ifcase #2
1057     \or\expandafter\XINT_xtrunc_BisOne
1058     \or\expandafter\XINT_xtrunc_BisTwo
1059     \or
1060     \or\expandafter\XINT_xtrunc_BisFour
1061     \or\expandafter\XINT_xtrunc_BisFive
1062     \or
1063     \or
1064     \or\expandafter\XINT_xtrunc_BisEight
1065     \fi\XINT_xtrunc_BisSmall {#2}%
1066 }%

1067 \def\XINT_xtrunc_BisOne\XINT_xtrunc_BisSmall #1#2#3#4%
1068     {\XINT_xtrunc_sp_e {#2}{#4}{#3}}%
1069 \def\XINT_xtrunc_BisTwo\XINT_xtrunc_BisSmall #1#2#3#4%
1070 {%
1071     \expandafter\XINT_xtrunc_sp_e\expandafter
1072     {\the\numexpr #2-\xint_c_i\expandafter}\expandafter
1073     {\romannumeral0\xintiimul 5{#4}}{#3}%
1074 }%
1075 \def\XINT_xtrunc_BisFour\XINT_xtrunc_BisSmall #1#2#3#4%
1076 {%
1077     \expandafter\XINT_xtrunc_sp_e\expandafter
1078     {\the\numexpr #2-\xint_c_ii\expandafter}\expandafter
1079     {\romannumeral0\xintiimul {25}{#4}}{#3}%
1080 }%
1081 \def\XINT_xtrunc_BisFive\XINT_xtrunc_BisSmall #1#2#3#4%
1082 {%
1083     \expandafter\XINT_xtrunc_sp_e\expandafter
1084     {\the\numexpr #2-\xint_c_i\expandafter}\expandafter
1085     {\romannumeral0\xintdouble {#4}}{#3}%
1086 }%
1087 \def\XINT_xtrunc_BisEight\XINT_xtrunc_BisSmall #1#2#3#4%
1088 {%
1089     \expandafter\XINT_xtrunc_sp_e\expandafter
1090     {\the\numexpr #2-\xint_c_iii\expandafter}\expandafter
1091     {\romannumeral0\xintiimul {125}{#4}}{#3}%
1092 }%
1093 \def\XINT_xtrunc_BisSmall #1%
1094 {%
1095     \expandafter\XINT_xtrunc_e\expandafter
1096     {\expandafter\XINT_xtrunc_small_a
1097     \the\numexpr #1/\xint_c_ii\expandafter

```

```

1098 .\the\numexpr \xint_c_x^viii+#1!}%
1099 }%

1100 \def\xINT_xtrunc_small_a #1.#2!#3%
1101 {%
1102   \expandafter\xINT_div_small_b\the\numexpr #1\expandafter
1103   \xint:\the\numexpr #2\expandafter!%
1104   \romannumeral0\xINT_div_small_ba #3\R\R\R\R\R\R\R\R{10}0000001\W
1105   #3\xINT_sepbyviii_Z_end 2345678\relax
1106 }%

1107 \def\xINT_xtrunc_prepare_b
1108   {\expandafter\xINT_xtrunc_prepare_c\romannumeral0\xINT_zeroes_forviii }%
1109 \def\xINT_xtrunc_prepare_c #1!%
1110 {%
1111   \xINT_xtrunc_prepare_d #1.00000000!{#1}%
1112 }%
1113 \def\xINT_xtrunc_prepare_d #1#2#3#4#5#6#7#8#9%
1114 {%
1115   \expandafter\xINT_xtrunc_prepare_e
1116   \xint_gob_til_dot #1#2#3#4#5#6#7#8#9!%
1117 }%
1118 \def\xINT_xtrunc_prepare_e #1!#2!#3#4%
1119 {%
1120   \xINT_xtrunc_prepare_f #4#3\X {#1}{#3}%
1121 }%
1122 \def\xINT_xtrunc_prepare_f #1#2#3#4#5#6#7#8#9\X
1123 {%
1124   \expandafter\xINT_xtrunc_prepare_g\expandafter
1125   \xINT_div_prepare_g
1126   \the\numexpr #1#2#3#4#5#6#7#8+\xint_c_i\expandafter
1127   \xint:\the\numexpr (#1#2#3#4#5#6#7#8+\xint_c_i)/\xint_c_ii\expandafter
1128   \xint:\the\numexpr #1#2#3#4#5#6#7#8\expandafter
1129   \xint:\romannumeral0\xINT_sepandrev_andcount
1130   #1#2#3#4#5#6#7#8#9\xINT_rsepbyviii_end_A 2345678%
1131   \xINT_rsepbyviii_end_B 2345678\relax\xint_c_ii\xint_c_i
1132   \R\xint:\xint_c_xii \R\xint:\xint_c_x \R\xint:\xint_c_viii \R\xint:\xint_c_vi
1133   \R\xint:\xint_c_iv \R\xint:\xint_c_ii \R\xint:\xint_c_\W
1134   \X
1135 }%

1136 \def\xINT_xtrunc_prepare_g #1;{\xINT_xtrunc_e {#1}}%

1137 \def\xINT_xtrunc_e #1#2%
1138 {%
1139   \ifnum #2<\xint_c_
1140     \expandafter\xINT_xtrunc_I
1141   \else
1142     \expandafter\xINT_xtrunc_II
1143   \fi #2\xint:{#1}%
1144 }%

```

```

1145 \def\XINT_xtrunc_I -#1\xint:#2#3#4%
1146 {%
1147   \expandafter\XINT_xtrunc_I_a\romannumeral0#2{#4}{#2}{#1}{#3}%
1148 }%

1149 \def\XINT_xtrunc_I_a #1#2#3#4#5%
1150 {%
1151   \expandafter\XINT_xtrunc_I_b\the\numexpr #4-#5\xint:#4\xint:{#5}{#2}{#3}{#1}%
1152 }%

1153 \def\XINT_xtrunc_I_b #1%
1154 {%
1155   \xint_UDsignfork
1156     #1\XINT_xtrunc_IA_c
1157     -\XINT_xtrunc_IB_c
1158   \krof #1%
1159 }%

1160 \def\XINT_xtrunc_IA_c -#1\xint:#2\xint:#3#4#5#6%
1161 {%
1162   \expandafter\XINT_xtrunc_IA_d
1163   \the\numexpr#2-\xintLength{#6}\xint:{#6}%
1164   \expandafter\XINT_xtrunc_IA_xd
1165   \the\numexpr (#1+\xint_c_ii^v)/\xint_c_ii^vi-\xint_c_i\xint:#1\xint:{#5}{#4}%
1166 }%

1167 \def\XINT_xtrunc_IA_d #1%
1168 {%
1169   \xint_UDsignfork
1170     #1\XINT_xtrunc_IAA_e
1171     -\XINT_xtrunc_IAB_e
1172   \krof #1%
1173 }%

1174 \def\XINT_xtrunc_IAA_e -#1\xint:#2%
1175 {%
1176   \romannumeral0\XINT_split_fromleft
1177   #1.#2\xint_gobble_i\xint_bye2345678\xint_bye. .%
1178 }%

1179 \def\XINT_xtrunc_IAB_e #1\xint:#2%
1180 {%
1181   0.\romannumeral\XINT_rep#1\endcsname0#2%
1182 }%

1183 \def\XINT_xtrunc_IA_xd #1\xint:#2\xint:%
1184 {%
1185   \expandafter\XINT_xtrunc_IA_xe\the\numexpr #2-\xint_c_ii^vi*#1\xint:#1\xint:%
1186 }%

```



```

1234 \def\XINT_xtrunc_transition
1235   \expandafter\XINT_xtrunc_loop_a\the\numexpr #1\xint:#2#3#4%
1236 {%
1237   \ifnum #4=\xint_c_ \expandafter\xint_gobble_vi\fi
1238   \expandafter\XINT_xtrunc_finish\expandafter
1239   {\romannumeral0\XINT_dsx_addzeros{#4}#2;}{#3}{#4}%
1240 }%
1241 \def\XINT_xtrunc_finish #1#2%
1242 {%
1243   \expandafter\XINT_xtrunc_finish_a\romannumeral0#2{#1}%
1244 }%
1245 \def\XINT_xtrunc_finish_a #1#2#3%
1246 {%
1247   \romannumeral\xintreplicate{#3-\xintLength{#1}}0#1%
1248 }%

1249 \def\XINT_xtrunc_sp_e #1%
1250 {%
1251   \ifnum #1<\xint_c_
1252     \expandafter\XINT_xtrunc_sp_I
1253   \else
1254     \expandafter\XINT_xtrunc_sp_II
1255   \fi #1\xint:%
1256 }%

1257 \def\XINT_xtrunc_sp_I -#1\xint:#2#3%
1258 {%
1259   \expandafter\XINT_xtrunc_sp_I_a\the\numexpr #1-#3\xint:#1\xint:{#3}{#2}%
1260 }%

1261 \def\XINT_xtrunc_sp_I_a #1%
1262 {%
1263   \xint_UDsignfork
1264   #1\XINT_xtrunc_sp_IA_b
1265   -\XINT_xtrunc_sp_IB_b
1266   \krof #1%
1267 }%

1268 \def\XINT_xtrunc_sp_IA_b -#1\xint:#2\xint:#3#4%
1269 {%
1270   \expandafter\XINT_xtrunc_sp_IA_c
1271   \the\numexpr#2-\xintLength{#4}\xint:{#4}\romannumeral\XINT_rep#1\endcsname0%
1272 }%

1273 \def\XINT_xtrunc_sp_IA_c #1%
1274 {%
1275   \xint_UDsignfork
1276   #1\XINT_xtrunc_sp_IAA
1277   -\XINT_xtrunc_sp_IAB
1278   \krof #1%
1279 }%

```

```

1280 \def\XINT_xtrunc_sp_IAA -#1\xint:#2%
1281 {%
1282     \romannumeral0\XINT_split_fromleft
1283     #1.#2\xint_gobble_i\xint_bye2345678\xint_bye.>%
1284 }%

1285 \def\XINT_xtrunc_sp_IAB #1\xint:#2%
1286 {%
1287     0.\romannumeral\XINT_rep#1\endcsname0#2%
1288 }%

1289 \def\XINT_xtrunc_sp_IB_b #1\xint:#2\xint:#3#4%
1290 {%
1291     \expandafter\XINT_xtrunc_sp_IB_c
1292     \romannumeral0\XINT_split_xfork #1.#4\xint_bye2345678\xint_bye..{#3}%
1293 }%

1294 \def\XINT_xtrunc_sp_IB_c #1.#2.#3%
1295 {%
1296     \expandafter\XINT_xtrunc_sp_IA_c\the\numexpr#3-\xintLength {#1}\xint:{#1}%
1297 }%

1298 \def\XINT_xtrunc_sp_II #1\xint:#2#3%
1299 {%
1300     #2\romannumeral\XINT_rep#1\endcsname0.\romannumeral\XINT_rep#3\endcsname0%
1301 }%

```

8.37 \xintAdd

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

```

1302 \def\xintAdd {\romannumeral0\xintadd }%
1303 \def\xintadd #1{\expandafter\XINT_fadd\romannumeral0\xintraw {#1}}%
1304 \def\XINT_fadd #1{\xint_gob_til_zero #1\XINT_fadd_Azero 0\XINT_fadd_a #1}%
1305 \def\XINT_fadd_Azero #1{\xintraw }%
1306 \def\XINT_fadd_a #1/#2[#3]#4%
1307     {\expandafter\XINT_fadd_b\romannumeral0\xintraw {#4}{#3}{#1}{#2}}%
1308 \def\XINT_fadd_b #1{\xint_gob_til_zero #1\XINT_fadd_Bzero 0\XINT_fadd_c #1}%
1309 \def\XINT_fadd_Bzero #1]#2#3#4{ #3/#4[#2]}%
1310 \def\XINT_fadd_c #1/#2[#3]#4%
1311 {%
1312     \expandafter\XINT_fadd_Aa\the\numexpr #4-#3.{#3}{#4}{#1}{#2}%
1313 }%
1314 \def\XINT_fadd_Aa #1%
1315 {%
1316     \xint_UDzerominusfork
1317     #1-\XINT_fadd_B
1318     0#1\XINT_fadd_Bb
1319     0-\XINT_fadd_Ba
1320     \krof #1%
1321 }%

```

```

1322 \def\XINT_fadd_B #1.#2#3#4#5#6#7{\XINT_fadd_C {#4}{#5}{#7}{#6}[#3]}%
1323 \def\XINT_fadd_Ba #1.#2#3#4#5#6#7%
1324 {%
1325     \expandafter\XINT_fadd_C\expandafter
1326         {\romannumeral0\XINT_dsx_addzeros {#1}#6;}%
1327     {#7}{#5}{#4}[#2]}%
1328 }%
1329 \def\XINT_fadd_Bb -#1.#2#3#4#5#6#7%
1330 {%
1331     \expandafter\XINT_fadd_C\expandafter
1332         {\romannumeral0\XINT_dsx_addzeros {#1}#4;}%
1333     {#5}{#7}{#6}[#3]}%
1334 }%
1335 \def\XINT_fadd_iszero #1[#2]{ 0/1[0]}% ou [#2] origine1?
1336 \def\XINT_fadd_C #1#2#3%
1337 {%
1338     \expandafter\XINT_fadd_D_b
1339     \romannumeral0\XINT_div_prepare{#2}{#3}{#2}{#2}{#3}{#1}%
1340 }%

```

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.

```

1341 \def\XINT_fadd_D_a #1#2%
1342 {%
1343     \expandafter\XINT_fadd_D_b
1344     \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
1345 }%
1346 \def\XINT_fadd_D_b #1#2{\XINT_fadd_D_c #2\Z}%
1347 \def\XINT_fadd_D_c #1#2\Z
1348 {%
1349     \xint_gob_til_zero #1\XINT_fadd_D_exit0\XINT_fadd_D_a {#1#2}%
1350 }%
1351 \def\XINT_fadd_D_exit0\XINT_fadd_D_a #1#2#3%
1352 {%
1353     \expandafter\XINT_fadd_E
1354     \romannumeral0\xintiiquo {#3}{#2}.{#2}%
1355 }%
1356 \def\XINT_fadd_E #1.#2#3%
1357 {%
1358     \expandafter\XINT_fadd_F
1359     \romannumeral0\xintiimul{#1}{#3}.{\xintiiQuo{#3}{#2}}{#1}%
1360 }%
1361 \def\XINT_fadd_F #1.#2#3#4#5%
1362 {%
1363     \expandafter\XINT_fadd_G
1364     \romannumeral0\xintiiaadd{\xintiiMul{#2}{#4}}{\xintiiMul{#3}{#5}}/#1%
1365 }%
1366 \def\XINT_fadd_G #1{%
1367 \def\XINT_fadd_G ##1{\if0##1\expandafter\XINT_fadd_iszero\fi#1##1}%
1368 }\XINT_fadd_G{ }%

```

8.38 \xintSub

Since 1.3 will use least common multiple of denominators.

```

1369 \def\xintSub {\romannumeral0\xintsub }%
1370 \def\xintsub #1{\expandafter\XINT_fsub\romannumeral0\xintra {#1}}%
1371 \def\XINT_fsub #1{\xint_gob_til_zero #1\XINT_fsub_Azero 0\XINT_fsub_a #1}%
1372 \def\XINT_fsub_Azero #1{\xintopp }%
1373 \def\XINT_fsub_a #1/#2[#3]#4%
1374 {\expandafter\XINT_fsub_b\romannumeral0\xintra {#4}{#3}{#1}{#2}}%
1375 \def\XINT_fsub_b #1{\xint_UDzerominusfork
1376 #1-\XINT_fadd_Bzero
1377 0#1\XINT_fadd_c
1378 0-{\XINT_fadd_c -#1}%
1379 \krof }%

```

8.39 \xintSum

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

Empty items in the input are not accepted by this macro, but the input may be empty.

Refactored slightly at 1.4. \XINT_Sum used in xintexpr code.

```

1380 \def\xintSum {\romannumeral0\xintsum }%
1381 \def\xintsum #1{\expandafter\XINT_sum\romannumeral`&&@#1^}%
1382 \def\XINT_Sum{\romannumeral0\XINT_sum}%
1383 \def\XINT_sum#1%
1384 {%
1385 \xint_gob_til_^ #1\XINT_sum_empty ^%
1386 \expandafter\XINT_sum_loop\romannumeral0\xintra{#1}\xint:
1387 }%
1388 \def\XINT_sum_empty ^#1\xint:{ 0/1[0]}%
1389 \def\XINT_sum_loop #1\xint:#2%
1390 {%
1391 \xint_gob_til_^ #2\XINT_sum_end ^%
1392 \expandafter\XINT_sum_loop
1393 \romannumeral0\xintadd{#1}{\romannumeral0\xintra{#2}}\xint:
1394 }%
1395 \def\XINT_sum_end ^#1\xintadd #2#3\xint:{ #2}%

```

8.40 \xintMul

```

1396 \def\xintMul {\romannumeral0\xintmul }%
1397 \def\xintmul #1{\expandafter\XINT_fmula\romannumeral0\xintra {#1}.}%
1398 \def\XINT_fmula #1{\xint_gob_til_zero #1\XINT_fmula_zero 0\XINT_fmula_a #1}%
1399 \def\XINT_fmula_a #1[#2].#3%
1400 {\expandafter\XINT_fmula_b\romannumeral0\xintra {#3}#1[#2.]}%
1401 \def\XINT_fmula_b #1{\xint_gob_til_zero #1\XINT_fmula_zero 0\XINT_fmula_c #1}%
1402 \def\XINT_fmula_c #1/#2[#3]#4/#5[#6.]%
1403 {%
1404 \expandafter\XINT_fmula_d
1405 \expandafter{\the\numexpr #3+#6\expandafter}%
1406 \expandafter{\romannumeral0\xintiimul {#5}{#2}}%

```

```

1407     {\romannumeral0\xintiimul {#4}{#1}}%
1408 }%
1409 \def\xINT_fmud #1#2#3%
1410 {%
1411     \expandafter \XINT_fmule \expandafter{#3}{#1}{#2}%
1412 }%
1413 \def\xINT_fmule #1#2{\XINT_outfrac {#2}{#1}}%
1414 \def\xINT_fmulo #1.#2{ 0/1[0]}%

```

8.41 \xintSqr

1.1 modifs comme xintMul.

```

1415 \def\xintSqr {\romannumeral0\xintsqr }%
1416 \def\xintsqr #1{\expandafter\xINT_fsqr\romannumeral0\xintraw {#1}}%
1417 \def\xINT_fsqr #1{\xint_gob_til_zero #1\xINT_fsqr_zero 0\xINT_fsqr_a #1}%
1418 \def\xINT_fsqr_a #1/#2[#3]%
1419 {%
1420     \expandafter\xINT_fsqr_b
1421     \expandafter{\the\numexpr #3+#3\expandafter}%
1422     \expandafter{\romannumeral0\xintiisqr {#2}}%
1423     {\romannumeral0\xintiisqr {#1}}%
1424 }%
1425 \def\xINT_fsqr_b #1#2#3{\expandafter \XINT_fmule \expandafter{#3}{#1}{#2}}%
1426 \def\xINT_fsqr_zero #1{ 0/1[0]}%

```

8.42 \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.

```

1427 \def\xintipow #1#2%
1428 {%
1429     \expandafter\xint_pow\the\numexpr \xintNum{#2}\expandafter
1430     .\romannumeral0\xintnum{#1}\xint:
1431 }%
1432 \def\xintPow {\romannumeral0\xintpow }%
1433 \def\xintpow #1%
1434 {%
1435     \expandafter\xINT_fpow\expandafter {\romannumeral0\xINT_infrac {#1}}%
1436 }%
1437 \def\xINT_fpow #1#2%
1438 {%
1439     \expandafter\xINT_fpow_fork\the\numexpr \xintNum{#2}\relax\Z #1%
1440 }%
1441 \def\xINT_fpow_fork #1#2\Z
1442 {%
1443     \xint_UDzerominusfork
1444     #1-\XINT_fpow_zero
1445     0#1\xINT_fpow_neg
1446     0-{\XINT_fpow_pos #1}%

```

```

1447 \krof
1448 {#2}%
1449 }%
1450 \def\XINT_fpow_zero #1#2#3#4{ 1/1[0]}%
1451 \def\XINT_fpow_pos #1#2#3#4#5%
1452 {%
1453 \expandafter\XINT_fpow_pos_A\expandafter
1454 {\the\numexpr #1#2*#3\expandafter}\expandafter
1455 {\romannumeral0\xintiipow {#5}{#1#2}}%
1456 {\romannumeral0\xintiipow {#4}{#1#2}}%
1457 }%
1458 \def\XINT_fpow_neg #1#2#3#4%
1459 {%
1460 \expandafter\XINT_fpow_pos_A\expandafter
1461 {\the\numexpr -#1*#2\expandafter}\expandafter
1462 {\romannumeral0\xintiipow {#3}{#1}}%
1463 {\romannumeral0\xintiipow {#4}{#1}}%
1464 }%
1465 \def\XINT_fpow_pos_A #1#2#3%
1466 {%
1467 \expandafter\XINT_fpow_pos_B\expandafter {#3}{#1}{#2}%
1468 }%
1469 \def\XINT_fpow_pos_B #1#2{\XINT_outfrac {#2}{#1}}%

```

8.43 \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.

```

1470 \def\xintFac {\romannumeral0\xintfac}%
1471 \def\xintfac #1{\expandafter\XINT_fac_fork\the\numexpr\xintNum{#1}.[0]}%

```

8.44 \xintBinomial

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

```

1472 \def\xintBinomial {\romannumeral0\xintbinomial}%
1473 \def\xintbinomial #1#2%
1474 {%
1475 \expandafter\XINT_binom_pre
1476 \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.[0]}%
1477 }%

```

8.45 \xintPFactorial

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

```

1478 \def\xintipfactorial #1#2%
1479 {%
1480 \expandafter\XINT_pfac_fork
1481 \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.%
1482 }%

```

```

1483 \def\xintPFactorial {\romannumeral0\xintpfactorial}%
1484 \def\xintpfactorial #1#2%
1485 {%
1486   \expandafter\XINT_pfac_fork
1487   \the\numexpr\xintNum{#1}\expandafter.\the\numexpr\xintNum{#2}.[0]%
1488 }%

```

8.46 \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).

```

1489 \def\xintPrd {\romannumeral0\xintprd }%
1490 \def\xintprd #1{\expandafter\XINT_prd\romannumeral`&&@#1^}%
1491 \def\XINT_Prd{\romannumeral0\XINT_prd}%
1492 \def\XINT_prd#1%
1493 {%
1494   \xint_gob_til_^ #1\XINT_prd_empty ^%
1495   \expandafter\XINT_prd_loop\romannumeral0\xintra{#1}\xint:
1496 }%
1497 \def\XINT_prd_empty ^#1\xint:{ 1/1[0]}%
1498 \def\XINT_prd_loop #1\xint:#2%
1499 {%
1500   \xint_gob_til_^ #2\XINT_prd_end ^%
1501   \expandafter\XINT_prd_loop
1502   \romannumeral0\xintmul{#1}{\romannumeral0\xintra{#2}}\xint:
1503 }%
1504 \def\XINT_prd_end ^#1\xintmul #2#3\xint:{ #2}%

```

8.47 \xintDiv

```

1505 \def\xintDiv {\romannumeral0\xintdiv }%
1506 \def\xintdiv #1%
1507 {%
1508   \expandafter\XINT_fdiv\expandafter {\romannumeral0\XINT_infrac {#1}}%
1509 }%
1510 \def\XINT_fdiv #1#2%
1511   {\expandafter\XINT_fdiv_A\romannumeral0\XINT_infrac {#2}#1}%
1512 \def\XINT_fdiv_A #1#2#3#4#5#6%
1513 {%
1514   \expandafter\XINT_fdiv_B
1515   \expandafter{\the\numexpr #4-#1\expandafter}%
1516   \expandafter{\romannumeral0\xintiimul {#2}{#6}}%
1517   {\romannumeral0\xintiimul {#3}{#5}}%
1518 }%
1519 \def\XINT_fdiv_B #1#2#3%
1520 {%
1521   \expandafter\XINT_fdiv_C
1522   \expandafter{#3}{#1}{#2}%
1523 }%
1524 \def\XINT_fdiv_C #1#2{\XINT_outfrac {#2}{#1}}%

```


8.48 `\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`.

```
1525 \def\xintDivFloor      {\romannumeral0\xintdivfloor }%
1526 \def\xintdivfloor #1#2{\xintifloor{\xintDiv {#1}{#2}}}%
```

8.49 `\xintDivTrunc`

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

```
1527 \def\xintDivTrunc      {\romannumeral0\xintdivtrunc }%
1528 \def\xintdivtrunc #1#2{\xintttrunc {\xintDiv {#1}{#2}}}%
```

8.50 `\xintDivRound`

1.1

```
1529 \def\xintDivRound      {\romannumeral0\xintdivround }%
1530 \def\xintdivround #1#2{\xintiround 0{\xintDiv {#1}{#2}}}%
```

8.51 `\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)`.

```
1531 \def\xintModTrunc {\romannumeral0\xintmodtrunc }%
1532 \def\xintmodtrunc #1{\expandafter\XINT_modtrunc_a\romannumeral0\xintra{#1}.}%
1533 \def\XINT_modtrunc_a #1#2.#3%
1534   {\expandafter\XINT_modtrunc_b\expandafter #1\romannumeral0\xintra{#3}#2.}%
1535 \def\XINT_modtrunc_b #1#2% #1 de A, #2 de B.
1536 {%
1537   \if0#2\xint_dothis{\XINT_modtrunc_divbyzero #1#2}\fi
1538   \if0#1\xint_dothis\XINT_modtrunc_aiszero\fi
1539   \if-#2\xint_dothis{\XINT_modtrunc_bneg #1}\fi
1540     \xint_orthat{\XINT_modtrunc_bpos #1#2}%
1541 }%
1542 \def\XINT_modtrunc_divbyzero #1#2[#3]#4.%
1543 {%
1544   \XINT_signalcondition{DivisionByZero}{Division by #2[#3] of #1#4}{0/1[0]}%
1545 }%
1546 \def\XINT_modtrunc_aiszero #1.{ 0/1[0]}%
1547 \def\XINT_modtrunc_bneg #1%
1548 {%
1549   \xint_UDsignfork
1550     #1{\xintiiopp\XINT_modtrunc_pos {}}%
1551     -{\XINT_modtrunc_pos #1}%
1552   \krof
1553 }%
1554 \def\XINT_modtrunc_bpos #1%
```

```

1555 {%
1556   \xint_UDsignfork
1557     #1{\xintiiopp\XINT_modtrunc_pos {}}%
1558     -{\XINT_modtrunc_pos #1}%
1559   \krof
1560 }%

```

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).

```

1561 \def\XINT_modtrunc_pos #1#2/#3[#4]#5/#6[#7].%
1562 {%
1563   \expandafter\XINT_modtrunc_pos_a
1564   \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.%
1565   \romannumeral0\expandafter\XINT_mod_D_b
1566   \romannumeral0\XINT_div_prepare{#3}{#6}{#3}{#3}{#6}%
1567   {#1#5}{#7-#4}{#2}{#4-#7}%
1568 }%
1569 \def\XINT_modtrunc_pos_a #1.#2#3#4{\xintiirem {#3}{#4}/#2[#1]}%

```

8.52 \xintDivMod

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

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

```

1570 \def\xintDivMod {\romannumeral0\xintdivmod }%
1571 \def\xintdivmod #1{\expandafter\XINT_divmod_a\romannumeral0\xintra{#1}.}%
1572 \def\XINT_divmod_a #1#2.#3%
1573   {\expandafter\XINT_divmod_b\expandafter #1\romannumeral0\xintra{#3}#2.}%
1574 \def\XINT_divmod_b #1#2% #1 de A, #2 de B.
1575 {%
1576   \if0#2\xint_dothis{\XINT_divmod_divbyzero #1#2}\fi
1577   \if0#1\xint_dothis\XINT_divmod_aiszero\fi
1578   \if-#2\xint_dothis{\XINT_divmod_bneg #1}\fi
1579   \xint_orthat{\XINT_divmod_bpos #1#2}%
1580 }%
1581 \def\XINT_divmod_divbyzero #1#2[#3]#4.%
1582 {%
1583   \XINT_signalcondition{DivisionByZero}{Division by #2[#3] of #1#4}{}%
1584   {{0}{0/1[0]}}% à revoir...
1585 }%
1586 \def\XINT_divmod_aiszero #1.{{0}{0/1[0]}}%
1587 \def\XINT_divmod_bneg #1% f // -g = (-f) // g, f % -g = -((-f) % g)
1588 {%
1589   \expandafter\XINT_divmod_bneg_finish
1590   \romannumeral0\xint_UDsignfork
1591     #1{\XINT_divmod_bpos {}}%
1592     -{\XINT_divmod_bpos {-#1}}%
1593   \krof
1594 }%
1595 \def\XINT_divmod_bneg_finish#1#2%

```

```

1596 {%
1597   \expandafter\xint_exchangetwo_keepbraces\expandafter
1598   {\romannumeral0\xintiioopp#2}{#1}%
1599 }%
1600 \def\xINT_divmod_bpos #1#2/#3[#4]#5/#6[#7].%
1601 {%
1602   \expandafter\xINT_divmod_bpos_a
1603   \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.%
1604   \romannumeral0\expandafter\xINT_mod_D_b
1605   \romannumeral0\xINT_div_prepare{#3}{#6}{#3}{#3}{#6}%
1606   {#1#5}{#7-#4}{#2}{#4-#7}%
1607 }%
1608 \def\xINT_divmod_bpos_a #1.#2#3#4%
1609 {%
1610   \expandafter\xINT_divmod_bpos_finish
1611   \romannumeral0\xintiidivision{#3}{#4}{/#2[#1]}%
1612 }%
1613 \def\xINT_divmod_bpos_finish #1#2#3{{#1}{#2#3}}%

```

8.53 \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.

```

1614 \def\xintMod {\romannumeral0\xintmod }%
1615 \def\xintmod #1{\expandafter\xINT_mod_a\romannumeral0\xintra{#1}.}%
1616 \def\xINT_mod_a #1#2.#3%
1617   {\expandafter\xINT_mod_b\expandafter #1\romannumeral0\xintra{#3}#2.}%
1618 \def\xINT_mod_b #1#2% #1 de A, #2 de B.
1619 {%
1620   \if0#2\xint_dothis{\XINT_mod_divbyzero #1#2}\fi
1621   \if0#1\xint_dothis\xINT_mod_aiszero\fi
1622   \if-#2\xint_dothis{\XINT_mod_bneg #1}\fi
1623   \xint_orthat{\XINT_mod_bpos #1#2}%
1624 }%

```

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

```

1625 \let\xINT_mod_divbyzero\xINT_modtrunc_divbyzero
1626 \let\xINT_mod_aiszero \XINT_modtrunc_aiszero
1627 \def\xINT_mod_bneg #1% f % -g = -((-f) % g), for g > 0
1628 {%
1629   \xintiioopp\xint_UDsignfork
1630   #1{\XINT_mod_bpos {}}%
1631   -{\XINT_mod_bpos {-#1}}%
1632   \krof
1633 }%
1634 \def\xINT_mod_bpos #1#2/#3[#4]#5/#6[#7].%
1635 {%
1636   \expandafter\xINT_mod_bpos_a
1637   \the\numexpr\ifnum#7>#4 #4\else #7\fi\expandafter.%

```

```

1638 \romannumeral0\expandafter\XINT_mod_D_b
1639 \romannumeral0\XINT_div_prepare{#3}{#6}{#3}{#3}{#6}%
1640 {#1#5}{#7-#4}{#2}{#4-#7}%
1641 }%
1642 \def\XINT_mod_D_a #1#2%
1643 {%
1644 \expandafter\XINT_mod_D_b
1645 \romannumeral0\XINT_div_prepare {#1}{#2}{#1}%
1646 }%
1647 \def\XINT_mod_D_b #1#2{\XINT_mod_D_c #2\Z}%
1648 \def\XINT_mod_D_c #1#2\Z
1649 {%
1650 \xint_gob_til_zero #1\XINT_mod_D_exit0\XINT_mod_D_a {#1#2}%
1651 }%
1652 \def\XINT_mod_D_exit0\XINT_mod_D_a #1#2#3%
1653 {%
1654 \expandafter\XINT_mod_E
1655 \romannumeral0\xintiipro {#3}{#2}.{#2}%
1656 }%
1657 \def\XINT_mod_E #1.#2#3%
1658 {%
1659 \expandafter\XINT_mod_F
1660 \romannumeral0\xintiimul{#1}{#3}.{\xintiiQuo{#3}{#2}}{#1}%
1661 }%
1662 \def\XINT_mod_F #1.#2#3#4#5#6#7%
1663 {%
1664 {#1}{\xintiiE{\xintiiMul{#4}{#3}}{#5}}%
1665 {\xintiiE{\xintiiMul{#6}{#2}}{#7}}%
1666 }%
1667 \def\XINT_mod_bpos_a #1.#2#3#4{\xintiiirem {#3}{#4}/#2[#1]}%

```

8.54 \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.

```

1668 \def\xintIsOne {\romannumeral0\xintisone }%
1669 \def\xintisone #1{\expandafter\XINT_fracisone
1670 \romannumeral0\xintrawwithzeros{#1}\Z }%
1671 \def\XINT_fracisone #1/#2\Z
1672 {\if0\xintiiCmp {#1}{#2}\xint_afterfi{ 1}\else\xint_afterfi{ 0}\fi}%

```

8.55 \xintGeq

```

1673 \def\xintGeq {\romannumeral0\xintgeq }%
1674 \def\xintgeq #1%
1675 {%
1676 \expandafter\XINT_fgeq\expandafter {\romannumeral0\xintabs {#1}}%
1677 }%
1678 \def\XINT_fgeq #1#2%
1679 {%
1680 \expandafter\XINT_fgeq_A \romannumeral0\xintabs {#2}#1%
1681 }%

```

```

1682 \def\XINT_fgeq_A #1%
1683 {%
1684   \xint_gob_til_zero #1\XINT_fgeq_Zii 0%
1685   \XINT_fgeq_B #1%
1686 }%
1687 \def\XINT_fgeq_Zii 0\XINT_fgeq_B #1[#2]#3[#4]{ 1}%
1688 \def\XINT_fgeq_B #1/#2[#3]#4#5/#6[#7]%
1689 {%
1690   \xint_gob_til_zero #4\XINT_fgeq_Zi 0%
1691   \expandafter\XINT_fgeq_C\expandafter
1692   {\the\numexpr #7-#3\expandafter}\expandafter
1693   {\romannumeral0\xintiimul {#4#5}{#2}}%
1694   {\romannumeral0\xintiimul {#6}{#1}}%
1695 }%
1696 \def\XINT_fgeq_Zi 0#1#2#3#4#5#6#7{ 0}%
1697 \def\XINT_fgeq_C #1#2#3%
1698 {%
1699   \expandafter\XINT_fgeq_D\expandafter
1700   {#3}{#1}{#2}%
1701 }%
1702 \def\XINT_fgeq_D #1#2#3%
1703 {%
1704   \expandafter\XINT_cntSgnFork\romannumeral`&&\expandafter\XINT_cntSgn
1705   \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\xint:
1706   { 0}{\XINT_fgeq_E #2\Z {#3}{#1}}{ 1}%
1707 }%
1708 \def\XINT_fgeq_E #1%
1709 {%
1710   \xint_UDsignfork
1711   #1\XINT_fgeq_Fd
1712   -{\XINT_fgeq_Fn #1}%
1713   \krof
1714 }%

1715 \def\XINT_fgeq_Fd #1\Z #2#3%
1716 {%
1717   \expandafter\XINT_fgeq_Fe
1718   \romannumeral0\XINT_dsx_addzeros {#1}#3;\xint:#2\xint:
1719 }%
1720 \def\XINT_fgeq_Fe #1\xint:#2#3\xint:{\XINT_geq_plusplus #2#1\xint:#3\xint:}%
1721 \def\XINT_fgeq_Fn #1\Z #2#3%
1722 {%
1723   \expandafter\XINT_fgeq_Fo
1724   \romannumeral0\XINT_dsx_addzeros {#1}#2;\xint:#3\xint:
1725 }%
1726 \def\XINT_fgeq_Fo #1#2\xint:#3\xint:{\XINT_geq_plusplus #1#3\xint:#2\xint:}%

8.56 \xintMax

1727 \def\xintMax {\romannumeral0\xintmax }%
1728 \def\xintmax #1%
1729 {%
1730   \expandafter\XINT_fmax\expandafter {\romannumeral0\xintraw {#1}}%
1731 }%

```

```

1732 \def\XINT_fmax #1#2%
1733 {%
1734   \expandafter\XINT_fmax_A\romannumeral0\xintra w {#2}#1%
1735 }%
1736 \def\XINT_fmax_A #1#2/#3[#4]#5#6/#7[#8]%
1737 {%
1738   \xint_UDsignsfork
1739     #1#5\XINT_fmax_minusminus
1740     -#5\XINT_fmax_firstneg
1741     #1-\XINT_fmax_secondneg
1742     --\XINT_fmax_nonneg_a
1743   \krof
1744   #1#5{#2/#3[#4]}{#6/#7[#8]}%
1745 }%
1746 \def\XINT_fmax_minusminus --%
1747   {\expandafter-\romannumeral0\XINT_fmin_nonneg_b }%
1748 \def\XINT_fmax_firstneg #1-#2#3{ #1#2}%
1749 \def\XINT_fmax_secondneg -#1#2#3{ #1#3}%
1750 \def\XINT_fmax_nonneg_a #1#2#3#4%
1751 {%
1752   \XINT_fmax_nonneg_b {#1#3}{#2#4}%
1753 }%
1754 \def\XINT_fmax_nonneg_b #1#2%
1755 {%
1756   \if0\romannumeral0\XINT_fgeq_A #1#2%
1757     \xint_afterfi{ #1}%
1758   \else \xint_afterfi{ #2}%
1759   \fi
1760 }%

```

8.57 \xintMaxof

1.21 protects \xintMaxof against items with non terminated \the\numexpr expressions.

1.4 renders the macro compatible with an empty argument and it also defines an accessor \XINT_Maxof suitable for xintexpr usage (formerly xintexpr had its own macro handling comma separated values, but it changed internal representation at 1.4).

```

1761 \def\xintMaxof {\romannumeral0\xintmaxof }%
1762 \def\xintmaxof #1{\expandafter\XINT_maxof\romannumeral`&&@#1^}%
1763 \def\XINT_Maxof{\romannumeral0\XINT_maxof}%
1764 \def\XINT_maxof#1%
1765 {%
1766   \xint_gob_til_^ #1\XINT_maxof_empty ^%
1767   \expandafter\XINT_maxof_loop\romannumeral0\xintra w{#1}\xint:
1768 }%
1769 \def\XINT_maxof_empty ^#1\xint:{ 0/1[0]}%
1770 \def\XINT_maxof_loop #1\xint:#2%
1771 {%
1772   \xint_gob_til_^ #2\XINT_maxof_e ^%
1773   \expandafter\XINT_maxof_loop
1774   \romannumeral0\xintmax{#1}{\romannumeral0\xintra w{#2}}\xint:
1775 }%
1776 \def\XINT_maxof_e ^#1\xintmax #2#3\xint:{ #2}%

```

8.58 \xintMin

```

1777 \def\xintMin {\romannumeral0\xintmin }%
1778 \def\xintmin #1%
1779 {%
1780   \expandafter\XINT_fmin\expandafter {\romannumeral0\xintraffrac {#1}}%
1781 }%
1782 \def\XINT_fmin #1#2%
1783 {%
1784   \expandafter\XINT_fmin_A\romannumeral0\xintraffrac {#2}#1%
1785 }%
1786 \def\XINT_fmin_A #1#2/#3[#4]#5#6/#7[#8]%
1787 {%
1788   \xint_UDsignsfork
1789     #1#5\XINT_fmin_minusminus
1790     -#5\XINT_fmin_firstneg
1791     #1-\XINT_fmin_secondneg
1792     --\XINT_fmin_nonneg_a
1793   \kroff
1794   #1#5{#2/#3[#4]}{#6/#7[#8]}%
1795 }%
1796 \def\XINT_fmin_minusminus --%
1797   {\expandafter-\romannumeral0\XINT_fmax_nonneg_b }%
1798 \def\XINT_fmin_firstneg #1-#2#3{ -#3}%
1799 \def\XINT_fmin_secondneg -#1#2#3{ -#2}%
1800 \def\XINT_fmin_nonneg_a #1#2#3#4%
1801 {%
1802   \XINT_fmin_nonneg_b {#1#3}{#2#4}%
1803 }%
1804 \def\XINT_fmin_nonneg_b #1#2%
1805 {%
1806   \if0\romannumeral0\XINT_fgeq_A #1#2%
1807     \xint_afterfi{ #2}%
1808   \else \xint_afterfi{ #1}%
1809   \fi
1810 }%

```

8.59 \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).

```

1811 \def\xintMinof {\romannumeral0\xintminof }%
1812 \def\xintminof #1{\expandafter\XINT_minof\romannumeral`&&@#1^}%
1813 \def\XINT_Minof{\romannumeral0\XINT_minof}%
1814 \def\XINT_minof#1%
1815 {%
1816   \xint_gob_til_^ #1\XINT_minof_empty ^%
1817   \expandafter\XINT_minof_loop\romannumeral0\xintraffrac{#1}\xint:
1818 }%
1819 \def\XINT_minof_empty ^#1\xint:{ 0/1[0]}%
1820 \def\XINT_minof_loop #1\xint:#2%
1821 {%
1822   \xint_gob_til_^ #2\XINT_minof_e ^%

```

```

1823 \expandafter\XINT_minof_loop\romannumeral0\xintmin{#1}{\romannumeral0\xintra{#2}}\xint:
1824 }%
1825 \def\XINT_minof_e ^#1\xintmin #2#3\xint:{ #2}%

```

8.60 \xintCmp

```

1826 \def\xintCmp {\romannumeral0\xintcmp }%
1827 \def\xintcmp #1%
1828 {%
1829 \expandafter\XINT_fcmp\expandafter {\romannumeral0\xintra{#1}}%
1830 }%
1831 \def\XINT_fcmp #1#2%
1832 {%
1833 \expandafter\XINT_fcmp_A\romannumeral0\xintra{#2}#1%
1834 }%
1835 \def\XINT_fcmp_A #1#2/#3[#4]#5#6/#7[#8]%
1836 {%
1837 \xint_UDsignsfork
1838 #1#5\XINT_fcmp_minusminus
1839 -#5\XINT_fcmp_firstneg
1840 #1-\XINT_fcmp_secondneg
1841 --\XINT_fcmp_nonneg_a
1842 \krof
1843 #1#5{#2/#3[#4]}{#6/#7[#8]}%
1844 }%
1845 \def\XINT_fcmp_minusminus --#1#2{\XINT_fcmp_B #2#1}%
1846 \def\XINT_fcmp_firstneg #1-#2#3{ -1}%
1847 \def\XINT_fcmp_secondneg -#1#2#3{ 1}%
1848 \def\XINT_fcmp_nonneg_a #1#2%
1849 {%
1850 \xint_UDzerosfork
1851 #1#2\XINT_fcmp_zerozero
1852 0#2\XINT_fcmp_firstzero
1853 #10\XINT_fcmp_secondzero
1854 00\XINT_fcmp_pos
1855 \krof
1856 #1#2%
1857 }%
1858 \def\XINT_fcmp_zerozero #1#2#3#4{ 0}%
1859 \def\XINT_fcmp_firstzero #1#2#3#4{ -1}%
1860 \def\XINT_fcmp_secondzero #1#2#3#4{ 1}%
1861 \def\XINT_fcmp_pos #1#2#3#4%
1862 {%
1863 \XINT_fcmp_B #1#3#2#4%
1864 }%
1865 \def\XINT_fcmp_B #1/#2[#3]#4/#5[#6]%
1866 {%
1867 \expandafter\XINT_fcmp_C\expandafter
1868 {\the\numexpr #6-#3\expandafter}\expandafter
1869 {\romannumeral0\xintiimul {#4}{#2}}%
1870 {\romannumeral0\xintiimul {#5}{#1}}%
1871 }%
1872 \def\XINT_fcmp_C #1#2#3%

```



```

1873 {%
1874   \expandafter\XINT_fcmp_D\expandafter
1875   {#3}{#1}{#2}%
1876 }%
1877 \def\XINT_fcmp_D #1#2#3%
1878 {%
1879   \expandafter\XINT_cntSgnFork\romannumeral`&&\expandafter\XINT_cntSgn
1880   \the\numexpr #2+\xintLength{#3}-\xintLength{#1}\relax\xint:
1881   { -1}{\XINT_fcmp_E #2\Z {#3}{#1}}{ 1}%
1882 }%
1883 \def\XINT_fcmp_E #1%
1884 {%
1885   \xint_UDsignfork
1886   #1\XINT_fcmp_Fd
1887   -{\XINT_fcmp_Fn #1}%
1888   \krof
1889 }%

1890 \def\XINT_fcmp_Fd #1\Z #2#3%
1891 {%
1892   \expandafter\XINT_fcmp_Fe
1893   \romannumeral0\XINT_dsx_addzeros {#1}#3;\xint:#2\xint:
1894 }%
1895 \def\XINT_fcmp_Fe #1\xint:#2#3\xint:{\XINT_cmp_plusplus #2#1\xint:#3\xint:}%
1896 \def\XINT_fcmp_Fn #1\Z #2#3%
1897 {%
1898   \expandafter\XINT_fcmp_Fo
1899   \romannumeral0\XINT_dsx_addzeros {#1}#2;\xint:#3\xint:
1900 }%
1901 \def\XINT_fcmp_Fo #1#2\xint:#3\xint:{\XINT_cmp_plusplus #1#3\xint:#2\xint:}%

```

8.61 \xintAbs

```

1902 \def\xintAbs {\romannumeral0\xintabs }%
1903 \def\xintabs #1{\expandafter\XINT_abs\romannumeral0\xintraw {#1}}%

```

8.62 \xintOpp

```

1904 \def\xintOpp {\romannumeral0\xintopp }%
1905 \def\xintopp #1{\expandafter\XINT_opp\romannumeral0\xintraw {#1}}%

```

8.63 \xintInv

1.3d.

```

1906 \def\xintInv {\romannumeral0\xintinv }%
1907 \def\xintinv #1{\expandafter\XINT_inv\romannumeral0\xintraw {#1}}%
1908 \def\XINT_inv #1%
1909 {%
1910   \xint_UDzerominusfork
1911   #1-\XINT_inv_iszero
1912   0#1\XINT_inv_a
1913   0-{\XINT_inv_a {}}%
1914   \krof #1%
1915 }%
1916 \def\XINT_inv_iszero #1]%

```

```

1917   {\XINT_signalcondition{DivisionByZero}{Division of 1 by zero (#1)}}{\{0/1[0]}}%
1918 \def\XINT_inv_a #1#2/#3[#4#5]%
1919 {%
1920   \xint_UDzerominusfork
1921   #4-\XINT_inv_expiszero
1922   0#4\XINT_inv_b
1923   0-{\XINT_inv_b -#4}%
1924   \krof #5.{#1#3/#2}%
1925 }%
1926 \def\XINT_inv_expiszero #1.#2{ #2[0]}%
1927 \def\XINT_inv_b #1.#2{ #2[#1]}%

```

8.64 \xintSgn

```

1928 \def\xintSgn {\romannumeral0\xintsgn}%
1929 \def\xintsgn #1{\expandafter\XINT_sgn\romannumeral0\xintra #1}\xint:%

```

8.65 \xintGCD, \xintLCM

1.4. They replace the former [xintgcd](#) macros of the same names which truncated to integers their arguments. Fraction-producing [gcd\(\)](#) and [lcm\(\)](#) functions were available since 1.3d [xintexpr](#), with non-public support macros handling comma separated values.

```

1930 \def\xintGCD {\romannumeral0\xintgcd}%
1931 \def\xintgcd #1#2{\XINT_fgcdof{#1}{#2}^}%
1932 \def\xintLCM {\romannumeral0\xintlcm}%
1933 \def\xintlcm #1#2{\XINT_flcmof{#1}{#2}^}%

```

8.66 \xintGCDof

1.4. This inherits from former non public [xintexpr](#) macro called [\xintGCDof:csv](#), handling comma separated items, and former [xintgcd](#) macro called [\xintGCDof](#) which handled braced items to which it applied [\xintNum](#) before handling the computations on integers only. The macro keeps the former name [xintgcd](#), and handles fractions presented as braced items. It is now the support macro for the [gcd\(\)](#) function in [\xintexpr](#) and [\xintfloatexpr](#).

The support macro for the [gcd\(\)](#) function in [\xintiexpr](#) is [\xintiiGCDof](#) which is located in [xint](#).

```

1934 \def\xintGCDof {\romannumeral0\xintgcdof}%
1935 \def\xintgcdof #1{\expandafter\XINT_fgcdof\romannumeral`&&@#1^}%
1936 \def\XINT_GCDof{\romannumeral0\XINT_fgcdof}%

```

This abuses the way [\xintiiabs](#) works in order to avoid fetching whole argument again: [\xintiiabs](#) ^ raises no error.

```

1937 \def\XINT_fgcdof #1%
1938 {%
1939   \xint_gob_til_ ^ #1\XINT_fgcdof_empty ^%
1940   \expandafter\XINT_fgcdof_loop\romannumeral0\xintiiabs#1\xint:
1941 }%
1942 \def\XINT_fgcdof_empty ^#1\xint:{ 1/1[0]}%
1943 \def\XINT_fgcdof_loop #1\xint:#2%
1944 {%
1945   \expandafter\XINT_fgcdof_loop_a\romannumeral0\xintiiabs#2\xint:#1\xint:
1946 }%

```

```

1947 \def\XINT_fgcdof_loop_a#1#2\xint:#3\xint:
1948 {%
1949   \xint_gob_til_^ #1\XINT_fgcdof_end ^%
1950   \xint_gob_til_zero #1\XINT_fgcdof_skip 0%
1951   \expandafter\XINT_fgcdof_loop_b\romannumeral0\xintmod{#1#2}{#3}\xint:#3\xint:
1952 }%
1953 \def\XINT_fgcdof_end ^#1\xint:#2\xint:{ #2}%
1954 \def\XINT_fgcdof_skip 0%
1955   \expandafter\XINT_fgcdof_loop_b\romannumeral0\xintmod#1\xint:
1956 {%
1957   \XINT_fgcdof_loop
1958 }%
1959 \def\XINT_fgcdof_loop_b#1#2\xint:#3\xint:
1960 {%
1961   \xint_gob_til_zero #1\XINT_fgcdof_next 0%
1962   \expandafter\XINT_fgcdof_loop_b\romannumeral0\xintmod{#3}{#1#2}\xint:#1#2\xint:
1963 }%
1964 \def\XINT_fgcdof_next 0%
1965   \expandafter\XINT_fgcdof_loop_b\romannumeral0\xintmod#1#2\xint:#3\xint:#4%
1966 {%
1967   \expandafter\XINT_fgcdof_loop_a\romannumeral0\xintiiabs#4\xint:#1\xint:
1968 }%

```

8.67 \xintLCMof

See comments for \xintGCDof. **xint** provides integer only \xintiilCMof.

```

1969 \def\xintLCMof {\romannumeral0\xintlcmof}%
1970 \def\xintlcmof #1{\expandafter\XINT_flcmof\romannumeral`&&@#1^}%
1971 \def\XINT_LCMof{\romannumeral0\XINT_flcmof}%
1972 \def\XINT_flcmof #1%
1973 {%
1974   \xint_gob_til_^ #1\XINT_flcmof_empty ^%
1975   \expandafter\XINT_flcmof_loop\romannumeral0\xintiiabs\xintRaw{#1}\xint:
1976 }%
1977 \def\XINT_flcmof_empty ^#1\xint:{ 0/1[0]}%

```

\XINT_inv expects A/B[N] format which is the case here.

```

1978 \def\XINT_flcmof_loop #1%
1979 {%
1980   \xint_gob_til_zero #1\XINT_flcmof_zero 0%
1981   \expandafter\XINT_flcmof_d\romannumeral0\XINT_inv #1%
1982 }%
1983 \def\XINT_flcmof_zero #1^{ 0/1[0]}%

```

\xintRaw{^} would raise an error thus we delay application of \xintRaw to new item. As soon as we hit against a zero item, the l.c.m is known to be zero itself. Else we need to inverse new item, but this requires full A/B[N] raw format, hence the \xintraw.

```

1984 \def\XINT_flcmof_d #1\xint:#2%
1985 {%
1986   \expandafter\XINT_flcmof_loop_a\romannumeral0\xintiiabs#2\xint:#1\xint:
1987 }%

```

```

1988 \def\XINT_flgcmof_loop_a #1#2\xint:%
1989 {%
1990     \xint_gob_til_^ #1\XINT_flgcmof_end ^%
1991     \xint_gob_til_zero #1\XINT_flgcmof_zero 0%
1992     \expandafter\XINT_flgcmof_loop_b\romannumeral0\expandafter\XINT_inv
1993     \romannumeral0\xintra{#1#2}\xint:
1994 }%
1995 \def\XINT_flgcmof_end ^#1\xint:#2\xint:{\XINT_inv #2}%

```

This is Euclide algorithm.

```

1996 \def\XINT_flgcmof_loop_b #1#2\xint:#3\xint:
1997 {%
1998     \xint_gob_til_zero #1\XINT_flgcmof_next 0%
1999     \expandafter\XINT_flgcmof_loop_b\romannumeral0\xintmod{#3}{#1#2}\xint:#1#2\xint:
2000 }%
2001 \def\XINT_flgcmof_next 0%
2002     \expandafter\XINT_flgcmof_loop_b\romannumeral0\xintmod#1#2\xint:#3\xint:#4%
2003 {%
2004     \expandafter\XINT_flgcmof_loop_a\romannumeral0\xintiiabs#4\xint:#1\xint:
2005 }%

```

8.68 Floating point macros

For a long time the float routines dating back to releases [1.07/1.08a](#) (May-June 2013) were not modified.

Since [1.2f](#) (March 2016) the four operations first round their arguments to `\xinttheDigits`-floats (or P -floats), not `(\xinttheDigits+2)`-floats or $(P+2)$ -floats as was the case with earlier releases.

The four operations addition, subtraction, multiplication, division have always produced the correct rounding of the theoretical exact value to P or `\xinttheDigits` digits when the inputs are decimal numbers with at most P digits, and arbitrary decimal exponent part.

From [1.08a](#) to [1.2j](#), `\xintFloat` (and `\XINTinFloat` which is used to parse inputs to other float macros) handled a fractional input A/B via an initial replacement to A'/B' where A' and B' were A and B truncated to $Q+2$ digits (where asked-for precision is Q), and then they correctly rounded A'/B' to Q digits. But this meant that this rounding of the input could differ (by up to one unit in the last place) from the correct rounding of the original A/B to the asked-for number of digits (which until [1.2f](#) in uses as auxiliary to the macros for the basic operations was 2 more than the prevailing precision).

Since [1.2k](#) all inputs are correctly rounded to the asked-for number of digits (this was, I think, the case in the [1.07](#) release -- there are no code comments -- but was, afaicr, not very efficiently done, and this is why the [1.08a](#) release opted for truncation of the numerator and denominator.)

Notice that in float expressions, the `/` is treated as operator, hence the above discussion makes a difference only for the special input form `qfloat(A/B)` or for an `\xintexpr A/B\relax` embedded in the float expression, with A or B having more digits than the prevailing float precision.

Internally there is no inner representation of P -floats as such !!!!!

The input parser will again compute the length of the mantissa on each use !!! This is obviously something that must be improved upon before implementation of higher functions.

Currently, special tricks are used to quickly recognize inputs having no denominators, or fractions whose numerators and denominators are not too long compared to the target precision P , and in particular P -floats or quotients of two such.

Another long-standing issue is that float multiplication will first compute the $2P$ or $2P-1$ digits of the exact product, and then round it to P digits. This is sub-optimal for large P particularly as the multiplication algorithm is basically the schoolbook one, hence worse than quadratic in the \TeX implementation which has extra cost of fetching long sequences of tokens.

8.69 `\xintDigits`, `\xintSetDigits`

1.3f modifies the (strange) original signature `#1#2` for `\xintDigits` macro into `#1=`, allowing usage without colon. It also adds `\xintSetDigits`. Starred variants are added by `xintexpr.sty`.

```
2006 \mathchardef\XINTdigits 16
2007 \def\xintDigits #1=%
2008   {\afterassignment \xint_gobble_i \mathchardef\XINTdigits=%}%
2009 \def\xinttheDigits {\number\XINTdigits }%
2010 \def\xintSetDigits #1{\mathchardef\XINTdigits=\numexpr#1\relax}%
```

8.70 `\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 δ (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...0e N 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`.

```

2011 \def\xintFloat    {\romannumeral0\xintfloat }%
2012 \def\xintfloat #1{\XINT_float_chkopt #1\xint:}%
2013 \def\XINT_float_chkopt #1%
2014 {%
2015     \ifx [#1\expandafter\XINT_float_opt
2016         \else\expandafter\XINT_float_noopt
2017     \fi #1%
2018 }%
2019 \def\XINT_float_noopt #1\xint:%
2020 {%
2021     \expandafter\XINT_float_post
2022     \romannumeral0\XINTinfloat[\XINTdigits]{#1}\XINTdigits.%
2023 }%
2024 \def\XINT_float_opt [\xint:#1]%
2025 {%
2026     \expandafter\XINT_float_opt_a\the\numexpr #1.%
2027 }%
2028 \def\XINT_float_opt_a #1.#2%
2029 {%
2030     \expandafter\XINT_float_post
2031     \romannumeral0\XINTinfloat[#1]{#2}#1.%
2032 }%
2033 \def\XINT_float_post #1%
2034 {%
2035     \xint_UDzerominusfork
2036     #1-\XINT_float_zero
2037     0#1\XINT_float_neg
2038     0-\XINT_float_pos
2039     \krof #1%
2040 }%[
2041 \def\XINT_float_zero #1]#2.{ 0.e0}%
2042 \def\XINT_float_neg-{\expandafter-\romannumeral0\XINT_float_pos}%
2043 \def\XINT_float_pos #1#2[#3]#4.%
2044 {%
2045     \expandafter\XINT_float_pos_done\the\numexpr#3+#4-\xint_c_i.#1.#2;%
2046 }%
2047 \def\XINT_float_pos_done #1.#2;{ #2e#1}%

```

8.71 \XINTinFloat, \XINTinFloatS, \XINTiLogTen

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

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

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

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

Each time `\XINTinFloat` is called it at least computes a length. Naturally if we had some format for floats that would be dispensed of... something like `<letterP><length of mantissa>.mantissa.exponent`, etc... not yet.

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

1.3e adds `\XINTiLogTen`.

```
2048 \def\XINTinFloat {\romannumeral0\XINTinfloat }%
2049 \def\XINTinfloat
2050   {\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).

```
2051 \def\XINT_infloat_clean #1%
2052   {\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

```
2053 \def\XINT_infloat_clean_a !#1.#2[#3]%
2054 {%
2055   \expandafter\XINT_infloat_done
2056   \the\numexpr #3-#1\expandafter.%
2057   \romannumeral0\XINT_dsx_addzeros {#1}#2;;%
2058 }%
2059 \def\XINT_infloat_done #1.#2;{ #2[#1]}%
```

variant which allows output with shorter mantissas.

```
2060 \def\XINTinFloatS {\romannumeral0\XINTinfloatS}%
2061 \def\XINTinfloatS
2062   {\expandafter\XINT_infloatS_clean\romannumeral0\XINT_infloat}%
2063 \def\XINT_infloatS_clean #1%
2064   {\if #1!\xint_dothis\XINT_infloatS_clean_a\fi\xint_orthat{ }#1}%
2065 \def\XINT_infloatS_clean_a !#1.{ }%
```

1.3e ajoute `\XINTiLogTen`. Le comportement pour un input nul est non encore finalisé. Il changera lorsque NaN, +Inf, -Inf existeront.

```
2066 \def\XINTFloatiLogTen {\the\numexpr\XINTfloatilogten}%
2067 \def\XINTfloatilogten [#1]#2%
2068   {\expandafter\XINT_floatilogten\romannumeral0\XINT_infloat[#1]{#2}#1.}%
2069 \def\XINT_floatilogten #1{%
2070   \if #10\xint_dothis\XINT_floatilogten_z\fi
2071   \if #1!\xint_dothis\XINT_floatilogten_a\fi
2072   \xint_orthat\XINT_floatilogten_b #1%
2073 }%
2074 \def\XINT_floatilogten_z 0[0]#1.-"7FFF8000\relax}%
2075 \def\XINT_floatilogten_a !#1.#2[#3]#4.{#3-#1+#4-1\relax}%
2076 \def\XINT_floatilogten_b #1[#2]#3.{#2+#3-1\relax}%
```

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

```
2077 \def\XINT_infloat [#1]#2%
```

```

2078 {%
2079   \expandafter\XINT_infloat_a\the\numexpr #1\expandafter.%
2080   \romannumeral0\XINT_infrac {#2}%
2081 }%

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

2082 \def\XINT_infloat_a #1.#2#3#4%
2083 {%
    micro boost au lieu d'utiliser \XINT_isOne{#4}, mais pas bon style.

2084   \if1\XINT_is_One#4XY%
2085     \expandafter\XINT_infloat_sp
2086   \else\expandafter\XINT_infloat_fork
2087   \fi #3.{#1}{#2}{#4}%
2088 }%

    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.

2089 \def\XINT_infloat_sp #1%
2090 {%
2091   \xint_UDzerominusfork
2092   #1-\XINT_infloat_spzero
2093   0#1\XINT_infloat_spneg
2094   0-\XINT_infloat_sppos
2095   \krof #1%
2096 }%

    Attention surtout pas 0/1[0] ici.

2097 \def\XINT_infloat_spzero 0.#1#2#3{ 0[0]}%
2098 \def\XINT_infloat_spneg-%
2099   {\expandafter\XINT_infloat_spnegend\romannumeral0\XINT_infloat_sppos}%
2100 \def\XINT_infloat_spnegend #1%
2101   {\if#1!\expandafter\XINT_infloat_spneg_needzeros\fi -#1}%
2102 \def\XINT_infloat_spneg_needzeros -!#1.{!#1.-}%

    in: A.{P}{N}{1}
    out: P-L.A.P.N.

2103 \def\XINT_infloat_sppos #1.#2#3#4%
2104 {%
2105   \expandafter\XINT_infloat_sp_b\the\numexpr#2-\xintLength{#1}.#1.#2.#3.%
2106 }%

    #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.

2107 \def\XINT_infloat_sp_b #1%
2108 {%
2109   \xint_UDzerominusfork
2110   #1-\XINT_infloat_sp_quick
2111   0#1\XINT_infloat_sp_c
2112   0-\XINT_infloat_sp_needzeros
2113   \krof #1%
2114 }%

```


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

```
2115 \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 `needzeros`. Si on appelle sous la forme `\XINTinFloatS`, on ne fait pas l'addition de zéros.

```
2116 \def\xINT_infloat_sp_needzeros #1.#2.#3.#4.{!#1.#2[#4]}%
```

$L-P=\#1$. $A=\#2$. $\#3=P$. $N=\#4$.
 Ici $P<L$. Il va falloir arrondir. Attention si on va à la puissance de 10 suivante. En $\#1$ on a $L-P$ qui est >0 . L'exposant final sera $N+L-P$, sauf dans le cas spécial, il sera alors $N+L-P+1$. L'ajustement final est fait par `\XINT_infloat_Y`.

```
2117 \def\xINT_infloat_sp_c -#1.#2#3.#4.#5.%
```

```
2118 {%
2119   \expandafter\xINT_infloat_Y
2120   \the\numexpr #5+#1\expandafter.%
2121   \romannumeral0\expandafter\xINT_infloat_sp_round
2122   \romannumeral0\xINT_split_fromleft
2123   (\xint_c_i+#4).#2#3\xint_bye2345678\xint_bye..#2%
2124 }%
2125 \def\xINT_infloat_sp_round #1.#2.%
2126 {%
2127   \XINT_dsrr#1\xint_bye\xint_Bye3456789\xint_bye/\xint_c_x\relax.%
2128 }%
```

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.

```
2129 \def\xINT_infloat_fork #1%
```

```
2130 {%
2131   \xint_UDsignfork
2132   #1\xINT_infloat_J
2133   -\XINT_infloat_K
2134   \krof #1%
2135 }%
2136 \def\xINT_infloat_J-{\expandafter-\romannumeral0\xINT_infloat_K }%
```

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

```
2137 \def\xINT_infloat_K #1.#2%
```

```
2138 {%
2139   \expandafter\xINT_infloat_L
2140   \the\numexpr\xintLength{#1}\expandafter.\the\numexpr #2+\xint_c_iv.{#1}\{#2}%
2141 }%
```

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

```
2142 \def\xINT_infloat_L #1.#2.%
```

```
2143 {%
```

```

2144 \ifnum #1>#2
2145 \expandafter\XINT_infloat_Ma
2146 \else
2147 \expandafter\XINT_infloat_Mb
2148 \fi #1.#2.%
2149 }%

```

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

```

2150 \def\XINT_infloat_Ma #1.#2.#3%
2151 {%
2152 \expandafter\XINT_infloat_MtoN\expandafter-\expandafter0\expandafter.%
2153 \romannumeral0\XINT_split_fromleft#2.#3\xint_bye2345678\xint_bye..%
2154 #2.#1.{#3}%
2155 }%

```

$|A|.P+4.\{A\}\{P\}\{n\}\{B\}$.
Here A is short. We set $u = P+4 - |A|$, and $A'' = A$ ($A' = 10^u A$)
output: $u.A''.$ $P+4.$ $|A|. \{A\}\{P\}\{n\}\{B\}$

```

2156 \def\XINT_infloat_Mb #1.#2.#3%
2157 {%
2158 \expandafter\XINT_infloat_MtoN\the\numexpr#2-#1.%
2159 #3..#2.#1.{#3}%
2160 }%

```

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

```

2161 \def\XINT_infloat_MtoN #1.#2.#3.#4.#5.#6#7#8#9%
2162 {%
2163 \expandafter\XINT_infloat_N
2164 \the\numexpr\xintLength{#9}.#4.{#9}#1.#2.#7.#5.#8.{#6}\{#9}%
2165 }%
2166 \def\XINT_infloat_N #1.#2.%
2167 {%
2168 \ifnum #1>#2
2169 \expandafter\XINT_infloat_Oa
2170 \else
2171 \expandafter\XINT_infloat_Ob
2172 \fi #1.#2.%
2173 }%

```

input $|B|.P+4.\{B\}u.A''.$ $P.$ $|A|.n.\{A\}\{B\}$
output $v=-0.B''.$ junk. $|B|.u.A''.$ $P.$ $|A|.n.\{A\}\{B\}$

```

2174 \def\XINT_infloat_Oa #1.#2.#3%
2175 {%
2176 \expandafter\XINT_infloat_P\expandafter-\expandafter0\expandafter.%
2177 \romannumeral0\XINT_split_fromleft#2.#3\xint_bye2345678\xint_bye..%
2178 #1.%
2179 }%

```

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

```
2180 \def\XINT_infloat_0b #1.#2.#3%
2181 {%
2182   \expandafter\XINT_infloat_P\the\numexpr#2-#1.#3..#1.%
2183 }%
```

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.

```
2184 \def\XINT_infloat_P #1#2.#3.#4.#5.#6#7.#8.#9.%
2185 {%
2186   \csname XINT_infloat_Q\if-#1\else\if-#6\else q\fi\fi\expandafter\endcsname
2187   \romannumeral0\xintiigo
2188   {\romannumeral0\XINT_dsx_addzerosnofuss
2189     {#6#7-#1#2+#9+\xint_c_iii\if-#1\else\if-#6\else-\xint_c_ii\fi\fi}#8;}%
2190   {#3}.#9.#5.%
2191 }%
```

«quick» branch.

```
2192 \def\XINT_infloat_Qq #1.#2.%
2193 {%
2194   \expandafter\XINT_infloat_Rq
2195   \romannumeral0\XINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..#2.%
2196 }%
2197 \def\XINT_infloat_Rq #1.#2#3.%
2198 {%
2199   \ifnum#2<\xint_c_v
2200     \expandafter\XINT_infloat_SEQ
2201   \else\expandafter\XINT_infloat_SUP
2202   \fi
2203   {\if.#3.\xint_c\else\xint_c_i\fi}#1.%
2204 }%
```

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

```
2205 \def\XINT_infloat_Q #1.#2.%
2206 {%
2207   \expandafter\XINT_infloat_R
2208   \romannumeral0\XINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..#2.%
2209 }%
2210 \def\XINT_infloat_R #1.#2#3#4#5.%
2211 {%
2212   \if.#5.\expandafter\XINT_infloat_Sa\else\expandafter\XINT_infloat_Sb\fi
2213   #2#3#4#5.#1.%
2214 }%
```

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

```

2215 \def\XINT_infloat_Sa #1.%
2216 {%
2217   \ifnum#1>500 \xint_dothis\XINT_infloat_SUP\fi
2218   \ifnum#1<499 \xint_dothis\XINT_infloat_SEq\fi
2219   \xint_orthat\XINT_infloat_X\xint_c_
2220 }%
2221 \def\XINT_infloat_Sb #1.%
2222 {%
2223   \ifnum#1>5009 \xint_dothis\XINT_infloat_SUP\fi
2224   \ifnum#1<4990 \xint_dothis\XINT_infloat_SEq\fi
2225   \xint_orthat\XINT_infloat_X\xint_c_i
2226 }%

```

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

```

2227 \def\XINT_infloat_SEq #1#2.#3.#4.#5.#6.#7#8%
2228 {%
2229   \expandafter\XINT_infloat_SY
2230   \the\numexpr #6+#5-#4-#3+#1.#2.%
2231 }%
2232 \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.

```

2233 \def\XINT_infloat_SUP #1#2#3.#4.#5.#6.#7.#8#9%
2234 {%
2235   \expandafter\XINT_infloat_Y
2236   \the\numexpr#7+#6-#5-#4+#1\expandafter.%
2237   \romannumeral0\xintinc{#2#3}.#2%
2238 }%

```

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

```

2239 \def\XINT_infloat_X #1#2#3.#4.#5.#6.#7.#8#9%
2240 {%
2241   \expandafter\XINT_infloat_Y
2242   \the\numexpr #7+#6-#5-#4+#1\expandafter.%
2243   \romannumeral`&&\romannumeral0\xintiiflt
2244   {\xintDSH{#6-#5-#4+#1}{\xintDouble{#8}}}%
2245   {\xintiiMul{\xintInc{\xintDouble{#2#3}}}{#9}}%
2246   \xint_firstofone
2247   \xintinc{#2#3}.#2%
2248 }%

```

check for rounding up to next power of ten.

```

2249 \def\XINT_infloat_Y #1{%

```

```

2250 \def\XINT_infloat_Y ##1.##2##3.##4%
2251 {%
2252   \if##49\if##21\expandafter\expandafter\expandafter\XINT_infloat_Z\fi\fi
2253   #1##2##3[##1]%
2254 }}\XINT_infloat_Y{ }%

    #1=1, #2=0.

2255 \def\XINT_infloat_Z #1#2#3[#4]%
2256 {%
2257   \expandafter\XINT_infloat_ZZ\the\numexpr#4+\xint_c_i.#3.%
2258 }%
2259 \def\XINT_infloat_ZZ #1.#2.{ 1#2[#1]}%

```

8.72 \xintPFloat, \xintPFloatE

1.1. This is a prettifying printing macro for floats.

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

Special case: the zero value is printed 0. (with a dot)

The coding got simpler with 1.2k as its `\xintFloat` always produces a mantissa with exactly P digits (no more $10.0\dots 0eN$ annoying exception).

1.4b adds `\xintPFloatE` allowing to customize whether to use e or E (or something else). For usage with `\xintfloateval{}` (anyhow only catcode 11 e is recognized by `xintfrac` macros proper), and to match similar `\xintFracToSciE`. For reasons commented upon in user manual (section «The three parsers»), I did not make an effort to let the macro be usable as a hook to grab the exponent.

Althout `\xintfloateval{}` will use `\xintPFloat` in an `\expanded` context we have to maintain `f-expandability` here.

```

2260 \def\xintPFloat   {\romannumeral0\xintpfloat }%
2261 \def\xintpfloat #1{\XINT_pfloat_chkopt #1\xint:}%
2262 \def\XINT_pfloat_chkopt #1%
2263 {%
2264   \ifx [#1\expandafter\XINT_pfloat_opt
2265     \else\expandafter\XINT_pfloat_noopt
2266     \fi #1%
2267 }%
2268 \def\XINT_pfloat_noopt #1\xint:%
2269 {%
2270   \expandafter\XINT_pfloat_a
2271   \romannumeral0\xintfloat [\XINTdigits]{#1};\XINTdigits.%
2272 }%

2273 \def\XINT_pfloat_opt [\xint:#1]%
2274 {%
2275   \expandafter\XINT_pfloat_opt_a \the\numexpr #1.%
2276 }%
2277 \def\XINT_pfloat_opt_a #1.#2%
2278 {%
2279   \expandafter\XINT_pfloat_a\romannumeral0\xintfloat [#1]{#2};#1.%
2280 }%
2281 \def\XINT_pfloat_a #1%

```

```

2282 {%
2283   \xint_UDzerominusfork
2284     #1-\XINT_pfloat_zero
2285     0#1\XINT_pfloat_neg
2286     0-\XINT_pfloat_pos
2287   \krof #1%
2288 }%

2289 \def\XINT_pfloat_zero #1;#2.{ 0.}%
2290 \def\XINT_pfloat_neg-{\expandafter-\romannumeral0\XINT_pfloat_pos }%

```

1.4b modifies the replacement pattern here #1{#2}{#3} in order to facilitate injection of once-expanded \xintPFloatE.

```

2291 \def\XINT_pfloat_pos #1.#2e#3;#4.%
2292 {%
2293   \ifnum #3>\xint_c_v \xint_dothis\XINT_pfloat_no\fi
2294   \ifnum #3<-\xint_c_v \xint_dothis\XINT_pfloat_no\fi
2295   \ifnum #3<\xint_c_ \xint_dothis\XINT_pfloat_N\fi
2296   \ifnum #3>\numexpr #4-\xint_c_i\relax \xint_dothis\XINT_pfloat_Ps\fi
2297   \xint_orthat\XINT_pfloat_P #1{#2}{#3}%
2298 }%
2299 \def\XINT_pfloat_no
2300 {%
2301   \expandafter\XINT_pfloat_no_e\expandafter{\xintPFloatE}%
2302 }%
2303 \def\XINT_pfloat_no_e #1#2#3#4{ #2.#3#1#4}%
2304 \def\xintPFloatE{e}%

```

This is all simpler coded, now that 1.2k's \xintFloat always outputs a mantissa with exactly one digits before decimal mark always.

```

2305 \def\XINT_pfloat_N #1#2#3%
2306 {%
2307   \csname XINT_pfloat_N_\romannumeral-#3\endcsname #1#2%
2308 }%
2309 \def\XINT_pfloat_N_i { 0.}%
2310 \def\XINT_pfloat_N_ii { 0.0}%
2311 \def\XINT_pfloat_N_iii{ 0.00}%
2312 \def\XINT_pfloat_N_iv { 0.000}%
2313 \def\XINT_pfloat_N_v { 0.0000}%

2314 \def\XINT_pfloat_P #1#2#3%
2315 {%
2316   \csname XINT_pfloat_P_\romannumeral#3\endcsname #1#2%
2317 }%
2318 \def\XINT_pfloat_P_ #1{ #1.}%
2319 \def\XINT_pfloat_P_i #1#2{ #1#2.}%
2320 \def\XINT_pfloat_P_ii #1#2#3{ #1#2#3.}%
2321 \def\XINT_pfloat_P_iii#1#2#3#4{ #1#2#3#4.}%
2322 \def\XINT_pfloat_P_iv #1#2#3#4#5{ #1#2#3#4#5.}%
2323 \def\XINT_pfloat_P_v #1#2#3#4#5#6{ #1#2#3#4#5#6.}%

```

```

2324 \def\XINT_pfloat_Ps #1#2#3%
2325 {%
2326   \csname XINT_pfloat_Ps\romannumeral#3\endcsname #1#200000;%
2327 }%
2328 \def\XINT_pfloat_Psi #1#2#3;{ #1#2.}%
2329 \def\XINT_pfloat_Psii #1#2#3#4;{ #1#2#3.}%
2330 \def\XINT_pfloat_Psiii#1#2#3#4#5;{ #1#2#3#4.}%
2331 \def\XINT_pfloat_Psiv #1#2#3#4#5#6;{ #1#2#3#4#5.}%
2332 \def\XINT_pfloat_Psv #1#2#3#4#5#6#7;{ #1#2#3#4#5#6.}%

```

8.73 \XINTinFloatFracdigits

1.09i, for frac function in `\xintfloatexpr`. This version computes exactly from the input the fractional part and then only converts it into a float with the asked-for number of digits. I will have to think it again some day, certainly.

1.1 removes optional argument for which there was anyhow no interface, for technical reasons having to do with `\xintNewExpr`.

1.1a renames the macro as `\XINTinFloatFracdigits` (from `\XINTinFloatFrac`) to be synchronous with the `\XINTinFloatSqrt` and `\XINTinFloat` habits related to `\xintNewExpr` problems.

Note to myself: I still have to rethink the whole thing about what is the best to do, the initial way of going through `\xintfrac` was just a first implementation.

```

2333 \def\XINTinFloatFracdigits {\romannumeral0\XINTinfloatfracdigits }%
2334 \def\XINTinfloatfracdigits #1%
2335 {%
2336   \expandafter\XINT_infloatfracdg_a\expandafter {\romannumeral0\xintfrac{#1}}%
2337 }%
2338 \def\XINT_infloatfracdg_a {\XINTinfloat [\XINTdigits]}%

```

8.74 \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.

```

2339 \def\xintFloatAdd      {\romannumeral0\xintfloatadd }%
2340 \def\xintfloatadd      #1{\XINT_fladd_chkopt \xintfloat #1\xint:}%
2341 \def\XINTinFloatAdd    {\romannumeral0\XINTinfloatadd }%
2342 \def\XINTinfloatadd    #1{\XINT_fladd_chkopt \XINTinfloatS #1\xint:}%
2343 \def\XINT_fladd_chkopt #1#2%
2344 {%
2345   \ifx [#2\expandafter\XINT_fladd_opt
2346     \else\expandafter\XINT_fladd_noopt
2347   \fi #1#2%
2348 }%
2349 \def\XINT_fladd_noopt #1#2\xint:#3%
2350 {%
2351   #1[\XINTdigits]%
2352   {\expandafter\XINT_FL_add_a
2353     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{#3}}%
2354 }%
2355 \def\XINT_fladd_opt #1[\xint:#2]#3#4%

```

```

2356 {%
2357   \expandafter\XINT_fladd_opt_a\the\numexpr #2.#1%
2358 }%
2359 \def\XINT_fladd_opt_a #1.#2#3#4%
2360 {%
2361   #2[#1]{\expandafter\XINT_FL_add_a\romannumeral0\XINTinfloat[#1]{#3}#1.{#4}}%
2362 }%
2363 \def\XINT_FL_add_a #1%
2364 {%
2365   \xint_gob_til_zero #1\XINT_FL_add_zero 0\XINT_FL_add_b #1%
2366 }%

2367 \def\XINT_FL_add_zero #1.#2{#2}%[
2368 \def\XINT_FL_add_b #1]#2.#3%
2369 {%
2370   \expandafter\XINT_FL_add_c\romannumeral0\XINTinfloat[#2]{#3}#2.#1%
2371 }%

2372 \def\XINT_FL_add_c #1%
2373 {%
2374   \xint_gob_til_zero #1\XINT_FL_add_zero 0\XINT_FL_add_d #1%
2375 }%

2376 \def\XINT_FL_add_d #1[#2]#3.#4[#5]%
2377 {%
2378   \ifnum\numexpr #2-#3-#5>\xint_c_\xint_dothis\xint_firstoftwo\fi
2379   \ifnum\numexpr #5-#3-#2>\xint_c_\xint_dothis\xint_secondoftwo\fi
2380   \xint_orthat\xintAdd {#1[#2]}{#4[#5]}%
2381 }%

```

8.75 \xintFloatSub, \XINTinFloatSub

First done 1.07.

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

```

2382 \def\xintFloatSub      {\romannumeral0\xintfloatsub }%
2383 \def\xintfloatsub     #1{\XINT_fsub_chkopt \xintfloat #1\xint:}%
2384 \def\XINTinFloatSub   {\romannumeral0\XINTinfloatsub }%
2385 \def\XINTinfloatsub   #1{\XINT_fsub_chkopt \XINTinfloatS #1\xint:}%
2386 \def\XINT_fsub_chkopt #1#2%
2387 {%
2388   \ifx [#2\expandafter\XINT_fsub_opt
2389     \else\expandafter\XINT_fsub_noopt
2390   \fi #1#2%
2391 }%
2392 \def\XINT_fsub_noopt #1#2\xint:#3%
2393 {%
2394   #1[\XINTdigits]%
2395   {\expandafter\XINT_FL_add_a
2396     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.{\xintOpp{#3}}}%
2397 }%

```



```

2398 \def\XINT_flsub_opt #1[\xint:#2]#3#4%
2399 {%
2400   \expandafter\XINT_flsub_opt_a\the\numexpr #2.#1%
2401 }%
2402 \def\XINT_flsub_opt_a #1.#2#3#4%
2403 {%
2404   #2[#1]{\expandafter\XINT_FL_add_a\romannumeral0\XINTinfloat[#1]{#3}#1.{\xintOpp{#4}}}%
2405 }%

```

8.76 \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).

```

2406 \def\xintFloatMul   {\romannumeral0\xintfloatmul   }%
2407 \def\xintfloatmul   #1{\XINT_flmul_chkopt \xintfloat #1\xint:}%
2408 \def\XINTinFloatMul {\romannumeral0\XINTinfloatmul }%
2409 \def\XINTinfloatmul #1{\XINT_flmul_chkopt \XINTinfloatS #1\xint:}%
2410 \def\XINT_flmul_chkopt #1#2%
2411 {%
2412   \ifx [#2\expandafter\XINT_flmul_opt
2413     \else\expandafter\XINT_flmul_noopt
2414   \fi #1#2%
2415 }%
2416 \def\XINT_flmul_noopt #1#2\xint:#3%
2417 {%
2418   #1[\XINTdigits]%
2419   {\expandafter\XINT_FL_mul_a
2420     \romannumeral0\XINTinfloatS[\XINTdigits]{#2}\XINTdigits.{#3}}%
2421 }%
2422 \def\XINT_flmul_opt #1[\xint:#2]#3#4%
2423 {%
2424   \expandafter\XINT_flmul_opt_a\the\numexpr #2.#1%
2425 }%
2426 \def\XINT_flmul_opt_a #1.#2#3#4%
2427 {%
2428   #2[#1]{\expandafter\XINT_FL_mul_a\romannumeral0\XINTinfloatS[#1]{#3}#1.#4}%
2429 }%
2430 \def\XINT_FL_mul_a #1[#2]#3.#4%
2431 {%
2432   \expandafter\XINT_FL_mul_b\romannumeral0\XINTinfloatS[#3]{#4}#1[#2]%
2433 }%
2434 \def\XINT_FL_mul_b #1[#2]#3[#4]{\xintiiMul{#3}{#1}/1[#4+#2]}%

```

8.77 \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.

```
2435 \def\XINTinFloatInv#1{\XINTinFloatS[\XINTdigits]{\xintInv{#1}}}%
```

8.78 \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.

```
2436 \def\xintFloatDiv  {\romannumeral0\xintfloatdiv  }%
2437 \def\xintfloatdiv  #1{\XINT_fldiv_chkopt \xintfloat #1\xint:}%
2438 \def\XINTinFloatDiv {\romannumeral0\XINTinfloatdiv }%
2439 \def\XINTinfloatdiv #1{\XINT_fldiv_chkopt \XINTinfloatS #1\xint:}%
2440 \def\XINT_fldiv_chkopt #1#2%
2441 {%
2442   \ifx [#2\expandafter\XINT_fldiv_opt
2443     \else\expandafter\XINT_fldiv_noopt
2444   \fi  #1#2%
2445 }%

2446 \def\XINT_fldiv_noopt #1#2\xint:#3%
2447 {%
2448   #1[\XINTdigits]%
2449   {\expandafter\XINT_FL_div_a
2450     \romannumeral0\XINTinfloatS[\XINTdigits]{#3}\XINTdigits.{#2}}%
2451 }%
2452 \def\XINT_fldiv_opt #1[\xint:#2]#3#4%
2453 {%
2454   \expandafter\XINT_fldiv_opt_a\the\numexpr #2.#1%
2455 }%

2456 \def\XINT_fldiv_opt_a #1.#2#3#4%
2457 {%
2458   #2[#1]{\expandafter\XINT_FL_div_a\romannumeral0\XINTinfloatS[#1]{#4}#1.#3}}%
2459 }%
2460 \def\XINT_FL_div_a #1[#2]#3.#4%
2461 {%
2462   \expandafter\XINT_FL_div_b\romannumeral0\XINTinfloatS[#3]{#4}/#1e#2%
2463 }%

2464 \def\XINT_FL_div_b #1[#2]{#1e#2}%
```

8.79 \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).

```

2465 \def\xintFloatPow {\romannumeral0\xintfloatpow}%
2466 \def\xintfloatpow #1{\XINT_flpow_chkopt \xintfloat #1\xint:}%
2467 \def\XINTinFloatPow {\romannumeral0\XINTinfloatpow}%
2468 \def\XINTinfloatpow #1{\XINT_flpow_chkopt \XINTinfloatS #1\xint:}%
2469 \def\XINT_flpow_chkopt #1#2%
2470 {%
2471   \ifx [#2\expandafter\XINT_flpow_opt
2472     \else\expandafter\XINT_flpow_noopt
2473   \fi
2474   #1#2%
2475 }%
2476 \def\XINT_flpow_noopt #1#2\xint:#3%
2477 {%
2478   \expandafter\XINT_flpow_checkB_a
2479   \the\numexpr #3.\XINTdigits.{#2}{#1[\XINTdigits]}%
2480 }%
2481 \def\XINT_flpow_opt #1[\xint:#2]%
2482 {%
2483   \expandafter\XINT_flpow_opt_a\the\numexpr #2.#1%
2484 }%
2485 \def\XINT_flpow_opt_a #1.#2#3#4%
2486 {%
2487   \expandafter\XINT_flpow_checkB_a\the\numexpr #4.#1.{#3}{#2[#1]}%
2488 }%
2489 \def\XINT_flpow_checkB_a #1%
2490 {%
2491   \xint_UDzerominusfork
2492   #1-\XINT_flpow_BisZero
2493   0#1{\XINT_flpow_checkB_b -}%
2494   0-{\XINT_flpow_checkB_b }{#1}%
2495   \krof
2496 }%
2497 \def\XINT_flpow_BisZero .#1.#2#3{#3{1[0]}}%

2498 \def\XINT_flpow_checkB_b #1#2.#3.%
2499 {%
2500   \expandafter\XINT_flpow_checkB_c
2501   \the\numexpr\xintLength{#2}+\xint_c_iii.#3.#2.{#1}%
2502 }%
2503 \def\XINT_flpow_checkB_c #1.#2.%
2504 {%
2505   \expandafter\XINT_flpow_checkB_d\the\numexpr#1+#2.#1.#2.%
2506 }%

```

1.2f rounds input to P digits, first.

```

2507 \def\XINT_flpow_checkB_d #1.#2.#3.#4.#5#6%
2508 {%
2509     \expandafter \XINT_flpow_aa
2510     \romannumeral0\XINTinfloat [#3]{#6}{#2}{#1}{#4}{#5}%
2511 }%

2512 \def\XINT_flpow_aa #1[#2]#3%
2513 {%
2514     \expandafter\XINT_flpow_ab\the\numexpr #2-#3\expandafter.%
2515     \romannumeral\XINT_rep #3\endcsname0.#1.%
2516 }%

2517 \def\XINT_flpow_ab #1.#2.#3.{\XINT_flpow_a #3#2[#1]}%

2518 \def\XINT_flpow_a #1%
2519 {%
2520     \xint_UDzerominusfork
2521     #1-\XINT_flpow_zero
2522     0#1{\XINT_flpow_b \iftrue}%
2523     0-{\XINT_flpow_b \iffalse#1}%
2524     \krof
2525 }%
2526 \def\XINT_flpow_zero #1[#2]#3#4#5#6%
2527 {%
2528     #6{\if 1#51\xint_dothis {0[0]}\fi
2529         \xint_orthat
2530         {\XINT_signalcondition{DivisionByZero}{0 to the power #4}{0[0]}}%
2531         }%
2532 }%

2533 \def\XINT_flpow_b #1#2[#3]#4#5%
2534 {%
2535     \XINT_flpow_loopI #5.#3.#2.#4.{#1\ifodd #5 \xint_c_i\fi\fi}%
2536 }%

2537 \def\XINT_flpow_truncate #1.#2.#3.%
2538 {%
2539     \expandafter\XINT_flpow_truncate_a
2540     \romannumeral0\XINT_split_fromleft
2541     #3.#2\xint_bye2345678\xint_bye..#1.#3.%
2542 }%

2543 \def\XINT_flpow_truncate_a #1.#2.#3.{#3+\xintLength{#2}.#1.}%
2544 \def\XINT_flpow_loopI #1.%
2545 {%
2546     \ifnum #1=\xint_c_i\expandafter\XINT_flpow_ItoIII\fi
2547     \ifodd #1
2548         \expandafter\XINT_flpow_loopI_odd

```

```

2549 \else
2550 \expandafter\XINT_flpow_loopI_even
2551 \fi
2552 #1.%
2553 }%

2554 \def\XINT_flpow_ItoIII\ifodd #1\fi #2.#3.#4.#5.#6%
2555 {%
2556 \expandafter\XINT_flpow_III\the\numexpr #6+\xint_c_.#3.#4.#5.%
2557 }%

2558 \def\XINT_flpow_loopI_even #1.#2.#3.%#4.%
2559 {%
2560 \expandafter\XINT_flpow_loopI
2561 \the\numexpr #1/\xint_c_ii\expandafter.%
2562 \the\numexpr\expandafter\XINT_flpow_truncate
2563 \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.%
2564 }%
2565 \def\XINT_flpow_loopI_odd #1.#2.#3.#4.%
2566 {%
2567 \expandafter\XINT_flpow_loopII
2568 \the\numexpr #1/\xint_c_ii-\xint_c_i\expandafter.%
2569 \the\numexpr\expandafter\XINT_flpow_truncate
2570 \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#2.#3.%
2571 }%
2572 \def\XINT_flpow_loopII #1.%
2573 {%
2574 \ifnum #1 = \xint_c_i\expandafter\XINT_flpow_ItoIII\fi
2575 \ifodd #1
2576 \expandafter\XINT_flpow_loopII_odd
2577 \else
2578 \expandafter\XINT_flpow_loopII_even
2579 \fi
2580 #1.%
2581 }%
2582 \def\XINT_flpow_loopII_even #1.#2.#3.%#4.%
2583 {%
2584 \expandafter\XINT_flpow_loopII
2585 \the\numexpr #1/\xint_c_ii\expandafter.%
2586 \the\numexpr\expandafter\XINT_flpow_truncate
2587 \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.%
2588 }%
2589 \def\XINT_flpow_loopII_odd #1.#2.#3.#4.#5.#6.%
2590 {%
2591 \expandafter\XINT_flpow_loopII_odda
2592 \the\numexpr\expandafter\XINT_flpow_truncate
2593 \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2594 #1.#2.#3.%
2595 }%
2596 \def\XINT_flpow_loopII_odda #1.#2.#3.#4.#5.#6.%
2597 {%
2598 \expandafter\XINT_flpow_loopII

```

```

2599 \the\numexpr #4/\xint_c_ii-\xint_c_i\expandafter.%
2600 \the\numexpr\expandafter\XINT_flpow_truncate
2601 \the\numexpr\xint_c_ii*#5\expandafter.\romannumeral0\xintiisqr{#6}.#3.%
2602 #1.#2.%
2603 }%

```

```

2604 \def\XINT_flpow_IItoIII\ifodd #1\fi #2.#3.#4.#5.#6.#7.#8%
2605 {%
2606 \expandafter\XINT_flpow_III\the\numexpr #8+\xint_c_\expandafter.%
2607 \the\numexpr\expandafter\XINT_flpow_truncate
2608 \the\numexpr#3+#6\expandafter.\romannumeral0\xintiimul{#4}{#7}.#5.%
2609 }%

```

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).

```

2610 \def\XINT_flpow_III #1.#2.#3.#4.#5%
2611 {%
2612 \expandafter\XINT_flpow_IIIend
2613 \xint_UDsignfork
2614 #5{{1/#3[-#2]}}%
2615 -{{#3[#2]}}%
2616 \krof #1%
2617 }%

```

```

2618 \def\XINT_flpow_IIIend #1#2#3%
2619 {#3{\if#21\xint_afterfi{\expandafter-\romannumeral`&&@\fi#1}}%

```

8.80 `\xintFloatPower`, `\XINTinFloatPower`

1.07. The core loop has been re-organized in 1.09j for some slight efficiency gain. The exponent B is given to `\xintNum`. The `^` in expressions is mapped to this routine.

Same modifications as in `\xintFloatPow` for 1.2f.

1.2f adds a special private macro for allowing half-integral exponents for use with `^` within `\xintfloatexpr`. The exponent will be first truncated to either an integer or an half-integer. The macro is not for general use.

1.2k does anew this 1.2f handling of half-integer exponents for the `\xintfloatexpr` parser: with 1.2f's code the final square-root extraction was applied to a value already rounded to the target precision, unneedlessly losing precision.

```

2620 \def\xintFloatPower    {\romannumeral0\xintfloatpower}%
2621 \def\xintfloatpower #1{\XINT_flpower_chkopt \xintfloat #1\xint:}%
2622 \def\XINTinFloatPower  {\romannumeral0\XINTinfloatpower }%
2623 \def\XINTinfloatpower #1{\XINT_flpower_chkopt \XINTinfloatS #1\xint:}%

```

First the special macro for use by the expression parser which checks if one raises to an half-integer exponent. This is always with `\XINTdigits` precision. Rewritten for 1.2k in order for the final square root to keep three guard digits.

We have to be careful that exponent #2 is not constrained by TeX bound. And we must allow fractions. The 1.2k variant does a rounding to nearest integer of half-integer, 1.2f did a truncation rather (this is done after truncation of #2 to fixed point with one digit after mark.) We try to recognize quickly the case of integer exponent, for speed, but there is overhead of going through `\xintiTrunc1`.

```

2624 \def\XINTinFloatPowerH {\romannumeral0\XINTinfloatpowerh }%

```

```

2625 \def\XINTinfloatpowerh #1#2%
2626 {%
2627   \expandafter\XINT_flpowerh_a\romannumeral0\xintitrunc1{#2};%
2628   \XINTdigits.{#1}\XINTinfloatS[\XINTdigits]}%
2629 }%

```

```

2630 \def\XINT_flpowerh_a #1;%
2631 {%
2632   \if0\xintLDg{#1}\expandafter\XINT_flpowerh_int
2633   \else\expandafter\XINT_flpowerh_b
2634   \fi #1.%
2635 }%
2636 \def\XINT_flpowerh_int #1%
2637 {%
2638   \if0#1\expandafter\XINT_flpower_BisZero
2639   \else\expandafter\XINT_flpowerh_i
2640   \fi #1%
2641 }%
2642 \def\XINT_flpowerh_i #10.{\expandafter\XINT_flpower_checkB_a#1.}%
2643 \def\XINT_flpowerh_b #1.%
2644 {%
2645   \expandafter\XINT_flpowerh_c\romannumeral0\xintdsrr{\xintDouble{#1}}.%
2646 }%
2647 \def\XINT_flpowerh_c #1.%
2648 {%
2649   \ifodd\xintLDg{#1} %<- intentional space
2650   \expandafter\XINT_flpowerh_d\else\expandafter\XINT_flpowerh_e
2651   \fi #1.%
2652 }%
2653 \def\XINT_flpowerh_d #1.\XINTdigits.#2#3%
2654 {%
2655   \XINT_flpower_checkB_a #1.\XINTdigits.{#2}\XINT_flpowerh_finish
2656 }%
2657 \def\XINT_flpowerh_finish #1%
2658   {\XINTinfloatS[\XINTdigits]{\XINTinFloatSqrt[\XINTdigits+\xint_c_iii]{#1}}}%
2659 \def\XINT_flpowerh_e #1.%
2660   {\expandafter\XINT_flpower_checkB_a\romannumeral0\xinthalff{#1}.}%

```

Start of macro. Check for optional argument.

```

2661 \def\XINT_flpower_chkopt #1#2%
2662 {%
2663   \ifx [#2\expandafter\XINT_flpower_opt
2664     \else\expandafter\XINT_flpower_noopt
2665   \fi
2666   #1#2%
2667 }%
2668 \def\XINT_flpower_noopt #1#2\xint:#3%
2669 {%
2670   \expandafter\XINT_flpower_checkB_a
2671   \romannumeral0\xintnum{#3}.\XINTdigits.{#2}{#1[\XINTdigits]}%
2672 }%
2673 \def\XINT_flpower_opt #1[\xint:#2]%
2674 {%
2675   \expandafter\XINT_flpower_opt_a\the\numexpr #2.#1%
2676 }%
2677 \def\XINT_flpower_opt_a #1.#2#3#4%
2678 {%
2679   \expandafter\XINT_flpower_checkB_a
2680   \romannumeral0\xintnum{#4}.#1.{#3}{#2[#1]}%
2681 }%
2682 \def\XINT_flpower_checkB_a #1%
2683 {%
2684   \xint_UDzerominusfork
2685   #1-{\XINT_flpower_BisZero 0}%
2686   0#1{\XINT_flpower_checkB_b -}%
2687   0-{\XINT_flpower_checkB_b }#1}%
2688   \krof
2689 }%
2690 \def\XINT_flpower_BisZero 0.#1.#2#3{#3{1[0]}}%
2691 \def\XINT_flpower_checkB_b #1#2.#3.%
2692 {%
2693   \expandafter\XINT_flpower_checkB_c
2694   \the\numexpr\xintLength{#2}+\xint_c_iii.#3.#2.{#1}%
2695 }%

2696 \def\XINT_flpower_checkB_c #1.#2.%
2697 {%
2698   \expandafter\XINT_flpower_checkB_d\the\numexpr#1+#2.#1.#2.%
2699 }%

2700 \def\XINT_flpower_checkB_d #1.#2.#3.#4.#5#6%
2701 {%
2702   \expandafter \XINT_flpower_aa
2703   \romannumeral0\XINTinfloat [#3]{#6}{#2}{#1}{#4}{#5}%
2704 }%

2705 \def\XINT_flpower_aa #1[#2]#3%
2706 {%

```



```

2707 \expandafter\XINT_flpower_ab\the\numexpr #2-#3\expandafter.%
2708 \romannumeral\XINT_rep #3\endcsname0.#1.%
2709 }%
2710 \def\XINT_flpower_ab #1.#2.#3.{\XINT_flpower_a #3#2[#1]}%
2711 \def\XINT_flpower_a #1%
2712 {%
2713 \xint_UDzerominusfork
2714 #1-\XINT_flpow_zero
2715 0#1{\XINT_flpower_b \iftrue}%
2716 0-{\XINT_flpower_b \iffalse#1}%
2717 \krof
2718 }%
2719 \def\XINT_flpower_b #1#2[#3]#4#5%
2720 {%
2721 \XINT_flpower_loopI #5.#3.#2.#4.{#1\xintiOdd{#5}\fi}%
2722 }%
2723 \def\XINT_flpower_loopI #1.%
2724 {%
2725 \if1\XINT_isOne {#1}\xint_dothis\XINT_flpower_ItoIII\fi
2726 \ifodd\xintLDg{#1} %<- intentional space
2727 \xint_dothis{\expandafter\XINT_flpower_loopI_odd}\fi
2728 \xint_orthat{\expandafter\XINT_flpower_loopI_even}%

2729 \romannumeral0\XINT_half
2730 #1\xint_bye\xint_Bye345678\xint_bye
2731 *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax.%
2732 }%
2733 \def\XINT_flpower_ItoIII #1.#2.#3.#4.#5%
2734 {%
2735 \expandafter\XINT_flpow_III\the\numexpr #5+\xint_c_.#2.#3.#4.%
2736 }%
2737 \def\XINT_flpower_loopI_even #1.#2.#3.#4.%
2738 {%
2739 \expandafter\XINT_flpower_toloopI
2740 \the\numexpr\expandafter\XINT_flpow_truncate
2741 \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#1.%
2742 }%
2743 \def\XINT_flpower_toloopI #1.#2.#3.#4.{\XINT_flpower_loopI #4.#1.#2.#3.}%
2744 \def\XINT_flpower_loopI_odd #1.#2.#3.#4.%
2745 {%
2746 \expandafter\XINT_flpower_toloopII
2747 \the\numexpr\expandafter\XINT_flpow_truncate
2748 \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.%
2749 #1.#2.#3.%
2750 }%
2751 \def\XINT_flpower_toloopII #1.#2.#3.#4.{\XINT_flpower_loopII #4.#1.#2.#3.}%
2752 \def\XINT_flpower_loopII #1.%
2753 {%
2754 \if1\XINT_isOne{#1}\xint_dothis\XINT_flpower_ItoIII\fi
2755 \ifodd\xintLDg{#1} %<- intentional space
2756 \xint_dothis{\expandafter\XINT_flpower_loopII_odd}\fi
2757 \xint_orthat{\expandafter\XINT_flpower_loopII_even}%

```

```

2758 \romannumeral0\XINT_half#1\xint_bye\xint_Bye345678\xint_bye
2759 *\xint_c_v+\xint_c_v)/\xint_c_x-\xint_c_i\relax.%
2760 }%
2761 \def\XINT_flpower_loopII_even #1.#2.#3.#4.%
2762 {%
2763 \expandafter\XINT_flpower_toloopII
2764 \the\numexpr\expandafter\XINT_flpow_truncate
2765 \the\numexpr\xint_c_ii*#2\expandafter.\romannumeral0\xintiisqr{#3}.#4.#1.%
2766 }%
2767 \def\XINT_flpower_loopII_odd #1.#2.#3.#4.#5.#6.%
2768 {%
2769 \expandafter\XINT_flpower_loopII_odda
2770 \the\numexpr\expandafter\XINT_flpow_truncate
2771 \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2772 #1.#2.#3.%
2773 }%
2774 \def\XINT_flpower_loopII_odda #1.#2.#3.#4.#5.#6.%
2775 {%
2776 \expandafter\XINT_flpower_toloopII
2777 \the\numexpr\expandafter\XINT_flpow_truncate
2778 \the\numexpr\xint_c_ii*#5\expandafter.\romannumeral0\xintiisqr{#6}.#3.%
2779 #4.#1.#2.%
2780 }%
2781 \def\XINT_flpower_IItoIII #1.#2.#3.#4.#5.#6.#7%
2782 {%
2783 \expandafter\XINT_flpow_III\the\numexpr #7+\xint_c_\expandafter.%
2784 \the\numexpr\expandafter\XINT_flpow_truncate
2785 \the\numexpr#2+#5\expandafter.\romannumeral0\xintiimul{#3}{#6}.#4.%
2786 }%

```

8.81 \xintFloatFac, \XINTFloatFac

Done at 1.2. At 1.3e \XINTinFloatFac outputs using \XINTinFloatS.

```

2787 \def\xintFloatFac {\romannumeral0\xintfloatfac}%
2788 \def\xintfloatfac #1{\XINT_flfac_chkopt \xintfloat #1\xint:}%
2789 \def\XINTinFloatFac {\romannumeral0\XINTinfloatfac }%
2790 \def\XINTinfloatfac #1{\XINT_flfac_chkopt \XINTinfloatS #1\xint:}%
2791 \def\XINT_flfac_chkopt #1#2%
2792 {%
2793 \ifx [#2\expandafter\XINT_flfac_opt
2794 \else\expandafter\XINT_flfac_noopt
2795 \fi
2796 #1#2%
2797 }%
2798 \def\XINT_flfac_noopt #1#2\xint:
2799 {%
2800 \expandafter\XINT_FL_fac_fork_a
2801 \the\numexpr \xintNum{#2}.\xint_c_i \XINTdigits\XINT_FL_fac_out{#1[\XINTdigits]}%
2802 }%
2803 \def\XINT_flfac_opt #1[\xint:#2]%
2804 {%
2805 \expandafter\XINT_flfac_opt_a\the\numexpr #2.#1%

```

```

2806 }%
2807 \def\XINT_flfac_opt_a #1.#2#3%
2808 {%
2809   \expandafter\XINT_FL_fac_fork_a\the\numexpr \xintNum{#3}.\xint_c_i {#1}\XINT_FL_fac_out{#2[#1]}%
2810 }%
2811 \def\XINT_FL_fac_fork_a #1%
2812 {%
2813   \xint_UDzerominusfork
2814   #1-\XINT_FL_fac_iszero
2815   0#1\XINT_FL_fac_isneg
2816   0-\XINT_FL_fac_fork_b #1}%
2817 \krof
2818 }%
2819 \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.

```

2820 \def\XINT_FL_fac_isneg #1.#2#3#4#5%
2821 {%
2822   #5{\XINT_signalcondition{InvalidOperation}
2823     {Factorial of negative: (-#1)!}}{0[0]}}%
2824 }%
2825 \def\XINT_FL_fac_fork_b #1.%
2826 {%
2827   \ifnum #1>\xint_c_x^viii_mone\xint_dothis\XINT_FL_fac_toobig\fi
2828   \ifnum #1>\xint_c_x^iv\xint_dothis\XINT_FL_fac_vbig \fi
2829   \ifnum #1>465 \xint_dothis\XINT_FL_fac_big\fi
2830   \ifnum #1>101 \xint_dothis\XINT_FL_fac_med\fi
2831   \xint_orthat\XINT_FL_fac_small
2832   #1.%
2833 }%
2834 \def\XINT_FL_fac_toobig #1.#2#3#4#5%
2835 {%
2836   #5{\XINT_signalcondition{InvalidOperation}
2837     {Factorial of too big: (#1)!}}{0[0]}}%
2838 }%

```

Computations are done with Q blocks of eight digits. When a multiplication has a carry, hence creates Q+1 blocks, the least significant one is dropped. The goal is to compute an approximate value X' to the exact value X, such that the final relative error (X-X')/X will be at most 10^{-P} with P the desired precision. Then, when we round X' to X'' with P significant digits, we can prove that the absolute error |X-X''| is bounded (strictly) by 0.6 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<=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 ceil((P+k+n)/8). Using \numexpr rounding division (ARRRRRGGGHHHH), if m is a positive integer, ceil(m/8) can be computed as (m+3)/8. Thus with m=P+10+k, this gives Q<-(P+13+k)/8. The routine actually computes 8(Q-1) for use in \XINT_FL_fac_addzeros.

With 1.2f the formula is m=P+9+k, Q<-(P+12+k)/8, and we use now 4=12-8 rather than the earlier 5=13-8. Whatever happens, the value computed in \XINT_FL_fac_increaseP is at least 8. There will

always be an extra block.

Note: with Digits:=32; Maple gives for 200!:

```
> factorial(200.);
```

```
375
```

```
0.78865786736479050355236321393218 10
```

My 1.2f routine (and also 1.2) outputs:

```
7.8865786736479050355236321393219e374
```

and this is the correct rounding because for 40 digits it computes

```
7.886578673647905035523632139321850622951e374
```

Maple's result (contrarily to xint) is thus not the correct rounding but still it is less than 0.6 ulp wrong.

```
2839 \def\XINT_FL_fac_vbig
2840   {\expandafter\XINT_FL_fac_vbigloop_a
2841     \the\numexpr \XINT_FL_fac_increaseP \xint_c_i   }%
2842 \def\XINT_FL_fac_big
2843   {\expandafter\XINT_FL_fac_bigloop_a
2844     \the\numexpr \XINT_FL_fac_increaseP \xint_c_ii  }%
2845 \def\XINT_FL_fac_med
2846   {\expandafter\XINT_FL_fac_medloop_a
2847     \the\numexpr \XINT_FL_fac_increaseP \xint_c_iii }%
2848 \def\XINT_FL_fac_small
2849   {\expandafter\XINT_FL_fac_smallloop_a
2850     \the\numexpr \XINT_FL_fac_increaseP \xint_c_iv  }%
2851 \def\XINT_FL_fac_increaseP #1#2.#3#4%
2852 {%
2853   #2\expandafter.\the\numexpr\xint_c_viii*%
2854   ((\xint_c_iv+#4+\expandafter\XINT_FL_fac_countdigits
2855     \the\numexpr #2/((#1*#3)\relax 87654321\Z)/\xint_c_viii).%
2856 }%
2857 \def\XINT_FL_fac_countdigits #1#2#3#4#5#6#7#8{\XINT_FL_fac_countdone }%
2858 \def\XINT_FL_fac_countdone  #1#2\Z {#1}%
2859 \def\XINT_FL_fac_out #1;![#2]#3%
2860   {#3{\romannumeral0\XINT_mul_out
2861     #1;!1\R!1\R!1\R!1\R!1\R!%
2862     1\R!1\R!1\R!1\R!1\R!\W [#2]}}%
2863 \def\XINT_FL_fac_vbigloop_a #1.#2.%
2864 {%
2865   \XINT_FL_fac_bigloop_a \xint_c_x^iv.#2.%
2866   {\expandafter\XINT_FL_fac_vbigloop_loop\the\numexpr 100010001\expandafter.%
2867     \the\numexpr \xint_c_x^viii+#1.}%
2868 }%
2869 \def\XINT_FL_fac_vbigloop_loop #1.#2.%
2870 {%
2871   \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2872   \expandafter\XINT_FL_fac_vbigloop_loop
2873   \the\numexpr #1+\xint_c_i\expandafter.%
2874   \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_mul #1!%
2875 }%
2876 \def\XINT_FL_fac_bigloop_a #1.%
2877 {%
2878   \expandafter\XINT_FL_fac_bigloop_b \the\numexpr
2879   #1+\xint_c_i-\xint_c_ii*((#1-464)/\xint_c_ii).#1.%
```

```

2880 }%
2881 \def\XINT_FL_fac_bigloop_b #1.#2.#3.%
2882 {%
2883   \expandafter\XINT_FL_fac_medloop_a
2884     \the\numexpr #1-\xint_c_i.#3.{\XINT_FL_fac_bigloop_loop #1.#2.}%
2885 }%
2886 \def\XINT_FL_fac_bigloop_loop #1.#2.%
2887 {%
2888   \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2889   \expandafter\XINT_FL_fac_bigloop_loop
2890     \the\numexpr #1+\xint_c_ii\expandafter.%
2891     \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_bigloop_mul #1!%
2892 }%
2893 \def\XINT_FL_fac_bigloop_mul #1!%
2894 {%
2895   \expandafter\XINT_FL_fac_mul
2896     \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
2897 }%
2898 \def\XINT_FL_fac_medloop_a #1.%
2899 {%
2900   \expandafter\XINT_FL_fac_medloop_b
2901     \the\numexpr #1+\xint_c_i-\xint_c_iii*((#1-100)/\xint_c_iii).#1.%
2902 }%
2903 \def\XINT_FL_fac_medloop_b #1.#2.#3.%
2904 {%
2905   \expandafter\XINT_FL_fac_smallloop_a
2906     \the\numexpr #1-\xint_c_i.#3.{\XINT_FL_fac_medloop_loop #1.#2.}%
2907 }%
2908 \def\XINT_FL_fac_medloop_loop #1.#2.%
2909 {%
2910   \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2911   \expandafter\XINT_FL_fac_medloop_loop
2912     \the\numexpr #1+\xint_c_iii\expandafter.%
2913     \the\numexpr #2\expandafter.\the\numexpr\XINT_FL_fac_medloop_mul #1!%
2914 }%
2915 \def\XINT_FL_fac_medloop_mul #1!%
2916 {%
2917   \expandafter\XINT_FL_fac_mul
2918     \the\numexpr
2919       \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
2920 }%
2921 \def\XINT_FL_fac_smallloop_a #1.%
2922 {%
2923   \csname
2924     XINT_FL_fac_smallloop_\the\numexpr #1-\xint_c_iv*(#1/\xint_c_iv)\relax
2925   \endcsname #1.%
2926 }%
2927 \expandafter\def\csname XINT_FL_fac_smallloop_1\endcsname #1.#2.%
2928 {%
2929   \XINT_FL_fac_addzeros #2.100000001!.{2.#1.}{#2}%
2930 }%
2931 \expandafter\def\csname XINT_FL_fac_smallloop_-2\endcsname #1.#2.%

```

```

2932 {%
2933   \XINT_FL_fac_addzeros #2.100000002!.{3.#1.}{#2}%
2934 }%
2935 \expandafter\def\csname XINT_FL_fac_smallloop_1\endcsname #1.#2.%
2936 {%
2937   \XINT_FL_fac_addzeros #2.100000006!.{4.#1.}{#2}%
2938 }%
2939 \expandafter\def\csname XINT_FL_fac_smallloop_0\endcsname #1.#2.%
2940 {%
2941   \XINT_FL_fac_addzeros #2.100000024!.{5.#1.}{#2}%
2942 }%
2943 \def\XINT_FL_fac_addzeros #1.%
2944 {%
2945   \ifnum #1=\xint_c_viii \expandafter\XINT_FL_fac_addzeros_exit\fi
2946   \expandafter\XINT_FL_fac_addzeros
2947   \the\numexpr #1-\xint_c_viii.100000000!%
2948 }%

```

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.

```

2949 \def\XINT_FL_fac_addzeros_exit #1.#2.#3#4{\XINT_FL_fac_smallloop_loop #3#21;![-#4]}%
2950 \def\XINT_FL_fac_smallloop_loop #1.#2.%
2951 {%
2952   \ifnum #1>#2 \expandafter\XINT_FL_fac_loop_exit\fi
2953   \expandafter\XINT_FL_fac_smallloop_loop
2954   \the\numexpr #1+\xint_c_iv\expandafter.%
2955   \the\numexpr #2\expandafter.\romannumeral0\XINT_FL_fac_smallloop_mul #1!%
2956 }%
2957 \def\XINT_FL_fac_smallloop_mul #1!%
2958 {%
2959   \expandafter\XINT_FL_fac_mul
2960   \the\numexpr
2961     \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
2962 }%[[
2963 \def\XINT_FL_fac_loop_exit #1!#2]#3{#3#2]}%
2964 \def\XINT_FL_fac_mul 1#1!%
2965   {\expandafter\XINT_FL_fac_mul_a\the\numexpr\XINT_FL_fac_smallmul 10!{#1]}%
2966 \def\XINT_FL_fac_mul_a #1-#2%
2967 {%
2968   \if#21\xint_afterfi{\expandafter\space\xint_gob_til_exclam}\else
2969   \expandafter\space\fi #11;!%
2970 }%
2971 \def\XINT_FL_fac_minimulwc_a #1#2#3#4#5!#6#7#8#9%
2972 {%
2973   \XINT_FL_fac_minimulwc_b {#1#2#3#4}{#5}{#6#7#8#9}%
2974 }%
2975 \def\XINT_FL_fac_minimulwc_b #1#2#3#4!#5%
2976 {%
2977   \expandafter\XINT_FL_fac_minimulwc_c
2978   \the\numexpr \xint_c_x^ix+#5+#2*#4!{#1}{#2}{#3}{#4}%
2979 }%

```

```

2980 \def\XINT_FL_fac_minimulwc_c 1#1#2#3#4#5#6!#7%
2981 {%
2982   \expandafter\XINT_FL_fac_minimulwc_d {#1#2#3#4#5}#7{#6}%
2983 }%
2984 \def\XINT_FL_fac_minimulwc_d #1#2#3#4#5%
2985 {%
2986   \expandafter\XINT_FL_fac_minimulwc_e
2987   \the\numexpr \xint_c_x^ix+#1+#2*#5+#3*#4!{#2}{#4}%
2988 }%
2989 \def\XINT_FL_fac_minimulwc_e 1#1#2#3#4#5#6!#7#8#9%
2990 {%
2991   1#6#9\expandafter!%
2992   \the\numexpr\expandafter\XINT_FL_fac_smallmul
2993   \the\numexpr \xint_c_x^viii+#1#2#3#4#5+#7*#8!%
2994 }%
2995 \def\XINT_FL_fac_smallmul 1#1!#21#3!%
2996 {%
2997   \xint_gob_til_sc #3\XINT_FL_fac_smallmul_end;%
2998   \XINT_FL_fac_minimulwc_a #2!#3!{#1}{#2}%
2999 }%

```

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.)

```

3000 \def\XINT_FL_fac_smallmul_end;\XINT_FL_fac_minimulwc_a #1!;!#2#3[#4]%
3001 {%
3002   \ifnum #2=\xint_c_
3003     \expandafter\xint_firstoftwo\else
3004     \expandafter\xint_secondoftwo
3005   \fi
3006   {-2\relax[#4]}%
3007   {1#2\expandafter!\expandafter-\expandafter1\expandafter
3008     [\the\numexpr #4+\xint_c_viii]}%
3009 }%

```

8.82 `\xintFloatPFactorial`, `\XINTinFloatPFactorial`

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

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

```

3010 \def\xintFloatPFactorial {\romannumeral0\xintfloatpfactorial}%
3011 \def\xintfloatpfactorial #1{\XINT_flpfac_chkopt \xintfloat #1\xint:}%
3012 \def\XINTinFloatPFactorial {\romannumeral0\XINTinfloatpfactorial}%
3013 \def\XINTinfloatpfactorial #1{\XINT_flpfac_chkopt \XINTinfloat #1\xint:}%
3014 \def\XINT_flpfac_chkopt #1#2%
3015 {%
3016   \ifx [#2\expandafter\XINT_flpfac_opt
3017     \else\expandafter\XINT_flpfac_noopt
3018   \fi

```

```

3019     #1#2%
3020 }%
3021 \def\XINT_flpfac_noopt #1#2\xint:#3%
3022 {%
3023     \expandafter\XINT_FL_pfac_fork
3024     \the\numexpr \xintNum{#2}\expandafter.%
3025     \the\numexpr \xintNum{#3}.\xint_c_i{\XINTdigits}{#1[\XINTdigits]]}%
3026 }%
3027 \def\XINT_flpfac_opt #1[\xint:#2]%
3028 {%
3029     \expandafter\XINT_flpfac_opt_b\the\numexpr #2.#1%
3030 }%
3031 \def\XINT_flpfac_opt_b #1.#2#3#4%
3032 {%
3033     \expandafter\XINT_FL_pfac_fork
3034     \the\numexpr \xintNum{#3}\expandafter.%
3035     \the\numexpr \xintNum{#4}.\xint_c_i{#1}{#2[#1]}%
3036 }%
3037 \def\XINT_FL_pfac_fork #1#2.#3#4.%
3038 {%
3039     \unless\ifnum #1#2<#3#4 \xint_dothis\XINT_FL_pfac_one\fi
3040     \if-#3\xint_dothis\XINT_FL_pfac_neg \fi
3041     \if-#1\xint_dothis\XINT_FL_pfac_zero\fi
3042     \ifnum #3#4>\xint_c_x^viii_mone\xint_dothis\XINT_FL_pfac_outofrange\fi
3043     \xint_orthat \XINT_FL_pfac_increaseP #1#2.#3#4.%
3044 }%
3045 \def\XINT_FL_pfac_outofrange #1.#2.#3#4#5%
3046 {%
3047     #5{\XINT_signalcondition{InvalidOperation}}
3048     {pfactorial second arg too big: 9999999 < #2}{0[0]}%
3049 }%
3050 \def\XINT_FL_pfac_one #1.#2.#3#4#5{#5{1[0]}}%
3051 \def\XINT_FL_pfac_zero #1.#2.#3#4#5{#5{0[0]}}%
3052 \def\XINT_FL_pfac_neg -#1.-#2.%
3053 {%
3054     \ifnum #1>\xint_c_x^viii\xint_dothis\XINT_FL_pfac_outofrange\fi
3055     \xint_orthat {%
3056         \ifodd\numexpr#2-#1\relax\xint_afterfi{\expandafter-\romannumeral`&&@}\fi
3057         \expandafter\XINT_FL_pfac_increaseP}%
3058     \the\numexpr #2-\xint_c_i\expandafter.\the\numexpr#1-\xint_c_i.%
3059 }%

```

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

```

3060 \def\XINT_FL_pfac_increaseP #1.#2.#3#4%
3061 {%
3062     \expandafter\XINT_FL_pfac_a

```



```

3063 \the\numexpr \xint_c_viii*((\xint_c_iv+#4+\expandafter
3064 \XINT_FL_fac_countdigits\the\numexpr (#2-#1-\xint_c_i)%
3065 /\ifnum #2>\xint_c_x^iv #3\else(#3*\xint_c_ii)\fi\relax
3066 87654321\Z)/\xint_c_viii).#1.#2.%
3067 }%
3068 \def\XINT_FL_pfac_a #1.#2.#3.%
3069 {%
3070 \expandafter\XINT_FL_pfac_b\the\numexpr \xint_c_i+#2\expandafter.%
3071 \the\numexpr#3\expandafter.%
3072 \romannumeral0\XINT_FL_pfac_addzeroes #1.100000001!1;![-#1]%
3073 }%
3074 \def\XINT_FL_pfac_addzeroes #1.%
3075 {%
3076 \ifnum #1=\xint_c_viii \expandafter\XINT_FL_pfac_addzeroes_exit\fi
3077 \expandafter\XINT_FL_pfac_addzeroes\the\numexpr #1-\xint_c_viii.100000000!%
3078 }%
3079 \def\XINT_FL_pfac_addzeroes_exit #1.{ }%
3080 \def\XINT_FL_pfac_b #1.%
3081 {%
3082 \ifnum #1>9999 \xint_dothis\XINT_FL_pfac_vbigloop \fi
3083 \ifnum #1>463 \xint_dothis\XINT_FL_pfac_bigloop \fi
3084 \ifnum #1>98 \xint_dothis\XINT_FL_pfac_medloop \fi
3085 \xint_orthat\XINT_FL_pfac_smallloop #1.%
3086 }%
3087 \def\XINT_FL_pfac_smallloop #1.#2.%
3088 {%
3089 \ifcase\numexpr #2-#1\relax
3090 \expandafter\XINT_FL_pfac_end_
3091 \or \expandafter\XINT_FL_pfac_end_i
3092 \or \expandafter\XINT_FL_pfac_end_ii
3093 \or \expandafter\XINT_FL_pfac_end_iii
3094 \else\expandafter\XINT_FL_pfac_smallloop_a
3095 \fi #1.#2.%
3096 }%
3097 \def\XINT_FL_pfac_smallloop_a #1.#2.%
3098 {%
3099 \expandafter\XINT_FL_pfac_smallloop_b
3100 \the\numexpr #1+\xint_c_iv\expandafter.%
3101 \the\numexpr #2\expandafter.%
3102 \romannumeral0\expandafter\XINT_FL_fac_mul
3103 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
3104 }%
3105 \def\XINT_FL_pfac_smallloop_b #1.%
3106 {%
3107 \ifnum #1>98 \expandafter\XINT_FL_pfac_medloop \else
3108 \expandafter\XINT_FL_pfac_smallloop \fi #1.%
3109 }%
3110 \def\XINT_FL_pfac_medloop #1.#2.%
3111 {%
3112 \ifcase\numexpr #2-#1\relax
3113 \expandafter\XINT_FL_pfac_end_
3114 \or \expandafter\XINT_FL_pfac_end_i

```

```

3115 \or \expandafter\XINT_FL_pfac_end_ii
3116 \else\expandafter\XINT_FL_pfac_medloop_a
3117 \fi #1.#2.%
3118 }%
3119 \def\XINT_FL_pfac_medloop_a #1.#2.%
3120 {%
3121 \expandafter\XINT_FL_pfac_medloop_b
3122 \the\numexpr #1+\xint_c_iii\expandafter.%
3123 \the\numexpr #2\expandafter.%
3124 \romannumeral0\expandafter\XINT_FL_fac_mul
3125 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
3126 }%
3127 \def\XINT_FL_pfac_medloop_b #1.%
3128 {%
3129 \ifnum #1>463 \expandafter\XINT_FL_pfac_bigloop \else
3130 \expandafter\XINT_FL_pfac_medloop \fi #1.%
3131 }%
3132 \def\XINT_FL_pfac_bigloop #1.#2.%
3133 {%
3134 \ifcase\numexpr #2-#1\relax
3135 \expandafter\XINT_FL_pfac_end_
3136 \or \expandafter\XINT_FL_pfac_end_i
3137 \else\expandafter\XINT_FL_pfac_bigloop_a
3138 \fi #1.#2.%
3139 }%
3140 \def\XINT_FL_pfac_bigloop_a #1.#2.%
3141 {%
3142 \expandafter\XINT_FL_pfac_bigloop_b
3143 \the\numexpr #1+\xint_c_ii\expandafter.%
3144 \the\numexpr #2\expandafter.%
3145 \romannumeral0\expandafter\XINT_FL_fac_mul
3146 \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
3147 }%
3148 \def\XINT_FL_pfac_bigloop_b #1.%
3149 {%
3150 \ifnum #1>9999 \expandafter\XINT_FL_pfac_vbigloop \else
3151 \expandafter\XINT_FL_pfac_bigloop \fi #1.%
3152 }%
3153 \def\XINT_FL_pfac_vbigloop #1.#2.%
3154 {%
3155 \ifnum #2=#1
3156 \expandafter\XINT_FL_pfac_end_
3157 \else\expandafter\XINT_FL_pfac_vbigloop_a
3158 \fi #1.#2.%
3159 }%
3160 \def\XINT_FL_pfac_vbigloop_a #1.#2.%
3161 {%
3162 \expandafter\XINT_FL_pfac_vbigloop
3163 \the\numexpr #1+\xint_c_i\expandafter.%
3164 \the\numexpr #2\expandafter.%
3165 \romannumeral0\expandafter\XINT_FL_fac_mul
3166 \the\numexpr\xint_c_x^viii+#1!%

```

```

3167 }%
3168 \def\XINT_FL_pfac_end_iii #1.#2.%
3169 {%
3170   \expandafter\XINT_FL_fac_out
3171   \romannumeral0\expandafter\XINT_FL_fac_mul
3172   \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)*(#1+\xint_c_iii)!%
3173 }%
3174 \def\XINT_FL_pfac_end_ii #1.#2.%
3175 {%
3176   \expandafter\XINT_FL_fac_out
3177   \romannumeral0\expandafter\XINT_FL_fac_mul
3178   \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)*(#1+\xint_c_ii)!%
3179 }%
3180 \def\XINT_FL_pfac_end_i #1.#2.%
3181 {%
3182   \expandafter\XINT_FL_fac_out
3183   \romannumeral0\expandafter\XINT_FL_fac_mul
3184   \the\numexpr \xint_c_x^viii+#1*(#1+\xint_c_i)!%
3185 }%
3186 \def\XINT_FL_pfac_end_ #1.#2.%
3187 {%
3188   \expandafter\XINT_FL_fac_out
3189   \romannumeral0\expandafter\XINT_FL_fac_mul
3190   \the\numexpr \xint_c_x^viii+#1!%
3191 }%

```

8.83 \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.

```

3192 \def\xintFloatBinomial   {\romannumeral0\xintfloatbinomial}%
3193 \def\xintfloatbinomial   #1{\XINT_flbinom_chkopt \xintfloat #1\xint:}%
3194 \def\XINTinFloatBinomial {\romannumeral0\XINTinfloatbinomial }%
3195 \def\XINTinfloatbinomial #1{\XINT_flbinom_chkopt \XINTinfloat #1\xint:}%
3196 \def\XINT_flbinom_chkopt #1#2%
3197 {%
3198   \ifx [#2\expandafter\XINT_flbinom_opt
3199     \else\expandafter\XINT_flbinom_noopt
3200   \fi #1#2%
3201 }%
3202 \def\XINT_flbinom_noopt #1#2\xint:#3%
3203 {%
3204   \expandafter\XINT_FL_binom_a
3205   \the\numexpr\xintNum{#2}\expandafter.\the\numexpr\xintNum{#3}.\XINTdigits.#1%
3206 }%
3207 \def\XINT_flbinom_opt #1[\xint:#2]#3#4%

```

```

3208 {%
3209   \expandafter\XINT_FL_binom_a
3210   \the\numexpr\xintNum{#3}\expandafter.\the\numexpr\xintNum{#4}\expandafter.%
3211   \the\numexpr #2.#1%
3212 }%
3213 \def\XINT_FL_binom_a #1.#2.%
3214 {%
3215   \expandafter\XINT_FL_binom_fork \the\numexpr #1-#2.#2.#1.%
3216 }%
3217 \def\XINT_FL_binom_fork #1#2.#3#4.#5#6.%
3218 {%
3219   \if-#5\xint_dothis \XINT_FL_binom_neg\fi
3220   \if-#1\xint_dothis \XINT_FL_binom_zero\fi
3221   \if-#3\xint_dothis \XINT_FL_binom_zero\fi
3222   \if0#1\xint_dothis \XINT_FL_binom_one\fi
3223   \if0#3\xint_dothis \XINT_FL_binom_one\fi
3224   \ifnum #5#6>\xint_c_x^viii_mone \xint_dothis\XINT_FL_binom_toobig\fi
3225   \ifnum #1#2>#3#4 \xint_dothis\XINT_FL_binom_ab \fi
3226   \xint_orthat\XINT_FL_binom_aa
3227   #1#2.#3#4.#5#6.%
3228 }%
3229 \def\XINT_FL_binom_neg #1.#2.#3.#4.#5%
3230 {%
3231   #5[#4]{\XINT_signalcondition{InvalidOperation}
3232           {binomial with first arg negative: #3}}{0[0]}}%
3233 }%
3234 \def\XINT_FL_binom_toobig #1.#2.#3.#4.#5%
3235 {%
3236   #5[#4]{\XINT_signalcondition{InvalidOperation}
3237           {binomial with first arg too big: 99999999 < #3}}{0[0]}}%
3238 }%
3239 \def\XINT_FL_binom_one #1.#2.#3.#4.#5{#5[#4]{1[0]}}%
3240 \def\XINT_FL_binom_zero #1.#2.#3.#4.#5{#5[#4]{0[0]}}%
3241 \def\XINT_FL_binom_aa #1.#2.#3.#4.#5%
3242 {%
3243   #5[#4]{\xintDiv{\XINT_FL_pfac_increaseP
3244             #2.#3.\xint_c_iv{#4+\xint_c_i}{\XINTinfloat[#4+\xint_c_iii]}}%
3245           {\XINT_FL_fac_fork_b
3246             #1.\xint_c_iv{#4+\xint_c_i}\XINT_FL_fac_out{\XINTinfloat[#4+\xint_c_iii]}}}%
3247 }%
3248 \def\XINT_FL_binom_ab #1.#2.#3.#4.#5%
3249 {%
3250   #5[#4]{\xintDiv{\XINT_FL_pfac_increaseP
3251             #1.#3.\xint_c_iv{#4+\xint_c_i}{\XINTinfloat[#4+\xint_c_iii]}}%
3252           {\XINT_FL_fac_fork_b
3253             #2.\xint_c_iv{#4+\xint_c_i}\XINT_FL_fac_out{\XINTinfloat[#4+\xint_c_iii]}}}%
3254 }%

```

8.84 \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.

```

3255 \def\xintFloatSqrt      {\romannumeral0\xintfloatsqr }%
3256 \def\xintfloatsqr      #1{\XINT_flsqr_chkopt \xintfloat #1\xint:}%
3257 \def\XINTinFloatSqr    {\romannumeral0\XINTinfloatsqr }%
3258 \def\XINTinfloatsqr    #1{\XINT_flsqr_chkopt \XINTinfloat #1\xint:}%
3259 \def\XINT_flsqr_chkopt #1#2%
3260 {%
3261   \ifx [#2\expandafter\XINT_flsqr_opt
3262     \else\expandafter\XINT_flsqr_noopt
3263   \fi #1#2%
3264 }%
3265 \def\XINT_flsqr_noopt #1#2\xint:%
3266 {%
3267   \expandafter\XINT_FL_sqrt_a
3268     \romannumeral0\XINTinfloat[\XINTdigits]{#2}\XINTdigits.#1%
3269 }%
3270 \def\XINT_flsqr_opt #1[\xint:#2]#3%
3271 {%
3272   \expandafter\XINT_flsqr_opt_a\the\numexpr #2.#1%
3273 }%
3274 \def\XINT_flsqr_opt_a #1.#2#3%
3275 {%
3276   \expandafter\XINT_FL_sqrt_a\romannumeral0\XINTinfloat[#1]{#3}#1.#2%
3277 }%
3278 \def\XINT_FL_sqrt_a #1%
3279 {%
3280   \xint_UDzerominusfork
3281   #1-\XINT_FL_sqrt_iszero
3282   0#1\XINT_FL_sqrt_isneg
3283   0-\XINT_FL_sqrt_pos #1}%
3284 \kroft
3285 }%[
3286 \def\XINT_FL_sqrt_iszero #1]#2.#3{#3[#2]{0[0]}}%
3287 \def\XINT_FL_sqrt_isneg #1]#2.#3%
3288 {%
3289   #3[#2]{\XINT_signalcondition{InvalidOperation}
3290     {Square root of negative: -#1}}{0[0]}}%
3291 }%

3292 \def\XINT_FL_sqrt_pos #1[#2]#3.%
3293 {%
3294   \expandafter\XINT_flsqr

```

```

3295 \the\numexpr #3\ifodd #2 \xint_dothis {+\xint_c_iii.(#2+\xint_c_i).0}\fi
3296 \xint_orthat {+\xint_c_ii.#2.{}#100.#3.%
3297 }%

3298 \def\xINT_flsqrt #1.#2.%
3299 {%
3300 \expandafter\xINT_flsqrt_a
3301 \the\numexpr #2/\xint_c_ii-(#1-\xint_c_i)/\xint_c_ii.#1.%
3302 }%

3303 \def\xINT_flsqrt_a #1.#2.#3#4.#5.%
3304 {%
3305 \expandafter\xINT_flsqrt_b
3306 \the\numexpr (#2-\xint_c_i)/\xint_c_ii\expandafter.%
3307 \romannumeral0\xINT_sqrt_start #2.#4#3.#5.#2.#4#3.#5.#1.%
3308 }%

3309 \def\xINT_flsqrt_b #1.#2#3%
3310 {%
3311 \expandafter\xINT_flsqrt_c
3312 \romannumeral0\xintiisub
3313 {\XINT_dsx_addzeros {#1}#2;}%
3314 {\xintiiDivRound{\XINT_dsx_addzeros {#1}#3;}%
3315 {\XINT_dbl#2\xint_bye2345678\xint_bye*\xint_c_ii\relax}}.%
3316 }%

3317 \def\xINT_flsqrt_c #1.#2.%
3318 {%
3319 \expandafter\xINT_flsqrt_d
3320 \romannumeral0\xINT_split_fromleft#2.#1\xint_bye2345678\xint_bye..%
3321 }%

3322 \def\xINT_flsqrt_d #1.#2#3.%
3323 {%
3324 \ifnum #2=\xint_c_v
3325 \expandafter\xINT_flsqrt_f\else\expandafter\xINT_flsqrt_finish\fi
3326 #2#3.#1.%
3327 }%

3328 \def\xINT_flsqrt_finish #1#2.#3.#4.#5.#6.#7.#8{#8[#6]{#3#1[#7]}}%

3329 \def\xINT_flsqrt_f 5#1.%
3330 {\expandafter\xINT_flsqrt_g\romannumeral0\xintinum{#1}\relax.%
3331 \def\xINT_flsqrt_g #1#2#3.{\if\relax#2\xint_dothis{\XINT_flsqrt_h #1}\fi
3332 \xint_orthat{\XINT_flsqrt_finish 5.}}%
3333 \def\xINT_flsqrt_h #1{\ifnum #1<\xint_c_iii\xint_dothis{\XINT_flsqrt_again}\fi
3334 \xint_orthat{\XINT_flsqrt_finish 5.}}%

```

```

3335 \def\XINT_flsqrt_again #1.#2.%
3336 {%
3337   \expandafter\XINT_flsqrt_again_a\the\numexpr #2+\xint_c_viii.%
3338 }%

```

```

3339 \def\XINT_flsqrt_again_a #1.#2.#3.%
3340 {%
3341   \expandafter\XINT_flsqrt_b
3342   \the\numexpr (#1-\xint_c_i)/\xint_c_ii\expandafter.%
3343   \romannumeral0\XINT_sqrt_start #1.#200000000.#3.%
3344   #1.#200000000.#3.%
3345 }%

```

8.85 \xintFloatE, \XINTinFloatE

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

1.2k had to rewrite this since there is no more a \XINT_float_a macro. Attention about \XINTinFloatE: it is for use by xintexpr.sty, contrarily to other \XINTinFloat<foo> macros it inserts itself the [\XINTdigits] thing, and with value 0 it produces on output 0[N], not 0[0].

```

3346 \def\xintFloatE   {\romannumeral0\xintfloate }%
3347 \def\xintfloate #1{\XINT_floate_chkopt #1\xint:}%
3348 \def\XINT_floate_chkopt #1%
3349 {%
3350   \ifx [#1\expandafter\XINT_floate_opt
3351     \else\expandafter\XINT_floate_noopt
3352   \fi #1%
3353 }%
3354 \def\XINT_floate_noopt #1\xint:%
3355 {%
3356   \expandafter\XINT_floate_post
3357   \romannumeral0\XINTinfloat[\XINTdigits]{#1}\XINTdigits.%
3358 }%
3359 \def\XINT_floate_opt [\xint:#1]%
3360 {%
3361   \expandafter\XINT_floate_opt_a\the\numexpr #1.%
3362 }%
3363 \def\XINT_floate_opt_a #1.#2%
3364 {%
3365   \expandafter\XINT_floate_post
3366   \romannumeral0\XINTinfloat[#1]{#2}#1.%
3367 }%
3368 \def\XINT_floate_post #1%
3369 {%
3370   \xint_UDzerominusfork
3371   #1-\XINT_floate_zero
3372   0#1\XINT_floate_neg
3373   0-\XINT_floate_pos
3374   \krof #1%
3375 }%[
3376 \def\XINT_floate_zero #1]#2.#3{ 0.e0}%
3377 \def\XINT_floate_neg-{\expandafter-\romannumeral0\XINT_floate_pos}%

```

```

3378 \def\XINT_floate_pos #1#2[#3]#4.#5%
3379 {%
3380   \expandafter\XINT_float_pos_done\the\numexpr#3+#4+#5-\xint_c_i.#1.#2;%
3381 }%
3382 \def\XINTinFloatE {\romannumeral0\XINTinfloate }%
3383 \def\XINTinfloate
3384   {\expandafter\XINT_infloate\romannumeral0\XINTinfloat[\XINTdigits]}%
3385 \def\XINT_infloate #1[#2]#3%
3386   {\expandafter\XINT_infloate_end\the\numexpr #3+#2.{#1}}%
3387 \def\XINT_infloate_end #1.#2{ #2[#1]}%

```

8.86 \XINTinFloatMod

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

```

3388 \def\XINTinFloatMod {\romannumeral0\XINTinfloatmod [\XINTdigits]}%
3389 \def\XINTinfloatmod [#1]#2#3%
3390 {%
3391   \XINTinfloat[#1]{\xintMod
3392     {\romannumeral0\XINTinfloat[#1]{#2}}%
3393     {\romannumeral0\XINTinfloat[#1]{#3}}}%
3394 }%

```

8.87 \XINTinFloatDivFloor

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

```

3395 \def\XINTinFloatDivFloor {\romannumeral0\XINTinfloatdivfloor [\XINTdigits]}%
3396 \def\XINTinfloatdivfloor [#1]#2#3%
3397 {%
3398   \xintdivfloor
3399     {\romannumeral0\XINTinfloat[#1]{#2}}%
3400     {\romannumeral0\XINTinfloat[#1]{#3}}%
3401 }%

```

8.88 \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.

```

3402 \def\XINTinFloatDivMod {\romannumeral0\XINTinfloatdivmod [\XINTdigits]}%
3403 \def\XINTinfloatdivmod [#1]#2#3%
3404 {%
3405   \expandafter\XINT_infloatdivmod
3406   \romannumeral0\xintdivmod
3407     {\romannumeral0\XINTinfloat[#1]{#2}}%

```



```

3408      {\romannumeral0\XINTinfloat[#1]{#3}}%
3409      {#1}%
3410 }%
3411 \def\XINT_infloatdivmod #1#2#3{\expanded{{#1}{\XINTinFloat[#3]{#2}}}}%

```

8.89 \xintifFloatInt

1.3a for ifint() function in \xintfloatexpr.

```

3412 \def\xintifFloatInt {\romannumeral0\xintiffloatint}%
3413 \def\xintiffloatint #1{\expandafter\XINT_iffloatint
3414      \romannumeral0\xintrez{\XINTinFloat[\XINTdigits]{#1}}}%
3415 \def\XINT_iffloatint #1#2/1[#3]%
3416 {%
3417   \if 0#1\xint_dothis\xint_stop_atfirstoftwo\fi
3418   \ifnum#3<\xint_c_\xint_dothis\xint_stop_atsecondoftwo\fi
3419   \xint_orthat\xint_stop_atfirstoftwo
3420 }%

```

8.90 \xintFloatIsInt

1.3d for isint() function in \xintfloatexpr.

```

3421 \def\xintFloatIsInt {\romannumeral0\xintfloatisint}%
3422 \def\xintfloatisint #1{\expandafter\XINT_iffloatint
3423      \romannumeral0\xintrez{\XINTinFloat[\XINTdigits]{#1}}10}%

```

8.91 \XINTinFloatdigits, \XINTinFloatSqrtdigits, \XINTinFloatFacdigits, \XINTiLogTendigits

For \xintNewExpr matters, mainly.

At 1.3e I add \XINTinFloatSdigits and use it at various places. I also modified \XINTinFloatFac to use S(hort) output format.

Also added \XINTiLogTendigits.

This whole stuff moved over from xintexpr.sty at 1.4

```

3424 \def\XINTinFloatdigits   {\XINTinFloat   [\XINTdigits]}%
3425 \def\XINTinFloatSdigits  {\XINTinFloatS  [\XINTdigits]}%
3426 \def\XINTinFloatSqrtdigits {\XINTinFloatSqrt[\XINTdigits]}%
3427 \def\XINTinFloatFacdigits {\XINTinFloatFac [\XINTdigits]}%
3428 \def\XINTFloatiLogTendigits{\XINTFloatiLogTen[\XINTdigits]}%

```

8.92 (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.

```

3429 \def\xINTinRandomFloatS{\romannumeral0\xINTinrandomfloatS}%
3430 \def\xINTinRandomFloatSdigits{\XINTinRandomFloatS[\XINTdigits]}%
3431 \def\xINTinrandomfloatS[#1]%
3432 {%
3433   \expandafter\xINT_inrandomfloatS\the\numexpr\xint_c_-(#1)\xint:
3434 }%
3435 \def\xINT_inrandomfloatS-#1\xint:
3436 {%
3437   \expandafter\xINT_inrandomfloatS_a
3438   \romannumeral0\xintrandomdigits{#1}[-#1]%
3439 }%

```

We add one macro to handle a tiny bit faster 90of cases, after all we also use one extra macro for the completely improbable all 0 case.

```

3440 \def\xINT_inrandomfloatS_a#1%
3441 {%
3442   \if#10\xint_dothis{\XINT_inrandomfloatS_b}\fi
3443   \xint_orthat{ #1}%
3444 }%[
3445 \def\xINT_inrandomfloatS_b#1%
3446 {%
3447   \if#1[\xint_dothis{\XINT_inrandomfloatS_zero}\fi% ]
3448   \if#10\xint_dothis{\XINT_inrandomfloatS_b}\fi
3449   \xint_orthat{ #1}%
3450 }%[
3451 \def\xINT_inrandomfloatS_zero#1]{ 0[0]}%

```

8.93 (WIP) `\XINTinRandomFloatSixteen`

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

```

3452 \def\xINTinRandomFloatSixteen%
3453 {%
3454   \romannumeral0\expandafter\xINT_inrandomfloatS_a
3455   \romannumeral`&&\expandafter\xINT_eightrandomdigits
3456   \romannumeral`&&\XINT_eightrandomdigits[-16]%
3457 }%

```

8.94 `\PoorManLogBaseTen`

1.3f. Code originally in `poormanlog v0.4` got transferred here. It produces the logarithm in base 10 with an error (believed to be at most) about 1 unit in the 9th (i.e. last) fractional digit. Testing seems to indicate error at most 2 units.

```

3458 \def\PoorManLogBaseTen{\romannumeral0\poormanlogbaseten}%
3459 \def\poormanlogbaseten #1%
3460   {\expandafter\PML@logbaseten\romannumeral0\XINTinfloat[9]{#1}}%
3461 \def\PML@logbaseten#1[#2]%
3462 {%
3463   \xintiiadd{\xintDSx{-9}}{\the\numexpr#2+8\relax}}{\the\numexpr\PML@#1.}%
3464   [-9]%
3465 }%

```

8.95 \PoorManPowerOfTen

1.3f. Transferred from poormanlog v0.4. Produces the $10^{\#1}$ with 9 digits of float precision, with an error (believed to be) at most 2 units in the last place. Of course for this the input must be precise enough to have 9 fractional digits of **fixed point** precision.

Attention that this breaks with low level Number too big error if integral part of argument exceeds TeX bound on integers. Indeed some `\numexpr` is used in the code to subtract 8... but anyway `xintfrac` allows for scientific exponents only integers within TeX bounds, so even if it did not break here it would quickly elsewhere.

```

3466 \def\PoorManPowerOfTen{\the\numexpr\poormanpoweroften}%
3467 \def\poormanpoweroften #1%
3468   {\expandafter\PML@powoften\romannumeral0\xintraw{#1}}%
3469 \def\PML@powoften#1%
3470 {%
3471   \xint_UDzerominusfork
3472   #1-\PML@powoften@zero
3473   0#1\PML@powoften@neg
3474   0-\PML@powoften@pos
3475   \krof #1%
3476 }%
3477 \def\PML@powoften@zero 0{1\relax}%/1[0]
3478 \def\PML@powoften@pos#1[#2]%
3479 {%
3480   \expandafter\PML@powoften@pos@a\romannumeral0\xintround{9}{#1[#2]}.%
3481 }%
3482 \def\PML@powoften@pos@a#1.#2.{\PML@Pa#2.\expandafter[\the\numexpr-8+#1]}%
3483 \def\PML@powoften@neg#1[#2]%
3484 {%
3485   \expandafter\PML@powoften@neg@a\romannumeral0\xintround{9}{#1[#2]}.%
3486 }%
3487 \def\PML@powoften@neg@a#1.#2.%
3488 {\ifnum#2=\xint_c_ \xint_afterfi{1\relax/1[#1]}\else
3489   \expandafter\expandafter\expandafter
3490   \PML@Pa\expandafter\xint_gobble_i\the\numexpr2000000000-#2.%
3491   \expandafter[\the\numexpr-9+#1\expandafter]\fi
3492 }%

```

8.96 \PoorManPower

1.3f. This code originally in poormanlog v0.4 transferred here. It does #1 to the power #2.

```

3493 \def\PoorManPower#1#2%
3494 {%

```

```
3495 \PoorManPowerOfTen{\xintMul{#2}{\PoorManLogBaseTen{#1}}}%  
3496 }%
```

8.97 Support macros for natural logarithm and exponential [xintexpr](#) functions

At 1.3f, the poormanlog v0.04 extension to xintfrac.sty got transferred here. These macros from xintlog.sty 1.3e got transferred here too.

```
3497 \def\xintLog#1{\xintMul{\PoorManLogBaseTen{#1}}{23025850923[-10]}}%  
3498 \def\XINTinFloatLog#1{\XINTinFloatMul{\PoorManLogBaseTen{#1}}{23025850923[-10]}}%  
3499 \def\xintExp#1{\PoorManPowerOfTen{\xintMul{#1}{434294481903[-12]}}}%  
3500 \def\XINTinFloatExp#1{\PoorManPowerOfTen{\XINTinFloatMul{#1}{434294481903[-12]}}}%  
3501 \let\XINTinFloatMaxof\XINT_Maxof  
3502 \let\XINTinFloatMinof\XINT_Minof  
3503 \let\XINTinFloatSum\XINT_Sum  
3504 \let\XINTinFloatPrd\XINT_Prd  
3505 \XINT_restorecatcodes_endinput%
```

9 Package *xintseries* implementation

<p>.1 Catcodes, ε-TeX and reload detection . . . 277</p> <p>.2 Package identification 278</p> <p>.3 <code>\xintSeries</code> 278</p> <p>.4 <code>\xintiSeries</code> 278</p> <p>.5 <code>\xintPowerSeries</code> 279</p> <p>.6 <code>\xintPowerSeriesX</code> 280</p>	<p>.7 <code>\xintRationalSeries</code> 280</p> <p>.8 <code>\xintRationalSeriesX</code> 281</p> <p>.9 <code>\xintFxFtPowerSeries</code> 282</p> <p>.10 <code>\xintFxFtPowerSeriesX</code> 283</p> <p>.11 <code>\xintFloatPowerSeries</code> 283</p> <p>.12 <code>\xintFloatPowerSeriesX</code> 285</p>
--	---

The commenting is currently (2020/02/25) very sparse.

9.1 Catcodes, ε -TeX and reload detection

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

```

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

```

```

38     \aftergroup\endinput % xintseries already loaded.
39     \fi
40     \fi
41     \fi
42 \z%
43 \XINTsetupcatcodes% defined in xintkernel.sty

```

9.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintseries}%
46 [2020/02/25 v1.4b Expandable partial sums with xint package (JFB)]%

```

9.3 \xintSeries

```

47 \def\xintSeries {\romannumeral0\xintseries }%
48 \def\xintseries #1#2%
49 {%
50     \expandafter\XINT_series\expandafter
51     {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
52 }%
53 \def\XINT_series #1#2#3%
54 {%
55     \ifnum #2<#1
56         \xint_afterfi { 0/1[0]}%
57     \else
58         \xint_afterfi {\XINT_series_loop {#1}{0}{#2}{#3}}%
59     \fi
60 }%
61 \def\XINT_series_loop #1#2#3#4%
62 {%
63     \ifnum #3>#1 \else \XINT_series_exit \fi
64     \expandafter\XINT_series_loop\expandafter
65     {\the\numexpr #1+1\expandafter }\expandafter
66     {\romannumeral0\xintadd {#2}{#4{#1}}}%
67     {#3}{#4}%
68 }%
69 \def\XINT_series_exit \fi #1#2#3#4#5#6#7#8%
70 {%
71     \fi\xint_gobble_ii #6%
72 }%

```

9.4 \xintiSeries

```

73 \def\xintiSeries {\romannumeral0\xintiseries }%
74 \def\xintiseries #1#2%
75 {%
76     \expandafter\XINT_ieries\expandafter
77     {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
78 }%
79 \def\XINT_ieries #1#2#3%
80 {%
81     \ifnum #2<#1
82         \xint_afterfi { 0}%
83     \else

```

```

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

```

9.5 \xintPowerSeries

The 1.03 version was very lame and created a build-up of denominators. (this was at a time \xintAdd always multiplied denominators, by the way) The Horner scheme for polynomial evaluation is used in 1.04, this cures the denominator problem and drastically improves the efficiency of the macro. Modified in 1.06 to give the indices first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that. 1.08a adds the forgotten optimization following that previous change.

```

99 \def\xintPowerSeries {\romannumeral0\xintpowerseries }%
100 \def\xintpowerseries #1#2%
101 {%
102     \expandafter\XINT_powseries\expandafter
103     {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
104 }%
105 \def\XINT_powseries #1#2#3#4%
106 {%
107     \ifnum #2<#1
108         \xint_afterfi { 0/1[0]}%
109     \else
110         \xint_afterfi
111         {\XINT_powseries_loop_i {#3{#2}}{#1}{#2}{#3}{#4}}%
112     \fi
113 }%
114 \def\XINT_powseries_loop_i #1#2#3#4#5%
115 {%
116     \ifnum #3>#2 \else\XINT_powseries_exit_i\fi
117     \expandafter\XINT_powseries_loop_ii\expandafter
118     {\the\numexpr #3-1\expandafter}\expandafter
119     {\romannumeral0\xintmul {#1}{#5}}{#2}{#4}{#5}%
120 }%
121 \def\XINT_powseries_loop_ii #1#2#3#4%
122 {%
123     \expandafter\XINT_powseries_loop_i\expandafter
124     {\romannumeral0\xintadd {#4{#1}}{#2}}{#3}{#1}{#4}%
125 }%
126 \def\XINT_powseries_exit_i\fi #1#2#3#4#5#6#7#8#9%
127 {%

```

```

128   \fi \XINT_powseries_exit_ii #6{#7}%
129 }%
130 \def\XINT_powseries_exit_ii #1#2#3#4#5#6%
131 {%
132   \xintmul{\xintPow {#5}{#6}}{#4}%
133 }%

```

9.6 \xintPowerSeriesX

Same as `\xintPowerSeries` except for the initial expansion of the `x` parameter. Modified in 1.06 to give the indices first to a `\numexpr` rather than expanding twice. I just use `\the\numexpr` and maintain the previous code after that. 1.08a adds the forgotten optimization following that previous change.

```

134 \def\xintPowerSeriesX {\romannumeral0\xintpowerseriesx }%
135 \def\xintpowerseriesx #1#2%
136 {%
137   \expandafter\XINT_powseriesx\expandafter
138   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
139 }%
140 \def\XINT_powseriesx #1#2#3#4%
141 {%
142   \ifnum #2<#1
143     \xint_afterfi { 0/1[0]}%
144   \else
145     \xint_afterfi
146     {\expandafter\XINT_powseriesx_pre\expandafter
147      {\romannumeral`&&@#4}{#1}{#2}{#3}%
148     }%
149   \fi
150 }%
151 \def\XINT_powseriesx_pre #1#2#3#4%
152 {%
153   \XINT_powseries_loop_i {#4{#3}}{#2}{#3}{#4}{#1}%
154 }%

```

9.7 \xintRationalSeries

This computes $F(a)+\dots+F(b)$ on the basis of the value of $F(a)$ and the ratios $F(n)/F(n-1)$. As in `\xintPowerSeries` we use an iterative scheme which has the great advantage to avoid denominator build-up. This makes exact computations possible with exponential type series, which would be completely inaccessible to `\xintSeries`. `#1=a`, `#2=b`, `#3=F(a)`, `#4=ratio` function Modified in 1.06 to give the indices first to a `\numexpr` rather than expanding twice. I just use `\the\numexpr` and maintain the previous code after that. 1.08a adds the forgotten optimization following that previous change.

```

155 \def\xintRationalSeries {\romannumeral0\xintratseries }%
156 \def\xintratseries #1#2%
157 {%
158   \expandafter\XINT_ratseries\expandafter
159   {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
160 }%
161 \def\XINT_ratseries #1#2#3#4%

```



```

162 {%
163   \ifnum #2<#1
164     \xint_afterfi { 0/1[0]}%
165   \else
166     \xint_afterfi
167     {\XINT_ratseries_loop {#2}{1}{#1}{#4}{#3}}%
168   \fi
169 }%
170 \def\XINT_ratseries_loop #1#2#3#4%
171 {%
172   \ifnum #1>#3 \else\XINT_ratseries_exit_i\fi
173   \expandafter\XINT_ratseries_loop\expandafter
174   {\the\numexpr #1-1\expandafter}\expandafter
175   {\romannumeral0\xintadd {1}{\xintMul {#2}{#4{#1}}}{#3}{#4}}%
176 }%
177 \def\XINT_ratseries_exit_i\fi #1#2#3#4#5#6#7#8%
178 {%
179   \fi \XINT_ratseries_exit_ii #6%
180 }%
181 \def\XINT_ratseries_exit_ii #1#2#3#4#5%
182 {%
183   \XINT_ratseries_exit_iii #5%
184 }%
185 \def\XINT_ratseries_exit_iii #1#2#3#4%
186 {%
187   \xintmul{#2}{#4}%
188 }%

```

9.8 \xintRationalSeriesX

a,b,initial,ratiofunction,x

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

```

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

```

```

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

```

9.9 \xintFxpPowerSeries

I am not too happy with this piece of code. Will make it more economical another day. Modified in 1.06 to give the indices first to a `\numexpr` rather than expanding twice. I just use `\the\numexpr` and maintain the previous code after that. 1.08a: forgot last time some optimization from the change to `\numexpr`.

```

210 \def\xintFxpPowerSeries {\romannumeral0\xintfxptpowerseries }%
211 \def\xintfxptpowerseries #1#2%
212 {%
213 \expandafter\XINT_fppowseries\expandafter
214 {\the\numexpr #1\expandafter}\expandafter{\the\numexpr #2}%
215 }%
216 \def\XINT_fppowseries #1#2#3#4#5%
217 {%
218 \ifnum #2<#1
219 \xint_afterfi { 0}%
220 \else
221 \xint_afterfi
222 {\expandafter\XINT_fppowseries_loop_pre\expandafter
223 {\romannumeral0\xinttrunc {#5}{\xintPow {#4}{#1}}}%
224 {#1}{#4}{#2}{#3}{#5}%
225 }%
226 \fi
227 }%
228 \def\XINT_fppowseries_loop_pre #1#2#3#4#5#6%
229 {%
230 \ifnum #4>#2 \else\XINT_fppowseries_dont_i \fi
231 \expandafter\XINT_fppowseries_loop_i\expandafter
232 {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
233 {\romannumeral0\xintitrunc {#6}{\xintMul {#5}{#2}}{#1}}%
234 {#1}{#3}{#4}{#5}{#6}%
235 }%
236 \def\XINT_fppowseries_dont_i \fi\expandafter\XINT_fppowseries_loop_i
237 {\fi \expandafter\XINT_fppowseries_dont_ii }%
238 \def\XINT_fppowseries_dont_ii #1#2#3#4#5#6#7{\xinttrunc {#7}{#2[-#7]}}%
239 \def\XINT_fppowseries_loop_i #1#2#3#4#5#6#7%
240 {%
241 \ifnum #5>#1 \else \XINT_fppowseries_exit_i \fi
242 \expandafter\XINT_fppowseries_loop_ii\expandafter
243 {\romannumeral0\xinttrunc {#7}{\xintMul {#3}{#4}}}%
244 {#1}{#4}{#2}{#5}{#6}{#7}%
245 }%
246 \def\XINT_fppowseries_loop_ii #1#2#3#4#5#6#7%
247 {%
248 \expandafter\XINT_fppowseries_loop_i\expandafter

```

```

249     {\the\numexpr #2+\xint_c_i\expandafter}\expandafter
250     {\romannumeral0\xintiiaadd {#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     {\xintiiaadd {#4}{\xintiTrunc {#7}{\xintMul {#6{#2}}{#1}}[-#7]}%
259 }%

```

9.10 \xintFxpPowerSeriesX

a,b,coeff,x,D

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

```

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

```

9.11 \xintFloatPowerSeries

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

```

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

```

```

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

```

```

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

```

9.12 \xintFloatPowerSeriesX

1.08a

```

348 \def\xintFloatPowerSeriesX {\romannumeral0\xintfloatpowerseriesx }%
349 \def\xintfloatpowerseriesx #1{\XINT_flpowseriesx_chkopt #1\xint:}%
350 \def\XINT_flpowseriesx_chkopt #1%
351 {%
352     \ifx [#1\expandafter\XINT_flpowseriesx_opt
353         \else\expandafter\XINT_flpowseriesx_noopt
354     \fi
355     #1%
356 }%
357 \def\XINT_flpowseriesx_noopt #1\xint:#2%
358 {%
359     \expandafter\XINT_flpowseriesx\expandafter
360     {\the\numexpr #1\expandafter}\expandafter
361     {\the\numexpr #2}\XINTdigits
362 }%
363 \def\XINT_flpowseriesx_opt [\xint:#1]#2#3%
364 {%
365     \expandafter\XINT_flpowseriesx\expandafter
366     {\the\numexpr #2\expandafter}\expandafter
367     {\the\numexpr #3\expandafter}{\the\numexpr #1}%
368 }%
369 \def\XINT_flpowseriesx #1#2#3#4#5%
370 {%
371     \ifnum #2<#1
372         \xint_afterfi { 0.e0}%
373     \else
374         \xint_afterfi
375         {\expandafter \XINT_flpowseriesx_pre \expandafter
376         {\romannumeral`&&#5}{#1}{#2}{#4}{#3}%
377         }%
378     \fi
379 }%
380 \def\XINT_flpowseriesx_pre #1#2#3#4#5%
381 {%
382     \expandafter\XINT_flpowseries_loop_pre\expandafter
383     {\romannumeral0\XINTinfloatpow [#5]{#1}{#2}}%
384     {#2}{#1}{#3}{#4}{#5}%
385 }%
386 \XINT_restorecatcodes_endinput%

```

10 Package *xintfrac* implementation

<p>.1 Catcodes, ε-TeX and reload detection . . . 286</p> <p>.2 Package identification 287</p> <p>.3 <code>\xintCFrac</code> 287</p> <p>.4 <code>\xintGCFrac</code> 288</p> <p>.5 <code>\xintGGCFrac</code> 290</p> <p>.6 <code>\xintGctoGCx</code> 291</p> <p>.7 <code>\xintFtoCs</code> 291</p> <p>.8 <code>\xintFtoCx</code> 292</p> <p>.9 <code>\xintFtoC</code> 292</p> <p>.10 <code>\xintFtoGC</code> 293</p> <p>.11 <code>\xintFGtoC</code> 293</p> <p>.12 <code>\xintFtoCC</code> 294</p> <p>.13 <code>\xintCtoF</code>, <code>\xintCstoF</code> 295</p> <p>.14 <code>\xintiCstoF</code> 296</p> <p>.15 <code>\xintGctoF</code> 296</p>	<p>.16 <code>\xintiGctoF</code> 298</p> <p>.17 <code>\xintCtoCv</code>, <code>\xintCstoCv</code> 299</p> <p>.18 <code>\xintiCstoCv</code> 300</p> <p>.19 <code>\xintGctoCv</code> 300</p> <p>.20 <code>\xintiGctoCv</code> 302</p> <p>.21 <code>\xintFtoCv</code> 303</p> <p>.22 <code>\xintFtoCCv</code> 303</p> <p>.23 <code>\xintCntoF</code> 303</p> <p>.24 <code>\xintGcntoF</code> 304</p> <p>.25 <code>\xintCntoCs</code> 305</p> <p>.26 <code>\xintCntoGC</code> 305</p> <p>.27 <code>\xintGcntoGC</code> 306</p> <p>.28 <code>\xintCstoGC</code> 307</p> <p>.29 <code>\xintGctoGC</code> 307</p>
---	---

The commenting is currently (2020/02/25) 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 -TeX and reload detection

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

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

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2  \catcode13=5    % ^^M
3  \endlinechar=13 %
4  \catcode123=1  % {
5  \catcode125=2  % }
6  \catcode64=11 % @
7  \catcode35=6   % #
8  \catcode44=12 % ,
9  \catcode45=12 % -
10 \catcode46=12 % .
11 \catcode58=12 % :
12 \let\z\endgroup
13 \expandafter\let\expandafter\x\csname ver@xintfrac.sty\endcsname
14 \expandafter\let\expandafter\w\csname ver@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{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 xintfrac.sty not yet loaded.
28       \def\z{\endgroup\input xintfrac.sty\relax}%
29     \fi
30   \else
31     \def\empty {}%
32     \ifx\x\empty % LaTeX, first loading,
33     % variable is initialized, but \ProvidesPackage not yet seen
34     \ifx\w\relax % xintfrac.sty not yet loaded.
35       \def\z{\endgroup\RequirePackage{xintfrac}}%
36     \fi
37   \else
38     \aftergroup\endinput % xintfrac already loaded.
39   \fi
40 \fi
41 \fi
42 \z%
43 \XINTsetupcatcodes% defined in xintkernel.sty

```

10.2 Package identification

```

44 \XINT_providespackage
45 \ProvidesPackage{xintfrac}%
46 [2020/02/25 v1.4b Expandable continued fractions with xint package (JFB)]%

```

10.3 \xintCFrac

```

47 \def\xintCFrac {\romannumeral0\xintfrac }%
48 \def\xintfrac #1%
49 {%
50   \XINT_cfrac_opt_a #1\xint:
51 }%
52 \def\XINT_cfrac_opt_a #1%
53 {%
54   \ifx[#1\XINT_cfrac_opt_b\fi \XINT_cfrac_noopt #1%
55 }%
56 \def\XINT_cfrac_noopt #1\xint:
57 {%
58   \expandafter\XINT_cfrac_A\romannumeral0\xintraawithzeros {#1}\Z
59   \relax\relax
60 }%
61 \def\XINT_cfrac_opt_b\fi\XINT_cfrac_noopt [\xint:#1]%
62 {%
63   \fi\csname XINT_cfrac_opt#1\endcsname
64 }%
65 \def\XINT_cfrac_optl #1%
66 {%
67   \expandafter\XINT_cfrac_A\romannumeral0\xintraawithzeros {#1}\Z
68   \relax\hfill
69 }%
70 \def\XINT_cfrac_optc #1%
71 {%
72   \expandafter\XINT_cfrac_A\romannumeral0\xintraawithzeros {#1}\Z

```

```

73   \relax\relax
74 }%
75 \def\XINT_cfrac_optr #1%
76 {%
77   \expandafter\XINT_cfrac_A\romannumeral0\xintraawithzeros {#1}\Z
78   \hfill\relax
79 }%
80 \def\XINT_cfrac_A #1/#2\Z
81 {%
82   \expandafter\XINT_cfrac_B\romannumeral0\xintiidivision {#1}{#2}{#2}%
83 }%
84 \def\XINT_cfrac_B #1#2%
85 {%
86   \XINT_cfrac_C #2\Z {#1}%
87 }%
88 \def\XINT_cfrac_C #1%
89 {%
90   \xint_gob_til_zero #1\XINT_cfrac_integer 0\XINT_cfrac_D #1%
91 }%
92 \def\XINT_cfrac_integer 0\XINT_cfrac_D 0#1\Z #2#3#4#5{ #2}%
93 \def\XINT_cfrac_D #1\Z #2#3{\XINT_cfrac_loop_a {#1}{#3}{#1}{#2}}%
94 \def\XINT_cfrac_loop_a
95 {%
96   \expandafter\XINT_cfrac_loop_d\romannumeral0\XINT_div_prepare
97 }%
98 \def\XINT_cfrac_loop_d #1#2%
99 {%
100   \XINT_cfrac_loop_e #2.{#1}%
101 }%
102 \def\XINT_cfrac_loop_e #1%
103 {%
104   \xint_gob_til_zero #1\xint_cfrac_loop_exit0\XINT_cfrac_loop_f #1%
105 }%
106 \def\XINT_cfrac_loop_f #1.#2#3#4%
107 {%
108   \XINT_cfrac_loop_a {#1}{#3}{#1}{#2#4}%
109 }%
110 \def\xint_cfrac_loop_exit0\XINT_cfrac_loop_f #1.#2#3#4#5#6%
111   {\XINT_cfrac_T #5#6{#2}#4\Z }%
112 \def\XINT_cfrac_T #1#2#3#4%
113 {%
114   \xint_gob_til_Z #4\XINT_cfrac_end\Z\XINT_cfrac_T #1#2{#4+\cfrac{#1#2}{#3}}%
115 }%
116 \def\XINT_cfrac_end\Z\XINT_cfrac_T #1#2#3%
117 {%
118   \XINT_cfrac_end_b #3%
119 }%
120 \def\XINT_cfrac_end_b \Z+\cfrac#1#2{ #2}%

```

10.4 \xintGCFrac

```

121 \def\xintGCFrac {\romannumeral0\xintgcfrac }%
122 \def\xintgcfrac #1{\XINT_gcfrac_opt_a #1\xint:}%
123 \def\XINT_gcfrac_opt_a #1%

```



```

124 {%
125   \ifx[#1\XINT_gcfrac_opt_b\fi \XINT_gcfrac_noopt #1%
126 }%
127 \def\XINT_gcfrac_noopt #1\xint:%
128 {%
129   \XINT_gcfrac #1+!\relax\relax
130 }%
131 \def\XINT_gcfrac_opt_b\fi\XINT_gcfrac_noopt [\xint:#1]%
132 {%
133   \fi\csname XINT_gcfrac_opt#1\endcsname
134 }%
135 \def\XINT_gcfrac_optl #1%
136 {%
137   \XINT_gcfrac #1+!\relax\hfill
138 }%
139 \def\XINT_gcfrac_optc #1%
140 {%
141   \XINT_gcfrac #1+!\relax\relax
142 }%
143 \def\XINT_gcfrac_optr #1%
144 {%
145   \XINT_gcfrac #1+!\hfill\relax
146 }%
147 \def\XINT_gcfrac
148 {%
149   \expandafter\XINT_gcfrac_enter\romannumeral`&&%
150 }%
151 \def\XINT_gcfrac_enter {\XINT_gcfrac_loop {}}%
152 \def\XINT_gcfrac_loop #1#2+#3/%
153 {%
154   \xint_gob_til_exclam #3\XINT_gcfrac_endloop!%
155   \XINT_gcfrac_loop {{#3}{#2}#1}%
156 }%
157 \def\XINT_gcfrac_endloop!\XINT_gcfrac_loop #1#2#3%
158 {%
159   \XINT_gcfrac_T #2#3#1!!%
160 }%
161 \def\XINT_gcfrac_T #1#2#3#4{\XINT_gcfrac_U #1#2{\xintFrac{#4}}}%
162 \def\XINT_gcfrac_U #1#2#3#4#5%
163 {%
164   \xint_gob_til_exclam #5\XINT_gcfrac_end!\XINT_gcfrac_U
165     #1#2{\xintFrac{#5}}%
166     \ifcase\xintSgn{#4}
167     +\or+\else-\fi
168     \cfrac{#1\xintFrac{\xintAbs{#4}}#2}{#3}}%
169 }%
170 \def\XINT_gcfrac_end!\XINT_gcfrac_U #1#2#3%
171 {%
172   \XINT_gcfrac_end_b #3%
173 }%
174 \def\XINT_gcfrac_end_b #1\cfrac#2#3{ #3}%

```

10.5 \xintGGCFrac

New with 1.09m

```

175 \def\xintGGCFrac {\romannumeral0\xintggcfrac }%
176 \def\xintggcfrac #1{\XINT_ggcfrac_opt_a #1\xint:}%
177 \def\XINT_ggcfrac_opt_a #1%
178 {%
179   \ifx[#1\XINT_ggcfrac_opt_b\fi \XINT_ggcfrac_noopt #1%
180 }%
181 \def\XINT_ggcfrac_noopt #1\xint:
182 {%
183   \XINT_ggcfrac #1+!\relax\relax
184 }%
185 \def\XINT_ggcfrac_opt_b\fi\XINT_ggcfrac_noopt [\xint:#1]%
186 {%
187   \fi\csname XINT_ggcfrac_opt#1\endcsname
188 }%
189 \def\XINT_ggcfrac_optl #1%
190 {%
191   \XINT_ggcfrac #1+!\relax\hfill
192 }%
193 \def\XINT_ggcfrac_optc #1%
194 {%
195   \XINT_ggcfrac #1+!\relax\relax
196 }%
197 \def\XINT_ggcfrac_optr #1%
198 {%
199   \XINT_ggcfrac #1+!\hfill\relax
200 }%
201 \def\XINT_ggcfrac
202 {%
203   \expandafter\XINT_ggcfrac_enter\romannumeral`&&@%
204 }%
205 \def\XINT_ggcfrac_enter {\XINT_ggcfrac_loop {}}%
206 \def\XINT_ggcfrac_loop #1#2+#3/%
207 {%
208   \xint_gob_til_exclam #3\XINT_ggcfrac_endloop!%
209   \XINT_ggcfrac_loop {{#3}{#2}{#1}}%
210 }%
211 \def\XINT_ggcfrac_endloop!\XINT_ggcfrac_loop #1#2#3%
212 {%
213   \XINT_ggcfrac_T #2#3#1!!%
214 }%
215 \def\XINT_ggcfrac_T #1#2#3#4{\XINT_ggcfrac_U #1#2{#4}}%
216 \def\XINT_ggcfrac_U #1#2#3#4#5%
217 {%
218   \xint_gob_til_exclam #5\XINT_ggcfrac_end!\XINT_ggcfrac_U
219     #1#2{#5+\cfrac{#1#4#2}{#3}}%
220 }%
221 \def\XINT_ggcfrac_end!\XINT_ggcfrac_U #1#2#3%
222 {%
223   \XINT_ggcfrac_end_b #3%

```

```
224 }%
225 \def\xINT_ggcfrac_end_b #1\cfrac#2#3{ #3}%
```

10.6 \xintGctoGCx

```
226 \def\xintGctoGCx {\romannumeral0\xintgctogcx }%
227 \def\xintgctogcx #1#2#3%
228 {%
229   \expandafter\xINT_gctgcx_start\expandafter {\romannumeral`&&@#3}{#1}{#2}%
230 }%
231 \def\xINT_gctgcx_start #1#2#3{\XINT_gctgcx_loop_a {}{#2}{#3}#1+!/%}
232 \def\xINT_gctgcx_loop_a #1#2#3#4+#5/%
233 {%
234   \xint_gob_til_exclam #5\xINT_gctgcx_end!%
235   \XINT_gctgcx_loop_b {#1{#4}}{#2{#5}#3}{#2}{#3}%
236 }%
237 \def\xINT_gctgcx_loop_b #1#2%
238 {%
239   \XINT_gctgcx_loop_a {#1#2}%
240 }%
241 \def\xINT_gctgcx_end!\XINT_gctgcx_loop_b #1#2#3#4{ #1}%
```

10.7 \xintFtoCs

Modified in 1.09m: a space is added after the inserted commas.

```
242 \def\xintFtoCs {\romannumeral0\xintftocs }%
243 \def\xintftocs #1%
244 {%
245   \expandafter\xINT_ftc_A\romannumeral0\xintraawwithzeros {#1}\Z
246 }%
247 \def\xINT_ftc_A #1/#2\Z
248 {%
249   \expandafter\xINT_ftc_B\romannumeral0\xintiidivision {#1}{#2}{#2}%
250 }%
251 \def\xINT_ftc_B #1#2%
252 {%
253   \XINT_ftc_C #2.{#1}%
254 }%
255 \def\xINT_ftc_C #1%
256 {%
257   \xint_gob_til_zero #1\xINT_ftc_integer 0\xINT_ftc_D #1%
258 }%
259 \def\xINT_ftc_integer 0\xINT_ftc_D 0#1.#2#3{ #2}%
260 \def\xINT_ftc_D #1.#2#3{\XINT_ftc_loop_a {#1}{#3}{#1}{#2, }}% 1.09m adds a space
261 \def\xINT_ftc_loop_a
262 {%
263   \expandafter\xINT_ftc_loop_d\romannumeral0\xINT_div_prepare
264 }%
265 \def\xINT_ftc_loop_d #1#2%
266 {%
267   \XINT_ftc_loop_e #2.{#1}%
268 }%
269 \def\xINT_ftc_loop_e #1%
```

```

270 {%
271   \xint_gob_til_zero #1\xint_ftc_loop_exit0\XINT_ftc_loop_f #1%
272 }%
273 \def\XINT_ftc_loop_f #1.#2#3#4%
274 {%
275   \XINT_ftc_loop_a {#1}{#3}{#1}{#4#2, }% 1.09m has an added space here
276 }%
277 \def\xint_ftc_loop_exit0\XINT_ftc_loop_f #1.#2#3#4{ #4#2}%

```

10.8 \xintFtoCx

```

278 \def\xintFtoCx {\romannumeral0\xintftocx }%
279 \def\xintftocx #1#2%
280 {%
281   \expandafter\XINT_ftcx_A\romannumeral0\xintraewithzeros {#2}\Z {#1}%
282 }%
283 \def\XINT_ftcx_A #1/#2\Z
284 {%
285   \expandafter\XINT_ftcx_B\romannumeral0\xintiidivision {#1}{#2}{#2}%
286 }%
287 \def\XINT_ftcx_B #1#2%
288 {%
289   \XINT_ftcx_C #2.{#1}%
290 }%
291 \def\XINT_ftcx_C #1%
292 {%
293   \xint_gob_til_zero #1\XINT_ftcx_integer 0\XINT_ftcx_D #1%
294 }%
295 \def\XINT_ftcx_integer 0\XINT_ftcx_D 0#1.#2#3#4{ #2}%
296 \def\XINT_ftcx_D #1.#2#3#4{\XINT_ftcx_loop_a {#1}{#3}{#1}{#2#4}{#4}}%
297 \def\XINT_ftcx_loop_a
298 {%
299   \expandafter\XINT_ftcx_loop_d\romannumeral0\XINT_div_prepare
300 }%
301 \def\XINT_ftcx_loop_d #1#2%
302 {%
303   \XINT_ftcx_loop_e #2.{#1}%
304 }%
305 \def\XINT_ftcx_loop_e #1%
306 {%
307   \xint_gob_til_zero #1\xint_ftcx_loop_exit0\XINT_ftcx_loop_f #1%
308 }%
309 \def\XINT_ftcx_loop_f #1.#2#3#4#5%
310 {%
311   \XINT_ftcx_loop_a {#1}{#3}{#1}{#4#2#5}{#5}%
312 }%
313 \def\xint_ftcx_loop_exit0\XINT_ftcx_loop_f #1.#2#3#4#5{ #4#2}%

```

10.9 \xintFtoC

New in 1.09m: this is the same as `\xintFtoCx` with empty separator. I had temporarily during preparation of 1.09m removed braces from `\xintFtoCx`, but I recalled later why that was useful (see doc), thus let's just here do `\xintFtoCx {}`

```
314 \def\xintFtoC {\romannumeral0\xintftoc }%
315 \def\xintftoc {\xintftocx {}}%
```

10.10 \xintFtoGC

```
316 \def\xintFtoGC {\romannumeral0\xintftogc }%
317 \def\xintftogc {\xintftocx {+1/}}%
```

10.11 \xintFGtoC

New with 1.09m of 2014/02/26. Computes the common initial coefficients for the two fractions *f* and *g*, and outputs them as a sequence of braced items.

```
318 \def\xintFGtoC {\romannumeral0\xintfgtoc}%
319 \def\xintfgtoc#1%
320 {%
321   \expandafter\XINT_fgtc_a\romannumeral0\xintraawithzeros {#1}\Z
322 }%
323 \def\XINT_fgtc_a #1/#2\Z #3%
324 {%
325   \expandafter\XINT_fgtc_b\romannumeral0\xintraawithzeros {#3}\Z #1/#2\Z { }%
326 }%
327 \def\XINT_fgtc_b #1/#2\Z
328 {%
329   \expandafter\XINT_fgtc_c\romannumeral0\xintiidivision {#1}{#2}{#2}%
330 }%
331 \def\XINT_fgtc_c #1#2#3#4/#5\Z
332 {%
333   \expandafter\XINT_fgtc_d\romannumeral0\xintiidivision
334     {#4}{#5}{#5}{#1}{#2}{#3}%
335 }%
336 \def\XINT_fgtc_d #1#2#3#4#5#6#7%
337 {%
338   \xintifEq {#1}{#4}{\XINT_fgtc_da {#1}{#2}{#3}{#4}}%
339     {\xint_thirdofthree}%
340 }%
341 \def\XINT_fgtc_da #1#2#3#4#5#6#7%
342 {%
343   \XINT_fgtc_e {#2}{#5}{#3}{#6}{#7}{#1}}%
344 }%
345 \def\XINT_fgtc_e #1%
346 {%
347   \xintiiifZero {#1}{\expandafter\xint_firstofone\xint_gobble_iii}%
348     {\XINT_fgtc_f {#1}}%
349 }%
350 \def\XINT_fgtc_f #1#2%
351 {%
352   \xintiiifZero {#2}{\xint_thirdofthree}{\XINT_fgtc_g {#1}{#2}}%
353 }%
354 \def\XINT_fgtc_g #1#2#3%
355 {%
356   \expandafter\XINT_fgtc_h\romannumeral0\XINT_div_prepare {#1}{#3}{#1}{#2}%
357 }%
358 \def\XINT_fgtc_h #1#2#3#4#5%
```

```

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

```

10.12 \xintFtoCC

```

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

```

```

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

```

10.13 \xintCtoF, \xintCstoF

1.09m uses `\xintCSVtoList` on the argument of `\xintCstoF` to allow spaces also before the commas. And the original `\xintCstoF` code became the one of the new `\xintCtoF` dealing with a braced rather than comma separated list.

```

434 \def\xintCstoF {\romannumeral0\xintcstof }%
435 \def\xintcstof #1%
436 {%
437   \expandafter\XINT_ctf_prep \romannumeral0\xintcsvtolist{#1}!%
438 }%
439 \def\xintCtoF {\romannumeral0\xintctof }%
440 \def\xintctof #1%
441 {%
442   \expandafter\XINT_ctf_prep \romannumeral`&&@#1!%
443 }%
444 \def\XINT_ctf_prep
445 {%
446   \XINT_ctf_loop_a 1001%
447 }%
448 \def\XINT_ctf_loop_a #1#2#3#4#5%
449 {%
450   \xint_gob_til_exclam #5\XINT_ctf_end!%
451   \expandafter\XINT_ctf_loop_b
452   \romannumeral0\xintraewithzeros {#5}.{#1}{#2}{#3}{#4}%
453 }%
454 \def\XINT_ctf_loop_b #1/#2.#3#4#5#6%
455 {%

```

```

456 \expandafter\XINT_ctf_loop_c\expandafter
457 {\romannumeral0\XINT_mul_fork #2\xint:#4\xint:}%
458 {\romannumeral0\XINT_mul_fork #2\xint:#3\xint:}%
459 {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#6\xint:}%
460 {\XINT_mul_fork #1\xint:#4\xint:}}%
461 {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#5\xint:}%
462 {\XINT_mul_fork #1\xint:#3\xint:}}%
463 }%
464 \def\XINT_ctf_loop_c #1#2%
465 {%
466 \expandafter\XINT_ctf_loop_d\expandafter {\expandafter{#2}{#1}}%
467 }%
468 \def\XINT_ctf_loop_d #1#2%
469 {%
470 \expandafter\XINT_ctf_loop_e\expandafter {\expandafter{#2}#1}%
471 }%
472 \def\XINT_ctf_loop_e #1#2%
473 {%
474 \expandafter\XINT_ctf_loop_a\expandafter{#2}#1%
475 }%
476 \def\XINT_ctf_end #1.#2#3#4#5{\xintrawithzeros {#2/#3}}% 1.09b removes [0]

```

10.14 \xintiCstoF

```

477 \def\xintiCstoF {\romannumeral0\xinticstof }%
478 \def\xinticstof #1%
479 {%
480 \expandafter\XINT_icstf_prep \romannumeral`&&@#1,!,%
481 }%
482 \def\XINT_icstf_prep
483 {%
484 \XINT_icstf_loop_a 1001%
485 }%
486 \def\XINT_icstf_loop_a #1#2#3#4#5,%
487 {%
488 \xint_gob_til_exclam #5\XINT_icstf_end!%
489 \expandafter
490 \XINT_icstf_loop_b \romannumeral`&&@#5.{#1}{#2}{#3}{#4}%
491 }%
492 \def\XINT_icstf_loop_b #1.#2#3#4#5%
493 {%
494 \expandafter\XINT_icstf_loop_c\expandafter
495 {\romannumeral0\xintiiadd {#5}{\XINT_mul_fork #1\xint:#3\xint:}}%
496 {\romannumeral0\xintiiadd {#4}{\XINT_mul_fork #1\xint:#2\xint:}}%
497 {#2}{#3}%
498 }%
499 \def\XINT_icstf_loop_c #1#2%
500 {%
501 \expandafter\XINT_icstf_loop_a\expandafter {#2}{#1}%
502 }%
503 \def\XINT_icstf_end#1.#2#3#4#5{\xintrawithzeros {#2/#3}}% 1.09b removes [0]

```

10.15 \xintGctoF


```

504 \def\xintGctoF {\romannumeral0\xintgctof }%
505 \def\xintgctof #1%
506 {%
507   \expandafter\XINT_gctf_prep \romannumeral`&&@#1+!/%
508 }%
509 \def\XINT_gctf_prep
510 {%
511   \XINT_gctf_loop_a 1001%
512 }%
513 \def\XINT_gctf_loop_a #1#2#3#4#5+%
514 {%
515   \expandafter\XINT_gctf_loop_b
516   \romannumeral0\xintraewithzeros {#5}.{#1}{#2}{#3}{#4}%
517 }%
518 \def\XINT_gctf_loop_b #1/#2.#3#4#5#6%
519 {%
520   \expandafter\XINT_gctf_loop_c\expandafter
521   {\romannumeral0\XINT_mul_fork #2\xint:#4\xint:}%
522   {\romannumeral0\XINT_mul_fork #2\xint:#3\xint:}%
523   {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#6\xint:}%
524                               {\XINT_mul_fork #1\xint:#4\xint:}}%
525   {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#5\xint:}%
526                               {\XINT_mul_fork #1\xint:#3\xint:}}%
527 }%
528 \def\XINT_gctf_loop_c #1#2%
529 {%
530   \expandafter\XINT_gctf_loop_d\expandafter {\expandafter{#2}{#1}}%
531 }%
532 \def\XINT_gctf_loop_d #1#2%
533 {%
534   \expandafter\XINT_gctf_loop_e\expandafter {\expandafter{#2}#1}%
535 }%
536 \def\XINT_gctf_loop_e #1#2%
537 {%
538   \expandafter\XINT_gctf_loop_f\expandafter {\expandafter{#2}#1}%
539 }%
540 \def\XINT_gctf_loop_f #1#2/%
541 {%
542   \xint_gob_til_exclam #2\XINT_gctf_end!%
543   \expandafter\XINT_gctf_loop_g
544   \romannumeral0\xintraewithzeros {#2}.#1%
545 }%
546 \def\XINT_gctf_loop_g #1/#2.#3#4#5#6%
547 {%
548   \expandafter\XINT_gctf_loop_h\expandafter
549   {\romannumeral0\XINT_mul_fork #1\xint:#6\xint:}%
550   {\romannumeral0\XINT_mul_fork #1\xint:#5\xint:}%
551   {\romannumeral0\XINT_mul_fork #2\xint:#4\xint:}%
552   {\romannumeral0\XINT_mul_fork #2\xint:#3\xint:}%
553 }%
554 \def\XINT_gctf_loop_h #1#2%
555 {%

```

```

556 \expandafter\XINT_gctf_loop_i\expandafter {\expandafter{#2}{#1}}%
557 }%
558 \def\XINT_gctf_loop_i #1#2%
559 {%
560 \expandafter\XINT_gctf_loop_j\expandafter {\expandafter{#2}{#1}}%
561 }%
562 \def\XINT_gctf_loop_j #1#2%
563 {%
564 \expandafter\XINT_gctf_loop_a\expandafter {#2}{#1}%
565 }%
566 \def\XINT_gctf_end #1.#2#3#4#5{\xintrawithzeros {#2/#3}}% 1.09b removes [0]

```

10.16 \xintiGctoF

```

567 \def\xintiGctoF {\romannumeral0\xintigctof }%
568 \def\xintigctof #1%
569 {%
570 \expandafter\XINT_igctf_prep \romannumeral`&&@#1+!/ %
571 }%
572 \def\XINT_igctf_prep
573 {%
574 \XINT_igctf_loop_a 1001%
575 }%
576 \def\XINT_igctf_loop_a #1#2#3#4#5+%
577 {%
578 \expandafter\XINT_igctf_loop_b
579 \romannumeral`&&@#5.{#1}{#2}{#3}{#4}%
580 }%
581 \def\XINT_igctf_loop_b #1.#2#3#4#5%
582 {%
583 \expandafter\XINT_igctf_loop_c\expandafter
584 {\romannumeral0\xintiiadd {#5}{\XINT_mul_fork #1\xint:#3\xint:}}%
585 {\romannumeral0\xintiiadd {#4}{\XINT_mul_fork #1\xint:#2\xint:}}%
586 {#2}{#3}%
587 }%
588 \def\XINT_igctf_loop_c #1#2%
589 {%
590 \expandafter\XINT_igctf_loop_f\expandafter {\expandafter{#2}{#1}}%
591 }%
592 \def\XINT_igctf_loop_f #1#2#3#4/%
593 {%
594 \xint_gob_til_exclam #4\XINT_igctf_end!%
595 \expandafter\XINT_igctf_loop_g
596 \romannumeral`&&@#4.{#2}{#3}#1%
597 }%
598 \def\XINT_igctf_loop_g #1.#2#3%
599 {%
600 \expandafter\XINT_igctf_loop_h\expandafter
601 {\romannumeral0\XINT_mul_fork #1\xint:#3\xint:}%
602 {\romannumeral0\XINT_mul_fork #1\xint:#2\xint:}%
603 }%
604 \def\XINT_igctf_loop_h #1#2%
605 {%
606 \expandafter\XINT_igctf_loop_i\expandafter {#2}{#1}%

```

```

607 }%
608 \def\XINT_igctf_loop_i #1#2#3#4%
609 {%
610   \XINT_igctf_loop_a {#3}{#4}{#1}{#2}%
611 }%
612 \def\XINT_igctf_end #1.#2#3#4#5{\xintrawwithzeros {#4/#5}}% 1.09b removes [0]

```

10.17 \xintCtoCv, \xintCstoCv

1.09m uses \xintCSVtoList on the argument of \xintCstoCv to allow spaces also before the commas. The original \xintCstoCv code became the one of the new \xintCtoF dealing with a braced rather than comma separated list.

```

613 \def\xintCstoCv {\romannumeral0\xintcstocv }%
614 \def\xintcstocv #1%
615 {%
616   \expandafter\XINT_ctcv_prep\romannumeral0\xintcsvtolist{#1}!%
617 }%
618 \def\xintCtoCv {\romannumeral0\xintctocv }%
619 \def\xintctocv #1%
620 {%
621   \expandafter\XINT_ctcv_prep\romannumeral`&&@#1!%
622 }%
623 \def\XINT_ctcv_prep
624 {%
625   \XINT_ctcv_loop_a {}1001%
626 }%
627 \def\XINT_ctcv_loop_a #1#2#3#4#5#6%
628 {%
629   \xint_gob_til_exclam #6\XINT_ctcv_end!%
630   \expandafter\XINT_ctcv_loop_b
631   \romannumeral0\xintrawwithzeros {#6}.{#2}{#3}{#4}{#5}{#1}%
632 }%
633 \def\XINT_ctcv_loop_b #1/#2.#3#4#5#6%
634 {%
635   \expandafter\XINT_ctcv_loop_c\expandafter
636   {\romannumeral0\XINT_mul_fork #2\xint:#4\xint:}%
637   {\romannumeral0\XINT_mul_fork #2\xint:#3\xint:}%
638   {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#6\xint:}%
639     {\XINT_mul_fork #1\xint:#4\xint:}}%
640   {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#5\xint:}%
641     {\XINT_mul_fork #1\xint:#3\xint:}}%
642 }%
643 \def\XINT_ctcv_loop_c #1#2%
644 {%
645   \expandafter\XINT_ctcv_loop_d\expandafter {\expandafter{#2}{#1}}%
646 }%
647 \def\XINT_ctcv_loop_d #1#2%
648 {%
649   \expandafter\XINT_ctcv_loop_e\expandafter {\expandafter{#2}#1}%
650 }%
651 \def\XINT_ctcv_loop_e #1#2%
652 {%
653   \expandafter\XINT_ctcv_loop_f\expandafter{#2}#1%

```

```

654 }%
655 \def\XINT_ctcv_loop_f #1#2#3#4#5%
656 {%
657   \expandafter\XINT_ctcv_loop_g\expandafter
658   {\romannumeral0\xintraawwithzeros {#1/#2}}{#5}{#1}{#2}{#3}{#4}%
659 }%
660 \def\XINT_ctcv_loop_g #1#2{\XINT_ctcv_loop_a {#2{#1}}}% 1.09b removes [0]
661 \def\XINT_ctcv_end #1.#2#3#4#5#6{ #6}%

```

10.18 \xintiCstoCv

```

662 \def\xintiCstoCv {\romannumeral0\xinticstocv }%
663 \def\xinticstocv #1%
664 {%
665   \expandafter\XINT_icstcv_prep \romannumeral`&&@#1,!,%
666 }%
667 \def\XINT_icstcv_prep
668 {%
669   \XINT_icstcv_loop_a {}1001%
670 }%
671 \def\XINT_icstcv_loop_a #1#2#3#4#5#6,%
672 {%
673   \xint_gob_til_exclam #6\XINT_icstcv_end!%
674   \expandafter
675   \XINT_icstcv_loop_b \romannumeral`&&@#6.{#2}{#3}{#4}{#5}{#1}%
676 }%
677 \def\XINT_icstcv_loop_b #1.#2#3#4#5%
678 {%
679   \expandafter\XINT_icstcv_loop_c\expandafter
680   {\romannumeral0\xintiiadd {#5}{\XINT_mul_fork #1\xint:#3\xint:}}%
681   {\romannumeral0\xintiiadd {#4}{\XINT_mul_fork #1\xint:#2\xint:}}%
682   {{#2}{#3}}%
683 }%
684 \def\XINT_icstcv_loop_c #1#2%
685 {%
686   \expandafter\XINT_icstcv_loop_d\expandafter {#2}{#1}%
687 }%
688 \def\XINT_icstcv_loop_d #1#2%
689 {%
690   \expandafter\XINT_icstcv_loop_e\expandafter
691   {\romannumeral0\xintraawwithzeros {#1/#2}}{#1}{#2}}%
692 }%
693 \def\XINT_icstcv_loop_e #1#2#3#4{\XINT_icstcv_loop_a {#4{#1}}#2#3}%
694 \def\XINT_icstcv_end #1.#2#3#4#5#6{ #6}% 1.09b removes [0]

```

10.19 \xintGctoCv

```

695 \def\xintGctoCv {\romannumeral0\xintgctocv }%
696 \def\xintgctocv #1%
697 {%
698   \expandafter\XINT_gctcv_prep \romannumeral`&&@#1+!/%
699 }%
700 \def\XINT_gctcv_prep
701 {%

```

```

702 \XINT_gctcv_loop_a {}1001%
703 }%
704 \def\XINT_gctcv_loop_a #1#2#3#4#5#6+%
705 {%
706 \expandafter\XINT_gctcv_loop_b
707 \romannumeral0\xintraewithzeros {#6}. {#2}{#3}{#4}{#5}{#1}%
708 }%
709 \def\XINT_gctcv_loop_b #1/#2.#3#4#5#6%
710 {%
711 \expandafter\XINT_gctcv_loop_c\expandafter
712 {\romannumeral0\XINT_mul_fork #2\xint:#4\xint:}%
713 {\romannumeral0\XINT_mul_fork #2\xint:#3\xint:}%
714 {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#6\xint:}%
715 {\XINT_mul_fork #1\xint:#4\xint:}}%
716 {\romannumeral0\xintiiadd {\XINT_mul_fork #2\xint:#5\xint:}%
717 {\XINT_mul_fork #1\xint:#3\xint:}}%
718 }%
719 \def\XINT_gctcv_loop_c #1#2%
720 {%
721 \expandafter\XINT_gctcv_loop_d\expandafter {\expandafter{#2}{#1}}%
722 }%
723 \def\XINT_gctcv_loop_d #1#2%
724 {%
725 \expandafter\XINT_gctcv_loop_e\expandafter {\expandafter{#2}{#1}}%
726 }%
727 \def\XINT_gctcv_loop_e #1#2%
728 {%
729 \expandafter\XINT_gctcv_loop_f\expandafter {#2}#1%
730 }%
731 \def\XINT_gctcv_loop_f #1#2%
732 {%
733 \expandafter\XINT_gctcv_loop_g\expandafter
734 {\romannumeral0\xintraewithzeros {#1/#2}}{#1}{#2}}%
735 }%
736 \def\XINT_gctcv_loop_g #1#2#3#4%
737 {%
738 \XINT_gctcv_loop_h {#4{#1}}{#2#3}% 1.09b removes [0]
739 }%
740 \def\XINT_gctcv_loop_h #1#2#3/%
741 {%
742 \xint_gob_til_exclam #3\XINT_gctcv_end!%
743 \expandafter\XINT_gctcv_loop_i
744 \romannumeral0\xintraewithzeros {#3}.#2{#1}%
745 }%
746 \def\XINT_gctcv_loop_i #1/#2.#3#4#5#6%
747 {%
748 \expandafter\XINT_gctcv_loop_j\expandafter
749 {\romannumeral0\XINT_mul_fork #1\xint:#6\xint:}%
750 {\romannumeral0\XINT_mul_fork #1\xint:#5\xint:}%
751 {\romannumeral0\XINT_mul_fork #2\xint:#4\xint:}%
752 {\romannumeral0\XINT_mul_fork #2\xint:#3\xint:}%
753 }%

```

```

754 \def\XINT_gctcv_loop_j #1#2%
755 {%
756   \expandafter\XINT_gctcv_loop_k\expandafter {\expandafter{#2}{#1}}%
757 }%
758 \def\XINT_gctcv_loop_k #1#2%
759 {%
760   \expandafter\XINT_gctcv_loop_l\expandafter {\expandafter{#2}#1}%
761 }%
762 \def\XINT_gctcv_loop_l #1#2%
763 {%
764   \expandafter\XINT_gctcv_loop_m\expandafter {\expandafter{#2}#1}%
765 }%
766 \def\XINT_gctcv_loop_m #1#2{\XINT_gctcv_loop_a {#2}#1}%
767 \def\XINT_gctcv_end #1.#2#3#4#5#6{ #6}%

```

10.20 \xintiGctoCv

```

768 \def\xintiGctoCv {\romannumeral0\xintigctocv }%
769 \def\xintigctocv #1%
770 {%
771   \expandafter\XINT_igctcv_prep \romannumeral`&&@#1+!/%
772 }%
773 \def\XINT_igctcv_prep
774 {%
775   \XINT_igctcv_loop_a {}1001%
776 }%
777 \def\XINT_igctcv_loop_a #1#2#3#4#5#6+%
778 {%
779   \expandafter\XINT_igctcv_loop_b
780   \romannumeral`&&@#6.{#2}{#3}{#4}{#5}{#1}%
781 }%
782 \def\XINT_igctcv_loop_b #1.#2#3#4#5%
783 {%
784   \expandafter\XINT_igctcv_loop_c\expandafter
785   {\romannumeral0\xintiiadd {#5}{\XINT_mul_fork #1\xint:#3\xint:}}%
786   {\romannumeral0\xintiiadd {#4}{\XINT_mul_fork #1\xint:#2\xint:}}%
787   {{#2}{#3}}%
788 }%
789 \def\XINT_igctcv_loop_c #1#2%
790 {%
791   \expandafter\XINT_igctcv_loop_f\expandafter {\expandafter{#2}{#1}}%
792 }%
793 \def\XINT_igctcv_loop_f #1#2#3#4/%
794 {%
795   \xint_gob_til_exclam #4\XINT_igctcv_end_a!%
796   \expandafter\XINT_igctcv_loop_g
797   \romannumeral`&&@#4.#1#2{#3}%
798 }%
799 \def\XINT_igctcv_loop_g #1.#2#3#4#5%
800 {%
801   \expandafter\XINT_igctcv_loop_h\expandafter
802   {\romannumeral0\XINT_mul_fork #1\xint:#5\xint:}%
803   {\romannumeral0\XINT_mul_fork #1\xint:#4\xint:}%
804   {{#2}{#3}}%

```

```

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

```

10.21 \xintFtoCv

Still uses \xinticstocv \xintFtoCs rather than \xintctocv \xintFtoC.

```

823 \def\xintFtoCv {\romannumeral0\xintftocv }%
824 \def\xintftocv #1%
825 {%
826   \xinticstocv {\xintFtoCs {#1}}%
827 }%

```

10.22 \xintFtoCCv

```

828 \def\xintFtoCCv {\romannumeral0\xintftoccv }%
829 \def\xintftoccv #1%
830 {%
831   \xintigctocv {\xintFtoCC {#1}}%
832 }%

```

10.23 \xintCntoF

Modified in 1.06 to give the N first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that.

```

833 \def\xintCntoF {\romannumeral0\xintcntof }%
834 \def\xintcntof #1%
835 {%
836   \expandafter\XINT_cntf\expandafter {\the\numexpr #1}%
837 }%
838 \def\XINT_cntf #1#2%
839 {%
840   \ifnum #1>\xint_c_
841     \xint_afterfi {\expandafter\XINT_cntf_loop\expandafter
842                   {\the\numexpr #1-1\expandafter}\expandafter
843                   {\romannumeral`&&@#2{#1}}{#2}}%
844   \else

```

```

845     \xint_afterfi
846     {\ifnum #1=\xint_c_
847         \xint_afterfi {\expandafter\space \romannumeral`&&@#2{0}}%
848         \else \xint_afterfi { }% 1.09m now returns nothing.
849         \fi}%
850 \fi
851 }%
852 \def\xINT_cntf_loop #1#2#3%
853 {%
854     \ifnum #1>\xint_c_ \else \XINT_cntf_exit \fi
855     \expandafter\xINT_cntf_loop\expandafter
856     {\the\numexpr #1-1\expandafter }\expandafter
857     {\romannumeral0\xintadd {\xintDiv {1[0]}\{#2}\{#3{#1}}}%
858     {#3}%
859 }%
860 \def\xINT_cntf_exit \fi
861     \expandafter\xINT_cntf_loop\expandafter
862     #1\expandafter #2#3%
863 {%
864     \fi\xint_gobble_ii #2%
865 }%

```

10.24 \xintGCntoF

Modified in 1.06 to give the N argument first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that.

```

866 \def\xintGCntoF {\romannumeral0\xintgcntof }%
867 \def\xintgcntof #1%
868 {%
869     \expandafter\xINT_gcntf\expandafter {\the\numexpr #1}%
870 }%
871 \def\xINT_gcntf #1#2#3%
872 {%
873     \ifnum #1>\xint_c_
874         \xint_afterfi {\expandafter\xINT_gcntf_loop\expandafter
875             {\the\numexpr #1-1\expandafter}\expandafter
876             {\romannumeral`&&@#2{#1}\{#2}\{#3}}%
877     \else
878         \xint_afterfi
879         {\ifnum #1=\xint_c_
880             \xint_afterfi {\expandafter\space\romannumeral`&&@#2{0}}%
881             \else \xint_afterfi { }% 1.09m now returns nothing rather than 0/1[0]
882             \fi}%
883     \fi
884 }%
885 \def\xINT_gcntf_loop #1#2#3#4%
886 {%
887     \ifnum #1>\xint_c_ \else \XINT_gcntf_exit \fi
888     \expandafter\xINT_gcntf_loop\expandafter
889     {\the\numexpr #1-1\expandafter }\expandafter
890     {\romannumeral0\xintadd {\xintDiv {#4{#1}}\{#2}\{#3{#1}}}%
891     {#3}\{#4}%

```



```

892 }%
893 \def\XINT_gcntf_exit \fi
894   \expandafter\XINT_gcntf_loop\expandafter
895   #1\expandafter #2#3#4%
896 {%
897   \fi\xint_gobble_ii #2%
898 }%

```

10.25 \xintCntoCs

Modified in 1.09m: added spaces after the commas in the produced list. Moreover the coefficients are not braced anymore. A slight induced limitation is that the macro argument should not contain some explicit comma (cf. `\XINT_cntcs_exit_b`), hence `\xintCntoCs {\macro,}` with `\def\macro,#1{<stuff>}` would crash. Not a very serious limitation, I believe.

```

899 \def\xintCntoCs {\romannumeral0\xintcntocs }%
900 \def\xintcntocs #1%
901 {%
902   \expandafter\XINT_cntcs\expandafter {\the\numexpr #1}%
903 }%
904 \def\XINT_cntcs #1#2%
905 {%
906   \ifnum #1<0
907     \xint_afterfi { }% 1.09i: a 0/1[0] was here, now the macro returns nothing
908   \else
909     \xint_afterfi {\expandafter\XINT_cntcs_loop\expandafter
910                   {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
911                   {\romannumeral`&&@#2{#1}}{#2}}% produced coeff not braced
912   \fi
913 }%
914 \def\XINT_cntcs_loop #1#2#3%
915 {%
916   \ifnum #1>-\xint_c_i \else \XINT_cntcs_exit \fi
917   \expandafter\XINT_cntcs_loop\expandafter
918   {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
919   {\romannumeral`&&@#3{#1}, #2}{#3}% space added, 1.09m
920 }%
921 \def\XINT_cntcs_exit \fi
922   \expandafter\XINT_cntcs_loop\expandafter
923   #1\expandafter #2#3%
924 {%
925   \fi\XINT_cntcs_exit_b #2%
926 }%
927 \def\XINT_cntcs_exit_b #1,{}% romannumeral stopping space already there

```

10.26 \xintCntoGC

Modified in 1.06 to give the N first to a `\numexpr` rather than expanding twice. I just use `\the\numexpr` and maintain the previous code after that.

1.09m maintains the braces, as the coeff are allowed to be fraction and the slash can not be naked in the GC format, contrarily to what happens in `\xintCntoCs`. Also the separators given to `\xintGCtoGCx` may then fetch the coefficients as argument, as they are braced.

```

928 \def\xintCntoGC {\romannumeral0\xintcntogc }%
929 \def\xintcntogc #1%
930 {%
931   \expandafter\XINT_cntgc\expandafter {\the\numexpr #1}%
932 }%
933 \def\XINT_cntgc #1#2%
934 {%
935   \ifnum #1<0
936     \xint_afterfi { }% 1.09i there was as strange 0/1[0] here, removed
937   \else
938     \xint_afterfi {\expandafter\XINT_cntgc_loop\expandafter
939                   {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
940                   {\expandafter{\romannumeral`&&@#2{#1}}{#2}}}%
941   \fi
942 }%
943 \def\XINT_cntgc_loop #1#2#3%
944 {%
945   \ifnum #1>-\xint_c_i \else \XINT_cntgc_exit \fi
946   \expandafter\XINT_cntgc_loop\expandafter
947   {\the\numexpr #1-\xint_c_i\expandafter }\expandafter
948   {\expandafter{\romannumeral`&&@#3{#1}}+1/#2}{#3}%
949 }%
950 \def\XINT_cntgc_exit \fi
951   \expandafter\XINT_cntgc_loop\expandafter
952   #1\expandafter #2#3%
953 {%
954   \fi\XINT_cntgc_exit_b #2%
955 }%
956 \def\XINT_cntgc_exit_b #1+1/{ }%

```

10.27 \xintGCntoGC

Modified in 1.06 to give the N first to a \numexpr rather than expanding twice. I just use \the\numexpr and maintain the previous code after that.

```

957 \def\xintGCntoGC {\romannumeral0\xintgcntogc }%
958 \def\xintgcntogc #1%
959 {%
960   \expandafter\XINT_gcntgc\expandafter {\the\numexpr #1}%
961 }%
962 \def\XINT_gcntgc #1#2#3%
963 {%
964   \ifnum #1<0
965     \xint_afterfi { }% 1.09i now returns nothing
966   \else
967     \xint_afterfi {\expandafter\XINT_gcntgc_loop\expandafter
968                   {\the\numexpr #1-\xint_c_i\expandafter}\expandafter
969                   {\expandafter{\romannumeral`&&@#2{#1}}{#2}{#3}}}%
970   \fi
971 }%
972 \def\XINT_gcntgc_loop #1#2#3#4%
973 {%
974   \ifnum #1>-\xint_c_i \else \XINT_gcntgc_exit \fi

```

```

975 \expandafter\XINT_gcntgc_loop_b\expandafter
976 {\expandafter{\romannumeral`&&@#4{#1}}/#2}{#3{#1}}{#1}{#3}{#4}%
977 }%
978 \def\XINT_gcntgc_loop_b #1#2#3%
979 {%
980 \expandafter\XINT_gcntgc_loop\expandafter
981 {\the\numexpr #3-\xint_c_i \expandafter}\expandafter
982 {\expandafter{\romannumeral`&&@#2}+#1}%
983 }%
984 \def\XINT_gcntgc_exit \fi
985 \expandafter\XINT_gcntgc_loop_b\expandafter #1#2#3#4#5%
986 {%
987 \fi\XINT_gcntgc_exit_b #1%
988 }%
989 \def\XINT_gcntgc_exit_b #1/{ }%

```

10.28 \xintCstoGC

```

990 \def\xintCstoGC {\romannumeral0\xintcstogc }%
991 \def\xintcstogc #1%
992 {%
993 \expandafter\XINT_cstc_prep \romannumeral`&&@#1,!,%
994 }%
995 \def\XINT_cstc_prep #1,{\XINT_cstc_loop_a {{#1}}}%
996 \def\XINT_cstc_loop_a #1#2,%
997 {%
998 \xint_gob_til_exclam #2\XINT_cstc_end!%
999 \XINT_cstc_loop_b {#1}{#2}%
1000 }%
1001 \def\XINT_cstc_loop_b #1#2{\XINT_cstc_loop_a {#1+1/{#2}}}%
1002 \def\XINT_cstc_end!\XINT_cstc_loop_b #1#2{ #1}%

```

10.29 \xintGctoGC

```

1003 \def\xintGctoGC {\romannumeral0\xintgctogc }%
1004 \def\xintgctogc #1%
1005 {%
1006 \expandafter\XINT_gctgc_start \romannumeral`&&@#1+!/%
1007 }%
1008 \def\XINT_gctgc_start {\XINT_gctgc_loop_a {}}%
1009 \def\XINT_gctgc_loop_a #1#2+#3/%
1010 {%
1011 \xint_gob_til_exclam #3\XINT_gctgc_end!%
1012 \expandafter\XINT_gctgc_loop_b\expandafter
1013 {\romannumeral`&&@#2}{#3}{#1}%
1014 }%
1015 \def\XINT_gctgc_loop_b #1#2%
1016 {%
1017 \expandafter\XINT_gctgc_loop_c\expandafter
1018 {\romannumeral`&&@#2}{#1}%
1019 }%
1020 \def\XINT_gctgc_loop_c #1#2#3%
1021 {%
1022 \XINT_gctgc_loop_a {#3{#2}+{#1}}/%

```

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

```
1023 }%
1024 \def\XINT_gctgc_end!\expandafter\XINT_gctgc_loop_b
1025 {%
1026   \expandafter\XINT_gctgc_end_b
1027 }%
1028 \def\XINT_gctgc_end_b #1#2#3{ #3{#1}}%
1029 \XINT_restorecatcodes_endinput%
```

11 Package *xintexpr* implementation

This is release 1.4b of 2020/02/25.

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	311
11.2	Old comments	312
11.3	Catcodes, ε -TeX and reload detection	313
11.4	Package identification	314
11.5	<code>\xintDigits*</code> , <code>\xintSetDigits*</code>	314
11.6	Support for output and transform of nested braced contents as core data type	315
11.6.1	Bracketed list rendering with prettifying of leaves from nested braced contents	315
11.6.2	Flattening nested braced contents	315
11.6.3	Braced contents rendering via a TeX alignment with prettifying of leaves	316
11.6.4	Transforming all leaves within nested braced contents	317
11.7	Top level user TeX interface: <code>\xinteval</code> , <code>\xintfloateval</code> , <code>\xintiieval</code>	318
11.7.1	<code>\xintexpr</code> , <code>\xintiexpr</code> , <code>\xintfloatexpr</code> , <code>\xintiieval</code>	318
11.7.2	<code>\XINT_expr_wrap</code> , <code>\XINT_iieval_wrap</code> , <code>\XINT_flexpr_wrap</code>	320
11.7.3	<code>\XINTexprprint</code> , <code>\XINTiievalprint</code> , <code>\XINTflexprprint</code>	320
11.7.4	<code>\xintthe</code> , <code>\xintthealign</code> , <code>\xinttheexpr</code> , <code>\xinttheiexpr</code> , <code>\xintthefloatexpr</code> , <code>\xinttheiieval</code>	320
11.7.5	<code>\thexintexpr</code> , <code>\thexintiieval</code> , <code>\thexintfloatexpr</code> , <code>\thexintiieval</code>	321
11.7.6	<code>\xintbareeval</code> , <code>\xintbarefloateval</code> , <code>\xintbareiieval</code>	321
11.7.7	<code>\xintthebareeval</code> , <code>\xintthebarefloateval</code> , <code>\xintthebareiieval</code>	321
11.7.8	<code>\xinteval</code> , <code>\xintiieval</code> , <code>\xintfloateval</code> , <code>\xintiieval</code>	322
11.7.9	<code>\xintboolexpr</code> , <code>\XINT_boolexpr_print</code> , <code>\xinttheboolexpr</code> , <code>\thexintboolexpr</code>	322
11.7.10	<code>\xintifboolexpr</code> , <code>\xintifboolfloatexpr</code> , <code>\xintifbooliieval</code>	322
11.7.11	<code>\xintifsgnexpr</code> , <code>\xintifsgnfloatexpr</code> , <code>\xintifsgniieval</code>	322
11.7.12	Small bits we have to put somewhere	322
11.8	Hooks into the numeric parser for usage by the <code>\xintdeffunc</code> symbolic parser	323
11.9	<code>\XINT_expr_getnext</code> : fetch some value then an operator and present them to last waiter with the found operator precedence, then the operator, then the value	324
11.10	<code>\XINT_expr_scan_nbr_or_func</code> : parsing the integer or decimal number or hexa-decimal number or function name or variable name or special hacky things	326
11.10.1	Integral part (skipping zeroes)	327
11.10.2	Fractional part	329
11.10.3	Scientific notation	330
11.10.4	Hexadecimal numbers	331
11.10.5	<code>\XINT_expr_scanfunc</code> : collecting names of functions and variables	333
11.10.6	<code>\XINT_expr_func</code> : dispatch to variable replacement or to function execution	334
11.11	<code>\XINT_expr_op_`</code> : launch function or pseudo-function, or evaluate variable and insert operator of multiplication in front of parenthesized contents	334
11.12	<code>\XINT_expr_op_</code> : replace a variable by its value and then fetch next operator	335
11.13	<code>\XINT_expr_getop</code> : fetch the next operator or closing parenthesis or end of expression	336
11.14	Expansion spanning; opening and closing parentheses	337
11.15	The comma as binary operator	340
11.16	The minus as prefix operator of variable precedence level	340
11.17	The <code>*</code> as Python-like «unpacking» prefix operator	341
11.18	Infix operators	342
11.18.1	<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>	342

11.18.2	.., ..[, and].. for a..b and a..[b]..c syntax	344
11.18.3	<, >, ==, <=, >=, != with Python-like chaining	346
11.18.4	Support macros for .., ..[and]..	348
11.19	Square brackets [] both as a container and a Python slicer	351
11.19.1	[...] as «oneple» constructor	351
11.19.2	[...] brackets and : operator for NumPy-like slicing and item indexing syntax	352
11.19.3	Macro layer implementing indexing and slicing	354
11.20	Support for raw A/B[N]	358
11.21	? as two-way and ?? as three-way «short-circuit» conditionals	358
11.22	! as postfix factorial operator	359
11.23	User defined variables	360
11.23.1	\xintdefvar, \xintdefiivar, \xintdeffloatvar	360
11.23.2	\xintunassignvar	362
11.24	Support for dummy variables	363
11.24.1	\xintnewdummy	363
11.24.2	\xintensuredummy, \xintrestorevariable	364
11.24.3	Checking (without expansion) that a symbolic expression contains correctly nested parentheses	365
11.24.4	Fetching balanced expressions E1, E2 and a variable name Name from E1, Name=E2)	366
11.24.5	Fetching a balanced expression delimited by a semi-colon	366
11.24.6	Low-level support for omit and abort keywords, the break() function, the n++ construct and the semi-colon as used in the syntax of seq(), add(), mul(), iter(), rseq(), iterr(), rrseq(), subsm(), subsn(), ndseq(), ndmap()	366
11.24.7	Reserved dummy variables @, @1, @2, @3, @4, @@, @@(1), ..., @@@, @@@(1), ... for recursions	368
11.25	Pseudo-functions involving dummy variables and generating scalars or sequences	369
11.25.1	Comments	369
11.25.2	subs(): substitution of one variable	371
11.25.3	subsm(): simultaneous independent substitutions	371
11.25.4	subsn(): leaner syntax for nesting (possibly dependent) substitutions	372
11.25.5	seq(): sequences from assigning values to a dummy variable	374
11.25.6	iter()	374
11.25.7	add(), mul()	376
11.25.8	rseq()	377
11.25.9	iterr()	378
11.25.10	rrseq()	379
11.26	Pseudo-functions related to N-dimensional hypercubic lists	380
11.26.1	ndseq()	380
11.26.2	ndmap()	381
11.26.3	ndfillraw()	383
11.27	Other pseudo-functions: bool(), togl(), protect(), qraw(), qint(), qfrac(), qfloat(), grand(), random(), rbit()	383
11.28	Regular built-in functions: num(), reduce(), preduce(), abs(), sgn(), frac(), floor(), ceil(), sqr(), ?(), !(), not(), odd(), even(), isint(), isone(), factorial(), sqrt(), sqrtr(), inv(), round(), trunc(), float(), sfloat(), ilog10(), divmod(), mod(), binomial(), pfactorial(), randrange(), quo(), rem(), gcd(), lcm(), max(), min(), `+`(), `*`(), all(), any(), xor(), len(), first(), last(), reversed(), if(), ifint(), ifone(), ifsgn(), nuple(), unpack(), flat() and zip()	384
11.29	User declared functions	398
11.29.1	\xintdeffunc, \xintdefiifunc, \xintdeffloatfunc	398
11.29.2	\xintdefufunc, \xintdefiifunc, \xintdeffloatufunc	401
11.29.3	\xintunassignexprfunc, \xintunassigniifunc, \xintunassignfloatexprfunc	402

11.29.4	<code>\xintNewFunction</code>	403
11.29.5	Mysterious stuff	404
11.29.6	<code>\XINT_expr_redefinemacros</code>	416
11.29.7	<code>\xintNewExpr</code> , <code>\xintNewIExpr</code> , <code>\xintNewFloatExpr</code> , <code>\xintNewIIExpr</code>	417
11.29.8	<code>\ifxintexprsafecatcodes</code> , <code>\xintexprSafeCatcodes</code> , <code>\xintexprRestoreCatcodes</code> . .	419

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 development 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 [2020/02/25 v1.4b Expandable expression parser (JFB)]%
59 \catcode`! 11
60 \let\XINT_Cmp \xintiiCmp
61 \def\XINTfstop{\noexpand\XINTfstop}%

```

11.5 *\xintDigits**, *\xintSetDigits**

1.3f

```

62 \def\xintDigits {\futurelet\XINT_token\xintDigitss}%
63 \def\xintDigitss #1={\afterassignment\xintDigitsss\mathchardef\XINTdigits=}%
64 \def\xintDigitsss#1{\ifx*\XINT_token\expandafter\xintreloadxinttrig\fi}%
65 \let\xintfracSetDigits\xintSetDigits
66 \def\xintSetDigits#1#{\if\relax\detokenize{#1}\relax
67   \else\afterassignment\xintreloadxinttrig\fi
68   \xintfracSetDigits}%

```

11.6 Support for output and transform of nested braced contents as core data type

New at 1.4, of course. The former `\csname.=... \endcsname` encapsulation technique made very difficult implementation of nested structures.

11.6.1 Bracketed list rendering with prettifying of leaves from nested braced contents

1.4 The braces in `\XINT:expr:toblistwith` are there because there is an `\expanded` trigger.

```

69 \def\XINT:expr:toblistwith#1#2%
70 {%
71     {\expandafter\XINT:expr:toblist_checkempty
72         \expanded{\noexpand#1!\expandafter}\detokenize{#2}^}%
73 }%
74 \def\XINT:expr:toblist_checkempty #1!#2%
75 {%
76     \if ^#2\expandafter\xint_gob_til_^ \else\expandafter\XINT:expr:toblist_a\fi
77     #1!#2%
78 }%
79 \catcode`< 1 \catcode`> 2 \catcode`{ 12 \catcode`} 12
80 \def\XINT:expr:toblist_a #1{#2%
81 <%
82     \if{#2\xint_dothis<[\XINT:expr:toblist_a]\fi
83     \xint_orthat\XINT:expr:toblist_b #1#2%
84 >%
85 \def\XINT:expr:toblist_b #1!#2}%
86 <%
87     \if\relax#2\relax\xintexprEmptyItem\else#1<#2>\fi\XINT:expr:toblist_c #1!}%
88 >%
89 \def\XINT:expr:toblist_c #1}{#2%
90 <%
91     \if ^#2\xint_dothis<\xint_gob_til_^>\fi
92     \if{#2\xint_dothis<, \XINT:expr:toblist_a>\fi
93     \xint_orthat<]\XINT:expr:toblist_c>#1#2%
94 >%
95 \catcode`{ 1 \catcode`} 2 \catcode`< 12 \catcode`> 12

```

11.6.2 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 `toblistwith`, 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.

```

96 \def\XINT:expr:flatten#1%
97 {%
98     {\expandafter\XINT:expr:flatten_checkempty\detokenize{#1}^}%
99 }%
100 \def\XINT:expr:flatten_checkempty #1%
101 {%
102     \if ^#1\expandafter\xint_gobble_i\else\expandafter\XINT:expr:flatten_a\fi

```

```

103 #1%
104 }%
105 \beginngroup % should I check lcode s generally if corrupted context at load?
106 \catcode`[ 1 \catcode`] 2 \lccode`[`{ \lccode`]` }
107 \catcode`< 1 \catcode`> 2 \catcode`{ 12 \catcode`} 12
108 \lowercase<\endgroup
109 \def\XINT:expr:flatten_a {#1%
110 <%
111 \if{#1\xint_dothis<\XINT:expr:flatten_a>\fi
112 \xint_orthat\XINT:expr:flatten_b #1%
113 >%
114 \def\XINT:expr:flatten_b #1}%
115 <%
116 [#1]\XINT:expr:flatten_c }%
117 >%
118 \def\XINT:expr:flatten_c }#1%
119 <%
120 \if ^#1\xint_dothis<\xint_gobble_i>\fi
121 \if{#1\xint_dothis<\XINT:expr:flatten_a>\fi
122 \xint_orthat<\XINT:expr:flatten_c>#1%
123 >%
124 >% back to normal catcodes

```

11.6.3 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 `\xintexpralignntab` nor `\xintexpraligninnercomma` or `\xintexpralignoutercomma` but `\xintexpraligninnersep`, etc... .

```

125 \catcode`& 4
126 \protected\def\xintexpralignbegin { \halign\bgroup\tabskip2ex\hfil##&&##\hfil\cr}%
127 \protected\def\xintexpralignend { \crcr\egroup}%
128 \protected\def\xintexpralignlinesep { ,\cr}%
129 \protected\def\xintexpralignleftbracket { [ ]%
130 \protected\def\xintexpralignrightbracket { } ]%
131 \protected\def\xintexpralignleftsep { & }%
132 \protected\def\xintexpralignrightsep { & }%
133 \protected\def\xintexpraligninnersep { , & }%
134 \catcode`& 7
135 \def\XINT:expr:toalignwith#1#2%
136 {%
137 { \expandafter\XINT:expr:toalign_checkempty
138 \expanded{\noexpand#1!\expandafter}\detokenize{#2}^\expandafter}%
139 \xintexpralignend
140 }%
141 \def\XINT:expr:toalign_checkempty #1!#2%
142 {%
143 \if ^#2\expandafter\xint_gob_til_^\else\expandafter\XINT:expr:toalign_a\fi
144 #1!#2%
145 }%
146 \catcode`< 1 \catcode`> 2 \catcode`{ 12 \catcode`} 12
147 \def\XINT:expr:toalign_a #1{#2%

```

```

148 <%
149   \if{#2\xint_dothis<\xintexpralignleftbracket\XINT:expr:toalign_a>\fi
150   \xint_orthat<\xintexpralignleftsep\XINT:expr:toalign_b>#1#2%
151 >%
152 \def\XINT:expr:toalign_b #1!#2}%
153 <%
154   \if\relax#2\relax\xintexprEmptyItem\else#1<#2>\fi\XINT:expr:toalign_c #1!}%
155 >%
156 \def\XINT:expr:toalign_c #1}#2%
157 <%
158   \if ^#2\xint_dothis<\xint_gob_til_^>\fi
159   \if {#2\xint_dothis<\xintexpralignninnersep\XINT:expr:toalign_A>\fi
160   \xint_orthat<\xintexpralignrightsep\xintexpralignrightbracket\XINT:expr:toalign_C>#1#2%
161 >%
162 \def\XINT:expr:toalign_A #1{#2%
163 <%
164   \if{#2\xint_dothis<\xintexpralignleftbracket\XINT:expr:toalign_A>\fi
165   \xint_orthat\XINT:expr:toalign_b #1#2%
166 >%
167 \def\XINT:expr:toalign_C #1}#2%
168 <%
169   \if ^#2\xint_dothis<\xint_gob_til_^>\fi
170   \if {#2\xint_dothis<\xintexpralignlinesep\XINT:expr:toalign_a>\fi
171   \xint_orthat<\xintexpralignrightbracket\XINT:expr:toalign_C>#1#2%
172 >%
173 \catcode`{ 1 \catcode`} 2 \catcode`< 12 \catcode`> 12

```

11.6.4 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 into braces :

```
\def\foo#1{\xintiRound{0}{#1}}
```

As the things will expand inside expanded, propagating expansion is not an issue.

This code is used by \xintiexpr and \xintfloatexpr in case of optional argument and by the «Universal functions».

```

174 \def\XINT:expr:mapwithin#1#2%
175 {%
176   {\expandafter\XINT:expr:mapwithin_checkempty
177   \expanded{\noexpand#1!\expandafter}\detokenize{#2}^}%
178 }%
179 \def\XINT:expr:mapwithin_checkempty #1!#2%
180 {%
181   \if ^#2\expandafter\xint_gob_til_^ \else\expandafter\XINT:expr:mapwithin_a\fi
182   #1!#2%
183 }%
184 \begingroup % should I check lccode s generally if corrupted context at load?
185 \catcode`[ 1 \catcode`] 2 \lccode`[` \lccode`]`

```

```

186 \catcode`< 1 \catcode`> 2 \catcode`{ 12 \catcode`} 12
187 \lowercase<\endgroup
188 \def\XINT:expr:mapwithin_a #1{#2%
189 <%
190   \if{#2\xint_dothis<[\iffalse]\fi\XINT:expr:mapwithin_a>\fi%
191   \xint_orthat\XINT:expr:mapwithin_b #1#2%
192 >%
193 \def\XINT:expr:mapwithin_b #1!#2}%
194 <%
195   #1<#2>\XINT:expr:mapwithin_c #1!}%
196 >%
197 \def\XINT:expr:mapwithin_c #1}#2%
198 <%
199   \if ^#2\xint_dothis<\xint_gob_til_^>\fi
200   \if{#2\xint_dothis<\XINT:expr:mapwithin_a>\fi%
201   \xint_orthat<[\iffalse[\fi]\XINT:expr:mapwithin_c>#1#2%
202 >%
203 >% back to normal catcodes

```

11.7 Top level user T_EX interface: `\xinteval`, `\xintfloateval`, `\xintiieval`

11.7.1	<code>\xintexpr</code> , <code>\xintiexpr</code> , <code>\xintfloatexpr</code> , <code>\xintiiepr</code>	318
11.7.2	<code>\XINT_expr_wrap</code> , <code>\XINT_iiexpr_wrap</code> , <code>\XINT_flexpr_wrap</code>	320
11.7.3	<code>\XINTexprprint</code> , <code>\XINTiiexprprint</code> , <code>\XINTflexprprint</code>	320
11.7.4	<code>\xintthe</code> , <code>\xintthealign</code> , <code>\xinttheexpr</code> , <code>\xinttheiexpr</code> , <code>\xintthefloatexpr</code> , <code>\xint-</code> <code>theiiepr</code>	320
11.7.5	<code>\thexintexpr</code> , <code>\thexintiexpr</code> , <code>\thexintfloatexpr</code> , <code>\thexintiiepr</code>	321
11.7.6	<code>\xintbareeval</code> , <code>\xintbarefloateval</code> , <code>\xintbareiieval</code>	321
11.7.7	<code>\xintthebareeval</code> , <code>\xintthebarefloateval</code> , <code>\xintthebareiieval</code>	321
11.7.8	<code>\xinteval</code> , <code>\xintiieval</code> , <code>\xintfloateval</code> , <code>\xintiieval</code>	322
11.7.9	<code>\xintboolexpr</code> , <code>\XINT_boolexpr_print</code> , <code>\xinttheboolexpr</code> , <code>\thexintboolexpr</code> . . .	322
11.7.10	<code>\xintifboolexpr</code> , <code>\xintifboolfloatexpr</code> , <code>\xintifbooliiepr</code>	322
11.7.11	<code>\xintifsgnexpr</code> , <code>\xintifsgnfloatexpr</code> , <code>\xintifsgniiepr</code>	322
11.7.12	Small bits we have to put somewhere	322

11.7.1 `\xintexpr`, `\xintiexpr`, `\xintfloatexpr`, `\xintiiepr`

`\xintiexpr` and `\xintfloatexpr` have an optional argument since 1.1.

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

The 1.4a version of optional argument [D] for `\xintiexpr` accepts a negative D, with same meaning as `\xintTrunc` from `xintfrac.sty`.

```

204 \def\xintexpr      {\romannumeral0\xintexpr0    }%
205 \def\xintiexpr     {\romannumeral0\xintiexpr0   }%
206 \def\xintfloatexpr {\romannumeral0\xintfloatexpr0}%
207 \def\xintiiepr     {\romannumeral0\xintiiepr0   }%
208 \def\xintexpr0     {\expandafter\XINT_expr_wrap\romannumeral0\xintbareeval }%
209 \def\xintiiepr0    {\expandafter\XINT_iiexpr_wrap\romannumeral0\xintbareiieval }%
210 \def\xintiiepr0 #1%
211 {%
212   \ifx [#1\expandafter\XINT_iiexpr_withopt\else\expandafter\XINT_iiexpr_noopt
213   \fi #1%
214 }%

```

```

215 \def\XINT_iexpr_noopt
216 {%
217   \expandafter\XINT_iexpr_iiround\romannumeral0\xintbareeval
218 }%
219 \def\XINT_iexpr_iiround
220 {%
221   \expandafter\XINT_expr_wrap
222   \expanded
223   \XINT:Nhook:x:mapwithin\XINT:expr:mapwithin{\XINTbracediRoundzero}%
224 }%
225 \def\XINTbracediRoundzero#1{{\xintiRound{0}{#1}}}%
226 \def\XINT_iexpr_withopt [#1]%
227 {%
228   \expandafter\XINT_iexpr_round
229   \the\numexpr \xint_zapspaces #1 \xint_gobble_i\expandafter.%
230   \romannumeral0\xintbareeval
231 }%
232 \def\XINT_iexpr_round #1.%
233 {%
234   \ifnum#1=\xint_c_\xint_dothis{\XINT_iexpr_iiround}\fi
235   \xint_orthat{\XINT_iexpr_round_a #1.}%
236 }%
237 \def\XINT_iexpr_round_a #1.%
238 {%
239   \expandafter\XINT_expr_wrap
240   \expanded
241   \XINT:Nhook:x:mapwithin\XINT:expr:mapwithin{\XINTbracedRound{#1}}%
242 }%
243 \def\XINTbracedRound#1#2{{\xintRound{#1}{#2}}}%
244 \def\xintfloatexpro #1%
245 {%
246   \ifx [#1\expandafter\XINT_flexpr_withopt\else\expandafter\XINT_flexpr_noopt
247   \fi #1%
248 }%
249 \def\XINT_flexpr_noopt
250 {%
251   \expandafter\XINT_flexpr_wrap\the\numexpr\XINTdigits\expandafter.%
252   \romannumeral0\xintbarefloateval
253 }%
254 \def\XINT_flexpr_withopt [#1]%
255 {%
256   \expandafter\XINT_flexpr_withopt_a
257   \the\numexpr\xint_zapspaces #1 \xint_gobble_i\expandafter.%
258   \romannumeral0\xintbarefloateval
259 }%
260 \def\XINT_flexpr_withopt_a #1#2.%
261 {%
262   \expandafter\XINT_flexpr_withopt_b\the\numexpr\if#1-\XINTdigits\fi#1#2.%
263 }%
264 \def\XINT_flexpr_withopt_b #1.%
265 {%
266   \expandafter\XINT_flexpr_wrap

```

```

267 \the\numexpr#1\expandafter.%
268 \expanded
269 \XINT:NEhook:x:mapwithin\XINT:expr:mapwithin{\XINTbracedinFloat[#1]}%
270 }%
271 \def\XINTbracedinFloat[#1]#2{{\XINTinFloat[#1]{#2}}}%

```

11.7.2 \XINT_expr_wrap, \XINT_iexpr_wrap, \XINT_flexpr_wrap

1.3e removes some leading space tokens which served nothing. There is no `\XINT_iexpr_wrap`, because `\XINT_expr_wrap` is used directly.

```

272 \def\XINT_expr_wrap {\XINTfstop\XINTexprprint.}%
273 \def\XINT_iexpr_wrap {\XINTfstop\XINTiexprprint.}%
274 \def\XINT_flexpr_wrap {\XINTfstop\XINTflexprprint}%

```

11.7.3 \XINTexprprint, \XINTiexprprint, \XINTflexprprint

Comments currently under reconstruction.

1.4: The reason for `\expanded` is to ensure `\xintthe` mechanism does expand completely in two steps, now that the print helper macros are not f-expandable anymore.

It is possible that the expression gave an empty object (e.g. `\xintexpr [4][1]\relax`). Thus we do not use `\romannumeral^^@` trigger else twice expansion of `\xinttheexpr` will propagate beyond empty expression. Thus `\romannumeral` ended via a chardef zero token.

```

275 \protected\def\XINTexprprint.%
276 {\XINT:NEhook:x:toblist\XINT:expr:toblistwith\xintexprPrintOne}%
277 \let\xintexprPrintOne\xintFracToSci
278 \def\xintexprEmptyItem{[]}%
279 \protected\def\XINTiexprprint.%
280 {\XINT:NEhook:x:toblist\XINT:expr:toblistwith\xintiexprPrintOne}%
281 \let\xintiexprPrintOne\xint_firstofone
282 \protected\def\XINTflexprprint #1.%
283 {\XINT:NEhook:x:toblist\XINT:expr:toblistwith{\xintfloatexprPrintOne{#1}}}%
284 \def\xintfloatexprPrintOne#1%
285 {\romannumeral0\XINT_pfloat_opt [\xint:#1]}% bad direct jump
286 \protected\def\XINTboolexprprint.%
287 {\XINT:NEhook:x:toblist\XINT:expr:toblistwith\xintboolexprPrintOne}%
288 \def\xintboolexprPrintOne#1{\xintiifNotZero{#1}{True}{False}}%

```

11.7.4 \xintthe, \xintthealign, \xinttheexpr, \xinttheiexpr, \xintthefloatexpr, \xinttheiexpr

The reason why `\xinttheiexpr` et `\xintthefloatexpr` are handled differently is that they admit an optional argument which acts via a custom «printing» stage.

We exploit here that `\expanded` expands forward until finding an implicit or explicit brace, and that this expansion overrules `\protected` macros, forcing them to expand, similarly as `\romannumeral` expands `\protected` macros, and contrarily to what happens *within* the actual `\expanded` scope. I discovered this fact by testing (with `pdftex`) and I don't know where this is documented apart from the source code of the relevant engines. This is useful to us because there are contexts where we will want to apply a complete expansion before printing, but in purely numerical context this is not needed (if I converted correctly after dropping at 1.4 the `\csname` governed expansions; however I rely at various places on the fact that the `xint` macros are f-expandable, so I have tried to not use zillions of `\expanded` all over the place), hence it is not needed to add the expansion overhead by default. But the `\expanded` here will allow `\xintNewExpr` to create

macro with suitable modification or the printing step, via some hook rather than having to duplicate all macros here with some new «NE» meaning (aliasing does not work or causes big issues due to desire to support `\xinteval` also in «NE» context as sub-constituent. The `\XINT:NEhook:x:toblist` is something else which serves to achieve this support of *sub* `\xinteval`, it serves nothing for the actual produced macros. For `\xintdeffunc`, things are simpler, but still we support the [N] optional argument of `\xintexpr` and `\xintfloatexpr`, which required some work...

```

289 \def\xintthe      #1{\expanded\expandafter\xint_gobble_i\romannumeral`&&@#1}%
290 \def\xintthealign #1{\expandafter\xintexpralignbegin
291         \expanded\expandafter\XINT:expr:toalignwith
292         \romannumeral0\expandafter\expandafter\expandafter\expandafter
293         \expandafter\expandafter\expandafter\xint_gob_andstop_ii
294         \expandafter\xint_gobble_i\romannumeral`&&@#1}%
295 \def\xinttheexpr
296   {\expanded\expandafter\XINTexprprint\expandafter.\romannumeral0\xintbareeval}%
297 \def\xinttheiexpr
298   {\expanded\expandafter\xint_gobble_i\romannumeral`&&@\xintexpr}%
299 \def\xintthefloatexpr
300   {\expanded\expandafter\xint_gobble_i\romannumeral`&&@\xintfloatexpr}%
301 \def\xinttheiiexpr
302   {\expanded\expandafter\XINTiiexprprint\expandafter.\romannumeral0\xintbareiieval}%

```

11.7.5 `\thexintexpr`, `\thexintiexpr`, `\thexintfloatexpr`, `\thexintiexpr`

New with 1.2h. I have been for the last three years very strict regarding macros with `\xint` or `\XINT`, but well.

1.4. Definitely I don't like those. I will remove them at 1.5.

```

303 \let\thexintexpr    \xinttheexpr
304 \let\thexintiexpr  \xinttheiexpr
305 \let\thexintfloatexpr\xintthefloatexpr
306 \let\thexintiexpr  \xinttheiiexpr

```

11.7.6 `\xintbareeval`, `\xintbarefloateval`, `\xintbareiieval`

At 1.4 added one expansion step via `_start` macros. Triggering is expected to be via either `\romannumeral`^^@` or `\romannumeral0` is also ok

```

307 \def\xintbareeval   {\XINT_expr_start }%
308 \def\xintbarefloateval{\XINT_flexpr_start}%
309 \def\xintbareiieval {\XINT_iiexpr_start}%

```

11.7.7 `\xintthebareeval`, `\xintthebarefloateval`, `\xintthebareiieval`

For matters of `\XINT_NewFunc`

```

310 \def\xintthebareeval   {\romannumeral0\expandafter\xint_stop_atfirstofone\romannumeral0\xintbareeval}%
311 \def\xintthebarefloateval {\romannumeral0\expandafter\xint_stop_atfirstofone\romannumeral0\xintbarefloateval}%
312 \def\xintthebareiieval {\romannumeral0\expandafter\xint_stop_atfirstofone\romannumeral0\xintbareiieval}%

```

11.7.8 `\xinteval`, `\xintieval`, `\xintfloateval`, `\xintiieval`

```

313 \def\xinteval #1%
314   {\expanded\expandafter\XINTexprprint\expandafter.\romannumeral0\xintbareeval#1\relax}%
315 \def\xintieval #1%
316   {\expanded\expandafter\xint_gobble_i\romannumeral`&&\xintiexpr#1\relax}%
317 \def\xintfloateval #1%
318   {\expanded\expandafter\xint_gobble_i\romannumeral`&&\xintfloatexpr#1\relax}%
319 \def\xintiieval #1%
320   {\expanded\expandafter\XINTiexprprint\expandafter.\romannumeral0\xintbareieval#1\relax}%

```

11.7.9 `\xintboolexpr`, `\XINT_boolexpr_print`, `\xinttheboolexpr`, `\thexintboolexpr`

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

Attention, the conversion to 1 or 0 is done only by the print macro. Perhaps I should force it also inside raw result.

```

321 \def\xintboolexpr
322 {%
323   \romannumeral0\expandafter\XINT_boolexpr_done\romannumeral0\xintexpr
324 }%
325 \def\XINT_boolexpr_done #1.{\XINTfstop\XINTboolexprprint.}%
326 \def\xinttheboolexpr
327 {%
328   \expanded\expandafter\XINTboolexprprint\expandafter.\romannumeral0\xintbareeval
329 }%
330 \let\thexintboolexpr\xinttheboolexpr

```

11.7.10 `\xintifboolexpr`, `\xintifboolfloatexpr`, `\xintifbooliiexpr`

They do not accept comma separated expressions input.

```

331 \def\xintifboolexpr      #1{\romannumeral0\xintiiifnotzero {\xinttheexpr #1\relax}}%
332 \def\xintifboolfloatexpr #1{\romannumeral0\xintiiifnotzero {\xintthefloatexpr #1\relax}}%
333 \def\xintifbooliiexpr   #1{\romannumeral0\xintiiifnotzero {\xinttheiiexpr #1\relax}}%

```

11.7.11 `\xintifsgnexpr`, `\xintifsgnfloatexpr`, `\xintifsgniiexpr`

1.3d.

They do not accept comma separated expressions.

```

334 \def\xintifsgnexpr      #1{\romannumeral0\xintiiifsgn {\xinttheexpr #1\relax}}%
335 \def\xintifsgnfloatexpr #1{\romannumeral0\xintiiifsgn {\xintthefloatexpr #1\relax}}%
336 \def\xintifsgniiexpr   #1{\romannumeral0\xintiiifsgn {\xinttheiiexpr #1\relax}}%

```

11.7.12 Small bits we have to put somewhere

Some renaming and modifications here with release 1.2 to switch from using chains of `\romannumeral-`0` in order to gather numbers, possibly hexadecimal, to using a `\csname` governed expansion. In this way no more limit at 5000 digits, and besides this is a logical move because the `\xintexpr` parser is already based on `\csname... \endcsname` storage of numbers as one token.

The limitation at 5000 digits didn't worry me too much because it was not very realistic to launch computations with thousands of digits... such computations are still slow with 1.2 but less so now.

Chains or `\romannumeral` are still used for the gathering of function names and other stuff which I have half-forgotten because the parser does many things.

In the earlier versions we used the `lockscan` macro after a chain of `\romannumeral-`0` had ended gathering digits; this uses has been replaced by direct processing inside a `\csname...\endcsname` and the macro is kept only for matters of dummy variables.

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

```
337 \def\xINT_embrace#1{{#1}}%
338 \def\xint_gob_til_! #1!{}% ! with catcode 11
339 \def\xintError:noopening
340 {}%
341 \XINT_expandableerror{Extra ) found during balancing, e(X)it before the worst.}%
342 }%
```

`\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!

```
343 \def\xintthecoords#1%
344 {\romannumeral`&&@\expandafter\xINT_thecoords_a\romannumeral0#1}%
345 \def\xINT_thecoords_a #1#2.#3% #2.=\XINTfloatprint<digits>. etc...
346 {\expanded{\expandafter\xINT_thecoords_b\expanded#2.{#3},!,!,^}%
347 \def\xINT_thecoords_b #1#2,#3#4,%
348 {\xint_gob_til_! #3\xINT_thecoords_c ! (#1#2, #3#4)\XINT_thecoords_b }%
349 \def\xINT_thecoords_c #1^{}%
```

`\xintthespaceseparated` 1.4a This is a utility macro which was distributed previously separately for usage with PSTricks `\listplot`

```
350 \def\xintthespaceseparated#1%
351 {\romannumeral`&&@\expandafter\xintthespaceseparated_a\romannumeral0#1}%
352 \def\xintthespaceseparated_a #1#2.#3%
353 {\expanded{\expandafter\xintthespaceseparated_b\expanded#2.{#3},!,!,!,!,!,!,!,!,!,^}%
354 \def\xintthespaceseparated_b #1,#2,#3,#4,#5,#6,#7,#8,#9,%
355 {\xint_gob_til_! #9\xintthespaceseparated_c !%
356 #1#2#3#4#5#6#7#8#9%
357 \xintthespaceseparated_b}%
358 \def\xintthespaceseparated_c !#1!#2^{#1}%
```

11.8 Hooks into the numeric parser for usage by the `\xintdeffunc` symbolic parser

This is new with 1.3 and considerably refactored at 1.4. See «Mysterious stuff».

```

359 \let\XINT:NEhook:f:one:from:one\expandafter
360 \let\XINT:NEhook:f:one:from:one:direct\empty
361 \let\XINT:NEhook:f:one:from:two\expandafter
362 \let\XINT:NEhook:f:one:from:two:direct\empty
363 \let\XINT:NEhook:x:one:from:two\empty
364 \let\XINT:NEhook:x:one:from:twoandone\empty
365 \let\XINT:NEhook:f:one:and:opt:direct \empty
366 \let\XINT:NEhook:f:tacitzeroifone:direct \empty
367 \let\XINT:NEhook:f:iitacitzeroifone:direct \empty
368 \let\XINT:NEhook:x:select:obey\empty
369 \let\XINT:NEhook:x:listsel\empty
370 \let\XINT:NEhook:f:reverse\empty
371 \def\XINT:NEhook:f:from:delim:u #1#2^{#1#2}%
372 \def\XINT:NEhook:f:noeval:from:braced:u#1#2^{#1{#2}}%
373 \let\XINT:NEhook:branch\expandafter
374 \let\XINT:NEhook:seqx\empty
375 \let\XINT:NEhook:iter\expandafter
376 \let\XINT:NEhook:opx\empty
377 \let\XINT:NEhook:rseq\expandafter
378 \let\XINT:NEhook:iterr\expandafter
379 \let\XINT:NEhook:rrseq\expandafter
380 \let\XINT:NEhook:x:toblist\empty
381 \let\XINT:NEhook:x:mapwithin\empty
382 \let\XINT:NEhook:x:ndmapx\empty

```

11.9 `\XINT_expr_getnext`: fetch some value then an operator and present them to last waiter with the found operator precedence, then the operator, then the value

Big change in 1.1, no attempt to detect braced stuff anymore as the [N] notation is implemented otherwise. Now, braces should not be used at all; one level removed, then `\romannumeral`0` expansion.

Refactored at 1.4 to put expansion of `\XINT_expr_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.

```

383 \def\XINT_expr_getnext #1%
384 {%
385   \expandafter\XINT_expr_put_op_first\romannumeral`&&@%
386   \expandafter\XINT_expr_getnext_a\romannumeral`&&@#1%
387 }%
388 \def\XINT_expr_put_op_first #1#2#3{\expandafter#2\expandafter#3\expandafter{#1}}%
389 \def\XINT_expr_getnext_a #1%
390 {%
391   \ifx\relax #1\xint_dothis\XINT_expr_foundprematuereend\fi
392   \ifx\XINTfstop#1\xint_dothis\XINT_expr_subexpr\fi

```

```

393   \ifcat\relax#1\xint_dothis\XINT_expr_countetc\fi
394   \xint_orthat{\XINT_expr_getnextfork #1%
395 }%
396 \def\XINT_expr_foundprematuereend\XINT_expr_getnextfork #1{{}\xint_c_\relax}%
397 \def\XINT_expr_subexpr #1.#2%
398 {%
399   \expanded{\unexpanded{{#2}}\expandafter}\romannumeral`&&\XINT_expr_getop
400 }%

```

1.2 adds \ht, \dp, \wd and the eTeX font things. 1.4 avoids big nested \if's, simply for code readability

```

401 \def\XINT_expr_countetc\XINT_expr_getnextfork#1%
402 {%
403   \if0\ifx\count#11\fi
404     \ifx\dimen#11\fi
405     \ifx\numexpr#11\fi
406     \ifx\dimexpr#11\fi
407     \ifx\skip#11\fi
408     \ifx\glueexpr#11\fi
409     \ifx\fontdimen#11\fi
410     \ifx\ht#11\fi
411     \ifx\dp#11\fi
412     \ifx\wd#11\fi
413     \ifx\fontcharht#11\fi
414     \ifx\fontcharwd#11\fi
415     \ifx\fontchardp#11\fi
416     \ifx\fontcharic#11\fi 0\expandafter\XINT_expr_fetch_as_number\fi
417   \expandafter\XINT_expr_getnext_a\number #1%
418 }%
419 \def\XINT_expr_fetch_as_number
420   \expandafter\XINT_expr_getnext_a\number #1%
421 {%
422   \expanded{{{\number#1}}\expandafter}\romannumeral`&&\XINT_expr_getop
423 }%

```

This is a key component which is involved in:

- support for \xintdeffunc via special handling of parameter character,
- support for skipping over ignored + signs,
- support for Python-like * «unpacking» unary operator (added at 1.4),
- support for [...] nutple constructor (1.4, formerly [...] by itself was like (...)),
- support for numbers starting with a decimal point,
- support for the minus as unary operator of variable precedence level,
- support for sub-expressions inside parenthesis (with possibly tacit multiplication)
- else starting the scan of explicit digits or letters for a number or a function name

```

424 \begingroup
425 \lccode`;=#
426 \lowercase{\endgroup
427 \def\XINT_expr_getnextfork #1{%
428   \if#1;\xint_dothis {\XINT_expr_scan_macropar ;}\fi
429   \if#1+\xint_dothis \XINT_expr_getnext_a \fi
430   \if#1*\xint_dothis {{}\xint_c_ii^v 0}\fi
431   \if#1[\xint_dothis {{}\xint_c_ii^v \XINT_expr_itself_obracket}\fi

```

```

432 \if#1.\xint_dothis {\XINT_expr_startdec}\fi
433 \if#1-\xint_dothis {{{}}-}\fi
434 \if#1(\xint_dothis {{{}\xint_c_ii^v (}\fi
435 \xint_orthat {\XINT_expr_scan_nbr_or_func #1}%
436 }}%
437 \def\xint_expr_scan_macropar #1#2%
438 {%
439 \expandafter{\expandafter{\expandafter#1\expandafter
440 #2\expandafter}\expandafter}\romannumeral`&&\XINT_expr_getop
441 }%

```

11.10 \XINT_expr_scan_nbr_or_func: parsing the integer or decimal number or hexa-decimal number or function name or variable name or special hacky things

11.10.1	Integral part (skipping zeroes)	327
11.10.2	Fractional part	329
11.10.3	Scientific notation	330
11.10.4	Hexadecimal numbers	331
11.10.5	\XINT_expr_scanfunc: collecting names of functions and variables	333
11.10.6	\XINT_expr_func: dispatch to variable replacement or to function execution	334

1.2 release has replaced chains of \romannumeral-`0 by \csname governed expansion. Thus there is no more the limit at about 5000 digits for parsed numbers.

In order to avoid having to lock and unlock in succession to handle the scientific part and adjust the exponent according to the number of digits of the decimal part, the parsing of this decimal part counts on the fly the number of digits it encounters.

There is some slight annoyance with \xintiexpr which should never be given a [n] inside its \csname.=<digits>\endcsname storage of numbers (because its arithmetic uses the ii macros which know nothing about the [N] notation). Hence if the parser has only seen digits when hitting something else than the dot or e (or E), it will not insert a [0]. Thus we very slightly compromise the efficiency of \xintexpr and \xintfloatexpr in order to be able to share the same code with \xintiexpr.

Indeed, the parser at this location is completely common to all, it does not know if it is working inside \xintexpr or \xintiexpr. On the other hand if a dot or a e (or E) is met, then the (common) parser has no scruples ending this number with a [n], this will provoke an error later if that was within an \xintiexpr, as soon as an arithmetic macro is used.

As the gathered numbers have no spaces, no pluses, no minuses, the only remaining issue is with leading zeroes, which are discarded on the fly. The hexadecimal numbers leading zeroes are stripped in a second stage by the \xintHexToDec macro.

With 1.2, \xinttheexpr .\relax does not work anymore (it did in earlier releases). There must be digits either before or after the decimal mark. Thus both \xinttheexpr 1.\relax and \xinttheexpr .1\relax are legal.

The ` syntax is here used for special constructs like `+`(..), `*`(..) where + or * will be treated as functions. Current implementation picks only one token (could have been braced stuff), here it will be + or *, and via \XINT_expr_op_` this then becomes a suitable \XINT_{expr|iiexpr|fexpr}_func_+ (or *). Documentation says to use `+`(...), but `+`(...) is also valid. The opening parenthesis must be there, it is not allowed to come from expansion.

Attention at this location #1 was of catcode 12 in all versions prior to 1.4.

Besides using principally \if tests, we will assume anyhow that catcodes of digits are 12...

```
442 \catcode96 11 % `
```

```

443 \def\XINT_expr_scan_nbr_or_func #1%
444 {%
445   \if "#1\xint_dothis \XINT_expr_scanhex_I\fi
446   \if `#1\xint_dothis {\XINT_expr_onliteral_}\fi
447   \ifnum \xint_c_ix<1\string#1 \xint_dothis \XINT_expr_startint\fi
448   \xint_orthat \XINT_expr_scanfunc #1%
449 }%
450 \def\XINT_expr_onliteral_` #1#2#3({#2}\xint_c_ii^v `}%
451 \catcode96 12 % `
452 \def\XINT_expr_startint #1%
453 {%
454   \if #10\expandafter\XINT_expr_gobz_a\else\expandafter\XINT_expr_scanint_a\fi #1%
455 }%
456 \def\XINT_expr_scanint_a #1#2%
457   {\expanded\bgroup{\iffalse}}\fi #1% spare a \string
458   \expandafter\XINT_expr_scanint_main\romannumeral`&&@#2}%
459 \def\XINT_expr_gobz_a #1#2%
460   {\expanded\bgroup{\iffalse}}\fi
461   \expandafter\XINT_expr_gobz_scanint_main\romannumeral`&&@#2}%
462 \def\XINT_expr_startdec #1%
463   {\expanded\bgroup{\iffalse}}\fi
464   \expandafter\XINT_expr_scandec_a\romannumeral`&&@#1}%

```

11.10.1 Integral part (skipping zeroes)

1.2 has modified the code to give highest priority to digits, the accelerating impact is non-negligible. I don't think the doubled \string is a serious penalty.

```

465 \def\XINT_expr_scanint_main #1%
466 {%
467   \ifcat \relax #1\expandafter\XINT_expr_scanint_hit_cs \fi
468   \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanint_next\fi
469   #1\XINT_expr_scanint_again
470 }%
471 \def\XINT_expr_scanint_again #1%
472 {%
473   \expandafter\XINT_expr_scanint_main\romannumeral`&&@#1%
474 }%
475 \def\XINT_expr_scanint_hit_cs \ifnum#1\fi#2\XINT_expr_scanint_again
476 {%
477   \iffalse{{\fi}}\expandafter\romannumeral`&&@\XINT_expr_getop#2%
478 }%

```

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.

```

479 \def\XINT_expr_scanint_next #1\XINT_expr_scanint_again
480 {%
481   \if [#1\xint_dothis\XINT_expr_rawxintfrac\fi
482   \if _#1\xint_dothis\XINT_expr_scanint_again\fi
483   \if e#1\xint_dothis{[\the\numexpr0\XINT_expr_scanexp_a +]\fi
484   \if E#1\xint_dothis{[\the\numexpr0\XINT_expr_scanexp_a +]\fi
485   \if .#1\xint_dothis{\XINT_expr_startdec_a .}\fi
486   \xint_orthat
487   {\iffalse{\fi}\expandafter\romannumeral`&&\XINT_expr_getop#1}%
488 }%
489 \def\XINT_expr_rawxintfrac
490 {%
491   \iffalse{\fi}\expandafter\csname XINT_expr_precedence_&&\endcsname&&%
492 }%
493 \def\XINT_expr_gobz_scanint_main #1%
494 {%
495   \ifcat \relax #1\expandafter\XINT_expr_gobz_scanint_hit_cs\fi
496   \ifnum\xint_c_x<1\string#1 \else\expandafter\XINT_expr_gobz_scanint_next\fi
497   #1\XINT_expr_scanint_again
498 }%
499 \def\XINT_expr_gobz_scanint_again #1%
500 {%
501   \expandafter\XINT_expr_gobz_scanint_main\romannumeral`&&#1%
502 }%
503 \def\XINT_expr_gobz_scanint_hit_cs\ifnum#1\fi#2\XINT_expr_scanint_again
504 {%
505   0\iffalse{\fi}\expandafter\romannumeral`&&\XINT_expr_getop#2%
506 }%
507 \def\XINT_expr_gobz_scanint_next #1\XINT_expr_scanint_again
508 {%
509   \if [#1\xint_dothis{\expandafter0\XINT_expr_rawxintfrac}\fi
510   \if _#1\xint_dothis\XINT_expr_gobz_scanint_again\fi
511   \if e#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +]\fi
512   \if E#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +]\fi
513   \if .#1\xint_dothis{\XINT_expr_gobz_startdec_a .}\fi
514   \if 0#1\xint_dothis\XINT_expr_gobz_scanint_again\fi
515   \xint_orthat

```



```
516     {0\iffalse{{{\fi}}\expandafter}\romannumeral`&&\XINT_expr_getop#1}%
517 }%
```

11.10.2 Fractional part

Annoying duplication of code to allow 0. as input.

1.2a corrects a very bad bug in 1.2 \XINT_expr_gobz_scandec_b which should have stripped leading zeroes in the fractional part but didn't; as a result \xinttheexpr 0.01\relax returned 0 =:-(((Thanks to Kroum Tzanev who reported the issue. Does it improve things if I say the bug was introduced in 1.2, it wasn't present before ?

```
518 \def\XINT_expr_startdec_a .#1%
519 {%
520     \expandafter\XINT_expr_scandec_a\romannumeral`&&#1%
521 }%
522 \def\XINT_expr_scandec_a #1%
523 {%
524     \if .#1\xint_dothis{\iffalse{{{\fi}}\expandafter}%
525         \romannumeral`&&\XINT_expr_getop..}\fi
526     \xint_orthat {\XINT_expr_scandec_main 0.#1}%
527 }%
528 \def\XINT_expr_gobz_startdec_a .#1%
529 {%
530     \expandafter\XINT_expr_gobz_scandec_a\romannumeral`&&#1%
531 }%
532 \def\XINT_expr_gobz_scandec_a #1%
533 {%
534     \if .#1\xint_dothis
535     {0\iffalse{{{\fi}}\expandafter}\romannumeral`&&\XINT_expr_getop..}\fi
536     \xint_orthat {\XINT_expr_gobz_scandec_main 0.#1}%
537 }%
538 \def\XINT_expr_scandec_main #1.#2%
539 {%
540     \ifcat \relax #2\expandafter\XINT_expr_scandec_hit_cs\fi
541     \ifnum\xint_c_ix<1\string#2 \else\expandafter\XINT_expr_scandec_next\fi
542     #2\expandafter\XINT_expr_scandec_again\the\numexpr #1-\xint_c_i.%
543 }%
544 \def\XINT_expr_scandec_again #1.#2%
545 {%
546     \expandafter\XINT_expr_scandec_main
547     \the\numexpr #1\expandafter.\romannumeral`&&#2%
548 }%
549 \def\XINT_expr_scandec_hit_cs\ifnum#1\fi
550     #2\expandafter\XINT_expr_scandec_again\the\numexpr#3-\xint_c_i.%
551 {%
552     [#3]\iffalse{{{\fi}}\expandafter}\romannumeral`&&\XINT_expr_getop#2%
553 }%
554 \def\XINT_expr_scandec_next #1#2\the\numexpr#3-\xint_c_i.%
555 {%
556     \if _#1\xint_dothis{\XINT_expr_scandec_again#3.}\fi
557     \if e#1\xint_dothis{[\the\numexpr#3\XINT_expr_scanexp_a +}\fi
558     \if E#1\xint_dothis{[\the\numexpr#3\XINT_expr_scanexp_a +}\fi
559     \xint_orthat
```

```

560     {[#3]\iffalse{ $\fi$ }\expandafter}\romannumeral`&&\XINT_expr_getop#1}%
561 }%

562 \def\XINT_expr_gobz_scandec_main #1.#2%
563 {%
564     \ifcat \relax #2\expandafter\XINT_expr_gobz_scandec_hit_cs\fi
565     \ifnum\xint_c_ix<1\string#2 \else\expandafter\XINT_expr_gobz_scandec_next\fi
566     \if0#2\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
567     {\expandafter\XINT_expr_gobz_scandec_main}%
568     {#2\expandafter\XINT_expr_scandec_again}\the\numexpr#1-\xint_c_i.%
569 }%

570 \def\XINT_expr_gobz_scandec_hit_cs \ifnum#1\fi\if0#2#3\xint_c_i.%
571 {%
572     0[0]\iffalse{ $\fi$ }\expandafter}\romannumeral`&&\XINT_expr_getop#2%
573 }%
574 \def\XINT_expr_gobz_scandec_next\if0#1#2\fi #3\numexpr#4-\xint_c_i.%
575 {%
576     \if _#1\xint_dothis{\XINT_expr_gobz_scandec_main #4.}\fi
577     \if e#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +]\fi
578     \if E#1\xint_dothis{0[\the\numexpr0\XINT_expr_scanexp_a +]\fi
579     \xint_orthat
580     {0[0]\iffalse{ $\fi$ }\expandafter}\romannumeral`&&\XINT_expr_getop#1}%
581 }%

```

11.10.3 Scientific notation

Some pluses and minuses are allowed at the start of the scientific part, however not later, and no parenthesis.

```

582 \def\XINT_expr_scanexp_a #1#2%
583 {%
584     #1\expandafter\XINT_expr_scanexp_main\romannumeral`&&#2%
585 }%
586 \def\XINT_expr_scanexp_main #1%
587 {%
588     \ifcat \relax #1\expandafter\XINT_expr_scanexp_hit_cs\fi
589     \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanexp_next\fi
590     #1\XINT_expr_scanexp_again
591 }%
592 \def\XINT_expr_scanexp_again #1%
593 {%
594     \expandafter\XINT_expr_scanexp_main_b\romannumeral`&&#1%
595 }%
596 \def\XINT_expr_scanexp_hit_cs\ifnum#1\fi#2\XINT_expr_scanexp_again
597 {%
598     ]\iffalse{ $\fi$ }\expandafter}\romannumeral`&&\XINT_expr_getop#2%
599 }%
600 \def\XINT_expr_scanexp_next #1\XINT_expr_scanexp_again
601 {%
602     \if _#1\xint_dothis \XINT_expr_scanexp_again \fi
603     \if +#1\xint_dothis {\XINT_expr_scanexp_a +}\fi
604     \if -#1\xint_dothis {\XINT_expr_scanexp_a -}\fi
605     \xint_orthat

```

```

606     []\iffalse{{{\fi}}\expandafter}\romannumeral`&&\XINT_expr_getop#1}%
607 }%
608 \def\XINT_expr_scanexp_main_b #1%
609 {%
610     \ifcat \relax #1\expandafter\XINT_expr_scanexp_hit_cs_b\fi
611     \ifnum\xint_c_ix<1\string#1 \else\expandafter\XINT_expr_scanexp_next_b\fi
612     #1\XINT_expr_scanexp_again_b
613 }%
614 \def\XINT_expr_scanexp_hit_cs_b\ifnum#1\fi#2\XINT_expr_scanexp_again_b
615 {%
616     []\iffalse{{{\fi}}\expandafter}\romannumeral`&&\XINT_expr_getop#2%
617 }%
618 \def\XINT_expr_scanexp_again_b #1%
619 {%
620     \expandafter\XINT_expr_scanexp_main_b\romannumeral`&&#1%
621 }%
622 \def\XINT_expr_scanexp_next_b #1\XINT_expr_scanexp_again_b
623 {%
624     \if_#1\xint_dothis\XINT_expr_scanexp_again\fi
625     \xint_orthat
626     []\iffalse{{{\fi}}\expandafter}\romannumeral`&&\XINT_expr_getop#1}%
627 }%

```

11.10.4 Hexadecimal numbers

1.2d has moved most of the handling of tacit multiplication to `\XINT_expr_getop`, but we have to do some of it here, because we apply `\string` before calling `\XINT_expr_scanhexI_aa`. I do not insert the `*` in `\XINT_expr_scanhexI_a`, because it is its higher precedence variant which will be expected, to do the same as when a non-hexadecimal number prefixes a sub-expression. Tacit multiplication in front of variable or function names will not work (because of this `\string`).

Extended for 1.2l to ignore underscore character `_` if encountered within digits.

```

628 \def\XINT_expr_hex_in #1.#2#3;%
629 {%
630     \expanded{{{\if#2>%
631         \xintHexToDec{#1}%
632     \else
633         \xintiiMul{\xintiiPow{625}{\xintLength{#3}}}{\xintHexToDec{#1#3}}%
634         [\the\numexpr-4*\xintLength{#3}]%
635     \fi}}\expandafter}\romannumeral`&&\XINT_expr_getop
636 }%
637 \def\XINT_expr_scanhex_I #1% #1="
638 {%
639     \expandafter\XINT_expr_hex_in\expanded\bgroup\XINT_expr_scanhexI_a
640 }%
641 \def\XINT_expr_scanhexI_a #1%
642 {%
643     \ifcat #1\relax\xint_dothis{.>}\iffalse{\fi}#1\fi
644     \xint_orthat {\XINT_expr_scanhexI_aa #1}%
645 }%
646 \def\XINT_expr_scanhexI_aa #1%
647 {%
648     \if\ifnum`#1>`/

```

```

649     \ifnum`#1>`9
650     \ifnum`#1>`@
651     \ifnum`#1>`F
652     0\else\fi\else0\fi\else1\fi\else0\fi 1%
653     \expandafter\XINT_expr_scanhexI_b
654 \else
655     \if_#1\xint_dothis{\expandafter\XINT_expr_scanhexI_bgob}\fi
656     \if_#1\xint_dothis{\expandafter\XINT_expr_scanhex_transition}\fi
657     \xint_orthat {\xint_afterfi {.>;\iffalse{\fi}}}%
658 \fi
659 #1%
660 }%
661 \def\XINT_expr_scanhexI_b #1#2%
662 {%
663     #1\expandafter\XINT_expr_scanhexI_a\romannumeral`&&@#2%
664 }%
665 \def\XINT_expr_scanhexI_bgob #1#2%
666 {%
667     \expandafter\XINT_expr_scanhexI_a\romannumeral`&&@#2%
668 }%
669 \def\XINT_expr_scanhex_transition_#1%
670 {%
671     \expandafter.\expandafter.\expandafter
672     \XINT_expr_scanhexII_a\romannumeral`&&@#1%
673 }%
674 \def\XINT_expr_scanhexII_a #1%
675 {%
676     \ifcat #1\relax\xint_dothis{;\iffalse{\fi}#1}\fi
677     \xint_orthat {\XINT_expr_scanhexII_aa #1}%
678 }%
679 \def\XINT_expr_scanhexII_aa #1%
680 {%
681     \if\ifnum`#1>`/
682     \ifnum`#1>`9
683     \ifnum`#1>`@
684     \ifnum`#1>`F
685     0\else\fi\else0\fi\else1\fi\else0\fi 1%
686     \expandafter\XINT_expr_scanhexII_b
687 \else
688     \if_#1\xint_dothis{\expandafter\XINT_expr_scanhexII_bgob}\fi
689     \xint_orthat{\xint_afterfi {;\iffalse{\fi}}}%
690 \fi
691 #1%
692 }%
693 \def\XINT_expr_scanhexII_b #1#2%
694 {%
695     #1\expandafter\XINT_expr_scanhexII_a\romannumeral`&&@#2%
696 }%
697 \def\XINT_expr_scanhexII_bgob #1#2%
698 {%
699     \expandafter\XINT_expr_scanhexII_a\romannumeral`&&@#2%
700 }%

```

11.10.5 \XINT_expr_scanfunc: collecting names of functions and variables

At 1.4 the first token left over in string has not been submitted to \string. We also know it is not a control sequence. So we can test catcode to identify if operator is found. And it is allowed to hit some operator such as a closing parenthesis we will then insert the «nil» value (which however can cause breakage of arithmetic operations, although xintfrac.sty converts empty to 0).

The @ causes a problem because it must work with both catcode 11 or 12.

The _ can be used internally for starting variables but it will have catcode 11 then.

There was prior to 1.4 solely the dispatch in \XINT_expr_scanfunc_b but now we do it immediately and issue \XINT_expr_func only in certain cases.

But we have to be careful that !(...) and ?(...) are part of the syntax and genuine functions. Because we now do earlier to getop we must filter them out.

```

701 \def\XINT_expr_scanfunc #1%
702 {%
703   \if 1\ifcat a#10\fi\if @#10\fi\if !#10\fi\if ?#10\fi 1%
704     \expandafter\xint_firstoftwo
705   \else\expandafter\xint_secondoftwo
706   \fi
707   {\expandafter{\expandafter}\romannumeral`&&\XINT_expr_getop#1}%
708   {\expandafter\XINT_expr_func\expanded\bgroup#1\XINT_expr_scanfunc_a}%
709 }%
710 \def\XINT_expr_scanfunc_a #1%
711 {%
712   \expandafter\XINT_expr_scanfunc_b\romannumeral`&&#1%
713 }%

```

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.

```

714 \def\XINT_expr_scanfunc_b #1%
715 {%
716   \ifcat \relax#1\xint_dothis{\iffalse{\fi}(_#1)\fi
717   \if (#1\xint_dothis{\iffalse{\fi}){`}\fi
718   \if 1\ifcat a#10\fi
719     \ifnum\xint_c_ix<1\string#1 0\fi
720     \if @#10\fi
721     \if _#10\fi
722     1%

```

```
723     \xint_dothis{\iffalse{\fi}(_#1)\fi
724 \xint_orthat {#1\XINT_expr_scanfunc_a}%
725 }%
```

11.10.6 \XINT_expr_func: dispatch to variable replacement or to function execution

Comments written 2015/11/12: earlier there was an `\ifcsname` test for checking if we had a variable in front of a `(`, for tacit multiplication for example in `x(y+z(x+w))` to work. But after I had implemented functions (that was yesterday...), I had the problem if was impossible to re-declare a variable name such as "f" as a function name. The problem is that here we can not test if the function is available because we don't know if we are in `expr`, `iiexpr` or `floatexpr`. The `\xint_c_ii^v` causes all fetching operations to stop and control is handed over to the routines which will be `expr`, `iiexpr` ou `floatexpr` specific, i.e. the `\XINT_{expr|iiexpr|flexpr}_op_{`|_}` which are invoked by the `until_<op>_b` macros earlier in the stream. Functions may exist for one but not the two other parsers. Variables are declared via one parser and usable in the others, but naturally `\xintiexpr` has its restrictions.

Thinking about this again I decided to treat a priori cases such as `x(...)` as functions, after having assigned to each variable a low-weight macro which will convert this into `_getop\.=<value of x>*(...)`. To activate that macro at the right time I could for this exploit the "onliteral" intercept, which is parser independent (1.2c).

This led to me necessarily to rewrite partially the `seq`, `add`, `mul`, `subs`, `iter` ... routines as now the variables fetch only one token. I think the thing is more efficient.

1.2c had `\def\XINT_expr_func #1(#2{\xint_c_ii^v #2{#1}}`

In `\XINT_expr_func` the `#2` is `_` if `#1` must be a variable name, or `#2=`` if `#1` must be either a function name or possibly a variable name which will then have to be followed by tacit multiplication before the opening parenthesis.

The `\xint_c_ii^v` is there because `_op_`` must know in which parser it works. Dispendious for `_`. Hence I modify for 1.2d.

```
726 \def\XINT_expr_func #1(#2{\if _#2\xint_dothis{\XINT_expr_op_{#1}}\fi
727     \xint_orthat{#1}\xint_c_ii^v #2}}%
```

11.11 \XINT_expr_op_`: launch function or pseudo-function, or evaluate variable and insert operator of multiplication in front of parenthesized contents

The "onliteral" intercepts is for `bool`, `togl`, `protect`, ... but also for `add`, `mul`, `seq`, etc... Genuine functions have `expr`, `iiexpr` and `flexpr` versions (or only one or two of the three).

With 1.2c "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.

```
728 \def\XINT_tmpa #1#2#3{%
729   \def #1##1%
730   {%
731     \ifcsname XINT_#3_func_##1\endcsname
732     \csname XINT_#3_func_##1\expandafter\endcsname
733     \romannumeral`&&\expandafter#2%
734   \else
735     \ifcsname XINT_expr_onliteral_##1\endcsname
736     \csname XINT_expr_onliteral_##1\expandafter\expandafter\expandafter
```

```

737     \endcsname
738   \else
739     \csname XINT_#3_func_\XINT_expr_unknown_function {##1}%
740     \expandafter\endcsname
741     \romannumeral`&&\expandafter\expandafter\expandafter#2%
742   \fi
743 \fi
744 }%
745 }%
746 \def\XINT_expr_unknown_function #1%
747   {\XINT_expandableerror{"#1" is unknown as function. (I)nsert correct name:}}%
748 \def\XINT_expr_func_ #1#2#3{#1#2{{0}}}%
749 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
750   \expandafter\XINT_tmpa
751     \csname XINT_#1_op_\` \expandafter\endcsname
752     \csname XINT_#1_oparen\endcsname
753     {#1}%
754 }%

```

11.12 \XINT_expr_op__: replace a variable by its value and then fetch next operator

The 1.1 mechanism for \XINT_expr_var_<varname> has been modified in 1.2c. The <varname> associated macro is now only expanded once, not twice. We arrive here via \XINT_expr_func.

At 1.4 \XINT_expr_getop is launched with accumulated result on its left. But the omit and abort keywords are implemented via dummy variables which rely on possibility to modify upstream tokens. If we did here something such as _var_#1\expandafter\endcsname\romannumeral`^^@\XINT_expr_getop the premature expansion of getop would break things. Thus we revert to former code which put \XINT_expr_getop (call it _legacy) in front of variable expansion (in xintexpr < 1.4 this expanded to a single token so the overhead was not serious).

Abusing variables to manipulate token stream is a bit bad, usually I prefer functions for this (such as the break() function) but then I have define 3 macros for the 3 parsers.

The situation here is not satisfactory. But 1.4 has to be released now.

```

755 \def\XINT_expr_op__ #1% op__ with two _'s
756 {%
757   \ifcsname XINT_expr_var_#1\endcsname
758     \expandafter\expandafter\expandafter\XINT_expr_getop_legacy
759     \csname XINT_expr_var_#1\expandafter\endcsname
760   \else
761     \expandafter\expandafter\expandafter\XINT_expr_getop_legacy
762     \csname XINT_expr_var_\XINT_expr_unknown_variable {#1}%
763     \expandafter\endcsname
764   \fi
765 }%
766 \def\XINT_expr_unknown_variable #1%
767   {\XINT_expandableerror {"#1" is unknown as a variable. (I)nsert correct one:}}%
768 \def\XINT_expr_var_{{0}}%
769 \let\XINT_flexpr_op__ \XINT_expr_op__
770 \let\XINT_iiexpr_op__ \XINT_expr_op__
771 \def\XINT_expr_getop_legacy #1%
772 {%
773   \expanded{\unexpanded{{#1}}\expandafter}\romannumeral`&&\XINT_expr_getop

```

774 }%

11.13 `\XINT_expr_getop`: fetch the next operator or closing parenthesis or end of expression

Release 1.1 implements multi-character operators.

1.2d adds tacit multiplication also in front of variable or functions names starting with a letter, not only a @ or a _ as was already the case. This is for (x+y)z situations. It also applies higher precedence in cases like x/2y or x/2@, or x/2max(3,5), or x/2\xintexpr 3\relax.

In fact, finally I decide that all sorts of tacit multiplication will always use the higher precedence.

Indeed I hesitated somewhat: with the current code one does not know if `\XINT_expr_getop` as invoked after a closing parenthesis or because a number parsing ended, and I felt distinguishing the two was unneeded extra stuff. This means cases like (a+b)/(c+d)(e+f) will first multiply the last two parenthesized terms.

The ! starting a sub-expression must be distinguished from the post-fix ! for factorial, thus we must not do a too early `\string`. In versions < 1.2c, the catcode 11 ! had to be identified in all branches of the number or function scans. Here it is simply treated as a special case of a letter.

1.2q adds tacit multiplication in cases such as (1+1)3 or 5!7!

1.4 has simplified coding here as `\XINT_expr_getop` expansion happens at a time when a fetched value has already being stored.

```
775 \def\XINT_expr_getop #1%
776 {%
777   \expandafter\XINT_expr_getop_a\romannumeral`&&#1%
778 }%
779 \catcode`* 11
780 \def\XINT_expr_getop_a #1%
781 {%
782   \ifx \relax #1\xint_dothis\xint_firstofthree\fi
783   \ifcat \relax #1\xint_dothis\xint_secondofthree\fi
784   \ifnum\xint_c_ix<1\string#1 \xint_dothis\xint_secondofthree\fi
785   \if :#1\xint_dothis \xint_thirdofthree\fi
786   \if _#1\xint_dothis \xint_secondofthree\fi
787   \if @#1\xint_dothis \xint_secondofthree\fi
788   \if (#1\xint_dothis \xint_secondofthree\fi %)
789   \ifcat a#1\xint_dothis \xint_secondofthree\fi
790   \xint_orthat \xint_thirdofthree
791   {\XINT_expr_foundend}%
```

tacit multiplication with higher precedence.

```
792   {\XINT_expr_precedence_*** #1}%
793   {\expandafter\XINT_expr_getop_b \string#1}%
794 }%
795 \catcode`* 12
```

`\relax` is a place holder here.

```
796 \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.

```

797 \def\XINT_expr_getop_b #1%
798 {%
799     \if &&@#1\xint_dothis{\csname XINT_expr_precedence_&&\endcsname}&&\fi
800     \if '#1\xint_dothis{\XINT_expr_binopwrд }\fi
801     \if ?#1\xint_dothis{\XINT_expr_precedence_? ?}\fi
802     \xint_orthat     {\XINT_expr_scanop_a #1}%
803 }%
804 \def\XINT_expr_binopwrд #1'%
805 {%
806     \expandafter\XINT_expr_foundop_a
807     \csname XINT_expr_itself_\xint_zapspaceс #1 \xint_gobble_i\endcsname
808 }%
809 \def\XINT_expr_scanop_a #1#2%
810 {%
811     \expandafter\XINT_expr_scanop_b\expandafter#1\romannumeral`&&@#2%
812 }%
813 \def\XINT_expr_scanop_b #1#2%
814 {%
815     \ifcat#2\relax\xint_dothis{\XINT_expr_foundop_a #1#2}\fi
816     \ifcsname XINT_expr_itself_#1#2\endcsname
817     \xint_dothis
818         {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#2\endcsname}\fi
819     \xint_orthat {\XINT_expr_foundop_a #1#2}%
820 }%
821 \def\XINT_expr_scanop_c #1#2%
822 {%
823     \expandafter\XINT_expr_scanop_d\expandafter#1\romannumeral`&&@#2%
824 }%
825 \def\XINT_expr_scanop_d #1#2%
826 {%
827     \ifcat#2\relax \xint_dothis{\XINT_expr_foundop #1#2}\fi
828     \ifcsname XINT_expr_itself_#1#2\endcsname
829     \xint_dothis
830         {\expandafter\XINT_expr_scanop_c\csname XINT_expr_itself_#1#2\endcsname } \fi
831     \xint_orthat {\csname XINT_expr_precedence_#1\endcsname #1#2}%
832 }%
833 \def\XINT_expr_foundop_a #1%
834 {%
835     \ifcsname XINT_expr_precedence_#1\endcsname
836         \csname XINT_expr_precedence_#1\endcsname\expandafter\endcsname
837         \expandafter #1%
838     \else
839         \xint_afterfi{\XINT_expr_getop\romannumeral0%
840         \XINT_expandableerror
841         {"#1" is unknown as operator. (I)nsert one:} }%<<deliberate space
842     \fi
843 }%
844 \def\XINT_expr_foundop #1{\csname XINT_expr_precedence_#1\endcsname #1}%

```

11.14 Expansion spanning; opening and closing parentheses

These comments apply to all definitions coming next relative to execution of operations from parsing of syntax.

Refactored (and unified) at 1.4. In particular the 1.4 scheme uses `op`, `exec`, `check-`, and `checkp`. Formerly it was `until_a` (`check-`) and `until_b` (now split into `checkp` and `exec`).

This way neither `check-` nor `checkp` have to grab the accumulated number so far (top of stack if you like) and besides one never has to go back to `check-` from `checkp` (and neither from `check-`).

Prior to 1.4, accumulated intermediate results were stored as one token, but now we have to use `\expanded` to propagate expansion beyond possibly arbitrary long braced nested data. With the 1.4 refactoring we do this only once and only grab a second time the data if we actually have to act upon it.

Version 1.1 had a hack inside the `until` macros for handling the `omit` and `abort` in iterations over dummy variables. This has been removed by 1.2c, see the subsection where `omit` and `abort` are discussed.

Exceptionally, the `check-` is here abbreviated to `check`.

```

845 \catcode`) 11
846 \def\XINT_tmpa #1#2#3#4#5#6%
847 {%
848   \def#1% start
849   {%
850     \expandafter#2\romannumeral`&&\XINT_expr_getnext
851   }%
852   \def##1% check
853   {%
854     \xint_UDsignfork
855     ##1{\expandafter#3\romannumeral`&&#4}%
856     -{#3##1}%
857     \krof
858   }%
859   \def3##1##2% checkp
860   {%
861     \ifcase ##1%
862       \expandafter\XINT_expr_done
863       \or\expandafter#5%
864     \else
865       \expandafter#3\romannumeral`&&\csname XINT_#6_op_##2\expandafter\endcsname
866     \fi
867   }%
868   \def#5%
869   {%
870     \XINT_expandableerror
871     {An extra ) has been removed. Hit Return, fingers crossed.}%
872     \expandafter#2\romannumeral`&&\expandafter\XINT_expr_put_op_first
873     \romannumeral`&&\XINT_expr_getop_legacy
874   }%
875 }%
876 \let\XINT_expr_done\space
877 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
878   \expandafter\XINT_tmpa
879   \csname XINT_#1_start\expandafter\endcsname
880   \csname XINT_#1_check\expandafter\endcsname
881   \csname XINT_#1_checkp\expandafter\endcsname
882   \csname XINT_#1_op_-xii\expandafter\endcsname

```

```
883 \csname XINT_#1_extra_)\endcsname
884 {#1}%
885 }%
```

Here also we take some shortcuts relative to general philosophy and have no explicit exec macro.

```
886 \def\XINT_tmpa #1#2#3#4#5#6#7%
887 {%
888 \def #1##1% op_(
889 {%
890 \expandafter #4\romannumeral`&&\XINT_expr_getnext
891 }%
892 \def #2##1% op_)
893 {%
894 \expanded{\unexpanded{\XINT_expr_put_op_first{##1}}\expandafter}\romannumeral`&&\XINT_expr_getnext
895 }%
896 \def #3% oparen
897 {%
898 \expandafter #4\romannumeral`&&\XINT_expr_getnext
899 }%
900 \def #4##1% check-
901 {%
902 \xint_UDsignfork
903 ##1{\expandafter#5\romannumeral`&&#6}%
904 -{#5##1}%
905 \krof
906 }%
907 \def #5##1##2% checkp
908 {%
909 \ifcase ##1\expandafter\XINT_expr_missing_)
910 \or \csname XINT_#7_op_##2\expandafter\endcsname
911 \else
912 \expandafter #5\romannumeral`&&\csname XINT_#7_op_##2\expandafter\endcsname
913 \fi
914 }%
915 }%
916 \def\XINT_expr_missing_)
917 {\XINT_expandableerror{Sorry to report a missing ) at the end of this journey.}%
918 \xint_c_ \XINT_expr_done }%
919 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
920 \expandafter\XINT_tmpa
921 \csname XINT_#1_op_(\expandafter\endcsname
922 \csname XINT_#1_op_)\expandafter\endcsname
923 \csname XINT_#1_oparen\expandafter\endcsname
924 \csname XINT_#1_check_)\expandafter\endcsname
925 \csname XINT_#1_checkp_)\expandafter\endcsname
926 \csname XINT_#1_op_-xii\endcsname
927 {#1}%
928 }%
929 \let\XINT_expr_precedence_)\xint_c_i
930 \catcode`) 12
```

11.15 The comma as binary operator

New with 1.09a. Refactored at 1.4.

```

931 \def\XINT_tmpa #1#2#3#4#5#6%
932 {%
933   \def #1##1% \XINT_expr_op_,
934   {%
935     \expanded{\unexpanded{#2{##1}}\expandafter}%
936     \romannumeral`&&\expandafter#3\romannumeral`&&\XINT_expr_getnext
937   }%
938   \def #2##1##2##3##4{##2##3{##1##4}}% \XINT_expr_exec_,
939   \def #3##1% \XINT_expr_check-,
940   {%
941     \xint_UDsignfork
942     ##1{\expandafter#4\romannumeral`&&@#5}%
943     -{#4##1}%
944     \krof
945   }%
946   \def #4##1##2% \XINT_expr_checkp_,
947   {%
948     \ifnum ##1>\xint_c_iii
949     \expandafter#4%
950     \romannumeral`&&\csname XINT_#6_op_#2\expandafter\endcsname
951     \else
952     \expandafter##1\expandafter##2%
953     \fi
954   }%
955 }%
956 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
957 \expandafter\XINT_tmpa
958   \csname XINT_#1_op_,\expandafter\endcsname
959   \csname XINT_#1_exec_,\expandafter\endcsname
960   \csname XINT_#1_check_,\expandafter\endcsname
961   \csname XINT_#1_checkp_,\expandafter\endcsname
962   \csname XINT_#1_op_-xii\endcsname {#1}%
963 }%
964 \expandafter\let\csname XINT_expr_precedence_,\endcsname\xint_c_iii

```

11.16 The minus as prefix operator of variable precedence level

Inherits the precedence level of the previous infix operator. Refactored at 1.4

```

965 \def\XINT_tmpb #1#2#3#4#5#6#7%
966 {%
967   \def #1% \XINT_expr_op_-<level>
968   {%
969     \expandafter #2\romannumeral`&&\expandafter#3%
970     \romannumeral`&&\XINT_expr_getnext
971   }%
972   \def #2##1##2##3% \XINT_expr_exec_-<level>
973   {%
974     \expandafter ##1\expandafter ##2\expandafter

```

```

975     {%
976     \romannumeral`&&\XINT:NEhook:f:one:from:one
977     {\romannumeral`&&@#7##3}%
978     }%
979 }%
980 \def #3##1% \XINT_expr_check_--<level>
981 {%
982     \xint_UDsignfork
983     ##1{\expandafter #4\romannumeral`&&@#1}%
984     -{#4##1}%
985     \krof
986 }%
987 \def #4##1##2% \XINT_expr_checkp_--<level>
988 {%
989     \ifnum ##1>#5%
990     \expandafter #4%
991     \romannumeral`&&\csname XINT_#6_op_##2\expandafter\endcsname
992     \else
993     \expandafter ##1\expandafter ##2%
994     \fi
995 }%
996 }%
997 \def\XINT_tmpa #1#2#3%
998 {%
999     \expandafter\XINT_tmpb
1000     \csname XINT_#1_op_#3\expandafter\endcsname
1001     \csname XINT_#1_exec_#3\expandafter\endcsname
1002     \csname XINT_#1_check_#3\expandafter\endcsname
1003     \csname XINT_#1_checkp_#3\expandafter\endcsname
1004     \csname xint_c_#3\endcsname {#1}#2%
1005 }%

```

1.2d needs precedence 8 for *** and 9 for ^. Earlier, precedence level for ^ was only 8 but nevertheless the code did also "ix" here, which I think was unneeded back then.

```

1006 \xintApplyInline{\XINT_tmpa {expr}\xintOpp}{{xii}{xiv}{xvi}{xviii}}%
1007 \xintApplyInline{\XINT_tmpa {fexpr}\xintOpp}{{xii}{xiv}{xvi}{xviii}}%
1008 \xintApplyInline{\XINT_tmpa {iexpr}\xintiiOpp}{{xii}{xiv}{xvi}{xviii}}%

```

11.17 The * as Python-like «unpacking» prefix operator

New with 1.4. Prior to 1.4 the internal data structure was the one of \csname encapsulated comma separated numbers. No hierarchical structure was (easily) possible. At 1.4, we can use TeX braces because there is no detokenization to catcode 12.

```

1009 \def\XINT_tmpa#1#2#3%
1010 {%
1011     \def#1##1{\expandafter#2\romannumeral`&&\XINT_expr_getnext}%
1012     \def#2##1##2%
1013     {%
1014         \ifnum ##1>\xint_c_xx
1015         \expandafter #2%
1016         \romannumeral`&&\csname XINT_#3_op_##2\expandafter\endcsname

```

```

1017     \else
1018     \expandafter##1\expandafter##2\romannumeral0\expandafter\XINT:NEhook:unpack
1019     \fi
1020 }%
1021 }%
1022 \def\XINT:NEhook:unpack{\xint_stop_atfirstofone}%
1023 \xintFor* #1 in {{expr}{fexpr}{iexpr}}:
1024   {\expandafter\XINT_tmpa\csname XINT_#1_op_0\expandafter\endcsname
1025     \csname XINT_#1_until_unpack\endcsname {#1}}%

```

11.18 Infix operators

11.18.1	&&, , //, /:, +, -, *, /, ^, **, 'and', 'or', 'xor', and 'mod'	342
11.18.2	.., ..[, and].. for a..b and a..[b]..c syntax	344
11.18.3	<, >, ==, <=, >=, != with Python-like chaining	346
11.18.4	Support macros for .., ..[and]..	348

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.

1.4b subtlety with catcode of ! in `\XINT_expr_itself_!=`, due to chaining of comparison operators which use it to reinject into stream, but we must then have it of catcode 12 there, whereas so far the `itself` macros were only expanded in `csname` context.

```

1026 \catcode`& 12
1027 \xintFor* #1 in {{==}{<=}{>=}{&&}{||}{**}{//}{/:}{..}{..[]{}..}}:
1028   \do {\expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}}%
1029 \catcode`& 7
1030 \expandafter\edef\csname XINT_expr_itself_!=\endcsname{\string !=}%
1031 \expandafter\let\csname XINT_expr_precedence_***\endcsname \xint_c_xvi

```

11.18.1 &&, ||, //, /:, +, -, *, /, ^, **, 'and', 'or', 'xor', and 'mod'

Usage of single character Boolean operators `&` and `|` is deprecated (for many years) and only `&&` and `||` should be used. `&` and `|` will be removed at next major release after 1.4.

```

1032 \def\XINT_expr_defbin_c #1#2#3#4#5#6#7#8%
1033 {%
1034   \def #1##1% \XINT_expr_op_<op>
1035   {%
1036     \expanded{\unexpanded{#2{##1}}\expandafter}%
1037     \romannumeral`&&\expandafter#3\romannumeral`&&\XINT_expr_getnext
1038   }%
1039   \def #2##1##2##3##4% \XINT_expr_exec_<op>
1040   {%
1041     \expandafter##2\expandafter##3\expandafter
1042     {\romannumeral`&&\XINT:NEhook:f:one:from:two{\romannumeral`&&#6##1##4}}%
1043   }%
1044   \def #3##1% \XINT_expr_check_<op>
1045   {%
1046     \xint_UDsignfork
1047     ##1{\expandafter#4\romannumeral`&&#5}%
1048     -{#4##1}%
1049   \krof

```

```

1050 }%
1051 \def #4#1##2% \XINT_expr_checkp_<op>
1052 {%
1053   \ifnum ##1>#7%
1054     \expandafter#4%
1055     \romannumeral`&&@\csname XINT_#8_op_##2\expandafter\endcsname
1056   \else
1057     \expandafter ##1\expandafter ##2%
1058   \fi
1059 }%
1060 }%
1061 \def\XINT_expr_defbin_b #1#2#3#4#5%
1062 {%
1063   \expandafter\XINT_expr_defbin_c
1064   \csname XINT_#1_op_#2\expandafter\endcsname
1065   \csname XINT_#1_exec_#2\expandafter\endcsname
1066   \csname XINT_#1_check_#2\expandafter\endcsname
1067   \csname XINT_#1_checkp_#2\expandafter\endcsname
1068   \csname XINT_#1_op_#4\expandafter\endcsname
1069   \csname #5\expandafter\endcsname
1070   \csname XINT_expr_precedence_#2\endcsname
1071   {#1}%
1072   \expandafter % done 3 times but well
1073   \let\csname XINT_expr_precedence_#2\expandafter\endcsname
1074     \csname xint_c_#3\endcsname
1075 }%
1076 \XINT_expr_defbin_b {expr} {||} {vi}{xii} {xintOR}%
1077 \XINT_expr_defbin_b {flexpr}{||} {vi}{xii} {xintOR}%
1078 \XINT_expr_defbin_b {iiexpr}{||} {vi}{xii} {xintOR}%
1079 \catcode`& 12
1080 \XINT_expr_defbin_b {expr} {&&} {viii}{xii} {xintAND}%
1081 \XINT_expr_defbin_b {flexpr}{&&} {viii}{xii} {xintAND}%
1082 \XINT_expr_defbin_b {iiexpr}{&&} {viii}{xii} {xintAND}%
1083 \catcode`& 7
1084 \XINT_expr_defbin_b {expr} {xor}{vi}{xii} {xintXOR}%
1085 \XINT_expr_defbin_b {flexpr}{xor}{vi}{xii} {xintXOR}%
1086 \XINT_expr_defbin_b {iiexpr}{xor}{vi}{xii} {xintXOR}%
1087 \XINT_expr_defbin_b {expr} {//} {xiv}{xiv}{xintDivFloor}%
1088 \XINT_expr_defbin_b {flexpr}{//} {xiv}{xiv}{XINTinFloatDivFloor}%
1089 \XINT_expr_defbin_b {iiexpr}{//} {xiv}{xiv}{xintiiDivFloor}%
1090 \XINT_expr_defbin_b {expr} {/:} {xiv}{xiv}{xintMod}%
1091 \XINT_expr_defbin_b {flexpr}{/:} {xiv}{xiv}{XINTinFloatMod}%
1092 \XINT_expr_defbin_b {iiexpr}{/:} {xiv}{xiv}{xintiiMod}%
1093 \XINT_expr_defbin_b {expr} + {xii}{xii}{xintAdd}%
1094 \XINT_expr_defbin_b {flexpr} + {xii}{xii}{XINTinFloatAdd}%
1095 \XINT_expr_defbin_b {iiexpr} + {xii}{xii}{xintiiAdd}%
1096 \XINT_expr_defbin_b {expr} - {xii}{xii}{xintSub}%
1097 \XINT_expr_defbin_b {flexpr} - {xii}{xii}{XINTinFloatSub}%
1098 \XINT_expr_defbin_b {iiexpr} - {xii}{xii}{xintiiSub}%
1099 \XINT_expr_defbin_b {expr} * {xiv}{xiv}{xintMul}%
1100 \XINT_expr_defbin_b {flexpr} * {xiv}{xiv}{XINTinFloatMul}%
1101 \XINT_expr_defbin_b {iiexpr} * {xiv}{xiv}{xintiiMul}%

```

```

1102 \XINT_expr_defbin_b {expr} / {xiv}{xiv}{xintDiv}%
1103 \XINT_expr_defbin_b {fexpr} / {xiv}{xiv}{XINTinFloatDiv}%
1104 \XINT_expr_defbin_b {iiexpr} / {xiv}{xiv}{xintiiDivRound}%
1105 \XINT_expr_defbin_b {expr} ^ {xviii}{xviii}{xintPow}%
1106 \XINT_expr_defbin_b {fexpr} ^ {xviii}{xviii}{XINTinFloatPowerH}%
1107 \XINT_expr_defbin_b {iiexpr} ^ {xviii}{xviii}{xintiiPow}%
1108 \catcode`& 12
1109 \xintFor #1 in {and,or,xor,mod} \do
1110 {%
1111   \expandafter\def\csname XINT_expr_itself_#1\endcsname {#1}%
1112 }%
1113 \expandafter\let\csname XINT_expr_precedence_and\expandafter\endcsname
1114   \csname XINT_expr_precedence_&&\endcsname
1115 \expandafter\let\csname XINT_expr_precedence_or\expandafter\endcsname
1116   \csname XINT_expr_precedence_||\endcsname
1117 \expandafter\let\csname XINT_expr_precedence_mod\expandafter\endcsname
1118   \csname XINT_expr_precedence_/\endcsname
1119 \xintFor #1 in {expr, fexpr, iiexpr} \do
1120 {%
1121   \expandafter\let\csname XINT_#1_op_and\expandafter\endcsname
1122     \csname XINT_#1_op_&&\endcsname
1123   \expandafter\let\csname XINT_#1_op_or\expandafter\endcsname
1124     \csname XINT_#1_op_||\endcsname
1125   \expandafter\let\csname XINT_#1_op_mod\expandafter\endcsname
1126     \csname XINT_#1_op_/\endcsname
1127 }%
1128 \expandafter\let\csname XINT_expr_precedence_&\expandafter\endcsname
1129   \csname XINT_expr_precedence_&&\endcsname
1130 \expandafter\let\csname XINT_expr_precedence_|\expandafter\endcsname
1131   \csname XINT_expr_precedence_||\endcsname
1132 \expandafter\let\csname XINT_expr_precedence_**\expandafter\endcsname
1133   \csname XINT_expr_precedence_^\endcsname
1134 \xintFor #1 in {expr, fexpr, iiexpr} \do
1135 {%
1136   \expandafter\let\csname XINT_#1_op_&\expandafter\endcsname
1137     \csname XINT_#1_op_&&\endcsname
1138   \expandafter\let\csname XINT_#1_op_|\expandafter\endcsname
1139     \csname XINT_#1_op_||\endcsname
1140   \expandafter\let\csname XINT_#1_op_**\expandafter\endcsname
1141     \csname XINT_#1_op_^\endcsname
1142 }%
1143 \catcode`& 7

```

11.18.2 .., ..[, 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.

```

1144 \def\XINT_expr_defbin_c #1#2#3#4#5#6#7%
1145 {%
1146   \def #1#1% \XINT_expr_op_...[
1147   {%

```



```

1148   \expanded{\unexpanded{#2{##1}}\expandafter}%
1149   \romannumeral`&&\expandafter#3\romannumeral`&&\XINT_expr_getnext
1150 }%
1151 \def #2##1##2##3##4% \XINT_expr_exec_... [
1152 {%
1153   ##2##3{{##1##4}}%
1154 }%
1155 \def #3##1% \XINT_expr_check-... [
1156 {%
1157   \xint_UDsignfork
1158   ##1{\expandafter#4\romannumeral`&&#5}%
1159   -{#4##1}%
1160   \krof
1161 }%
1162 \def #4##1##2% \XINT_expr_checkp_... [
1163 {%
1164   \ifnum ##1>#6%
1165     \expandafter#4%
1166     \romannumeral`&&\csname XINT_#7_op_##2\expandafter\endcsname
1167   \else
1168     \expandafter ##1\expandafter ##2%
1169   \fi
1170 }%
1171 }%
1172 \def\XINT_expr_defbin_b #1%
1173 {%
1174   \expandafter\XINT_expr_defbin_c
1175   \csname XINT_#1_op_...[\expandafter\endcsname
1176   \csname XINT_#1_exec_...[\expandafter\endcsname
1177   \csname XINT_#1_check-...[\expandafter\endcsname
1178   \csname XINT_#1_checkp_...[\expandafter\endcsname
1179   \csname XINT_#1_op_-xii\expandafter\endcsname
1180   \csname XINT_expr_precedence_...[\endcsname
1181   {#1}%
1182 }%
1183 \XINT_expr_defbin_b {expr}%
1184 \XINT_expr_defbin_b {flexpr}%
1185 \XINT_expr_defbin_b {iiexpr}%
1186 \expandafter\let\csname XINT_expr_precedence_...[\endcsname\xint_c_vi
1187 \def\XINT_expr_defbin_c #1#2#3#4#5#6#7#8%
1188 {%
1189   \def #1##1% \XINT_expr_op_<op>
1190   {%
1191     \expanded{\unexpanded{#2{##1}}\expandafter}%
1192     \romannumeral`&&\expandafter#3\romannumeral`&&\XINT_expr_getnext
1193   }%
1194   \def #2##1##2##3##4% \XINT_expr_exec_<op>
1195   {%
1196     \expandafter##2\expandafter##3\expanded
1197     {{\XINT:NEhook:x:one:from:two#8##1##4}}%
1198   }%
1199   \def #3##1% \XINT_expr_check-_<op>

```

```

1200 {%
1201   \xint_UDsignfork
1202   ##1{\expandafter#4\romannumeral`&&@#5}%
1203   -{#4##1}%
1204   \krof
1205 }%
1206 \def #4##1##2% \XINT_expr_checkp_<op>
1207 {%
1208   \ifnum ##1>#6%
1209     \expandafter#4%
1210     \romannumeral`&&@\csname XINT_#7_op_##2\expandafter\endcsname
1211   \else
1212     \expandafter ##1\expandafter ##2%
1213   \fi
1214 }%
1215 }%
1216 \def\XINT_expr_defbin_b #1#2#3%
1217 {%
1218   \expandafter\XINT_expr_defbin_c
1219   \csname XINT_#1_op_#2\expandafter\endcsname
1220   \csname XINT_#1_exec_#2\expandafter\endcsname
1221   \csname XINT_#1_check_#2\expandafter\endcsname
1222   \csname XINT_#1_checkp_#2\expandafter\endcsname
1223   \csname XINT_#1_op_-xii\expandafter\endcsname
1224   \csname XINT_expr_precedence_#2\endcsname
1225   {#1}#3%
1226   \expandafter\let
1227   \csname XINT_expr_precedence_#2\expandafter\endcsname\xint_c_vi
1228 }%
1229 \XINT_expr_defbin_b {expr} {..}\xintSeq:tl:x
1230 \XINT_expr_defbin_b {flexpr} {..}\xintSeq:tl:x
1231 \XINT_expr_defbin_b {iiexpr} {..}\xintiiSeq:tl:x
1232 \XINT_expr_defbin_b {expr} {[]}\xintSeqB:tl:x
1233 \XINT_expr_defbin_b {flexpr}{[]}\xintSeqB:tl:x
1234 \XINT_expr_defbin_b {iiexpr}{[]}\xintiiSeqB:tl:x

```

11.18.3 <, >, ==, <=, >=, != with Python-like chaining

Usage of single character comparison operator = is deprecated (since many years) and only == should be used. = will be removed at next major release after 1.4.

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.

```

1235 \def\XINT_expr_defbin_d #1#2%
1236 {%
1237   \def #1##1##2##3##4% \XINT_expr_exec_<op>
1238   {%
1239     \expandafter##2\expandafter##3\expandafter
1240     {\romannumeral`&&@\XINT:NEhook:f:one:from:two{\romannumeral`&&@#2##1##4}}%
1241   }%
1242 }%

```

```

1243 \def\XINT_expr_defbin_c #1#2#3#4#5#6#7#8#9%
1244 {%
1245 \def #1##1% \XINT_expr_op_<op>
1246 {%
1247 \expanded{\unexpanded{#2{##1}}\expandafter}%
1248 \romannumeral`&&\expandafter#7%
1249 \romannumeral`&&\expandafter#3\romannumeral`&&\XINT_expr_getnext
1250 }%
1251 \def #3##1% \XINT_expr_check_-_<op>
1252 {%
1253 \xint_UDsignfork
1254 ##1{\expandafter#4\romannumeral`&&#5}%
1255 -{#4##1}%
1256 \krof
1257 }%
1258 \def #4##1##2% \XINT_expr_checkp_<op>
1259 {%
1260 \ifnum ##1>#6%
1261 \expandafter#4%
1262 \romannumeral`&&\csname XINT_#9_op_##2\expandafter\endcsname
1263 \else
1264 \expandafter ##1\expandafter ##2%
1265 \fi
1266 }%
1267 \let #6\xint_c_x
1268 \def #7##1% \XINT_expr_checkc_<op>
1269 {%
1270 \ifnum ##1=\xint_c_x\expandafter#8\fi ##1%
1271 }%
1272 \edef #8##1##2##3% \XINT_expr_execc_<op>
1273 {%
1274 \csname XINT_#9_precedence_\string&\string&\endcsname
1275 \expandafter\noexpand\csname XINT_#9_itself_\string&\string&\endcsname
1276 {##3}%
1277 \XINTfstop.{##3}##2%
1278 }%
1279 \XINT_expr_defbin_d #2% \XINT_expr_exec_<op>
1280 }%
1281 \def\XINT_expr_defbin_b #1#2#3%
1282 {%
1283 \expandafter\XINT_expr_defbin_c
1284 \csname XINT_#1_op_#2\expandafter\endcsname
1285 \csname XINT_#1_exec_#2\expandafter\endcsname
1286 \csname XINT_#1_check_-_#2\expandafter\endcsname
1287 \csname XINT_#1_checkp_#2\expandafter\endcsname
1288 \csname XINT_#1_op_-xii\expandafter\endcsname
1289 \csname XINT_expr_precedence_#2\expandafter\endcsname
1290 \csname XINT_#1_checkc_#2\expandafter\endcsname
1291 \csname XINT_#1_execc_#2\endcsname
1292 {#1}%#3%
1293 }%

```

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?

```

1294 \XINT_expr_defbin_b {expr} <\xintLt
1295 \XINT_expr_defbin_b {fexpr}<\xintLt
1296 \XINT_expr_defbin_b {iiexpr}<\xintiLt
1297 \XINT_expr_defbin_b {expr} >\xintGt
1298 \XINT_expr_defbin_b {fexpr}>\xintGt
1299 \XINT_expr_defbin_b {iiexpr}>\xintiGt
1300 \XINT_expr_defbin_b {expr} {==}\xintEq
1301 \XINT_expr_defbin_b {fexpr}{==}\xintEq
1302 \XINT_expr_defbin_b {iiexpr}{==}\xintiiEq
1303 \XINT_expr_defbin_b {expr} {<=}\xintLtorEq
1304 \XINT_expr_defbin_b {fexpr}{<=}\xintLtorEq
1305 \XINT_expr_defbin_b {iiexpr}{<=}\xintiLtorEq
1306 \XINT_expr_defbin_b {expr} {>=}\xintGtorEq
1307 \XINT_expr_defbin_b {fexpr}{>=}\xintGtorEq
1308 \XINT_expr_defbin_b {iiexpr}{>=}\xintiGtorEq
1309 \XINT_expr_defbin_b {expr} {!=}\xintNotEq
1310 \XINT_expr_defbin_b {fexpr}{!=}\xintNotEq
1311 \XINT_expr_defbin_b {iiexpr}{!=}\xintiNotEq
1312 \expandafter\let\csname XINT_expr_precedence_\endcsname\xint_c_x
1313 \xintFor #1 in {expr, fexpr, iiexpr} \do
1314 {%
1315   \expandafter\let\csname XINT_#1_op_\endcsname
1316     \csname XINT_#1_op_=\endcsname
1317 }%

```

11.18.4 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.

```

1318 \def\xintSeq:tl:x #1#2%
1319 {%
1320   \expandafter\XINT_Seq:tl:x
1321   \the\numexpr \xintiCeil{#1}\expandafter.\the\numexpr \xintiFloor{#2}.%
1322 }%
1323 \def\XINT_Seq:tl:x #1.#2.%
1324 {%
1325   \ifnum #2=#1 \xint_dothis\XINT_Seq:tl:x_z\fi
1326   \ifnum #2<#1 \xint_dothis\XINT_Seq:tl:x_n\fi
1327   \xint_orthat\XINT_Seq:tl:x_p
1328   #1.#2.%
1329 }%
1330 \def\XINT_Seq:tl:x_z #1.#2.{{#1/1[0]}}%
1331 \def\XINT_Seq:tl:x_p #1.#2.%
1332 {%
1333   {#1/1[0]}\ifnum #1=#2 \XINT_Seq:tl:x_e\fi
1334   \expandafter\XINT_Seq:tl:x_p \the\numexpr #1+\xint_c_i.#2.%
1335 }%

```

```

1336 \def\XINT_Seq:tl:x_n #1.#2.%
1337 {%
1338     {#1/1[0]}\ifnum #1=#2 \XINT_Seq:tl:x_e\fi
1339     \expandafter\XINT_Seq:tl:x_n \the\numexpr #1-\xint_c_i.#2.%
1340 }%
1341 \def\XINT_Seq:tl:x_e#1#2.#3.{#1}%

\xintiSeq:tl:x
1342 \def\xintiSeq:tl:x #1#2%
1343 {%
1344     \expandafter\XINT_iiSeq:tl:x
1345     \the\numexpr \xintiCeil{#1}\expandafter.\the\numexpr \xintiFloor{#2}.%
1346 }%
1347 \def\XINT_iiSeq:tl:x #1.#2.%
1348 {%
1349     \ifnum #2=#1 \xint_dothis\XINT_iiSeq:tl:x_z\fi
1350     \ifnum #2<#1 \xint_dothis\XINT_iiSeq:tl:x_n\fi
1351     \xint_orthat\XINT_iiSeq:tl:x_p
1352     #1.#2.%
1353 }%
1354 \def\XINT_iiSeq:tl:x_z #1.#2.{{#1}}%
1355 \def\XINT_iiSeq:tl:x_p #1.#2.%
1356 {%
1357     {#1}\ifnum #1=#2 \XINT_Seq:tl:x_e\fi
1358     \expandafter\XINT_iiSeq:tl:x_p \the\numexpr #1+\xint_c_i.#2.%
1359 }%
1360 \def\XINT_iiSeq:tl:x_n #1.#2.%
1361 {%
1362     {#1}\ifnum #1=#2 \XINT_Seq:tl:x_e\fi
1363     \expandafter\XINT_iiSeq:tl:x_n \the\numexpr #1-\xint_c_i.#2.%
1364 }%

```

Contrarily to `a..b` which is limited to small integers, this works with `a`, `b`, and `d` (big) fractions. It will produce a «nil» list, if `a>b` and `d<0` or `a<b` and `d>0`.

\xintSeqA, \xintiSeqA

```

1365 \def\xintSeqA      {\expandafter\XINT_SeqA\romannumeral0\xintra}%
1366 \def\xintiSeqA    #1{\expandafter\XINT_iiSeqA\romannumeral`&&@#1;}%
1367 \def\XINT_SeqA  #1]#2{\expandafter\XINT_SeqA_a\romannumeral0\xintra {#2}#1]}%
1368 \def\XINT_iiSeqA#1;#2{\expandafter\XINT_SeqA_a\romannumeral`&&@#2;#1;}%
1369 \def\XINT_SeqA_a #1{\xint_UDzerominusfork
1370                      #1-{z}}%
1371                      0#1{n}}%
1372                      0-{p}}%
1373                      \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`.

```

1374 \def\xintSeqB:tl:x #1{\expandafter\XINT_SeqB:tl:x\romannumeral`&&\xintSeqA#1}%
1375 \def\XINT_SeqB:tl:x #1{\csname XINT_SeqB#1:tl:x\endcsname}%
1376 \def\XINT_SeqBz:tl:x #1]#2]#3{{#2}}}%
1377 \def\XINT_SeqBp:tl:x #1]#2]#3{\expandafter\XINT_SeqBp:tl:x_a\romannumeral0\xintraw{#3}#2]#1]}%
1378 \def\XINT_SeqBp:tl:x_a #1]#2]#3]%
1379 {%
1380     \xintifCmp{#1}]{#2}}}%
1381     {{{#2}}}{#2}}\expandafter\XINT_SeqBp:tl:x_b\romannumeral0\xintadd{#3}]{#2}]{#1}#3}}}%
1382 }%
1383 \def\XINT_SeqBp:tl:x_b #1]#2]#3]%
1384 {%
1385     \xintifCmp{#1}]{#2}}}%
1386     {#1}}\expandafter\XINT_SeqBp:tl:x_b\romannumeral0\xintadd{#3}]{#1}]{#2}]{#3}}{#1}}}%
1387 }%
1388 \def\XINT_SeqBn:tl:x #1]#2]#3{\expandafter\XINT_SeqBn:tl:x_a\romannumeral0\xintraw{#3}#2]#1]}%
1389 \def\XINT_SeqBn:tl:x_a #1]#2]#3]%
1390 {%
1391     \xintifCmp{#1}]{#2}}}%
1392     {#2}}\expandafter\XINT_SeqBn:tl:x_b\romannumeral0\xintadd{#3}]{#2}]{#1}#3}}{#2}}}%
1393 }%
1394 \def\XINT_SeqBn:tl:x_b #1]#2]#3]%
1395 {%
1396     \xintifCmp{#1}]{#2}}}%
1397     {{{#1}}}{#1}}\expandafter\XINT_SeqBn:tl:x_b\romannumeral0\xintadd{#3}]{#1}]{#2}]{#3}}}%
1398 }%

```

\xintiiSeqB:tl:x

```

1399 \def\xintiiSeqB:tl:x #1{\expandafter\XINT_iiSeqB:tl:x\romannumeral`&&\xintiiSeqA#1}%
1400 \def\XINT_iiSeqB:tl:x #1{\csname XINT_iiSeqB#1:tl:x\endcsname}%
1401 \def\XINT_iiSeqBz:tl:x #1;#2;#3{{#2}}}%
1402 \def\XINT_iiSeqBp:tl:x #1;#2;#3{\expandafter\XINT_iiSeqBp:tl:x_a\romannumeral`&&#3;#2;#1;}%
1403 \def\XINT_iiSeqBp:tl:x_a #1;#2;#3;%
1404 {%
1405     \xintiiifCmp{#1}]{#2}}}%
1406     {{{#2}}}{#2}}\expandafter\XINT_iiSeqBp:tl:x_b\romannumeral0\xintiiadd{#3}]{#2};#1;#3;}}}%
1407 }%
1408 \def\XINT_iiSeqBp:tl:x_b #1;#2;#3;%
1409 {%
1410     \xintiiifCmp{#1}]{#2}}}%
1411     {#1}}\expandafter\XINT_iiSeqBp:tl:x_b\romannumeral0\xintiiadd{#3}]{#1};#2;#3;}}{#1}}}%
1412 }%
1413 \def\XINT_iiSeqBn:tl:x #1;#2;#3{\expandafter\XINT_iiSeqBn:tl:x_a\romannumeral`&&#3;#2;#1;}%
1414 \def\XINT_iiSeqBn:tl:x_a #1;#2;#3;%
1415 {%
1416     \xintiiifCmp{#1}]{#2}}}%
1417     {#2}}\expandafter\XINT_iiSeqBn:tl:x_b\romannumeral0\xintiiadd{#3}]{#2};#1;#3;}}{#2}}}%
1418 }%
1419 \def\XINT_iiSeqBn:tl:x_b #1;#2;#3;%
1420 {%
1421     \xintiiifCmp{#1}]{#2}}}%
1422     {{{#1}}}{#1}}\expandafter\XINT_iiSeqBn:tl:x_b\romannumeral0\xintiiadd{#3}]{#1};#2;#3;}}}%
1423 }%

```

11.19 Square brackets [] both as a container and a Python slicer

Refactored at 1.4

The architecture allows to implement separately a «left» and a «right» precedence and this is crucial.

11.19.1	[...] as «oneple» constructor	351
11.19.2	[...] brackets and : operator for NumPy-like slicing and item indexing syntax	352
11.19.3	Macro layer implementing indexing and slicing	354

11.19.1 [...] as «oneple» constructor

In the definition of `\XINT_expr_op_obracket` the parameter is trash `{}`. The `[` is intercepted by the `getnextfork` and handled via the `\xint_c_ii^v` highest precedence trick to get `op_obracket` executed.

```

1424 \def\xint_expr_itself_obracket{obracket}%
1425 \catcode`] 11 \catcode`[ 11
1426 \def\xint_expr_defbin_c #1#2#3#4#5#6%
1427 {%
1428   \def #1##1%
1429   {%
1430     \expandafter#3\romannumerical`&&\xint_expr_getnext
1431   }%
1432   \def #2##1% op_]
1433   {%
1434     \expanded{\unexpanded{\xint_expr_put_op_first{##1}}\expandafter}%
1435     \romannumerical`&&\xint_expr_getop
1436   }%
1437   \def #3##1% until_cbracket_a
1438   {%
1439     \xint_UDsignfork
1440     ##1{\expandafter#4\romannumerical`&&#5}% #5 = op_-xii
1441     -{##1}%
1442     \krof
1443   }%
1444   \def #4##1##2% until_cbracket_b
1445   {%
1446     \ifcase #1\expandafter\xint_expr_missing_]
1447     \or \expandafter\xint_expr_missing_]
1448     \or \expandafter#2%
1449     \else
1450     \expandafter #4%
1451     \romannumerical`&&\csname XINT_#6_op_##2\expandafter\endcsname
1452     \fi
1453   }%
1454 }%
1455 \def\xint_expr_defbin_b #1%
1456 {%
1457   \expandafter\xint_expr_defbin_c
1458   \csname XINT_#1_op_obracket\expandafter\endcsname
1459   \csname XINT_#1_op_] \expandafter\endcsname
1460   \csname XINT_#1_until_cbracket_a\expandafter\endcsname
1461   \csname XINT_#1_until_cbracket_b\expandafter\endcsname

```

```

1462 \csname XINT_#1_op_-xii\endcsname
1463 {#1}%
1464 }%
1465 \XINT_expr_defbin_b {expr}%
1466 \XINT_expr_defbin_b {fexpr}%
1467 \XINT_expr_defbin_b {iexpr}%
1468 \def\XINT_expr_missing_
1469 {\XINT_expandableerror{Ooops, looks like we are missing a ] here. Goodbye!}%
1470 \xint_c_ \XINT_expr_done}%
1471 \let\XINT_expr_precedence_]\xint_c_ii

```

11.19.2 [...] brackets and : operator for NumPy-like slicing and item indexing syntax

The opening bracket [for the nutple constructor is filtered out by \XINT_expr_getnextfork and becomes «obracquet» 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.

```

1472 \def\XINT_expr_defbin_c #1#2#3#4#5#6%
1473 {%
1474 \def #1##1% \XINT_expr_op_[
1475 {%
1476 \expanded{\unexpanded{#2{##1}}\expandafter}%
1477 \romannumerical`&&@\expandafter#3\romannumerical`&&@\XINT_expr_getnext
1478 }%
1479 \def #2##1##2##3##4% \XINT_expr_exec_]
1480 {%
1481 \expandafter\XINT_expr_put_op_first
1482 \expanded
1483 {%
1484 {\XINT:NEhook:x:listsel\XINT_ListSel_top ##1_##4&({##1})\expandafter}%
1485 \expandafter
1486 }%
1487 \romannumerical`&&@\XINT_expr_getop
1488 }%
1489 \def #3##1% \XINT_expr_check-_]
1490 {%
1491 \xint_UDsignfork
1492 ##1{\expandafter#4\romannumerical`&&@#5}%
1493 -{#4##1}%
1494 \krof
1495 }%
1496 \def #4##1##2% \XINT_expr_checkp_]
1497 {%

```



```

1498     \ifcase ##1\XINT_expr_missing_]
1499     \or \XINT_expr_missing_]
1500     \or \expandafter##1\expandafter##2%
1501     \else \expandafter#4%
1502         \romannumeral`&&\csname XINT_#6_op_##2\expandafter\endcsname
1503     \fi
1504 }%
1505 }%
1506 \let\XINT_expr_precedence_ \xint_c_xx
1507 \def\XINT_expr_defbin_b #1%
1508 {%
1509     \expandafter\XINT_expr_defbin_c
1510     \csname XINT_#1_op_[\expandafter\endcsname
1511     \csname XINT_#1_exec_]\expandafter\endcsname
1512     \csname XINT_#1_check_]\expandafter\endcsname
1513     \csname XINT_#1_checkp_]\expandafter\endcsname
1514     \csname XINT_#1_op_-xii\endcsname
1515     {#1}%
1516 }%
1517 \XINT_expr_defbin_b {expr}%
1518 \XINT_expr_defbin_b {flexpr}%
1519 \XINT_expr_defbin_b {iiexpr}%
1520 \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.

```

1521 \def\XINT_expr_defbin_c #1#2#3#4#5#6%
1522 {%
1523     \def #1##1% \XINT_expr_op_:
1524     {%
1525         \expanded{\unexpanded{#2{##1}}\expandafter}%
1526         \romannumeral`&&\expandafter#3\romannumeral`&&\XINT_expr_getnext
1527     }%
1528     \def #2##1##2##3##4% \XINT_expr_exec_:
1529     {%
1530         ##2##3{:#1{0};##4:_}%
1531     }%
1532     \def #3##1% \XINT_expr_check_::
1533     {\xint_UDsignfork
1534         ##1{\expandafter#4\romannumeral`&&@#5}%
1535         -{#4##1}%
1536     \krof
1537     }%
1538     \def #4##1##2% \XINT_expr_checkp_:
1539     {%
1540         \ifnum ##1>\XINT_expr_precedence_:
1541             \expandafter #4\romannumeral`&&@%
1542             \csname XINT_#6_op_##2\expandafter\endcsname
1543         \else

```

```

1544     \expandafter##1\expandafter##2%
1545     \fi
1546   }%
1547 }%
1548 \let\XINT_expr_precedence_:\xint_c_vi
1549 \def\XINT_expr_defbin_b #1%
1550 {%
1551   \expandafter\XINT_expr_defbin_c
1552   \csname XINT_#1_op_:\expandafter\endcsname
1553   \csname XINT_#1_exec_:\expandafter\endcsname
1554   \csname XINT_#1_check_:\expandafter\endcsname
1555   \csname XINT_#1_checkp_:\expandafter\endcsname
1556   \csname XINT_#1_op_-xii\endcsname {#1}%
1557 }%
1558 \XINT_expr_defbin_b {expr}%
1559 \XINT_expr_defbin_b {flexpr}%
1560 \XINT_expr_defbin_b {iiexpr}%

```

11.19.3 Macro layer implementing indexing and slicing

xintexpr applies slicing not only to «objects» (which can be passed as arguments to functions) but also to «oples».

Our «nlists» are not necessarily regular N-dimensional arrays à la NumPy. Leaves can be at arbitrary depths. If we were handling regular «ndarrays», we could proceed a bit differently.

For the related explanations, refer to the user manual.

Notice that currently the code uses f-expandable (and not using \expanded) macros \xintApply, \xintApplyUnbraced, \xintKeep, \xintTrim, \xintNthOne from *xinttools*.

But the whole expansion happens inside an \expanded context, so possibly some gain could be achieved with x-expandable variants (*xintexpr* < 1.4 had an \xintKeep:x:csv).

I coded \xintApply:x and \xintApplyUnbraced:x in *xinttools*, Brief testing indicated they were perhaps a bit better for 5x5x5x5 and 15x15x15x15 arrays of 8 digits numbers and for 30x30x15 with 16 digits numbers: say 1% gain... this seems to raise to between 4% and 5% for 400x400 array of 1 digit...

Currently sticking with old macros.

```

1561 \def\XINT_ListSel_deeper #1%
1562 {%
1563   \if :#1\xint_dothis\XINT_ListSel_slice_next\fi
1564   \xint_orthat {\XINT_ListSel_extract_next {#1}}%
1565 }%
1566 \def\XINT_ListSel_slice_next #1(%
1567 {%
1568   \xintApply{\XINT_ListSel_recurse{:#1}}%
1569 }%
1570 \def\XINT_ListSel_extract_next #1(%
1571 {%
1572   \xintApplyUnbraced{\XINT_ListSel_recurse{#1}}%
1573 }%
1574 \def\XINT_ListSel_recurse #1#2%
1575 {%
1576   \XINT_ListSel_check #2_#1({#2}\expandafter\empty\empty
1577 }%
1578 \def\XINT_ListSel_check{\expandafter\XINT_ListSel_check_a \string}%

```

```

1579 \def\XINT_ListSel_check_a #1%
1580 {%
1581     \if #1\bgroup\xint_dothis\XINT_ListSel_check_is_ok\fi
1582     \xint_orthat\XINT_ListSel_check_leaf
1583 }%
1584 \def\XINT_ListSel_check_leaf #1\expandafter{\expandafter}%
1585 \def\XINT_ListSel_check_is_ok
1586 {%
1587     \expandafter\XINT_ListSel_check_is_ok_a\expandafter{\string}%
1588 }%
1589 \def\XINT_ListSel_check_is_ok_a #1__#2%
1590 {%
1591     \if :#2\xint_dothis{\XINT_ListSel_slice}\fi
1592     \xint_orthat {\XINT_ListSel_nthone {#2}}%
1593 }%
1594 \def\XINT_ListSel_top #1#2%
1595 {%
1596     \if _\noexpand#2%
1597         \expandafter\XINT_ListSel_top_one_or_none\string#1.\else
1598         \expandafter\XINT_ListSel_top_at_least_two\fi
1599 }%
1600 \def\XINT_ListSel_top_at_least_two #1__{\XINT_ListSel_top_ople}%
1601 \def\XINT_ListSel_top_one_or_none #1%
1602 {%
1603     \if #1_\xint_dothis\XINT_ListSel_top_nil\fi
1604     \if #1.\xint_dothis\XINT_ListSel_top_nutple_a\fi
1605     \if #1\bgroup\xint_dothis\XINT_ListSel_top_nutple\fi
1606     \xint_orthat\XINT_ListSel_top_number
1607 }%
1608 \def\XINT_ListSel_top_nil #1\expandafter#2\expandafter{\fi\expandafter}%
1609 \def\XINT_ListSel_top_nutple
1610 {%
1611     \expandafter\XINT_ListSel_top_nutple_a\expandafter{\string}%
1612 }%
1613 \def\XINT_ListSel_top_nutple_a #1_#2#3(#4%
1614 {%
1615     \fi\if :#2\xint_dothis{\XINT_ListSel_slice #3(#4)}\fi
1616     \xint_orthat {\XINT_ListSel_nthone {#2}#3(#4}%
1617 }%
1618 \def\XINT_ListSel_top_number #1_{\fi\XINT_ListSel_top_ople}%
1619 \def\XINT_ListSel_top_ople #1%
1620 {%
1621     \if :#1\xint_dothis\XINT_ListSel_slice\fi
1622     \xint_orthat {\XINT_ListSel_nthone {#1}}%
1623 }%
1624 \def\XINT_ListSel_slice #1%
1625 {%
1626     \expandafter\XINT_ListSel_slice_a \expandafter{\romannumeral0\xintnum{#1}}%
1627 }%
1628 \def\XINT_ListSel_slice_a #1#2;#3#4%
1629 {%
1630     \if _#4\expandafter\XINT_ListSel_s_b

```

```

1631         \else\expandafter\XINT_ListSel_slice_b\fi
1632     #1;#3%
1633 }%
1634 \def\XINT_ListSel_s_b #1#2;#3#4%
1635 {%
1636     \if &#4\expandafter\XINT_ListSel_s_last\fi
1637     \XINT_ListSel_s_c #1{#1#2}{#4}%
1638 }%
1639 \def\XINT_ListSel_s_last\XINT_ListSel_s_c #1#2#3(#4%
1640 {%
1641     \if-#1\expandafter\xintKeep\else\expandafter\xintTrim\fi {#2}{#4}%
1642 }%
1643 \def\XINT_ListSel_s_c #1#2#3(#4%
1644 {%
1645     \expandafter\XINT_ListSel_deeper
1646     \expanded{\unexpanded{#3}(\expandafter)\expandafter{%
1647     \romannumeral0%
1648     \if-#1\expandafter\xintkeep\else\expandafter\xinttrim\fi {#2}{#4}}%
1649 }%

```

`\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 ople `{}`, not the empty bracketed ople `{}}`.

I have thus added `\xintNthOne` to `xinttools` which does not strip brace pair from an extracted item.

Attention: `\XINT_nthoney_a` does no expansion on second argument. But here arguments are either numerical or already expanded. Normally.

```

1650 \def\XINT_ListSel_nthone #1#2%
1651 {%
1652     \if &#2\expandafter\XINT_ListSel_nthone_last\fi
1653     \XINT_ListSel_nthone_a {#1}{#2}%
1654 }%
1655 \def\XINT_ListSel_nthone_a #1#2(#3%
1656 {%
1657     \expandafter\XINT_ListSel_deeper
1658     \expanded{\unexpanded{#2}(\expandafter)\expandafter{%
1659     \romannumeral0\expandafter\XINT_nthoney_a\the\numexpr\xintNum{#1}.#3}}%
1660 }%
1661 \def\XINT_ListSel_nthone_last\XINT_ListSel_nthone_a #1#2(%#3%
1662 {%
1663     \romannumeral0\expandafter\XINT_nthoney_a\the\numexpr\xintNum{#1}.#3}
1664 }%

```

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.

```

1665 \def\XINT_ListSel_slice_b #1;#2_#3%

```

```

1666 {%
1667   \if &#3\expandafter\XINT_ListSel_slice_last\fi
1668   \expandafter\XINT_ListSel_slice_c \expandafter{\romannumeral0\xintnum{#2}};#1;{#3}%
1669 }%
1670 \def\XINT_ListSel_slice_last\expandafter\XINT_ListSel_slice_c #1;#2;#3(%#4
1671 {%
1672   \expandafter\XINT_ListSel_slice_last_c #1;#2;{%#4}
1673 }%
1674 \def\XINT_ListSel_slice_last_c #1;#2;#3%
1675 {%
1676   \romannumeral0\XINT_ListSel_slice_d #2;#1;{#3}%
1677 }%
1678 \def\XINT_ListSel_slice_c #1;#2;#3(%#4%
1679 {%
1680   \expandafter\XINT_ListSel_deeper
1681   \expanded{\unexpanded{#3}(\expandafter)\expandafter{%
1682     \romannumeral0\XINT_ListSel_slice_d #2;#1;{#4}}}%
1683 }%
1684 \def\XINT_ListSel_slice_d #1#2;#3#4;%
1685 {%
1686   \xint_UDsignsfork
1687     #1#3\XINT_ListSel_N:N
1688     #1-\XINT_ListSel_N:P
1689     -#3\XINT_ListSel_P:N
1690     --\XINT_ListSel_P:P
1691   \krof #1#2;#3#4;%
1692 }%
1693 \def\XINT_ListSel_P:P #1;#2;#3%
1694 {%
1695   \unless\ifnum #1<#2 \expandafter\xint_gob_andstop_iii\fi
1696   \xintkeep{#2-#1}{\xintTrim{#1}{#3}}%
1697 }%
1698 \def\XINT_ListSel_N:N #1;#2;#3%
1699 {%
1700   \expandafter\XINT_ListSel_N:N_a
1701   \the\numexpr #2-#1\expandafter;\the\numexpr#1+\xintLength{#3};{#3}%
1702 }%
1703 \def\XINT_ListSel_N:N_a #1;#2;#3%
1704 {%
1705   \unless\ifnum #1>\xint_c_ \expandafter\xint_gob_andstop_iii\fi
1706   \xintkeep{#1}{\xintTrim{\ifnum#2<\xint_c_\xint_c_\else#2\fi}{#3}}%
1707 }%
1708 \def\XINT_ListSel_N:P #1;#2;#3%
1709 {%
1710   \expandafter\XINT_ListSel_N:P_a
1711   \the\numexpr #1+\xintLength{#3};#2;{#3}%
1712 }%
1713 \def\XINT_ListSel_N:P_a #1#2;%
1714   {\if -#1\expandafter\XINT_ListSel_0:P\fi\XINT_ListSel_P:P #1#2;}%
1715 \def\XINT_ListSel_0:P\XINT_ListSel_P:P #1;{\XINT_ListSel_P:P 0;}%
1716 \def\XINT_ListSel_P:N #1;#2;#3%
1717 {%

```

```

1718 \expandafter\XINT_ListSel_P:N_a
1719 \the\numexpr #2+\xintLength{#3};#1;{#3}%
1720 }%
1721 \def\XINT_ListSel_P:N_a #1#2;#3;%
1722 {\if -#1\expandafter\XINT_ListSel_P:0\fi\XINT_ListSel_P:P #3;#1#2;}%
1723 \def\XINT_ListSel_P:0\XINT_ListSel_P:P #1;#2;{\XINT_ListSel_P:P #1;0;}%

```

11.20 Support for raw A/B[N]

Releases earlier than 1.1 required the use of braces around A/B[N] input. The [N] is now implemented directly. *BUT* this uses a delimited macro! thus N is not allowed to be itself an expression (I could add it...). `\xintE`, `\xintiiE`, and `\XINTinFloatE` all put #2 in a `\numexpr`. But attention to the fact that `\numexpr` stops at spaces separating digits: `\the\numexpr 3 + 7 9\relax` gives `109\relax` !! Hence we have to be careful.

`\numexpr` will not handle catcode 11 digits, but adding a `\detokenize` will suddenly make illicit for N to rely on macro expansion.

At 1.4, [is already overloaded and it is not easy to support this. We do this by a kludge maintaining more or less former (very not efficient) way but using \$ sign which is free for time being. No, finally I use the null character, should be safe enough! (I hesitated about using R with catcode 12).

As for ? operator we needed to hack into `\XINT_expr_getop_b` for intercepting that pseudo operator. See also `\XINT_expr_scanint_c` (`\XINT_expr_rawxintfrac`).

```

1724 \catcode0 11
1725 \let\XINT_expr_precedence_&&& \xint_c_xiv
1726 \def\XINT_expr_op_&&& #1#2]%
1727 {%
1728 \expandafter\XINT_expr_put_op_first
1729 \expanded{{{ \xintE#1{ \xint_zapspaces #2 \xint_gobble_i}}}%
1730 \expandafter}\romannumeral`&&&\XINT_expr_getop
1731 }%
1732 \def\XINT_iiexpr_op_&&& #1#2]%
1733 {%
1734 \expandafter\XINT_expr_put_op_first
1735 \expanded{{{ \xintiiE#1{ \xint_zapspaces #2 \xint_gobble_i}}}%
1736 \expandafter}\romannumeral`&&&\XINT_expr_getop
1737 }%
1738 \def\XINT_flexpr_op_&&& #1#2]%
1739 {%
1740 \expandafter\XINT_expr_put_op_first
1741 \expanded{{{ \XINTinFloatE#1{ \xint_zapspaces #2 \xint_gobble_i}}}%
1742 \expandafter}\romannumeral`&&&\XINT_expr_getop
1743 }%
1744 \catcode0 12

```

11.21 ? as two-way and ?? as three-way «short-circuit» conditionals

Comments undergoing reconstruction.

```

1745 \let\XINT_expr_precedence_? \xint_c_xx
1746 \catcode`- 11
1747 \def\XINT_expr_op_? {\XINT_expr_op__? \XINT_expr_op_-xii}%
1748 \def\XINT_flexpr_op_?{\XINT_expr_op__? \XINT_flexpr_op_-xii}%

```

```

1749 \def\XINT_iiexpr_op_?{\XINT_expr_op__? \XINT_iiexpr_op_-xii}%
1750 \catcode`- 12
1751 \def\XINT_expr_op__? #1#2#3%
1752     {\XINT_expr_op__?_a #3!\xint_bye\XINT_expr_exec_? {#1}{#2}{#3}}%
1753 \def\XINT_expr_op__?_a #1{\expandafter\XINT_expr_op__?_b\detokenize{#1}}%
1754 \def\XINT_expr_op__?_b #1%
1755     {\if ?#1\expandafter\XINT_expr_op__?_c\else\expandafter\xint_bye\fi }%
1756 \def\XINT_expr_op__?_c #1{\xint_gob_til_! #1\XINT_expr_op_?? !\xint_bye}%
1757 \def\XINT_expr_op_?? !\xint_bye\xint_bye\XINT_expr_exec_?{\XINT_expr_exec_??}%
1758 \catcode`- 11
1759 \def\XINT_expr_exec_? #1#2%
1760 {%
1761     \expandafter\XINT_expr_check_-_after?\expandafter#1%
1762     \romannumeral`&&\expandafter\XINT_expr_getnext\romannumeral0\xintiiifnotzero#2%
1763 }%
1764 \def\XINT_expr_exec_?? #1#2#3%
1765 {%
1766     \expandafter\XINT_expr_check_-_after?\expandafter#1%
1767     \romannumeral`&&\expandafter\XINT_expr_getnext\romannumeral0\xintiiifsgn#2%
1768 }%
1769 \def\XINT_expr_check_-_after? #1{%
1770 \def\XINT_expr_check_-_after? ##1##2%
1771 {%
1772     \xint_UDsignfork
1773     ##2{##1}%
1774     #1{##2}%
1775     \krof
1776 }}\expandafter\XINT_expr_check_-_after?\string -%
1777 \catcode`- 12

```

11.22 ! as postfix factorial operator

```

1778 \let\XINT_expr_precedence_! \xint_c_xx
1779 \def\XINT_expr_op_! #1%
1780 {%
1781     \expandafter\XINT_expr_put_op_first
1782     \expanded{{\romannumeral`&&\XINT:NEhook:f:one:from:one
1783     {\romannumeral`&&\xintFac#1}}\expandafter}\romannumeral`&&\XINT_expr_getop
1784 }%
1785 \def\XINT_flexpr_op_! #1%
1786 {%
1787     \expandafter\XINT_expr_put_op_first
1788     \expanded{{\romannumeral`&&\XINT:NEhook:f:one:from:one
1789     {\romannumeral`&&\XINTinFloatFac#1}}\expandafter}\romannumeral`&&\XINT_expr_getop
1790 }%
1791 \def\XINT_iiexpr_op_! #1%
1792 {%
1793     \expandafter\XINT_expr_put_op_first
1794     \expanded{{\romannumeral`&&\XINT:NEhook:f:one:from:one
1795     {\romannumeral`&&\xintiiFac#1}}\expandafter}\romannumeral`&&\XINT_expr_getop
1796 }%

```

11.23 User defined variables

11.23.1	<code>\xintdefvar</code> , <code>\xintdefiivar</code> , <code>\xintdeffloatvar</code>	360
11.23.2	<code>\xintunassignvar</code>	362

11.23.1 `\xintdefvar`, `\xintdefiivar`, `\xintdeffloatvar`

1.1.

1.2p (2017/12/01). extends `\xintdefvar` et al. to accept simultaneous assignments to multiple variables.

1.3c (2018/06/17). Use `\xintexprSafeCatcodes` (to palliate issue with active semi-colon from Babel+French if in body of a \TeX document).

And allow usage with both syntaxes `name:=expr`; or `name=expr`; . Also the colon may have catcode 11, 12, or 13 with no issue. Variable names may contain letters, digits, underscores, and must not start with a digit. Names starting with @ or an underscore are reserved.

- currently @, @1, @2, @3, and @4 are reserved because they have special meanings for use in iterations,
- @@, @@@, @@@@ are also reserved but are technically functions, not variables: a user may possibly define @@ as a variable name, but if it is followed by parentheses, the function interpretation will be applied (rather than the variable interpretation followed by a tacit multiplication),
- since 1.21, the underscore _ may be used as separator of digits in long numbers. Hence a variable whose name starts with _ will not play well with the mechanism of tacit multiplication of variables by numbers: the underscore will be removed from input stream by the number scanner, thus creating an undefined or wrong variable name, or none at all if the variable name was an initial _ followed by digits.

Note that the optional argument [P] as usable with `\xintfloatexpr` is **not** supported by `\xintdeffloatvar`. One must do `\xintdeffloatvar foo = \xintfloatexpr[16] blabla \relax`; to achieve the effect.

1.4 (2020/01/27). The expression will be fetched up to final semi-colon in a manner allowing inner semi-colons as used in the `iter()`, `rseq()`, `subsm()`, `subsn()` etc... syntax. They don't need to be hidden within a braced pair anymore.

TODO: prior to 1.4 a variable «value» was passed along as a single token. Now it is managed, like everything else, as explicit braced contents. But most of the code is ready for passing it along again as a single (braced, now) token again, because all needed `\expanded/\unexpanded` things are in place. However this is «most of the code». I am really eager to get 1.4 released now, because I can't devote more time in immediate future. It is too late to engage into an umpteenth deep refactoring at a time where things work and many new features were added and most aspects of inner working got adapted. However in future it could be that variables holding large data will be managed much faster.

```

1797 \catcode`* 11
1798 \def\xINT_expr_defvar_one #1#2%
1799 {%
1800   \XINT_global
1801   \expandafter\edef\csname XINT_expr_varvalue_#1\endcsname {#2}%
1802   \XINT_expr_defvar_one_b {#1}%
1803 }%
1804 \def\xINT_expr_defvar_one_b #1%
1805 {%
1806   \XINT_global
1807   \expandafter\edef\csname XINT_expr_var_#1\endcsname

```



```

1808     {{\expandafter\noexpand\csname XINT_expr_varvalue_#1\endcsname}}%
1809 \XINT_global
1810 \expandafter\edef\csname XINT_expr_onliteral_#1\endcsname
1811     {\noexpand\expandafter\XINT_expr_precedence_***
1812     \noexpand\expandafter *\expandafter
1813     \noexpand\csname XINT_expr_var_#1\endcsname{}}%
1814 \ifxintverbose\xintMessage{xintexpr}{Info}
1815     {Variable "#1" \ifxintglobaldefs globally \fi
1816     defined with value \csname XINT_expr_varvalue_#1\endcsname.}%
1817 \fi
1818 }%
1819 \catcode`* 12
1820 \catcode`~ 13
1821 \catcode`: 12
1822 \def\XINT_expr_defvar_getname #1:#2~%
1823 {%
1824     \endgroup
1825     \def\XINT_defvar_tmpa{#1}\edef\XINT_defvar_tmpc{\xintCSVLength{#1}}%
1826 }%
1827 \def\XINT_expr_defvar #1#2%
1828 {%
1829     \def\XINT_defvar_tmpa{#2}%
1830     \expandafter\XINT_expr_defvar_a\expandafter#1\romannumeral\XINT_expr_fetch_to_semicolon
1831 }%
1832 \def\XINT_expr_defvar_a #1#2%
1833 {%
1834     \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.

```

1835 \begingroup\lccode`~: \lowercase{\let~}\empty
1836 \edef\XINT_defvar_tmpa{\XINT_defvar_tmpa}%
1837 \edef\XINT_defvar_tmpa{\xint_zapspace_o\XINT_defvar_tmpa}%
1838 \expandafter\XINT_expr_defvar_getname
1839     \detokenize\expandafter{\XINT_defvar_tmpa}:~%
1840 \ifcase\XINT_defvar_tmpc\space
1841     \xintMessage {xintexpr}{Error}
1842     {Aborting: not allowed to declare variable with empty name.}%
1843 \or
1844     \XINT_global
1845     \expandafter\edef\csname XINT_expr_varvalue_\XINT_defvar_tmpa\endcsname
1846         {\romannumeral0#1#2\relax}%
1847     \XINT_expr_defvar_one_b\XINT_defvar_tmpa
1848 \else
1849     \edef\XINT_defvar_tmpb{\romannumeral0#1#2\relax}%
1850     \edef\XINT_defvar_tmpe{\expandafter\xintLength\expandafter{\XINT_defvar_tmpb}}%
1851     \let\XINT_defvar_tmpe\empty
1852     \if1\XINT_defvar_tmpe
1853         \def\XINT_defvar_tmpe{unpacked }%
1854         \oodef\XINT_defvar_tmpe{\expandafter\xint_firstofone\XINT_defvar_tmpe}%

```

```

1855     \edef\XINT_defvar_tmpd{\expandafter\xintLength\expandafter{\XINT_defvar_tmpb}}%
1856     \fi
1857     \ifnum\XINT_defvar_tmpe=\XINT_defvar_tmpd\space
1858     \xintAssignArray\xintCSVtoList\XINT_defvar_tmpe\to\XINT_defvar_tmpevar
1859     \xintAssignArray\xintApply\XINT_embrace{\XINT_defvar_tmpb}\to\XINT_defvar_tmpeval
1860     \def\XINT_defvar_tmpe{1}%
1861     \xintloop
1862     \expandafter\XINT_expr_defvar_one
1863     \csname XINT_defvar_tmpevar\XINT_defvar_tmpe\expandafter\endcsname
1864     \csname XINT_defvar_tmpeval\XINT_defvar_tmpe\endcsname
1865     \ifnum\XINT_defvar_tmpe<\XINT_defvar_tmpe\space
1866     \edef\XINT_defvar_tmpe{\the\numexpr\XINT_defvar_tmpe+1}%
1867     \repeat
1868     \xintRelaxArray\XINT_defvar_tmpevar
1869     \xintRelaxArray\XINT_defvar_tmpeval
1870     \else
1871     \xintMessage {xintexpr}{Error}
1872     {Aborting: mismatch between number of variables (\XINT_defvar_tmpe)
1873     and number of \XINT_defvar_tmpe values (\XINT_defvar_tmpe).}%
1874     \fi
1875     \fi
1876     \let\XINT_defvar_tmpe\empty
1877     \let\XINT_defvar_tmpeb\empty
1878     \let\XINT_defvar_tmpe\empty
1879     \let\XINT_defvar_tmpe\empty
1880 }%
1881 \catcode`~ 3
1882 \catcode`: 11

```

This `SafeCatcodes` is mainly in the hope that semi-colon ending the expression can still be sanitized.

```

1883 \def\xintdefvar      {\xintexprSafeCatcodes\xintdefvar_a}%
1884 \def\xintdefiivar   {\xintexprSafeCatcodes\xintdefiivar_a}%
1885 \def\xintdeffloatvar {\xintexprSafeCatcodes\xintdeffloatvar_a}%
1886 \def\xintdefvar_a   #1={\XINT_expr_defvar\xintthebareeval    {#1}}%
1887 \def\xintdefiivar_a #1={\XINT_expr_defvar\xintthebareieeval  {#1}}%
1888 \def\xintdeffloatvar_a #1={\XINT_expr_defvar\xintthebarefloateval {#1}}%

```

11.23.2 \xintunassignvar

1.2e.

1.3d. Embarrassingly I had for a long time a misunderstanding of `\ifcsname` (let's blame its documentation) and I was not aware that it chooses FALSE branch if tested control sequence has been `\let` to `\undefined`... So earlier version didn't do the right thing (and had another bug: failure to protect `\.=0` from expansion).

The `\ifcsname` tests are done in `\XINT_expr_op__` and `\XINT_expr_op``.

```

1889 \def\xintunassignvar #1{%
1890     \edef\XINT_unvar_tmpe{#1}%
1891     \edef\XINT_unvar_tmpe {\xint_zapspace_o\XINT_unvar_tmpe}%
1892     \ifcsname XINT_expr_var_\XINT_unvar_tmpe\endcsname
1893     \ifnum\expandafter\xintLength\expandafter{\XINT_unvar_tmpe}=\@ne
1894     \expandafter\xintnewdummy\XINT_unvar_tmpe

```

```

1895     \else
1896     \XINT_global\expandafter
1897     \let\csname XINT_expr_varvalue_\XINT_unvar_tmpa\endcsname\xint_undefined
1898     \XINT_global\expandafter
1899     \let\csname XINT_expr_var_\XINT_unvar_tmpa\endcsname\xint_undefined
1900     \XINT_global\expandafter
1901     \let\csname XINT_expr_onliteral_\XINT_unvar_tmpa\endcsname\xint_undefined
1902     \ifxintverbose\xintMessage {xintexpr}{Info}
1903     {Variable \XINT_unvar_tmpa\space has been
1904     \ifxintglobaldefs globally \fi ``unassigned''.}%
1905     \fi
1906     \fi
1907 \else
1908     \xintMessage {xintexpr}{Warning}
1909     {Error: there was no such variable \XINT_unvar_tmpa\space to unassign.}%
1910     \fi
1911 }%

```

11.24 Support for dummy variables

11.24.1	<code>\xintnewdummy</code>	363
11.24.2	<code>\xintensuredummy</code> , <code>\xintrestorevariable</code>	364
11.24.3	Checking (without expansion) that a symbolic expression contains correctly nested parentheses	365
11.24.4	Fetching balanced expressions E1, E2 and a variable name Name from E1, Name=E2)	366
11.24.5	Fetching a balanced expression delimited by a semi-colon	366
11.24.6	Low-level support for omit and abort keywords, the <code>break()</code> function, the <code>n++</code> construct and the semi-colon as used in the syntax of <code>seq()</code> , <code>add()</code> , <code>mul()</code> , <code>iter()</code> , <code>rseq()</code> , <code>iterr()</code> , <code>rrseq()</code> , <code>subsm()</code> , <code>subsn()</code> , <code>ndseq()</code> , <code>ndmap()</code>	366
11.24.7	Reserved dummy variables <code>@</code> , <code>@1</code> , <code>@2</code> , <code>@3</code> , <code>@4</code> , <code>@@</code> , <code>@@(1)</code> , . . . , <code>@@@</code> , <code>@@@(1)</code> , . . . for recursions	368

11.24.1 `\xintnewdummy`

Comments under reconstruction.

1.4 adds multi-letter names as usable dummy variables!

```

1912 \catcode`* 11
1913 \def\XINT_expr_makedummy #1%
1914 {%
1915     \edef\XINT_tmpa{\xint_zapspace #1 \xint_gobble_i}%
1916     \ifcsname XINT_expr_var_\XINT_tmpa\endcsname
1917     \XINT_global
1918     \expandafter\let\csname XINT_expr_var_\XINT_tmpa\old\expandafter\endcsname
1919     \csname XINT_expr_var_\XINT_tmpa\expandafter\endcsname
1920     \fi
1921     \ifcsname XINT_expr_onliteral_\XINT_tmpa\endcsname
1922     \XINT_global
1923     \expandafter\let\csname XINT_expr_onliteral_\XINT_tmpa\old\expandafter\endcsname
1924     \csname XINT_expr_onliteral_\XINT_tmpa\expandafter\endcsname
1925     \fi
1926     \expandafter\XINT_global
1927     \expanded

```

```

1928 {\edef\expandafter\noexpand
1929     \csname XINT_expr_var_\XINT_tmpa\endcsname ##1\relax !\XINT_tmpa##2}%
1930     {##2}##1\relax !\XINT_tmpa{##2}}%
1931 \expandafter\XINT_global
1932 \expanded
1933 {\edef\expandafter\noexpand
1934     \csname XINT_expr_onliteral_\XINT_tmpa\endcsname ##1\relax !\XINT_tmpa##2}%
1935     {\XINT_expr_precedence_*** *##2}(##1\relax !\XINT_tmpa{##2}}%
1936 }%
1937 \xintApplyUnbraced \XINT_expr_makedummy {abcdefghijklmnopqrstuvwxyz}%
1938 \xintApplyUnbraced \XINT_expr_makedummy {ABCDEFGHIJKLMNOPQRSTUVWXYZ}%
1939 \def\xintnewdummy #1{%
1940     \XINT_expr_makedummy{#1}%
1941     \ifxintverbose\xintMessage {xintexpr}{Info}%
1942     {\XINT_tmpa\space now
1943     \ifxintglobaldefs globally \fi usable as dummy variable.}%
1944     \fi
1945 }%
1946 % \begin{macrocode}
1947 % Je ne définis pas de onliteral for them (it only serves for allowing
1948 % tacit multiplication if variable name is in front of an opening
1949 % parenthesis).
1950 %
1951 % The |nil| variable was need in |xint < 1.4| (with some other meaning)
1952 % in places the syntax could not allow emptiness, such as |,,|, and
1953 % other things, but at |1.4| meaning as changed.
1954 %
1955 % The other variables are new with |1.4|.
1956 % Don't use the |None|, it is tentative, and may be input as |[ ]|.
1957 % \begin{macrocode}
1958 \def\XINT_expr_var_nil{ }%
1959 \def\XINT_expr_var_None{ }% ? tentative
1960 \def\XINT_expr_var_false{0}% Maple, TeX
1961 \def\XINT_expr_var_true{1}%
1962 \def\XINT_expr_var_False{0}% Python
1963 \def\XINT_expr_var_True{1}%
1964 \catcode`* 12

```

11.24.2 \xintensuredummy, \xintrestorevariable

1.3e \xintensuredummy differs from \xintnewdummy only in the informational message... Attention that this is not meant to be nested.

1.4 fixes that the message mentioned non-existent \xintrestoredummy (real name was \xintrestorelettervar and renames the latter to \xintrestorevariable as it applies also to multi-letter names.

```

1965 \def\xintensuredummy #1{%
1966     \XINT_expr_makedummy{#1}%
1967     \ifxintverbose\xintMessage {xintexpr}{Info}%
1968     {\XINT_tmpa\space now
1969     \ifxintglobaldefs globally \fi usable as dummy variable.&&}
1970     Issue \string\xintrestorevariable{\XINT_tmpa} to restore former meaning.}%
1971     \fi

```

```

1972 }%
1973 \def\xintrestorevariablesilently #1{%
1974   \edef\XINT_tmpa{\xint_zapspace #1 \xint_gobble_i}%
1975   \ifcsname XINT_expr_var_\XINT_tmpa/old\endcsname
1976     \XINT_global
1977     \expandafter\let\csname XINT_expr_var_\XINT_tmpa\expandafter\endcsname
1978       \csname XINT_expr_var_\XINT_tmpa/old\expandafter\endcsname
1979   \fi
1980   \ifcsname XINT_expr_onliteral_\XINT_tmpa/old\endcsname
1981     \XINT_global
1982     \expandafter\let\csname XINT_expr_onliteral_\XINT_tmpa\expandafter\endcsname
1983       \csname XINT_expr_onliteral_\XINT_tmpa/old\expandafter\endcsname
1984   \fi
1985 }%
1986 \def\xintrestorevariable #1{%
1987   \xintrestorevariablesilently {#1}%
1988   \ifxintverbose\xintMessage {xintexpr}{Info}%
1989     {\XINT_tmpa\space
1990     \ifxintglobaldefs globally \fi restored to its earlier status, if any.}%
1991   \fi
1992 }%

```

11.24.3 Checking (without expansion) that a symbolic expression contains correctly nested parentheses

Expands to `\xint_c_mone` in case a closing `)` had no opening `(` matching it, to `\@ne` if opening `(` had no closing `)` matching it, to `\z@` if expression was balanced. Call it as:

```
\XINT_isbalanced_a \relax #1(\xint_bye)\xint_bye
```

This is legacy f-expandable code not using `\expanded` even at 1.4.

```

1993 \def\XINT_isbalanced_a #1({\XINT_isbalanced_b #1)\xint_bye }%
1994 \def\XINT_isbalanced_b #1)#2%
1995   {\xint_bye #2\XINT_isbalanced_c\xint_bye\XINT_isbalanced_error }%
      if #2 is not \xint_bye, a ) was found, but there was no (. Hence error -> -1
1996 \def\XINT_isbalanced_error #1)\xint_bye {\xint_c_mone}%
      #2 was \xint_bye, was there a ) in original #1?
1997 \def\XINT_isbalanced_c\xint_bye\XINT_isbalanced_error #1%
1998   {\xint_bye #1\XINT_isbalanced_yes\xint_bye\XINT_isbalanced_d #1}%
      #1 is \xint_bye, there was never ( nor ) in original #1, hence OK.
1999 \def\XINT_isbalanced_yes\xint_bye\XINT_isbalanced_d\xint_bye )\xint_bye {\xint_c_ }%
      #1 is not \xint_bye, there was indeed a ( in original #1. We check if we see a ). If we do, we then
      loop until no ( nor ) is to be found.
2000 \def\XINT_isbalanced_d #1)#2%
2001   {\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.
2002 \def\XINT_isbalanced_no\xint_bye #1\xint_bye\xint_bye {\xint_c_i }%

```

11.24.4 Fetching balanced expressions E1, E2 and a variable name Name from E1, Name=E2)

Multi-letter dummy variables added at 1.4.

```

2003 \def\XINT_expr_fetch_E_comma_V_equal_E_a #1#2,%
2004 {%
2005     \ifcase\XINT_isbalanced_a \relax #1#2(\xint_bye)\xint_bye
2006         \expandafter\XINT_expr_fetch_E_comma_V_equal_E_c
2007         \or\expandafter\XINT_expr_fetch_E_comma_V_equal_E_b
2008         \else\expandafter\xintError:noopening
2009     \fi #1#2},%
2010 }%
2011 \def\XINT_expr_fetch_E_comma_V_equal_E_b #1,%
2012     {\XINT_expr_fetch_E_comma_V_equal_E_a {#1},}%
2013 \def\XINT_expr_fetch_E_comma_V_equal_E_c #1,#2#3=%
2014 {%
2015     \expandafter\XINT_expr_fetch_E_comma_V_equal_E_d\expandafter
2016     {\expanded{{\xint_zapspaces #2#3 \xint_gobble_i}}{#1}}{}}%
2017 }%
2018 \def\XINT_expr_fetch_E_comma_V_equal_E_d #1#2#3)%
2019 {%
2020     \ifcase\XINT_isbalanced_a \relax #2#3(\xint_bye)\xint_bye
2021         \or\expandafter\XINT_expr_fetch_E_comma_V_equal_E_e
2022         \else\expandafter\xintError:noopening
2023     \fi
2024     {#1}{#2#3}%
2025 }%
2026 \def\XINT_expr_fetch_E_comma_V_equal_E_e #1#2{\XINT_expr_fetch_E_comma_V_equal_E_d {#1}{#2}}}%

```

11.24.5 Fetching a balanced expression delimited by a semi-colon

1.4. For `subsn()` leaner syntax of nested substitutions.

Will also serve to `\xintdeffunc`, to not have to hide inner semi-colons in for example an `iter()` from `\xintdeffunc`.

Adding brace removal protection for no serious reason, anyhow the `xintexpr` parsers always removes braces when moving forward, but well.

Trigger by `\romannumeral\XINT_expr_fetch_to_semicolon` upfront.

```

2027 \def\XINT_expr_fetch_to_semicolon {\XINT_expr_fetch_to_semicolon_a {}\empty}%
2028 \def\XINT_expr_fetch_to_semicolon_a #1#2;%
2029 {%
2030     \ifcase\XINT_isbalanced_a \relax #1#2(\xint_bye)\xint_bye
2031         \xint_dothis{\expandafter\XINT_expr_fetch_to_semicolon_c}%
2032         \or\xint_dothis{\expandafter\XINT_expr_fetch_to_semicolon_b}%
2033         \else\expandafter\xintError:noopening
2034     \fi\xint_orthat{} \expandafter{#2}{#1}%
2035 }%
2036 \def\XINT_expr_fetch_to_semicolon_b #1#2{\XINT_expr_fetch_to_semicolon_a {#2#1};\empty}%
2037 \def\XINT_expr_fetch_to_semicolon_c #1#2{\xint_c_{#2#1}}}%

```

11.24.6 Low-level support for omit and abort keywords, the `break()` function, the `n++` construct and the semi-colon as used in the syntax of `seq()`, `add()`, `mul()`, `iter()`, `rseq()`, `iterr()`, `rrseq()`, `subsm()`, `subsn()`, `ndseq()`, `ndmap()`

There is some clever play simply based on setting suitable precedence levels combined with special meanings given to op macros.

The special !? internal operator is a helper for omit and abort keywords in list generators.

Prior to 1.4 support for +[, *[, ...,]+,]*, had some elements here.

The n++ construct 1.1 2014/10/29 did `\expandafter\.=+\xintiCeil` which transformed it into `\romannumeral0\xinticeil`, which seems a bit weird. This exploited the fact that dummy variables macros could back then pick braced material (which in the case at hand here ended being `{\romannumeral0\xinticeil...}`) and were submitted to two expansions. The result of this was to provide a not value which got expanded only in the first loop of the `:_A` and following macros of `seq`, `iter`, `rseq`, etc...

Anyhow with 1.2c I have changed the implementation of dummy variables which now need to fetch a single locked token, which they do not expand.

The `\xintiCeil` appears a bit dispendious, but I need the starting value in a `\numexpr` compatible form in the iteration loops.

```
2038 \expandafter\def\csname XINT_expr_itself_++\endcsname {++}%
2039 \expandafter\def\csname XINT_expr_itself_++\endcsname {++})%
2040 \expandafter\let\csname XINT_expr_precedence_++\endcsname \xint_c_i
2041 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
2042     \expandafter\def\csname XINT_#1_op_++\endcsname ##1##2\relax
2043     {\expandafter\XINT_expr_foundend
2044         \expanded{{+\{\XINT:NEhook:f:one:from:one:direct\xintiCeil##1}}}%
2045     }%
2046 }%
```

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.

```
2047 \catcode`? 3
2048 \def\XINT_expr_func_break #1#2#3{#1#2{?#3}}%
2049 \catcode`? 11
2050 \let\XINT_flexpr_func_break \XINT_expr_func_break
2051 \let\XINT_iiexpr_func_break \XINT_expr_func_break
```

The omit and abort keywords Comments are currently undergoing reconstruction.

```
2052 \edef\XINT_expr_var_omit #1\relax !{1\string !?!\relax !}%
2053 \edef\XINT_expr_var_abort #1\relax !{1\string !?^\relax !}%
2054 \def\XINT_expr_itself_!? {!?!}%
2055 \def\XINT_expr_op_!? #1#2\relax{\XINT_expr_foundend{#2}}%
2056 \let\XINT_iiexpr_op_!? \XINT_expr_op_!?
2057 \let\XINT_flexpr_op_!? \XINT_expr_op_!?
2058 \let\XINT_expr_precedence_!? \xint_c_iv
```

The semi-colon Obsolete comments undergoing re-construction

```
2059 \xintFor #1 in {expr,flexpr,iiexpr} \do {%
2060     \expandafter\def\csname XINT_#1_op_;\endcsname {\xint_c_i ;}%
2061 }%
2062 \expandafter\let\csname XINT_expr_precedence_;\endcsname\xint_c_i
2063 \expandafter\def\csname XINT_expr_itself_;\endcsname {})%
2064 \expandafter\let\csname XINT_expr_precedence_;\endcsname\xint_c_i
```

11.24.7 Reserved dummy variables @, @1, @2, @3, @4, @@, @@(1), ..., @@@, @@@(1), ... for recursions

Comments currently under reconstruction.

1.4 breaking change: @ and @1 behave differently and one can not use @ in place of @1 in iterr() and rrseq(). Formerly @ and @1 had the same definition.

Brace stripping in \XINT_expr_func_@@ is prevented by some ending 0 or other token see iterr() and rrseq() code.

For the record, the ~ and ? have catcode 3 in this code.

```

2065 \catcode`* 11
2066 \def\XINT_expr_var_@ #1~#2{{#2}#1~{#2}}%
2067 \def\XINT_expr_onliteral_@ #1~#2{\XINT_expr_precedence_*** *{#2}(#1~{#2})}%
2068 \expandafter
2069 \def\csname XINT_expr_var_@1\endcsname #1~#2{{#2}#1~{#2}}%
2070 \expandafter
2071 \def\csname XINT_expr_var_@2\endcsname #1~#2#3{{#3}#1~{#2}{#3}}%
2072 \expandafter
2073 \def\csname XINT_expr_var_@3\endcsname #1~#2#3#4{{#4}#1~{#2}{#3}{#4}}%
2074 \expandafter
2075 \def\csname XINT_expr_var_@4\endcsname #1~#2#3#4#5{{#5}#1~{#2}{#3}{#4}{#5}}%
2076 \expandafter\def\csname XINT_expr_onliteral_@1\endcsname #1~#2%
2077     {\XINT_expr_precedence_*** *{#2}(#1~{#2})}%
2078 \expandafter\def\csname XINT_expr_onliteral_@2\endcsname #1~#2#3%
2079     {\XINT_expr_precedence_*** *{#3}(#1~{#2}{#3})}%
2080 \expandafter\def\csname XINT_expr_onliteral_@3\endcsname #1~#2#3#4%
2081     {\XINT_expr_precedence_*** *{#4}(#1~{#2}{#3}{#4})}%
2082 \expandafter\def\csname XINT_expr_onliteral_@4\endcsname #1~#2#3#4#5%
2083     {\XINT_expr_precedence_*** *{#5}(#1~{#2}{#3}{#4}{#5})}%
2084 \catcode`* 12
2085 \catcode`? 3
2086 \def\XINT_expr_func_@@ #1#2#3#4~#5?%
2087 {%
2088     \expandafter#1\expandafter#2\expandafter{\expandafter{%
2089         \romannumeral0\xintntheltnoexpand{\xintNum#3}{#5}}#4~#5?%
2090 }}%
2091 \def\XINT_expr_func_@@@ #1#2#3#4~#5~#6?%
2092 {%
2093     \expandafter#1\expandafter#2\expandafter{\expandafter{%
2094         \romannumeral0\xintntheltnoexpand{\xintNum#3}{#6}}#4~#5~#6?%
2095 }}%
2096 \def\XINT_expr_func_@@@@ #1#2#3#4~#5~#6~#7?%
2097 {%
2098     \expandafter#1\expandafter#2\expandafter{\expandafter{%
2099         \romannumeral0\xintntheltnoexpand{\xintNum#3}{#7}}#4~#5~#6~#7?%
2100 }}%
2101 \let\XINT_flexpr_func_@@\XINT_expr_func_@@
2102 \let\XINT_flexpr_func_@@@\XINT_expr_func_@@@
2103 \let\XINT_flexpr_func_@@@@\XINT_expr_func_@@@@
2104 \def\XINT_iiexpr_func_@@ #1#2#3#4~#5?%
2105 {%
2106     \expandafter#1\expandafter#2\expandafter{\expandafter{%
2107         \romannumeral0\xintntheltnoexpand{\xint_firstofone#3}{#5}}#4~#5?%

```



```

2108 }%
2109 \def\XINT_iiexpr_func_@@@ #1#2#3#4~#5~#6?%
2110 {%
2111   \expandafter#1\expandafter#2\expandafter{\expandafter{%
2112     \romannumeral0\xintntheltnoexpand{\xint_firstofone#3}{#6}}#4~#5~#6?%
2113 }%
2114 \def\XINT_iiexpr_func_@@@ #1#2#3#4~#5~#6~#7?%
2115 {%
2116   \expandafter#1\expandafter#2\expandafter{\expandafter{%
2117     \romannumeral0\xintntheltnoexpand{\xint_firstofone#3}{#7}}#4~#5~#6~#7?%
2118 }%
2119 \catcode`? 11

```

11.25 Pseudo-functions involving dummy variables and generating scalars or sequences

11.25.1	Comments	369
11.25.2	subs(): substitution of one variable	371
11.25.3	subsm(): simultaneous independent substitutions	371
11.25.4	subsn(): leaner syntax for nesting (possibly dependent) substitutions	372
11.25.5	seq(): sequences from assigning values to a dummy variable	374
11.25.6	iter()	374
11.25.7	add(), mul()	376
11.25.8	rseq()	377
11.25.9	iterr()	378
11.25.10	rrseq()	379

11.25.1 Comments

Comments added 2020/01/16.

The mechanism for «seq» is the following. When the parser encounters «seq», which means it parsed these letters and encountered (from expansion) an opening parenthesis, the `\XINT_expr_func` mechanism triggers the «`» operator which realizes that «seq» is a pseudo-function (there is no `_func_seq`) and thus spans the `\XINT_expr_onliteral_seq` macro (currently this means however that the knowledge of which parser we are in is lost, see comments of `\XINT_expr_op_`` code). The latter will use delimited macros and parenthesis check to fetch (without any expansion), the symbolic expression `ExprSeq` to evaluate, the `Name` (now possibly multi-letter) of the variable and the expression `ExprValues` to evaluate which will give the values to assign to the dummy variable `Name`. It then positions upstream `ExprValues` suitably terminated (see next) and after it `{{Name}{ExprSeq}}`. Then it inserts a second call to the «`» operator with now «seqx» as argument hence the appropriate «`{,fl,ii}expr_func_seqx`» macros gets executed. The general way function macros work is that first all their arguments are evaluated via a call not to `\xintbare{,float,ii}eval` but to the suitable `\XINT_{expr,flexpr,iiexpr}_oparen` core macro which does almost same excepts it expects a final closing parenthesis (of course allowing nested parenthesis in-between) and stops there. Here, this closing parenthesis got positioned deliberately with a `\relax` after it, so the parser, which always after having gathered a value looks ahead to find the next operator, thinks it has hit the end of the expression and as result inserts a `\xint_c_` (i.e. `\z@`) token for precedence level and a dummy `\relax` token (place-holder for a non-existing operator). Generally speaking «`func_foo`» macros expect to be executed with three parameters `#1#2#3`, `#1` = precedence, `#2` = operator, `#3` = values (call it «`args`») i.e. the fully evaluated list of all its arguments. The special «`func_seqx`» and cousins know that the first two tokens are trash and they now proceed forward, having thus lying before them upstream the values to loop over, now fully evaluated, and `{{Name}{ExprSeq}}`.

It then positions appropriately ExprSeq inside a sub-expression and after it, following suitable delimiter, Name and the evaluated values to assign to Name.

Dummy variables are essentially simply delimited macros where the delimiter is the variable name preceded by a \relax token and a catcode 11 exclamation point. Thus the various «subsx», «seq», «iterx» position the tokens appropriately and launch suitable loops.

All of this nests well, inner «seq»'s (or more often in practice «subs»'s) being allowed to refer to the dummy variables used by outer «seq»'s because the outer «seq»'s have the values to assign to their variables evaluated first and their ExprSeq evaluated last. For inner dummy variables to be able to refer to outer dummy variables the author must be careful of course to not use in the implementation braces { and } which would break dummy variables to fetch values beyond the closing brace.

The above «seq» mechanism was done around June 15-25th 2014 at the time of the transition from 1.09n to 1.1 but already in October 2014 I made a note that I had a hard time to understand it again:

« [START OF YEAR 2014 COMMENTS]

All of seq, add, mul, rseq, etc... (actually all of the extensive changes from xintexpr 1.09n to 1.1) was done around June 15-25th 2014, but the problem is that I did not document the code enough, and I had a hard time understanding in October what I had done in June. Despite the lesson, again being short on time, I do not document enough my current understanding of the innards of the beast...

I added subs, and iter in October (also the [:n], [n:] list extractors), proving I did at least understand a bit (or rather could imitate) my earlier code (but don't ask me to explain \xintNewExpr !)

The \XINT_expr_fetch_E_comma_V_equal_E_a parses: "expression, variable=list)" (when it is called the opening (has been swallowed, and it looks for the ending one.) Both expression and list may themselves contain parentheses and commas, we allow nesting. For example "x^2,x=1..10)", at the end of seq_a we have {variable{expression}}{list}, in this example {x{x^2}}{1..10}, or more complicated "seq(add(y,y=1..x),x=1..10)" will work too. The variable is a single lowercase Latin letter.

The complications with \xint_c_ii^v in seq_f is for the recurrent thing that we don't know in what type of expressions we are, hence we must move back up, with some loss of efficiency (superfluous check for minus sign, etc...). But the code manages simultaneously expr, flexpr and iiexpr.

[END OF YEAR 2014 OLD COMMENTS]»

On Jeudi 16 janvier 2020 à 15:13:32 I finally did the documentation as above.

The case of «iter», «rseq», «iterr», «rrseq» differs slightly because the initial values need evaluation. This is done by genuine functions \XINT_<parser>_func_iter etc... (there was no \XINT_<parser>_func_seq). The trick is via the semi-colon ; which is a genuine operator having the precedence of a closing parenthesis and whose action is only to stop expansion. Thus this first step of gathering the initial values is done as part of the regular expansion job of the parser not using delimited macros and the ; can be hidden in braces {;} because the three parsers when moving forward remove one level of braces always. Thus \XINT_<parser>_func_seq simply hand over to \XINT_allexpr_iter which will then trigger the fetching without expansion of ExprIter, Name=ExprValues as described previously for «seq».

With 1.4, multi-letter names for dummy variables are allowed.

Also there is the additional 1.4 ambition to make the whole thing parsable by \xintNewExpr/\xintdeffunc. This is done by checking if all is numerical, because the omit, abort and break() mechanisms have no translation into macros, and the only solution for symbolic material is to simply keep it as is, so that expansion will again activate the xintexpr parsers. At 1.4 this approach is fine although the initial goals of \xintNewExpr/\xintdeffunc was to completely replace the parsers (whose storage method hit the string pool formerly) by macros. Now that 1.4 does not impact the string pool we can make \xintdeffunc much more powerful but it will not be a construct using only xintfrac macros, it will still be partially the \xintexpr etc... parsers in such cases.

Got simpler with 1.2c as now the dummy variable fetches an already encapsulated value, which is anyhow the form in which we get it.

Refactored at 1.4 using `\expanded` rather than `\csname`.

And support for multi-letter variables, which means function declarations can now use multi-letter variables !

11.25.2 `subs()`: substitution of one variable

```
2120 \def\XINT_expr_onliteral_subs
2121 {%
2122   \expandafter\XINT_allexpr_subs_f
2123   \romannumeral`&&\XINT_expr_fetch_E_comma_V_equal_E_a {}%
2124 }%
2125 \def\XINT_allexpr_subs_f #1#2{\xint_c_ii^v `{subsx}#2)\relax #1}%
2126 \def\XINT_expr_func_subsx #1#2{\XINT_allexpr_subsx \xintbareeval }%
2127 \def\XINT_flexpr_func_subsx #1#2{\XINT_allexpr_subsx \xintbarefloateval}%
2128 \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

```
2129 \def\XINT_allexpr_subsx #1#2#3#4%
2130 {%
2131   \expandafter\XINT_expr_put_op_first
2132   \expanded
2133   \bgroup\romannumeral0#1#4\relax \iffalse\relax !#3{#2}{\fi
2134   \expandafter}\romannumeral`&&\XINT_expr_getop
2135 }%
```

11.25.3 `subsm()`: simultaneous independent substitutions

New with 1.4. Globally the `var1=expr1; var2=expr2; var2=expr3;...` part can arise from expansion, except that once a semi-colon has been found (from expansion) the `varK=` thing following it must be there. And as for `subs()` the final parenthesis must be there from the start.

```
2136 \def\XINT_expr_onliteral_subsm
2137 {%
2138   \expandafter\XINT_allexpr_subsm_f
2139   \romannumeral`&&\XINT_expr_fetch_E_comma_V_equal_E_a {}%
2140 }%
2141 \def\XINT_allexpr_subsm_f #1#2{\xint_c_ii^v `{subsmx}#2)\relax #1}%
2142 \def\XINT_expr_func_subsmx
2143 {%
2144   \expandafter\XINT_allexpr_subsmx\expandafter\xintbareeval
2145   \expanded\bgroup{\iffalse}\fi\XINT_allexpr_subsm_A\XINT_expr_oparen
2146 }%
2147 \def\XINT_flexpr_func_subsmx
2148 {%
2149   \expandafter\XINT_allexpr_subsmx\expandafter\xintbarefloateval
2150   \expanded\bgroup{\iffalse}\fi\XINT_allexpr_subsm_A\XINT_flexpr_oparen
2151 }%
2152 \def\XINT_iiexpr_func_subsmx
2153 {%
2154   \expandafter\XINT_allexpr_subsmx\expandafter\xintbareiieval
```

```

2155 \expanded\bgroup{\iffalse}\fi\XINT_allexpr_subsm_A\XINT_iiexpr_oparen
2156 }%
2157 \def\XINT_allexpr_subsm_A #1#2#3%
2158 {%
2159 \ifx#2\xint_c_
2160 \expandafter\XINT_allexpr_subsm_done
2161 \else
2162 \expandafter\XINT_allexpr_subsm_B
2163 \fi #1%
2164 }%
2165 \def\XINT_allexpr_subsm_B #1#2#3#4=%
2166 {%
2167 {#2}\relax !\xint_zapspaces#3#4 \xint_gobble_i
2168 \expandafter\XINT_allexpr_subsm_A\expandafter#1\romannumeral`&&@#1%
2169 }%

```

#1 = \xintbareeval, or \xintbarefloateval or \xintbareiieval #2 = evaluation of last variable assignment

```

2170 \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

```

2171 \def\XINT_allexpr_subsmx #1#2#3#4%
2172 {%
2173 \expandafter\XINT_expr_put_op_first
2174 \expanded
2175 \bgroup\romannumeral0#1#4\relax \iffalse\relax !#3#2{\fi
2176 \expandafter}\romannumeral`&&@\XINT_expr_getop
2177 }%

```

11.25.4 subsn(): leaner syntax for nesting (possibly dependent) substitutions

New with 1.4. 2020/01/24

```

2178 \def\XINT_expr_onliteral_subsn
2179 {%
2180 \expandafter\XINT_allexpr_subsn_f
2181 \romannumeral`&&@\XINT_expr_fetch_E_comma_V_equal_E_a }%
2182 }%
2183 \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

```

2184 \def\XINT_allexpr_subsn_g #1#2#3%
2185 {%
2186 \expandafter\XINT_allexpr_subsn_h
2187 \expanded\bgroup{\iffalse}\fi\expandafter\XINT_allexpr_subsn_B
2188 \romannumeral\XINT_expr_fetch_to_semicolon #1=#3;\hbox=;;^{#2}%
2189 }%
2190 \def\XINT_allexpr_subsn_B #1{\XINT_allexpr_subsn_C #1\vbox}%

```

```

2191 \def\XINT_allexpr_subsn_C #1#2=#3\vbox
2192 {%
2193   \ifx\hbox#1\iffalse{\fi}\expandafter\else
2194   {\xint_zapspaces #1#2 \xint_gobble_i};\unexpanded{{{#3}}}%
2195   \expandafter\XINT_allexpr_subsn_B
2196   \romannumeral\expandafter\XINT_expr_fetch_to_semicolon\fi
2197 }%
2198 \def\XINT_allexpr_subsn_h
2199 {%
2200   \xint_c_ii^v `{subsnx}\romannumeral0\xintreverseorder
2201 }%
2202 \def\XINT_expr_func_subsnx #1#2#3#4#5;#6%
2203 {%
2204   \xint_gob_til_ ^ #6\XINT_allexpr_subsnx_H ^%
2205   \expandafter\XINT_allexpr_subsnx\expandafter
2206   \xintbareeval\romannumeral0\xintbareeval #5\relax !#4{#3}\xintundefined
2207   {\relax !#4{#3}\relax !#6}%
2208 }%
2209 \def\XINT_iiexpr_func_subsnx #1#2#3#4#5;#6%
2210 {%
2211   \xint_gob_til_ ^ #6\XINT_allexpr_subsnx_H ^%
2212   \expandafter\XINT_allexpr_subsnx\expandafter
2213   \xintbareieeval\romannumeral0\xintbareieeval #5\relax !#4{#3}\xintundefined
2214   {\relax !#4{#3}\relax !#6}%
2215 }%
2216 \def\XINT_flexpr_func_subsnx #1#2#3#4#5;#6%
2217 {%
2218   \xint_gob_til_ ^ #6\XINT_allexpr_subsnx_H ^%
2219   \expandafter\XINT_allexpr_subsnx\expandafter
2220   \xintbarefloateval\romannumeral0\xintbarefloateval #5\relax !#4{#3}\xintundefined
2221   {\relax !#4{#3}\relax !#6}%
2222 }%
2223 \def\XINT_allexpr_subsnx #1#2!#3\xintundefined#4#5;#6%
2224 {%
2225   \xint_gob_til_ ^ #6\XINT_allexpr_subsnx_I ^%
2226   \expandafter\XINT_allexpr_subsnx\expandafter
2227   #1\romannumeral0#1#5\relax !#4{#2}\xintundefined
2228   {\relax !#4{#2}\relax !#6}%
2229 }%
2230 \def\XINT_allexpr_subsnx_H ^#1\romannumeral0#2#3!#4\xintundefined #5#6%
2231 {%
2232   \expandafter\XINT_allexpr_subsnx_J\romannumeral0#2#6#5%
2233 }%
2234 \def\XINT_allexpr_subsnx_I ^#1\romannumeral0#2#3\xintundefined #4#5%
2235 {%
2236   \expandafter\XINT_allexpr_subsnx_J\romannumeral0#2#5#4%
2237 }%
2238 \def\XINT_allexpr_subsnx_J #1#2^%
2239 {%
2240   \expandafter\XINT_expr_put_op_first
2241   \expanded{\unexpanded{#1}}\expandafter\romannumeral`&&\XINT_expr_getop
2242 }%

```

11.25.5 seq(): sequences from assigning values to a dummy variable

In `seq_f`, the `#2` is the `ExprValues` expression which needs evaluation to provide the values to the dummy variable and `#1` is `{Name}{ExprSeq}` where `Name` is the name of dummy variable and `{ExprSeq}` the expression which will have to be evaluated.

```
2243 \def\XINT_allexpr_seq_f #1#2{\xint_c_ii^v `{seqx}#2)\relax #1}%
2244 \def\XINT_expr_onliteral_seq
2245 {\expandafter\XINT_allexpr_seq_f\romannumeral`&&\XINT_expr_fetch_E_comma_V_equal_E_a {}}%
2246 \def\XINT_expr_func_seqx #1#2{\XINT:NEhook:seqx\XINT_allexpr_seqx\xintbareeval }%
2247 \def\XINT_flexpr_func_seqx #1#2{\XINT:NEhook:seqx\XINT_allexpr_seqx\xintbarefloateval}%
2248 \def\XINT_iexpr_func_seqx #1#2{\XINT:NEhook:seqx\XINT_allexpr_seqx\xintbareiieval }%
2249 \def\XINT_allexpr_seqx #1#2#3#4%
2250 {%
2251     \expandafter\XINT_expr_put_op_first
2252     \expanded \bgroup {\iffalse}\fi\XINT_expr_seq:_b {#1#4}\relax !#3}#2^%
2253     \XINT_expr_cb_and_getop
2254 }%
2255 \def\XINT_expr_cb_and_getop{\iffalse{\fi\expandafter}\romannumeral`&&\XINT_expr_getop}%
```

Comments undergoing reconstruction.

```
2256 \catcode`? 3
2257 \def\XINT_expr_seq:_b #1#2%
2258 {%
2259     \ifx +#2\xint_dothis\XINT_expr_seq:_Ca\fi
2260     \ifx !#2!\xint_dothis\XINT_expr_seq:_noop\fi
2261     \ifx ^#2\xint_dothis\XINT_expr_seq:_end\fi
2262     \xint_orthat{\XINT_expr_seq:_c}{#2}{#1}%
2263 }%
2264 \def\XINT_expr_seq:_noop #1{\XINT_expr_seq:_b }%
2265 \def\XINT_expr_seq:_end #1#2{\iffalse{\fi}}%
2266 \def\XINT_expr_seq:_c #1#2{\expandafter\XINT_expr_seq:_d\romannumeral0#2{{#1}}{#2}}%
2267 \def\XINT_expr_seq:_d #1{\ifx ^#1\xint_dothis\XINT_expr_seq:_abort\fi
2268     \ifx ?#1\xint_dothis\XINT_expr_seq:_break\fi
2269     \ifx !#1\xint_dothis\XINT_expr_seq:_omit\fi
2270     \xint_orthat{\XINT_expr_seq:_goon {#1}}}%
2271 \def\XINT_expr_seq:_abort #1!#2^{\iffalse{\fi}}%
2272 \def\XINT_expr_seq:_break #1!#2^{#1\iffalse{\fi}}%
2273 \def\XINT_expr_seq:_omit #1!#2#{\expandafter\XINT_expr_seq:_b\xint_gobble_i}%
2274 \def\XINT_expr_seq:_goon #1!#2#{#1\expandafter\XINT_expr_seq:_b\xint_gobble_i}%
2275 \def\XINT_expr_seq:_Ca #1#2#3{\XINT_expr_seq:_Cc#3.#2}%
2276 \def\XINT_expr_seq:_Cb #1{\expandafter\XINT_expr_seq:_Cc\the\numexpr#1+\xint_c_i.}%
2277 \def\XINT_expr_seq:_Cc #1.#2{\expandafter\XINT_expr_seq:_D\romannumeral0#2{{#1}}{#1}{#2}}%
2278 \def\XINT_expr_seq:_D #1{\ifx ^#1\xint_dothis\XINT_expr_seq:_abort\fi
2279     \ifx ?#1\xint_dothis\XINT_expr_seq:_break\fi
2280     \ifx !#1\xint_dothis\XINT_expr_seq:_Omit\fi
2281     \xint_orthat{\XINT_expr_seq:_Goon {#1}}}%
2282 \def\XINT_expr_seq:_Omit #1!#2#{\expandafter\XINT_expr_seq:_Cb\xint_gobble_i}%
2283 \def\XINT_expr_seq:_Goon #1!#2#{#1\expandafter\XINT_expr_seq:_Cb\xint_gobble_i}%
```

11.25.6 iter()

Prior to 1.2g, the iter keyword was what is now called iterr, analogous with rrseq. Somehow I forgot an iter functioning like rseq with the sole difference of printing only the last iteration. Both rseq and iter work well with list selectors, as @ refers to the whole comma separated sequence of the initial values. I have thus deliberately done the backwards incompatible renaming of iter to iterr, and the new iter.

To understand the tokens which are presented to \XINT_allexpr_iter it is needed to check elsewhere in the source code how the ; hack is done.

The #2 in \XINT_allexpr_iter is \xint_c_i from the ; hack. Formerly (xint < 1.4) there was no such token. The change is motivated to using ; also in subsm() syntax.

```

2284 \def\XINT_expr_func_iter  {\XINT_allexpr_iter \xintbareeval      }%
2285 \def\XINT_flexpr_func_iter {\XINT_allexpr_iter \xintbarefloateval }%
2286 \def\XINT_iiexpr_func_iter {\XINT_allexpr_iter \xintbareiieval   }%
2287 \def\XINT_allexpr_iter #1#2#3#4%
2288 {%
2289   \expandafter\XINT_expr_iterx
2290   \expandafter#1\expanded{\unexpanded{#4}}\expandafter}%
2291   \romannumeral`&&\XINT_expr_fetch_E_comma_V_equal_E_a }%
2292 }%
2293 \def\XINT_expr_iterx #1#2#3#4%
2294 {%
2295   \XINT:NEhook:iter\XINT_expr_itery\romannumeral0#1(#4)\relax {#2}#3#1%
2296 }%
2297 \def\XINT_expr_itery #1#2#3#4#5%
2298 {%
2299   \expandafter\XINT_expr_put_op_first
2300   \expanded \bgroup {\iffalse}\fi
2301   \XINT_expr_iter:_b {#5#4\relax !#3}#1^~{#2}\XINT_expr_cb_and_getop
2302 }%
2303 \def\XINT_expr_iter:_b #1#2%
2304 {%
2305   \ifx +#2\xint_dothis\XINT_expr_iter:_Ca\fi
2306   \ifx !#2!\xint_dothis\XINT_expr_iter:_noop\fi
2307   \ifx ^#2\xint_dothis\XINT_expr_iter:_end\fi
2308   \xint_orthat{\XINT_expr_iter:_c}{#2}{#1}%
2309 }%
2310 \def\XINT_expr_iter:_noop #1{\XINT_expr_iter:_b }%
2311 \def\XINT_expr_iter:_end #1#2~#3{#3\iffalse{\fi}}%
2312 \def\XINT_expr_iter:_c #1#2{\expandafter\XINT_expr_iter:_d\romannumeral0#2{{#1}}{#2}}%
2313 \def\XINT_expr_iter:_d #1{\ifx ^#1\xint_dothis\XINT_expr_iter:_abort\fi
2314   \ifx ?#1\xint_dothis\XINT_expr_iter:_break\fi
2315   \ifx !#1\xint_dothis\XINT_expr_iter:_omit\fi
2316   \xint_orthat{\XINT_expr_iter:_goon {#1}}}%
2317 \def\XINT_expr_iter:_abort #1!#2^~#3{#3\iffalse{\fi}}%
2318 \def\XINT_expr_iter:_break #1!#2^~#3{#1\iffalse{\fi}}%
2319 \def\XINT_expr_iter:_omit #1!#2#{\expandafter\XINT_expr_iter:_b\xint_gobble_i}%
2320 \def\XINT_expr_iter:_goon #1!#2#{\XINT_expr_iter:_goon_a {#1}}%
2321 \def\XINT_expr_iter:_goon_a #1#2#3~#4{\XINT_expr_iter:_b #3~{#1}}%
2322 \def\XINT_expr_iter:_Ca #1#2#3{\XINT_expr_iter:_Cc#3.{#2}}%
2323 \def\XINT_expr_iter:_Cb #1{\expandafter\XINT_expr_iter:_Cc\the\numexpr#1+\xint_c_i.}%
2324 \def\XINT_expr_iter:_Cc #1.#2{\expandafter\XINT_expr_iter:_D\romannumeral0#2{{#1}}{#1}{#2}}%
2325 \def\XINT_expr_iter:_D #1{\ifx ^#1\xint_dothis\XINT_expr_iter:_abort\fi
2326   \ifx ?#1\xint_dothis\XINT_expr_iter:_break\fi

```

```

2327             \ifx !#1\xint_dothis\XINT_expr_iter:_Omit\fi
2328             \xint_orthat{\XINT_expr_iter:_Goon {#1}}}%
2329 \def\XINT_expr_iter:_Omit #1!#2#\expandafter\XINT_expr_iter:_Cb\xint_gobble_i}%
2330 \def\XINT_expr_iter:_Goon #1!#2#\XINT_expr_iter:_Goon_a {#1}}%
2331 \def\XINT_expr_iter:_Goon_a #1!#2#3~#4{\XINT_expr_iter:_Cb #3~{#1}}%

```

11.25.7 add(), mul()

Comments under reconstruction.

These were a bit anomalous as they did not implement omit and abort keyword and the break() function (and per force then neither the n++ syntax).

At 1.4 they are simply mapped to using adequately iter(). Thus, there is small loss in efficiency, but supporting omit, abort and break is important. Using dedicated macros here would have caused also slight efficiency drop. Simpler to remove the old approach.

```

2332 \def\XINT_expr_onliteral_add
2333 {\expandafter\XINT_allexpr_add_f\romannumeral`&&\XINT_expr_fetch_E_comma_V_equal_E_a {}}%
2334 \def\XINT_allexpr_add_f #1#2{\xint_c_ii^v `{opx}#2)\relax #1{+}{0}}%
2335 \def\XINT_expr_onliteral_mul
2336 {\expandafter\XINT_allexpr_mul_f\romannumeral`&&\XINT_expr_fetch_E_comma_V_equal_E_a {}}%
2337 \def\XINT_allexpr_mul_f #1#2{\xint_c_ii^v `{opx}#2)\relax #1{*}{1}}%
2338 \def\XINT_expr_func_opx {\XINT:NEhook:opx \XINT_allexpr_opx \xintbareeval }%
2339 \def\XINT_flexpr_func_opx {\XINT:NEhook:opx \XINT_allexpr_opx \xintbarefloateval}%
2340 \def\XINT_iiexpr_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.

```

2341 \def\XINT_allexpr_opx #1#2#3#4#5#6#7#8%
2342 {%
2343   \expandafter\XINT_expr_put_op_first
2344   \expanded \bgroup {\iffalse}\fi
2345   \XINT_expr_iter:_b {#1%
2346   \expandafter\XINT_allexpr_opx_ifnotomitted
2347   \romannumeral0#1#6\relax#7@\relax !#5}#4^~{#8}}\XINT_expr_cb_and_getop
2348 }%
2349 \def\XINT_allexpr_opx_ifnotomitted #1%
2350 {%
2351   \ifx !#1\xint_dothis{@\relax}\fi
2352   \ifx ^#1\xint_dothis{\XINTfstop. ^\relax}\fi
2353   \if ?\xintFirstItem{#1}\xint_dothis{\XINT_allexpr_opx_break{#1}}\fi
2354   \xint_orthat{\XINTfstop.{#1}}}%
2355 }%
2356 \def\XINT_allexpr_opx_break #1#2\relax
2357 {%
2358   break(\expandafter\XINTfstop\expandafter.\expandafter{\xint_gobble_i#1}#2)\relax
2359 }%

```


11.25.8 rseq()

When `func_rseq` has its turn, initial segment has been scanned by `oparen`, the `;` mimicking the rôle of a closing parenthesis, and stopping further expansion (and leaving a `\xint_c_i` left-over token since 1.4). The `;` is discovered during standard parsing mode, it may be for example `{;}` or arise from expansion as `rseq` does not use a delimited macro to locate it.

```

2360 \def\xINT_expr_func_rseq  {\XINT_allexpr_rseq \xintbareeval      }%
2361 \def\xINT_flexpr_func_rseq {\XINT_allexpr_rseq \xintbarefloateval }%
2362 \def\xINT_iexpr_func_rseq  {\XINT_allexpr_rseq \xintbareiieval    }%
2363 \def\xINT_allexpr_rseq #1#2#3#4%
2364 {%
2365   \expandafter\xINT_expr_rseqx
2366   \expandafter #1\expanded{\unexpanded{#4}}\expandafter}%
2367   \romannumeral &&@\XINT_expr_fetch_E_comma_V_equal_E_a }%
2368 }%
2369 \def\xINT_expr_rseqx #1#2#3#4%
2370 {%
2371   \XINT:NEhook:rseq \XINT_expr_rseq\romannumeral0#1(#4)\relax {#2}#3#1%
2372 }%
2373 \def\xINT_expr_rseqy #1#2#3#4#5%
2374 {%
2375   \expandafter\xINT_expr_put_op_first
2376   \expanded \bgroup {\iffalse}\fi
2377   #2%
2378   \XINT_expr_rseq:_b {#5#4\relax !#3}#1^~{#2}\XINT_expr_cb_and_getop
2379 }%
2380 \def\xINT_expr_rseq:_b #1#2%
2381 {%
2382   \ifx +#2\xint_dothis\xINT_expr_rseq:_Ca\fi
2383   \ifx !#2!\xint_dothis\xINT_expr_rseq:_noop\fi
2384   \ifx ^#2\xint_dothis\xINT_expr_rseq:_end\fi
2385   \xint_orthat{\XINT_expr_rseq:_c}{#2}{#1}%
2386 }%
2387 \def\xINT_expr_rseq:_noop #1{\XINT_expr_rseq:_b }%
2388 \def\xINT_expr_rseq:_end #1#2~#3{\iffalse\fi}%
2389 \def\xINT_expr_rseq:_c #1#2{\expandafter\xINT_expr_rseq:_d\romannumeral0#2{{#1}}{#2}}%
2390 \def\xINT_expr_rseq:_d #1{\ifx ^#1\xint_dothis\xINT_expr_rseq:_abort\fi
2391   \ifx ?#1\xint_dothis\xINT_expr_rseq:_break\fi
2392   \ifx !#1\xint_dothis\xINT_expr_rseq:_omit\fi
2393   \xint_orthat{\XINT_expr_rseq:_goon {#1}}}%
2394 \def\xINT_expr_rseq:_abort #1!#2^~#3{\iffalse\fi}%
2395 \def\xINT_expr_rseq:_break #1!#2^~#3{#1\iffalse\fi}%
2396 \def\xINT_expr_rseq:_omit #1!#2#{\expandafter\xINT_expr_rseq:_b\xint_gobble_i}%
2397 \def\xINT_expr_rseq:_goon #1!#2#{\XINT_expr_rseq:_goon_a {#1}}%
2398 \def\xINT_expr_rseq:_goon_a #1#2#3~#4{#1\xINT_expr_rseq:_b #3~{#1}}%
2399 \def\xINT_expr_rseq:_Ca #1#2#3{\XINT_expr_rseq:_Cc#3.{#2}}%
2400 \def\xINT_expr_rseq:_Cb #1{\expandafter\xINT_expr_rseq:_Cc\the\numexpr#1+\xint_c_i.}%
2401 \def\xINT_expr_rseq:_Cc #1.#2{\expandafter\xINT_expr_rseq:_D\romannumeral0#2{{#1}}{#1}{#2}}%
2402 \def\xINT_expr_rseq:_D #1{\ifx ^#1\xint_dothis\xINT_expr_rseq:_abort\fi
2403   \ifx ?#1\xint_dothis\xINT_expr_rseq:_break\fi
2404   \ifx !#1\xint_dothis\xINT_expr_rseq:_Omit\fi
2405   \xint_orthat{\XINT_expr_rseq:_Goon {#1}}}%

```

```
2406 \def\XINT_expr_rseq:_Omit #1!#2#{\expandafter\XINT_expr_rseq:_Cb\xint_gobble_i}%
2407 \def\XINT_expr_rseq:_Goon #1!#2#\XINT_expr_rseq:_Goon_a {#1}}%
2408 \def\XINT_expr_rseq:_Goon_a #1#2#3~#4{#1\XINT_expr_rseq:_Cb #3~{#1}}%
```

11.25.9 iterr()

ATTENTION! at 1.4 the @ and @1 are not synonymous anymore. One *must* use @1 in iterr() context.

```
2409 \def\XINT_expr_func_iterr {XINT_allexpr_iterr \xintbareeval }%
2410 \def\XINT_flexpr_func_iterr {XINT_allexpr_iterr \xintbarefloateval }%
2411 \def\XINT_iexpr_func_iterr {XINT_allexpr_iterr \xintbareiieval }%
2412 \def\XINT_allexpr_iterr #1#2#3#4%
2413 {%
2414 \expandafter\XINT_expr_iterrx
2415 \expandafter #1\expanded{\xintRevWithBraces{#4}}\expandafter}%
2416 \romannumeral`&&\XINT_expr_fetch_E_comma_V_equal_E_a }%
2417 }%
2418 \def\XINT_expr_iterrx #1#2#3#4%
2419 {%
2420 \XINT:NEhook:iterr\XINT_expr_iterr\romannumeral0#1(#4)\relax {#2}#3#1%
2421 }%
2422 \def\XINT_expr_iterr #1#2#3#4#5%
2423 {%
2424 \expandafter\XINT_expr_put_op_first
2425 \expanded \bgroup {\iffalse}\fi
2426 \XINT_expr_iterr:_b {#5#4\relax !#3}#1^~#20?\XINT_expr_cb_and_getop
2427 }%
2428 \def\XINT_expr_iterr:_b #1#2%
2429 {%
2430 \ifx +#2\xint_dothis\XINT_expr_iterr:_Ca\fi
2431 \ifx !#2!\xint_dothis\XINT_expr_iterr:_noop\fi
2432 \ifx ^#2\xint_dothis\XINT_expr_iterr:_end\fi
2433 \xint_orthat{\XINT_expr_iterr:_c}{#2}{#1}%
2434 }%
2435 \def\XINT_expr_iterr:_noop #1{\XINT_expr_iterr:_b }%
2436 \def\XINT_expr_iterr:_end #1#2~#3#4?{{#3}\iffalse{\fi}}%
2437 \def\XINT_expr_iterr:_c #1#2{\expandafter\XINT_expr_iterr:_d\romannumeral0#2{{#1}}{#2}}%
2438 \def\XINT_expr_iterr:_d #1{\ifx ^#1\xint_dothis\XINT_expr_iterr:_abort\fi
2439 \ifx ?#1\xint_dothis\XINT_expr_iterr:_break\fi
2440 \ifx !#1\xint_dothis\XINT_expr_iterr:_omit\fi
2441 \xint_orthat{\XINT_expr_iterr:_goon {#1}}}%
2442 \def\XINT_expr_iterr:_abort #1!#2^~#3?{\iffalse{\fi}}%
2443 \def\XINT_expr_iterr:_break #1!#2^~#3?{#1\iffalse{\fi}}%
2444 \def\XINT_expr_iterr:_omit #1!#2#\expandafter\XINT_expr_iterr:_b\xint_gobble_i}%
2445 \def\XINT_expr_iterr:_goon #1!#2#\XINT_expr_iterr:_goon_a{#1}}%
2446 \def\XINT_expr_iterr:_goon_a #1#2#3~#4?%
2447 {%
2448 \expandafter\XINT_expr_iterr:_b \expanded{\unexpanded{#3~}\xintTrim{-2}{#1#4}}0?%
2449 }%
2450 \def\XINT_expr_iterr:_Ca #1#2#3{\XINT_expr_iterr:_Cc#3.#2}}%
2451 \def\XINT_expr_iterr:_Cb #1{\expandafter\XINT_expr_iterr:_Cc\the\numexpr#1+\xint_c_i.}%
2452 \def\XINT_expr_iterr:_Cc #1.#2{\expandafter\XINT_expr_iterr:_D\romannumeral0#2{{#1}}{#1}}{#2}}%
2453 \def\XINT_expr_iterr:_D #1{\ifx ^#1\xint_dothis\XINT_expr_iterr:_abort\fi
```

```

2454             \ifx ?#1\xint_dothis\XINT_expr_iterr:_break\fi
2455             \ifx !#1\xint_dothis\XINT_expr_iterr:_Omit\fi
2456             \xint_orthat{\XINT_expr_iterr:_Goon {#1}}}%
2457 \def\XINT_expr_iterr:_Omit #1!#2#{\expandafter\XINT_expr_iterr:_Cb\xint_gobble_i}%
2458 \def\XINT_expr_iterr:_Goon #1!#2#{\XINT_expr_iterr:_Goon_a{#1}}}%
2459 \def\XINT_expr_iterr:_Goon_a #1#2#3~#4?%
2460 {%
2461     \expandafter\XINT_expr_iterr:_Cb \expanded{\unexpanded{#3~}\xintTrim{-2}{#1#4}}0?%
2462 }%

```

11.25.10 rrseq()

When func_rrseq has its turn, initial segment has been scanned by oparen, the ; mimicking the rôle of a closing parenthesis, and stopping further expansion. #2 = \xint_c_i and #3 are left-over trash.

```

2463 \def\XINT_expr_func_rrseq {\XINT_allexpr_rrseq \xintbareeval }%
2464 \def\XINT_flexpr_func_rrseq {\XINT_allexpr_rrseq \xintbarefloateval }%
2465 \def\XINT_iexpr_func_rrseq {\XINT_allexpr_rrseq \xintbareiieval }%
2466 \def\XINT_allexpr_rrseq #1#2#3#4%
2467 {%
2468     \expandafter\XINT_expr_rrseq\expandafter#1\expanded
2469     {\unexpanded{#4}}{\xintRevWithBraces{#4}}\expandafter}%
2470     \romannumeral &&@\XINT_expr_fetch_E_comma_V_equal_E_a }%
2471 }%
2472 \def\XINT_expr_rrseqx #1#2#3#4#5%
2473 {%
2474     \XINT:NEhook:rrseq\XINT_expr_rrseq\romannumeral0#1(#5)\relax {#2}{#3}#4#1%
2475 }%
2476 \def\XINT_expr_rrseqy #1#2#3#4#5#6%
2477 {%
2478     \expandafter\XINT_expr_put_op_first
2479     \expanded \bgroup {\iffalse}\fi
2480     #2\XINT_expr_rrseq:_b {#6#5}\relax !#4#1^~#30?\XINT_expr_cb_and_getop
2481 }%
2482 \def\XINT_expr_rrseq:_b #1#2%
2483 {%
2484     \ifx +#2\xint_dothis\XINT_expr_rrseq:_Ca\fi
2485     \ifx !#2!\xint_dothis\XINT_expr_rrseq:_noop\fi
2486     \ifx ^#2\xint_dothis\XINT_expr_rrseq:_end\fi
2487     \xint_orthat{\XINT_expr_rrseq:_c}{#2}{#1}%
2488 }%
2489 \def\XINT_expr_rrseq:_noop #1{\XINT_expr_rrseq:_b }%
2490 \def\XINT_expr_rrseq:_end #1#2~#3?{\iffalse\fi}%
2491 \def\XINT_expr_rrseq:_c #1#2{\expandafter\XINT_expr_rrseq:_d\romannumeral0#2{{#1}}{#2}}%
2492 \def\XINT_expr_rrseq:_d #1{\ifx ^#1\xint_dothis\XINT_expr_rrseq:_abort\fi
2493     \ifx ?#1\xint_dothis\XINT_expr_rrseq:_break\fi
2494     \ifx !#1\xint_dothis\XINT_expr_rrseq:_omit\fi
2495     \xint_orthat{\XINT_expr_rrseq:_goon {#1}}}%
2496 \def\XINT_expr_rrseq:_abort #1!#2^~#3?{\iffalse\fi}%
2497 \def\XINT_expr_rrseq:_break #1!#2^~#3?{#1\iffalse\fi}%
2498 \def\XINT_expr_rrseq:_omit #1!#2#{\expandafter\XINT_expr_rrseq:_b\xint_gobble_i}%
2499 \def\XINT_expr_rrseq:_goon #1!#2#{\XINT_expr_rrseq:_goon_a {#1}}%

```

```

2500 \def\XINT_expr_rrseq:_goon_a #1#2#3~#4?%
2501 {%
2502     #1\expandafter\XINT_expr_rrseq:_b\expanded{\unexpanded{#3~}\xintTrim{-2}{#1#4}}0?%
2503 }%
2504 \def\XINT_expr_rrseq:_Ca #1#2#3{\XINT_expr_rrseq:_Cc#3.#2}%
2505 \def\XINT_expr_rrseq:_Cb #1{\expandafter\XINT_expr_rrseq:_Cc\the\numexpr#1+\xint_c_i.}%
2506 \def\XINT_expr_rrseq:_Cc #1.#2{\expandafter\XINT_expr_rrseq:_D\romannumeral0#2{#1}{#1}{#2}}%
2507 \def\XINT_expr_rrseq:_D #1{\ifx ^#1\xint_dothis\XINT_expr_rrseq:_abort\fi
2508     \ifx ?#1\xint_dothis\XINT_expr_rrseq:_break\fi
2509     \ifx !#1\xint_dothis\XINT_expr_rrseq:_Omit\fi
2510     \xint_orthat{\XINT_expr_rrseq:_Goon {#1}}}%
2511 \def\XINT_expr_rrseq:_Omit #1!#2#{\expandafter\XINT_expr_rrseq:_Cb\xint_gobble_i}%
2512 \def\XINT_expr_rrseq:_Goon #1!#2#{\XINT_expr_rrseq:_Goon_a {#1}}%
2513 \def\XINT_expr_rrseq:_Goon_a #1#2#3~#4?%
2514 {%
2515     #1\expandafter\XINT_expr_rrseq:_Cb\expanded{\unexpanded{#3~}\xintTrim{-2}{#1#4}}0?%
2516 }%
2517 \catcode`? 11

```

11.26 Pseudo-functions related to N-dimensional hypercubic lists

11.26.1 ndseq()

New with 1.4. 2020/01/23. It is derived from subsm() but instead of evaluating one expression according to one value per variable, it constructs a nested bracketed seq... this means the expression is parsed each time ! Anyway, proof of concept. Nota Bene : omit, abort, break() work !

```

2518 \def\XINT_expr_onliteral_ndseq
2519 {%
2520     \expandafter\XINT_allexpr_ndseq_f
2521     \romannumeral`&&\XINT_expr_fetch_E_comma_V_equal_E_a {}%
2522 }%
2523 \def\XINT_allexpr_ndseq_f #1#2{\xint_c_ii^v `{ndseqx}#2)\relax #1}%
2524 \def\XINT_expr_func_ndseqx
2525 {%
2526     \expandafter\XINT_allexpr_ndseqx\expandafter\xintbareeval
2527     \expandafter{\romannumeral0\expandafter\xint_gobble_i\string}%
2528     \expandafter\xintrevwithbraces
2529     \expanded\bgroup{\iffalse}\fi\XINT_allexpr_ndseq_A\XINT_expr_oparen
2530 }%
2531 \def\XINT_flexpr_func_ndseqx
2532 {%
2533     \expandafter\XINT_allexpr_ndseqx\expandafter\xintbarefloateval
2534     \expandafter{\romannumeral0\expandafter\xint_gobble_i\string}%
2535     \expandafter\xintrevwithbraces
2536     \expanded\bgroup{\iffalse}\fi\XINT_allexpr_ndseq_A\XINT_flexpr_oparen
2537 }%
2538 \def\XINT_iiexpr_func_ndseqx
2539 {%
2540     \expandafter\XINT_allexpr_ndseqx\expandafter\xintbareiieval
2541     \expandafter{\romannumeral0\expandafter\xint_gobble_i\string}%
2542     \expandafter\xintrevwithbraces

```

```

2543   \expanded\bgroup{\iffalse}\fi\XINT_allexpr_ndseq_A\XINT_iiexpr_oparen
2544 }%
2545 \def\XINT_allexpr_ndseq_A #1#2#3%
2546 {%
2547   \ifx#2\xint_c_
2548     \expandafter\XINT_allexpr_ndseq_C
2549   \else
2550     \expandafter\XINT_allexpr_ndseq_B
2551   \fi #1%
2552 }%
2553 \def\XINT_allexpr_ndseq_B #1#2#3#4=%
2554 {%
2555   {#2}{\xint_zapspaces#3#4 \xint_gobble_i}%
2556   \expandafter\XINT_allexpr_ndseq_A\expandafter#1\romannumeral`&&@#1%
2557 }%

#1 = \xintbareeval, or \xintbarefloateval or \xintbareiieval #2 = values for last coordinate

2558 \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

2559 \def\XINT_allexpr_ndseqx #1#2#3#4%
2560 {%
2561   \expandafter\XINT_expr_put_op_first
2562   \expanded
2563   \bgroup
2564   \romannumeral0#1\empty
2565   \expanded{\xintReplicate{\xintLength{{#3}#2}/2}{[seq(}%
2566     \unexpanded{#4}%
2567     \XINT_allexpr_ndseqx_a #2{#3}^^%
2568   ]}%
2569   \relax
2570   \iffalse{\fi\expandafter}\romannumeral`&&\XINT_expr_getop
2571 }%
2572 \def\XINT_allexpr_ndseqx_a #1#2%
2573 {%
2574   \xint_gob_til_^ #1\XINT_allexpr_ndseqx_e ^%
2575   \unexpanded{,#2=\XINTfstop.{#1}}}\XINT_allexpr_ndseqx_a
2576 }%
2577 \def\XINT_allexpr_ndseqx_e ^#1\XINT_allexpr_ndseqx_a{%

```

11.26.2 ndmap()

New with 1.4. 2020/01/24.

```

2578 \def\XINT_expr_onliteral_ndmap #1,{\xint_c_ii^v `{ndmapx}\XINTfstop.{#1};}%
2579 \def\XINT_expr_func_ndmapx #1#2#3%
2580 {%
2581   \expandafter\XINT_allexpr_ndmapx
2582   \csname XINT_expr_func_\xint_zapspaces #3 \xint_gobble_i\endcsname
2583   \XINT_expr_oparen
2584 }%

```

```

2585 \def\XINT_flexpr_func_ndmapx #1#2#3%
2586 {%
2587   \expandafter\XINT_allexpr_ndmapx
2588   \csname XINT_flexpr_func_\xint_zapspaces #3 \xint_gobble_i\endcsname
2589   \XINT_flexpr_oparen
2590 }%
2591 \def\XINT_iiexpr_func_ndmapx #1#2#3%
2592 {%
2593   \expandafter\XINT_allexpr_ndmapx
2594   \csname XINT_iiexpr_func_\xint_zapspaces #3 \xint_gobble_i\endcsname
2595   \XINT_iiexpr_oparen
2596 }%
2597 \def\XINT_allexpr_ndmapx #1#2%
2598 {%
2599   \expandafter\XINT_expr_put_op_first
2600   \expanded\bgroup{\iffalse}\fi
2601   \expanded
2602   {\noexpand\XINT:NEhook:x:ndmapx
2603    \noexpand\XINT_allexpr_ndmapx_a
2604    \noexpand#1}\expandafter}%
2605   \expanded\bgroup\expandafter\XINT_allexpr_ndmap_A
2606           \expandafter#2\romannumeral`&&@#2%
2607 }%
2608 \def\XINT_allexpr_ndmap_A #1#2#3%
2609 {%
2610   \ifx#3;%
2611     \expandafter\XINT_allexpr_ndmap_B
2612   \else
2613     \xint_afterfi{\XINT_allexpr_ndmap_C#2#3}%
2614   \fi #1%
2615 }%
2616 \def\XINT_allexpr_ndmap_B #1#2%
2617 {%
2618   {#2}\expandafter\XINT_allexpr_ndmap_A\expandafter#1\romannumeral`&&@#1%
2619 }%
2620 \def\XINT_allexpr_ndmap_C #1#2#3#4%
2621 {%
2622   {#4}^\relax\iffalse{{\fi}}#1#2%
2623 }%
2624 \def\XINT_allexpr_ndmapx_a #1#2#3%
2625 {%
2626   \xint_gob_til_^ #3\XINT_allexpr_ndmapx_l ^%
2627   \XINT_allexpr_ndmapx_b #1{#2}{#3}%
2628 }%
2629 \def\XINT_allexpr_ndmapx_l ^#1\XINT_allexpr_ndmapx_b #2#3#4\relax
2630 {%
2631   #2\empty\xint_firstofone{#3}%
2632 }%
2633 \def\XINT_allexpr_ndmapx_b #1#2#3#4\relax
2634 {%
2635   {\iffalse}\fi\XINT_allexpr_ndmapx_c {#4\relax}#1{#2}{#3}%
2636 }%

```

```

2637 \def\XINT_allexpr_ndmapx_c #1#2#3#4%
2638 {%
2639   \xint_gob_til_^ #4\XINT_allexpr_ndmapx_e ^%
2640   \XINT_allexpr_ndmapx_a #2#{#3#{#4}}#1%
2641   \XINT_allexpr_ndmapx_c  {#1}#2#{#3}%
2642 }%
2643 \def\XINT_allexpr_ndmapx_e ^#1\XINT_allexpr_ndmapx_c
2644  {\iffalse{\fi}\xint_gobble_iii}%

```

11.26.3 ndfillraw()

New with 1.4. 2020/01/24. J'hésite à autoriser un #1 quelconque, ou plutôt à le wrapper dans un `\xintbareval`. Mais il faut alors distinguer les trois. De toute façon les variables ne marcheraient pas donc j'hésite à mettre un wrapper automatique. Mais ce n'est pas bien d'autoriser l'injection de choses quelconques.

Pour des choses comme `ndfillraw(\xintRandomBit,[10,10])`.

Je n'aime pas le nom !. Le changer. `ndconst`? Surtout je n'aime pas que dans le premier argument il faut rajouter explicitement si nécessaire `\xintiexpr wrap`.

```

2645 \def\XINT_expr_onliteral_ndfillraw #1,{\xint_c_ii^v `{\ndfillrawx}\XINTfstop.{{#1}},}%
2646 \def\XINT_expr_func_ndfillrawx #1#2#3%
2647 {%
2648   \expandafter#1\expandafter#2\expanded{{{XINT_allexpr_ndfillrawx_a #3}}}%
2649 }%
2650 \let\XINT_iiexpr_func_ndfillrawx\XINT_expr_func_ndfillrawx
2651 \let\XINT_flexpr_func_ndfillrawx\XINT_expr_func_ndfillrawx
2652 \def\XINT_allexpr_ndfillrawx_a #1#2%
2653 {%
2654   \expandafter\XINT_allexpr_ndfillrawx_b
2655   \romannumeral0\xintApply{\xintNum}{#2}^{\relax {#1}}%
2656 }%
2657 \def\XINT_allexpr_ndfillrawx_b #1#2\relax#3%
2658 {%
2659   \xint_gob_til_^ #1\XINT_allexpr_ndfillrawx_c ^%
2660   \xintReplicate{#1}{{XINT_allexpr_ndfillrawx_b #2\relax {#3}}}%
2661 }%
2662 \def\XINT_allexpr_ndfillrawx_c ^\xintReplicate #1#2%
2663 {%
2664   \expandafter\XINT_allexpr_ndfillrawx_d\xint_firstofone #2%
2665 }%
2666 \def\XINT_allexpr_ndfillrawx_d\XINT_allexpr_ndfillrawx_b \relax #1{#1}%

```

11.27 Other pseudo-functions: `bool()`, `togl()`, `protect()`, `qraw()`, `qint()`, `qfrac()`, `qfloat()`, `qrand()`, `random()`, `rbit()`

`bool`, `togl` and `protect` use delimited macros. They are not true functions, they turn off the parser to gather their "variable".

1.2. adds `qint()`, `qfrac()`, `qfloat()`.

1.3c. adds `qraw()`. Useful to limit impact on \TeX memory from abuse of `\csname`'s storage when generating many comma separated values from a loop.

1.3e. `qfloat()` keeps a short mantissa if possible.

They allow the user to hand over quickly a big number to the parser, spaces not immediately removed but should be harmless in general. The `qraw()` does no post-processing at all apart complete expansion, useful for comma-separated values, but must be obedient to (non really documented) expected format. Each uses a delimited macro, the closing parenthesis can not emerge from expansion.

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.

```

2667 \def\XINT_expr_onliteral_bool #1)%
2668   {\expandafter\XINT_expr_put_op_first\expanded{{{xintBool{#1}}}\expandafter
2669     }\romannumeral`&&\XINT_expr_getop}%
2670 \def\XINT_expr_onliteral_togl #1)%
2671   {\expandafter\XINT_expr_put_op_first\expanded{{{xintToggle{#1}}}\expandafter
2672     }\romannumeral`&&\XINT_expr_getop}%
2673 \def\XINT_expr_onliteral_protect #1)%
2674   {\expandafter\XINT_expr_put_op_first\expanded{{{detokenize{#1}}}\expandafter
2675     }\romannumeral`&&\XINT_expr_getop}%
2676 \def\XINT_expr_onliteral_qint #1)%
2677   {\expandafter\XINT_expr_put_op_first\expanded{{{xintiNum{#1}}}\expandafter
2678     }\romannumeral`&&\XINT_expr_getop}%
2679 \def\XINT_expr_onliteral_qfrac #1)%
2680   {\expandafter\XINT_expr_put_op_first\expanded{{{xintRaw{#1}}}\expandafter
2681     }\romannumeral`&&\XINT_expr_getop}%
2682 \def\XINT_expr_onliteral_qfloat #1)%
2683   {\expandafter\XINT_expr_put_op_first\expanded{{{XINTinFloatSdigits{#1}}}\expandafter
2684     }\romannumeral`&&\XINT_expr_getop}%
2685 \def\XINT_expr_onliteral_qraw #1)%
2686   {\expandafter\XINT_expr_put_op_first\expanded{#{#1}\expandafter
2687     }\romannumeral`&&\XINT_expr_getop}%
2688 \def\XINT_expr_onliteral_random #1)%
2689   {\expandafter\XINT_expr_put_op_first\expanded{{{XINTinRandomFloatSdigits}}\expandafter
2690     }\romannumeral`&&\XINT_expr_getop}%
2691 \def\XINT_expr_onliteral_grand #1)%
2692   {\expandafter\XINT_expr_put_op_first\expanded{{{XINTinRandomFloatSixteen}}\expandafter
2693     }\romannumeral`&&\XINT_expr_getop}%
2694 \def\XINT_expr_onliteral_rbit #1)%
2695   {\expandafter\XINT_expr_put_op_first\expanded{{{xintRandBit}}\expandafter
2696     }\romannumeral`&&\XINT_expr_getop}%

```

11.28 Regular built-in functions: `num()`, `reduce()`, `preduce()`, `abs()`, `sgn()`, `frac()`, `floor()`, `ceil()`, `sqr()`, `?()`, `!()`, `not()`, `odd()`, `even()`, `isint()`, `isone()`, `factorial()`, `sqrtr()`, `inv()`, `round()`, `trunc()`, `float()`, `sfloat()`, `ilog10()`, `divmod()`, `mod()`, `binomial()`, `pfactorial()`, `randrange()`, `quo()`, `rem()`, `gcd()`, `lcm()`, `max()`, `min()`, ``+`()`, ``*`()`, `all()`, `any()`, `xor()`, `len()`, `first()`, `last()`, `reversed()`, `if()`, `ifint()`, `ifone()`, `ifsgn()`, `nuple()`, `unpack()`, `flat()` and `zip()`

```

2697 \def\XINT:expr:f:one:and:opt #1#2#3!#4#5%
2698 {%

```



```

2699 \if\relax#3\relax\expandafter\xint_firstoftwo\else
2700 \expandafter\xint_secondoftwo\fi
2701 {#4}{#5[\xintNum{#2}]}{#1}%
2702 }%
2703 \def\xINT:expr:f:tacitzeroifone #1#2#3!#4#5%
2704 {%
2705 \if\relax#3\relax\expandafter\xint_firstoftwo\else
2706 \expandafter\xint_secondoftwo\fi
2707 {#4{0}}{#5{\xintNum{#2}}}{#1}%
2708 }%
2709 \def\xINT:expr:f:iitacitzeroifone #1#2#3!#4%
2710 {%
2711 \if\relax#3\relax\expandafter\xint_firstoftwo\else
2712 \expandafter\xint_secondoftwo\fi
2713 {#4{0}}{#4{#2}}{#1}%
2714 }%
2715 \def\xINT_expr_func_num #1#2#3%
2716 {%
2717 \expandafter #1\expandafter #2\expandafter{%
2718 \romannumeral`&&\xINT:NEhook:f:one:from:one
2719 {\romannumeral`&&\xintNum#3}}%
2720 }%
2721 \let\xINT_flexpr_func_num\xINT_expr_func_num
2722 \let\xINT_iiexpr_func_num\xINT_expr_func_num
2723 \def\xINT_expr_func_reduce #1#2#3%
2724 {%
2725 \expandafter #1\expandafter #2\expandafter{%
2726 \romannumeral`&&\xINT:NEhook:f:one:from:one
2727 {\romannumeral`&&\xintIrr#3}}%
2728 }%
2729 \let\xINT_flexpr_func_reduce\xINT_expr_func_reduce
2730 \def\xINT_expr_func_preduce #1#2#3%
2731 {%
2732 \expandafter #1\expandafter #2\expandafter{%
2733 \romannumeral`&&\xINT:NEhook:f:one:from:one
2734 {\romannumeral`&&\xintPIrr#3}}%
2735 }%
2736 \let\xINT_flexpr_func_preduce\xINT_expr_func_preduce
2737 \def\xINT_expr_func_abs #1#2#3%
2738 {%
2739 \expandafter #1\expandafter #2\expandafter{%
2740 \romannumeral`&&\xINT:NEhook:f:one:from:one
2741 {\romannumeral`&&\xintAbs#3}}%
2742 }%
2743 \let\xINT_flexpr_func_abs\xINT_expr_func_abs
2744 \def\xINT_iiexpr_func_abs #1#2#3%
2745 {%
2746 \expandafter #1\expandafter #2\expandafter{%
2747 \romannumeral`&&\xINT:NEhook:f:one:from:one
2748 {\romannumeral`&&\xintiAbs#3}}%
2749 }%
2750 \def\xINT_expr_func_sgn #1#2#3%

```

```

2751 {%
2752   \expandafter #1\expandafter #2\expandafter{%
2753   \romannumeral`&&\XINT:NEhook:f:one:from:one
2754   {\romannumeral`&&\xintSgn#3}}%
2755 }%
2756 \let\XINT_flexpr_func_sgn\XINT_expr_func_sgn
2757 \def\XINT_iiexpr_func_sgn #1#2#3%
2758 {%
2759   \expandafter #1\expandafter #2\expandafter{%
2760   \romannumeral`&&\XINT:NEhook:f:one:from:one
2761   {\romannumeral`&&\xintiSgn#3}}%
2762 }%
2763 \def\XINT_expr_func_frac #1#2#3%
2764 {%
2765   \expandafter #1\expandafter #2\expandafter{%
2766   \romannumeral`&&\XINT:NEhook:f:one:from:one
2767   {\romannumeral`&&\xintTFrac#3}}%
2768 }%
2769 \def\XINT_flexpr_func_frac #1#2#3%
2770 {%
2771   \expandafter #1\expandafter #2\expandafter{%
2772   \romannumeral`&&\XINT:NEhook:f:one:from:one
2773   {\romannumeral`&&\XINTinFloatFracdigits#3}}%
2774 }%

```

no \XINT_iiexpr_func_frac

```

2775 \def\XINT_expr_func_floor #1#2#3%
2776 {%
2777   \expandafter #1\expandafter #2\expandafter{%
2778   \romannumeral`&&\XINT:NEhook:f:one:from:one
2779   {\romannumeral`&&\xintFloor#3}}%
2780 }%
2781 \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.

```

2782 \def\XINT_iiexpr_func_floor #1#2#3%
2783 {%
2784   \expandafter #1\expandafter #2\expandafter{%
2785   \romannumeral`&&\XINT:NEhook:f:one:from:one
2786   {\romannumeral`&&\xintiFloor#3}}%
2787 }%
2788 \def\XINT_expr_func_ceil #1#2#3%
2789 {%
2790   \expandafter #1\expandafter #2\expandafter{%
2791   \romannumeral`&&\XINT:NEhook:f:one:from:one
2792   {\romannumeral`&&\xintCeil#3}}%
2793 }%
2794 \let\XINT_flexpr_func_ceil\XINT_expr_func_ceil
2795 \def\XINT_iiexpr_func_ceil #1#2#3%
2796 {%
2797   \expandafter #1\expandafter #2\expandafter{%
2798   \romannumeral`&&\XINT:NEhook:f:one:from:one

```

```

2799     {\romannumeral`&&\xintiCeil#3}}%
2800 }%
2801 \def\XINT_expr_func_sqr #1#2#3%
2802 {%
2803     \expandafter #1\expandafter #2\expandafter{%
2804     \romannumeral`&&\XINT:NEhook:f:one:from:one
2805     {\romannumeral`&&\xintSqr#3}}%
2806 }%
2807 \def\XINTinFloatSqr#1{\XINTinFloatMul{#1}{#1}}%
2808 \def\XINT_flexpr_func_sqr #1#2#3%
2809 {%
2810     \expandafter #1\expandafter #2\expandafter{%
2811     \romannumeral`&&\XINT:NEhook:f:one:from:one
2812     {\romannumeral`&&\XINTinFloatSqr#3}}%
2813 }%
2814 \def\XINT_iiexpr_func_sqr #1#2#3%
2815 {%
2816     \expandafter #1\expandafter #2\expandafter{%
2817     \romannumeral`&&\XINT:NEhook:f:one:from:one
2818     {\romannumeral`&&\xintiiSqr#3}}%
2819 }%
2820 \def\XINT_expr_func_? #1#2#3%
2821 {%
2822     \expandafter #1\expandafter #2\expandafter{%
2823     \romannumeral`&&\XINT:NEhook:f:one:from:one
2824     {\romannumeral`&&\xintiiIsNotZero#3}}%
2825 }%
2826 \let\XINT_flexpr_func_? \XINT_expr_func_?
2827 \let\XINT_iiexpr_func_? \XINT_expr_func_?
2828 \def\XINT_expr_func_! #1#2#3%
2829 {%
2830     \expandafter #1\expandafter #2\expandafter{%
2831     \romannumeral`&&\XINT:NEhook:f:one:from:one
2832     {\romannumeral`&&\xintiiIsZero#3}}%
2833 }%
2834 \let\XINT_flexpr_func_! \XINT_expr_func_!
2835 \let\XINT_iiexpr_func_! \XINT_expr_func_!
2836 \def\XINT_expr_func_not #1#2#3%
2837 {%
2838     \expandafter #1\expandafter #2\expandafter{%
2839     \romannumeral`&&\XINT:NEhook:f:one:from:one
2840     {\romannumeral`&&\xintiiIsZero#3}}%
2841 }%
2842 \let\XINT_flexpr_func_not \XINT_expr_func_not
2843 \let\XINT_iiexpr_func_not \XINT_expr_func_not
2844 \def\XINT_expr_func_odd #1#2#3%
2845 {%
2846     \expandafter #1\expandafter #2\expandafter{%
2847     \romannumeral`&&\XINT:NEhook:f:one:from:one
2848     {\romannumeral`&&\xintOdd#3}}%
2849 }%
2850 \let\XINT_flexpr_func_odd\XINT_expr_func_odd

```

```

2851 \def\XINT_iiexpr_func_odd #1#2#3%
2852 {%
2853   \expandafter #1\expandafter #2\expandafter{%
2854   \romannumeral`&&\XINT:NEhook:f:one:from:one
2855   {\romannumeral`&&\xintiiOdd#3}}%
2856 }%
2857 \def\XINT_expr_func_even #1#2#3%
2858 {%
2859   \expandafter #1\expandafter #2\expandafter{%
2860   \romannumeral`&&\XINT:NEhook:f:one:from:one
2861   {\romannumeral`&&\xintEven#3}}%
2862 }%
2863 \let\XINT_flexpr_func_even\XINT_expr_func_even
2864 \def\XINT_iiexpr_func_even #1#2#3%
2865 {%
2866   \expandafter #1\expandafter #2\expandafter{%
2867   \romannumeral`&&\XINT:NEhook:f:one:from:one
2868   {\romannumeral`&&\xintiiEven#3}}%
2869 }%
2870 \def\XINT_expr_func_isint #1#2#3%
2871 {%
2872   \expandafter #1\expandafter #2\expandafter{%
2873   \romannumeral`&&\XINT:NEhook:f:one:from:one
2874   {\romannumeral`&&\xintIsInt#3}}%
2875 }%
2876 \def\XINT_flexpr_func_isint #1#2#3%
2877 {%
2878   \expandafter #1\expandafter #2\expandafter{%
2879   \romannumeral`&&\XINT:NEhook:f:one:from:one
2880   {\romannumeral`&&\xintFloatIsInt#3}}%
2881 }%
2882 \let\XINT_iiexpr_func_isint\XINT_expr_func_isint % ? perhaps rather always 1
2883 \def\XINT_expr_func_isone #1#2#3%
2884 {%
2885   \expandafter #1\expandafter #2\expandafter{%
2886   \romannumeral`&&\XINT:NEhook:f:one:from:one
2887   {\romannumeral`&&\xintIsOne#3}}%
2888 }%
2889 \let\XINT_flexpr_func_isone\XINT_expr_func_isone
2890 \def\XINT_iiexpr_func_isone #1#2#3%
2891 {%
2892   \expandafter #1\expandafter #2\expandafter{%
2893   \romannumeral`&&\XINT:NEhook:f:one:from:one
2894   {\romannumeral`&&\xintiiIsOne#3}}%
2895 }%
2896 \def\XINT_expr_func_factorial #1#2#3%
2897 {%
2898   \expandafter #1\expandafter #2\expandafter{\expandafter{%
2899   \romannumeral`&&\XINT:NEhook:f:one:and:opt:direct
2900   \XINT:expr:f:one:and:opt #3,! \xintFac\XINTinFloatFac
2901   }}%
2902 }%

```

```

2903 \def\XINT_flexpr_func_factorial #1#2#3%
2904 {%
2905   \expandafter #1\expandafter #2\expandafter{\expandafter{%
2906     \romannumeral`&&\XINT:NEhook:f:one:and:opt:direct
2907     \XINT:expr:f:one:and:opt#3,! \XINTinFloatFacdigits\XINTinFloatFac
2908   }}%
2909 }%
2910 \def\XINT_iiexpr_func_factorial #1#2#3%
2911 {%
2912   \expandafter #1\expandafter #2\expandafter{%
2913     \romannumeral`&&\XINT:NEhook:f:one:from:one
2914     {\romannumeral`&&\xintiiFac#3}}%
2915 }%
2916 \def\XINT_expr_func_sqrt #1#2#3%
2917 {%
2918   \expandafter #1\expandafter #2\expandafter{\expandafter{%
2919     \romannumeral`&&\XINT:NEhook:f:one:and:opt:direct
2920     \XINT:expr:f:one:and:opt #3,! \XINTinFloatSqrtdigits\XINTinFloatSqrt
2921   }}%
2922 }%
2923 \let\XINT_flexpr_func_sqrt\XINT_expr_func_sqrt
2924 \def\XINT_expr_func_sqrt_ #1#2#3%
2925 {%
2926   \expandafter #1\expandafter #2\expandafter{%
2927     \romannumeral`&&\XINT:NEhook:f:one:from:one
2928     {\romannumeral`&&\XINTinFloatSqrtdigits#3}}%
2929 }%
2930 \let\XINT_flexpr_func_sqrt_\XINT_expr_func_sqrt_
2931 \def\XINT_iiexpr_func_sqrt #1#2#3%
2932 {%
2933   \expandafter #1\expandafter #2\expandafter{%
2934     \romannumeral`&&\XINT:NEhook:f:one:from:one
2935     {\romannumeral`&&\xintiiSqrt#3}}%
2936 }%
2937 \def\XINT_iiexpr_func_sqrtr #1#2#3%
2938 {%
2939   \expandafter #1\expandafter #2\expandafter{%
2940     \romannumeral`&&\XINT:NEhook:f:one:from:one
2941     {\romannumeral`&&\xintiiSqrtr#3}}%
2942 }%
2943 \def\XINT_expr_func_inv #1#2#3%
2944 {%
2945   \expandafter #1\expandafter #2\expandafter{%
2946     \romannumeral`&&\XINT:NEhook:f:one:from:one
2947     {\romannumeral`&&\xintInv#3}}%
2948 }%
2949 \def\XINT_flexpr_func_inv #1#2#3%
2950 {%
2951   \expandafter #1\expandafter #2\expandafter{%
2952     \romannumeral`&&\XINT:NEhook:f:one:from:one
2953     {\romannumeral`&&\XINTinFloatInv#3}}%
2954 }%

```

```

2955 \def\XINT_expr_func_round #1#2#3%
2956 {%
2957   \expandafter #1\expandafter #2\expandafter{\expandafter{%
2958     \romannumeral`&&\XINT:NEhook:f:tacitzeroifone:direct
2959     \XINT:expr:f:tacitzeroifone #3,! \xintiRound\xintRound
2960   }}%
2961 }%
2962 \let\XINT_flexpr_func_round\XINT_expr_func_round
2963 \def\XINT_iiexpr_func_round #1#2#3%
2964 {%
2965   \expandafter #1\expandafter #2\expandafter{\expandafter{%
2966     \romannumeral`&&\XINT:NEhook:f:iitacitzeroifone:direct
2967     \XINT:expr:f:iitacitzeroifone #3,! \xintiRound
2968   }}%
2969 }%
2970 \def\XINT_expr_func_trunc #1#2#3%
2971 {%
2972   \expandafter #1\expandafter #2\expandafter{\expandafter{%
2973     \romannumeral`&&\XINT:NEhook:f:tacitzeroifone:direct
2974     \XINT:expr:f:tacitzeroifone #3,! \xintiTrunc\xintTrunc
2975   }}%
2976 }%
2977 \let\XINT_flexpr_func_trunc\XINT_expr_func_trunc
2978 \def\XINT_iiexpr_func_trunc #1#2#3%
2979 {%
2980   \expandafter #1\expandafter #2\expandafter{\expandafter{%
2981     \romannumeral`&&\XINT:NEhook:f:iitacitzeroifone:direct
2982     \XINT:expr:f:iitacitzeroifone #3,! \xintiTrunc
2983   }}%
2984 }%

```

Hesitation at 1.3e about using `\XINTinFloatSdigits` and `\XINTinFloatS`. Finally I add a `sfloat()` function. It helps for `xinttrig.sty`.

```

2985 \def\XINT_expr_func_float #1#2#3%
2986 {%
2987   \expandafter #1\expandafter #2\expandafter{\expandafter{%
2988     \romannumeral`&&\XINT:NEhook:f:one:and:opt:direct
2989     \XINT:expr:f:one:and:opt #3,! \XINTinFloatdigits\XINTinFloat
2990   }}%
2991 }%
2992 \let\XINT_flexpr_func_float\XINT_expr_func_float

```

`float_()` added at 1.4. Does not check for optional argument. Useful to transfer functions defined with `\xintdefunc` to functions usable in `\xintfloateval`. I hesitated briefly about notation but here we go. Unfortunately I will have to document it (contrarily to `sqrt_()`).

No need to do same for `sfloat()` currently used in `xinttrig.sty` to go from `float` to `expr`, because `sfloat(x)` sees there is no optional argument.

Still I wonder if better would not be to have some function «`single()`» which signals to outer one it is a single argument? Must think about this. Too late now for 1.4.

```

2993 \def\XINT_expr_func_float_ #1#2#3%
2994 {%
2995   \expandafter #1\expandafter #2\expandafter{%
2996     \romannumeral`&&\XINT:NEhook:f:one:from:one

```

```

2997     {\romannumeral`&&\XINTinFloatdigits#3}}%
2998 }%
2999 \let\XINT_flexpr_func_float_\XINT_expr_func_float_
3000 \def\XINT_expr_func_sfloat #1#2#3%
3001 {%
3002     \expandafter #1\expandafter #2\expandafter{\expandafter{%
3003     \romannumeral`&&\XINT:NEhook:f:one:and:opt:direct
3004     \XINT:expr:f:one:and:opt #3,! \XINTinFloatSdigits\XINTinFloatS
3005     }}%
3006 }%
3007 \let\XINT_flexpr_func_sfloat\XINT_expr_func_sfloat
3008 % \XINT_iiexpr_func_sfloat not defined
3009 \expandafter\def\csname XINT_expr_func_ilog10\endcsname #1#2#3%
3010 {%
3011     \expandafter #1\expandafter #2\expandafter{\expandafter{%
3012     \romannumeral`&&\XINT:NEhook:f:one:and:opt:direct
3013     \XINT:expr:f:one:and:opt #3,! \xintiLogTen\XINTFloatiLogTen
3014     }}%
3015 }%
3016 \expandafter\def\csname XINT_flexpr_func_ilog10\endcsname #1#2#3%
3017 {%
3018     \expandafter #1\expandafter #2\expandafter{\expandafter{%
3019     \romannumeral`&&\XINT:NEhook:f:one:and:opt:direct
3020     \XINT:expr:f:one:and:opt #3,! \XINTFloatiLogTendigits\XINTFloatiLogTen
3021     }}%
3022 }%
3023 \expandafter\def\csname XINT_iiexpr_func_ilog10\endcsname #1#2#3%
3024 {%
3025     \expandafter #1\expandafter #2\expandafter{%
3026     \romannumeral`&&\XINT:NEhook:f:one:from:one
3027     {\romannumeral`&&\xintiiLogTen#3}}%
3028 }%
3029 \def\XINT_expr_func_divmod #1#2#3%
3030 {%
3031     \expandafter #1\expandafter #2\expandafter{\romannumeral`&&%
3032     \XINT:NEhook:f:one:from:two
3033     {\romannumeral`&&\xintDivMod #3}}%
3034 }%
3035 \def\XINT_flexpr_func_divmod #1#2#3%
3036 {%
3037     \expandafter #1\expandafter #2\expandafter{\romannumeral`&&%
3038     \XINT:NEhook:f:one:from:two
3039     {\romannumeral`&&\XINTinFloatDivMod #3}}%
3040 }%
3041 \def\XINT_iiexpr_func_divmod #1#2#3%
3042 {%
3043     \expandafter #1\expandafter #2\expandafter{\romannumeral`&&%
3044     \XINT:NEhook:f:one:from:two
3045     {\romannumeral`&&\xintiiDivMod #3}}%
3046 }%
3047 \def\XINT_expr_func_mod #1#2#3%
3048 {%

```

```

3049 \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
3050 \XINT:NEhook:f:one:from:two
3051 {\romannumeral`&&\xintMod#3}}%
3052 }%
3053 \def\XINT_flexpr_func_mod #1#2#3%
3054 {%
3055 \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
3056 \XINT:NEhook:f:one:from:two
3057 {\romannumeral`&&\XINTinFloatMod#3}}%
3058 }%
3059 \def\XINT_iiexpr_func_mod #1#2#3%
3060 {%
3061 \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
3062 \XINT:NEhook:f:one:from:two
3063 {\romannumeral`&&\xintiiMod#3}}%
3064 }%
3065 \def\XINT_expr_func_binomial #1#2#3%
3066 {%
3067 \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
3068 \XINT:NEhook:f:one:from:two
3069 {\romannumeral`&&\xintBinomial #3}}%
3070 }%
3071 \def\XINT_flexpr_func_binomial #1#2#3%
3072 {%
3073 \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
3074 \XINT:NEhook:f:one:from:two
3075 {\romannumeral`&&\XINTinFloatBinomial #3}}%
3076 }%
3077 \def\XINT_iiexpr_func_binomial #1#2#3%
3078 {%
3079 \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
3080 \XINT:NEhook:f:one:from:two
3081 {\romannumeral`&&\xintiiBinomial #3}}%
3082 }%
3083 \def\XINT_expr_func_pfactorial #1#2#3%
3084 {%
3085 \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
3086 \XINT:NEhook:f:one:from:two
3087 {\romannumeral`&&\xintPFactorial #3}}%
3088 }%
3089 \def\XINT_flexpr_func_pfactorial #1#2#3%
3090 {%
3091 \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
3092 \XINT:NEhook:f:one:from:two
3093 {\romannumeral`&&\XINTinFloatPFactorial #3}}%
3094 }%
3095 \def\XINT_iiexpr_func_pfactorial #1#2#3%
3096 {%
3097 \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
3098 \XINT:NEhook:f:one:from:two
3099 {\romannumeral`&&\xintiiPFactorial #3}}%
3100 }%

```



```

3101 \def\XINT_expr_func_randrange #1#2#3%
3102 {%
3103   \expandafter #1\expandafter #2\expanded{{{%
3104   \XINT:expr:randrange #3,!%
3105   }}}%
3106 }%
3107 \let\XINT_flexpr_func_randrange\XINT_expr_func_randrange
3108 \def\XINT_iiexpr_func_randrange #1#2#3%
3109 {%
3110   \expandafter #1\expandafter #2\expanded{{{%
3111   \XINT:iiexpr:randrange #3,!%
3112   }}}%
3113 }%
3114 \def\XINT:expr:randrange #1#2#3!%
3115 {%
3116   \if\relax#3\relax\expandafter\xint_firstoftwo\else
3117     \expandafter\xint_secondoftwo\fi
3118   {\xintiRandRange{\XINT:NEhook:f:one:from:one:direct\xintNum{#1}}}%
3119   {\xintiRandRangeAtoB{\XINT:NEhook:f:one:from:one:direct\xintNum{#1}}%
3120     {\XINT:NEhook:f:one:from:one:direct\xintNum{#2}}}%
3121   }%
3122 }%
3123 \def\XINT:iiexpr:randrange #1#2#3!%
3124 {%
3125   \if\relax#3\relax\expandafter\xint_firstoftwo\else
3126     \expandafter\xint_secondoftwo\fi
3127   {\xintiRandRange{#1}}%
3128   {\xintiRandRangeAtoB{#1}{#2}}%
3129 }%
3130 \def\XINT_expr_func_quo #1#2#3%
3131 {%
3132   \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
3133   \XINT:NEhook:f:one:from:two
3134   {\romannumeral`&&\xintiQuo #3}}%
3135 }%
3136 \let\XINT_flexpr_func_quo\XINT_expr_func_quo
3137 \def\XINT_iiexpr_func_quo #1#2#3%
3138 {%
3139   \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
3140   \XINT:NEhook:f:one:from:two
3141   {\romannumeral`&&\xintiQuo #3}}%
3142 }%
3143 \def\XINT_expr_func_rem #1#2#3%
3144 {%
3145   \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%
3146   \XINT:NEhook:f:one:from:two
3147   {\romannumeral`&&\xintiRem #3}}%
3148 }%
3149 \let\XINT_flexpr_func_rem\XINT_expr_func_rem
3150 \def\XINT_iiexpr_func_rem #1#2#3%
3151 {%
3152   \expandafter #1\expandafter #2\expandafter{\romannumeral`&&@%

```

```

3153 \XINT:NEhook:f:one:from:two
3154 {\romannumeral`&&\xintiiRem #3}}%
3155 }%
3156 \def\xINT_expr_func_gcd #1#2#3%
3157 {%
3158 \expandafter #1\expandafter #2\expandafter{\expandafter
3159 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\xINT_GCDof#3^}}%
3160 }%
3161 \let\xINT_flexpr_func_gcd\xINT_expr_func_gcd
3162 \def\xINT_iiexpr_func_gcd #1#2#3%
3163 {%
3164 \expandafter #1\expandafter #2\expandafter{\expandafter
3165 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\xINT_iiGCDof#3^}}%
3166 }%
3167 \def\xINT_expr_func_lcm #1#2#3%
3168 {%
3169 \expandafter #1\expandafter #2\expandafter{\expandafter
3170 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\xINT_LCMof#3^}}%
3171 }%
3172 \let\xINT_flexpr_func_lcm\xINT_expr_func_lcm
3173 \def\xINT_iiexpr_func_lcm #1#2#3%
3174 {%
3175 \expandafter #1\expandafter #2\expandafter{\expandafter
3176 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\xINT_iiLCMof#3^}}%
3177 }%
3178 \def\xINT_expr_func_max #1#2#3%
3179 {%
3180 \expandafter #1\expandafter #2\expandafter{\expandafter
3181 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\xINT_Maxof#3^}}%
3182 }%
3183 \def\xINT_iiexpr_func_max #1#2#3%
3184 {%
3185 \expandafter #1\expandafter #2\expandafter{\expandafter
3186 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\xINT_iiMaxof#3^}}%
3187 }%
3188 \def\xINT_flexpr_func_max #1#2#3%
3189 {%
3190 \expandafter #1\expandafter #2\expandafter{\expandafter
3191 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\xINTinFloatMaxof#3^}}%
3192 }%
3193 \def\xINT_expr_func_min #1#2#3%
3194 {%
3195 \expandafter #1\expandafter #2\expandafter{\expandafter
3196 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\xINT_Minof#3^}}%
3197 }%
3198 \def\xINT_iiexpr_func_min #1#2#3%
3199 {%
3200 \expandafter #1\expandafter #2\expandafter{\expandafter
3201 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\xINT_iiMinof#3^}}%
3202 }%
3203 \def\xINT_flexpr_func_min #1#2#3%
3204 {%

```

```

3205 \expandafter #1\expandafter #2\expandafter{\expandafter
3206 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINTinFloatMinof#3^}}%
3207 }%
3208 \expandafter
3209 \def\csname XINT_expr_func_+\endcsname #1#2#3%
3210 {%
3211 \expandafter #1\expandafter #2\expandafter{\expandafter
3212 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_Sum#3^}}%
3213 }%
3214 \expandafter
3215 \def\csname XINT_flexpr_func_+\endcsname #1#2#3%
3216 {%
3217 \expandafter #1\expandafter #2\expandafter{\expandafter
3218 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINTinFloatSum#3^}}%
3219 }%
3220 \expandafter
3221 \def\csname XINT_iiexpr_func_+\endcsname #1#2#3%
3222 {%
3223 \expandafter #1\expandafter #2\expandafter{\expandafter
3224 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_iiSum#3^}}%
3225 }%
3226 \expandafter
3227 \def\csname XINT_expr_func_*\endcsname #1#2#3%
3228 {%
3229 \expandafter #1\expandafter #2\expandafter{\expandafter
3230 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_Prd#3^}}%
3231 }%
3232 \expandafter
3233 \def\csname XINT_flexpr_func_*\endcsname #1#2#3%
3234 {%
3235 \expandafter #1\expandafter #2\expandafter{\expandafter
3236 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINTinFloatPrd#3^}}%
3237 }%
3238 \expandafter
3239 \def\csname XINT_iiexpr_func_*\endcsname #1#2#3%
3240 {%
3241 \expandafter #1\expandafter #2\expandafter{\expandafter
3242 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_iiPrd#3^}}%
3243 }%
3244 \def\XINT_expr_func_all #1#2#3%
3245 {%
3246 \expandafter #1\expandafter #2\expandafter{\expandafter
3247 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_ANDof#3^}}%
3248 }%
3249 \let\XINT_flexpr_func_all\XINT_expr_func_all
3250 \let\XINT_iiexpr_func_all\XINT_expr_func_all
3251 \def\XINT_expr_func_any #1#2#3%
3252 {%
3253 \expandafter #1\expandafter #2\expandafter{\expandafter
3254 {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_ORof#3^}}%
3255 }%
3256 \let\XINT_flexpr_func_any\XINT_expr_func_any

```

```

3257 \let\XINT_iiexpr_func_any\XINT_expr_func_any
3258 \def\XINT_expr_func_xor #1#2#3%
3259 {%
3260   \expandafter #1\expandafter #2\expandafter{\expandafter
3261     {\romannumeral`&&\XINT:NEhook:f:from:delim:u\XINT_XORof#3^}}%
3262 }%
3263 \let\XINT_flexpr_func_xor\XINT_expr_func_xor
3264 \let\XINT_iiexpr_func_xor\XINT_expr_func_xor
3265 \def\XINT_expr_func_len #1#2#3%
3266 {%
3267   \expandafter#1\expandafter#2\expandafter{\expandafter{%
3268     \romannumeral`&&\XINT:NEhook:f:noeval:from:braced:u\xintLength#3^%
3269   }}%
3270 }%
3271 \let\XINT_flexpr_func_len \XINT_expr_func_len
3272 \let\XINT_iiexpr_func_len \XINT_expr_func_len
3273 \def\XINT_expr_func_first #1#2#3%
3274 {%
3275   \expandafter #1\expandafter #2\expandafter{%
3276     \romannumeral`&&\XINT:NEhook:f:noeval:from:braced:u\xintFirstOne#3^%
3277   }%
3278 }%
3279 \let\XINT_flexpr_func_first\XINT_expr_func_first
3280 \let\XINT_iiexpr_func_first\XINT_expr_func_first
3281 \def\XINT_expr_func_last #1#2#3%
3282 {%
3283   \expandafter #1\expandafter #2\expandafter{%
3284     \romannumeral`&&\XINT:NEhook:f:noeval:from:braced:u\xintLastOne#3^%
3285   }%
3286 }%
3287 \let\XINT_flexpr_func_last\XINT_expr_func_last
3288 \let\XINT_iiexpr_func_last\XINT_expr_func_last
3289 \def\XINT_expr_func_reversed #1#2#3%
3290 {%
3291   \expandafter #1\expandafter #2\expandafter{%
3292     \romannumeral`&&\XINT:NEhook:f:reverse\XINT_expr_reverse
3293     #3^^#3\xint:\xint:\xint:\xint:
3294     \xint:\xint:\xint:\xint:\xint_bye
3295   }%
3296 }%
3297 \def\XINT_expr_reverse #1#2%
3298 {%
3299   \if ^\noexpand#2%
3300     \expandafter\XINT_expr_reverse:_one_or_none\string#1.%
3301   \else
3302     \expandafter\XINT_expr_reverse:_at_least_two
3303   \fi
3304 }%
3305 \def\XINT_expr_reverse:_at_least_two #1^^{\XINT_revwbr_loop {}}%
3306 \def\XINT_expr_reverse:_one_or_none #1%
3307 {%
3308   \if #1\bgroup\xint_dothis\XINT_expr_reverse:_nutple\fi

```

```

3309 \if #1^\xint_dothis\XINT_expr_reverse:_nil\fi
3310 \xint_orthat\XINT_expr_reverse:_leaf
3311 }%
3312 \edef\XINT_expr_reverse:_nil #1\xint_bye{\noexpand\fi\space}%
3313 \def\XINT_expr_reverse:_leaf#1\fi #2\xint:#3\xint_bye{\fi\xint_gob_andstop_i#2}%
3314 \def\XINT_expr_reverse:_nutple%
3315 {%
3316 \expandafter\XINT_expr_reverse:_nutple_a\expandafter{\string}%
3317 }%
3318 \def\XINT_expr_reverse:_nutple_a #1^#2\xint:#3\xint_bye
3319 {%
3320 \fi\expandafter
3321 {\romannumeral0\XINT_revwbr_loop{ }#2\xint:#3\xint_bye}%
3322 }%
3323 \let\XINT_flexpr_func_reversed\XINT_expr_func_reversed
3324 \let\XINT_iiexpr_func_reversed\XINT_expr_func_reversed
3325 \def\XINT_expr_func_if #1#2#3%
3326 {%
3327 \expandafter #1\expandafter #2\expandafter{%
3328 \romannumeral`&&\XINT:NEhook:branch{\romannumeral`&&\xintiifNotZero #3}}%
3329 }%
3330 \let\XINT_flexpr_func_if\XINT_expr_func_if
3331 \let\XINT_iiexpr_func_if\XINT_expr_func_if
3332 \def\XINT_expr_func_ifint #1#2#3%
3333 {%
3334 \expandafter #1\expandafter #2\expandafter{%
3335 \romannumeral`&&\XINT:NEhook:branch{\romannumeral`&&\xintifInt #3}}%
3336 }%
3337 \let\XINT_iiexpr_func_ifint\XINT_expr_func_ifint
3338 \def\XINT_flexpr_func_ifint #1#2#3%
3339 {%
3340 \expandafter #1\expandafter #2\expandafter{%
3341 \romannumeral`&&\XINT:NEhook:branch{\romannumeral`&&\xintifFloatInt #3}}%
3342 }%
3343 \def\XINT_expr_func_ifone #1#2#3%
3344 {%
3345 \expandafter #1\expandafter #2\expandafter{%
3346 \romannumeral`&&\XINT:NEhook:branch{\romannumeral`&&\xintifOne #3}}%
3347 }%
3348 \let\XINT_flexpr_func_ifone\XINT_expr_func_ifone
3349 \def\XINT_iiexpr_func_ifone #1#2#3%
3350 {%
3351 \expandafter #1\expandafter #2\expandafter{%
3352 \romannumeral`&&\XINT:NEhook:branch{\romannumeral`&&\xintiifOne #3}}%
3353 }%
3354 \def\XINT_expr_func_ifsgn #1#2#3%
3355 {%
3356 \expandafter #1\expandafter #2\expandafter{%
3357 \romannumeral`&&\XINT:NEhook:branch{\romannumeral`&&\xintiifSgn #3}}%
3358 }%
3359 \let\XINT_flexpr_func_ifsgn\XINT_expr_func_ifsgn
3360 \let\XINT_iiexpr_func_ifsgn\XINT_expr_func_ifsgn

```

```

3361 \def\XINT_expr_func_nuple #1#2#3{#1#2{{#3}}}%
3362 \let\XINT_flexpr_func_nuple\XINT_expr_func_nuple
3363 \let\XINT_iiexpr_func_nuple\XINT_expr_func_nuple
3364 \def\XINT_expr_func_unpack #1#2#3%
3365   {\expandafter#1\expandafter#2\romannumeral0\XINT:NEhook:unpack}%
3366 \let\XINT_flexpr_func_unpack\XINT_expr_func_unpack
3367 \let\XINT_iiexpr_func_unpack\XINT_expr_func_unpack
3368 \def\XINT_expr_func_flat #1#2#3%
3369 {%
3370   \expandafter#1\expandafter#2\expanded
3371   \XINT:NEhook:x:flatten\XINT:expr:flatten
3372 }%
3373 \let\XINT_flexpr_func_flat\XINT_expr_func_flat
3374 \let\XINT_iiexpr_func_flat\XINT_expr_func_flat
3375 \let\XINT:NEhook:x:flatten\empty
3376 \def\XINT_expr_func_zip #1#2#3%
3377 {%
3378   \expandafter#1\expandafter#2\romannumeral`&&@%
3379   \XINT:NEhook:x:zip\XINT:expr:zip
3380 }%
3381 \let\XINT_flexpr_func_zip\XINT_expr_func_zip
3382 \let\XINT_iiexpr_func_zip\XINT_expr_func_zip
3383 \let\XINT:NEhook:x:zip\empty
3384 \def\XINT:expr:zip#1{\expandafter{\expanded\XINT_zip_A#1\xint_bye\xint_bye}}%

```

11.29 User declared functions

It is possible that the author actually does understand at this time the `\xintNewExpr`/`\xintdeffunc` refactored code and mechanisms for the first time since 2014: past evolutions such as the 2018 1.3 refactoring were done a bit in the fog (although they did accomplish a crucial step).

The 1.4 version of function and macro definitions is much more powerful than 1.3 one. But the mechanisms such as «omit», «abort» and «break()» in `iter()` et al. can't be translated into much else than their actual code when they potentially have to apply to non-numeric only context. The 1.4 `\xintdeffunc` is thus apparently able to digest them but its pre-parsing benefits are limited compared to simply assigning such parts of an expression to a mock-function created by `\xintNewFunction` (which creates simply a TeX macro from its substitution expression in macro parameters and add syntactic sugar to let it appear to `\xintexpr` as a genuine «function» although nothing of the syntax has really been pre-parsed.)

At 1.4 fetching the expression up to final semi-colon is done using `\XINT_expr_fetch_to_semicolon`, hence semi-colons arising in the syntax do not need to be hidden inside braces.

11.29.1	<code>\xintdeffunc</code> , <code>\xintdefiifunc</code> , <code>\xintdeffloatfunc</code>	398
11.29.2	<code>\xintdefufunc</code> , <code>\xintdefiiufunc</code> , <code>\xintdeffloatufunc</code>	401
11.29.3	<code>\xintunassignexprfunc</code> , <code>\xintunassigniiexprfunc</code> , <code>\xintunassignfloatexprfunc</code>	402
11.29.4	<code>\xintNewFunction</code>	403
11.29.5	Mysterious stuff	404
11.29.6	<code>\XINT_expr_redefinemacros</code>	416
11.29.7	<code>\xintNewExpr</code> , <code>\xintNewIExpr</code> , <code>\xintNewFloatExpr</code> , <code>\xintNewIIExpr</code>	417
11.29.8	<code>\ifxintexprsafecatcodes</code> , <code>\xintexprSafeCatcodes</code> , <code>\xintexprRestoreCatcodes</code>	419

11.29.1 `\xintdeffunc`, `\xintdefiifunc`, `\xintdeffloatfunc`

1.2c (2015/11/12).

Note: it is possible to have same name assigned both to a variable and a function: things such as `add(f(f), f=1..10)` are possible.

1.2c (2015/11/13).

Function names first expanded then detokenized and cleaned of spaces.

1.2e (2015/11/21).

No `\detokenize` anymore on the function names. And `#1(#2)#3=#4` parameter pattern to avoid to have to worry if a `:` is there and it is active.

1.2f (2016/02/22).

La macro associée à la fonction ne débute plus par un `\romannumeral`, car de toute façon elle est pour emploi dans `\csname..\endcsname`.

1.2f (2016/03/08).

Comma separated expressions allowed (formerly this required using parenthesis `\xintdeffunc foo(x,..):=(.., .., ..)`;

1.3c (2018/06/17).

Usage of `\xintexprSafeCatcodes` to be compatible with an active semi-colon at time of use; the colon was not a problem (see `##3`) already.

1.3e (??).

`\xintdefefunc` variant added for functions which will expand completely if used with numeric arguments in other function definitions. They can't be used for recursive definitions.

1.4 (2020/01/10).

Multi-letter variables can be used (with no prior declaration)

1.4 (2020/01/11).

The new internal data model has caused many worries initially (such as whether to allow functions with «ople» outputs in contrast to «numbers» or «nutples») but in the end all is simpler again and the refactoring of `?` and `??` in function definitions allows to fuse inert functions (allowing recursive definitions) and expanding functions (expanding completely if with numeric arguments) into a single entity.

Thus the 1.3e `\xintdefefunc`, `\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.

```

3385 \def\xINT_tmpa #1#2#3#4#5%
3386 {%
3387   \def #1##1(##2)##3={%
3388     \edef\xINT_deffunc_tmpa {##1}%
3389     \edef\xINT_deffunc_tmpa {\xint_zapspace_o \xINT_deffunc_tmpa}%
3390     \def\xINT_deffunc_tmpb {0}%
3391     \edef\xINT_deffunc_tmpd {##2}%
3392     \edef\xINT_deffunc_tmpd {\xint_zapspace_o\xINT_deffunc_tmpd}%
3393     \def\xINT_deffunc_tmpe {0}%
3394     \expandafter#5\romannumeral\xINT_expr_fetch_to_semicolon
3395   }% end of \xintdeffunc_a definition
3396   \def#5##1{%
3397     \def\xINT_deffunc_tmpe{##1}%

```

```

3398 \ifnum\xintLength:f:csv{\XINT_deffunc_tmpd}>\xint_c_
3399 \xintFor #####1 in {\XINT_deffunc_tmpd}\do
3400 {%
3401 \xintifForFirst{\let\XINT_deffunc_tmpd\empty}{}%
3402 \def\XINT_deffunc_tmpf{#####1}%
3403 \if*\xintFirstItem{#####1}%
3404 \xintifForLast
3405 {%
3406 \def\XINT_deffunc_tmpe{1}%
3407 \edef\XINT_deffunc_tmpf{\xintTrim{1}{#####1}}%
3408 }%
3409 {%
3410 \edef\XINT_deffunc_tmpf{\xintTrim{1}{#####1}}%
3411 \xintMessage{xintexpr}{Error}
3412 {Only the last positional argument can be variadic. Trimmed #####1 to
3413 \XINT_deffunc_tmpf}%
3414 }%
3415 \fi
3416 \XINT_expr_makedummy{\XINT_deffunc_tmpf}%
3417 \edef\XINT_deffunc_tmpd{\XINT_deffunc_tmpd{\XINT_deffunc_tmpf}}%
3418 \edef\XINT_deffunc_tmpe {\the\numexpr\XINT_deffunc_tmpe+\xint_c_i}%
3419 \edef\XINT_deffunc_tmpe {subs(\unexpanded\expandafter{\XINT_deffunc_tmpe},%
3420 \XINT_deffunc_tmpe=#####\XINT_deffunc_tmpe)}%
3421 }%
3422 \fi

```

Place holder for comments. Logic at 1.4 is simplified here compared to earlier releases.

```

3423 \ifcase\XINT_deffunc_tmpe\space
3424 \expandafter\XINT_expr_defuserfunc_none\csname
3425 \else
3426 \expandafter\XINT_expr_defuserfunc\csname
3427 \fi
3428 \XINT_#2_func_\XINT_deffunc_tmpe\expandafter\endcsname
3429 \csname XINT_#2_userfunc_\XINT_deffunc_tmpe\expandafter\endcsname
3430 \expandafter{\XINT_deffunc_tmpe}{#2}%
3431 \expandafter#3\csname XINT_#2_userfunc_\XINT_deffunc_tmpe\endcsname
3432 [\XINT_deffunc_tmpe]{\XINT_deffunc_tmpe}%
3433 \ifxintverbose\xintMessage {xintexpr}{Info}
3434 {Function \XINT_deffunc_tmpe\space for \string\xint #4 parser
3435 associated to \string\XINT_#2_userfunc_\XINT_deffunc_tmpe\space
3436 with \ifxintglobaldefs global \fi meaning \expandafter\meaning
3437 \csname XINT_#2_userfunc_\XINT_deffunc_tmpe\endcsname}%
3438 \fi
3439 \xintFor* #####1 in {\XINT_deffunc_tmpd}:{\xintrestorevariablesilently{#####1}}%
3440 \xintexprRestoreCatcodes
3441 }% end of \xintdeffunc_b definition
3442 }%
3443 \def\xintdeffunc {\xintexprSafeCatcodes\xintdeffunc_a}%
3444 \def\xintdefiifunc {\xintexprSafeCatcodes\xintdefiifunc_a}%
3445 \def\xintdeffloatfunc {\xintexprSafeCatcodes\xintdeffloatfunc_a}%
3446 \XINT_tmpe\xintdeffunc_a {expr} \XINT_NewFunc {expr}\xintdeffunc_b
3447 \XINT_tmpe\xintdefiifunc_a {iiexpr}\XINT_NewIIFunc {iiexpr}\xintdefiifunc_b
3448 \XINT_tmpe\xintdeffloatfunc_a{flexpr}\XINT_NewFloatFunc{floatexpr}\xintdeffloatfunc_b

```



```

3449 \def\XINT_expr_defuserfunc_none #1#2#3#4%
3450 {%
3451   \XINT_global
3452   \def #1##1##2##3%
3453   {%
3454     \expandafter##1\expandafter##2\expanded{%
3455       {\XINT:NEhook:usernoargfunc\csname XINT_#4_userfunc_#3\endcsname}%
3456     }%
3457   }%
3458 }%
3459 \let\XINT:NEhook:usernoargfunc \empty
3460 \def\XINT_expr_defuserfunc #1#2#3#4%
3461 {%
3462   \if0\XINT_deffunc_tmpe
3463   \XINT_global
3464   \def #1##1##2##3%
3465   {%
3466     \expandafter ##1\expandafter##2\expanded\bgroup{\iffalse}\fi
3467     \XINT:NEhook:userfunc{XINT_#4_userfunc_#3}##2##3%
3468   }%
3469   \else
3470   \def #1##1{%
3471     \XINT_global\def #1####1####2####3%
3472     {%
3473       \expandafter ####1\expandafter####2\expanded\bgroup{\iffalse}\fi
3474       \XINT:NEhook:userfunc:argv{##1}{XINT_#4_userfunc_#3}##2####3%
3475     }%\expandafter#1\expandafter{\the\numexpr\XINT_deffunc_tmpe-1}%
3476     \fi
3477   }%
3478 \def\XINT:NEhook:userfunc #1#2#3{#2#3\iffalse{\fi}}%
3479 \def\XINT:NEhook:userfunc:argv #1#2#3#4%
3480   {\expandafter#3\expanded{\xintKeep{#1}{#4}{\xintTrim{#1}{#4}}\iffalse{\fi}}%
3481 \let\xintdefefunc\xintdeffunc
3482 \let\xintdefiifunc\xintdefiifunc
3483 \let\xintdeffloatefunc\xintdeffloatfunc

```

11.29.2 \xintdefefunc, \xintdefiifunc, \xintdeffloatufunc

1.4

```

3484 \def\XINT_tmpa #1#2#3#4#5#6%
3485 {%
3486   \def #1##1(##2)##3={%
3487     \edef\XINT_defufunc_tmpa {##1}%
3488     \edef\XINT_defufunc_tmpa {\xint_zapspaces_o \XINT_defufunc_tmpa}%
3489     \edef\XINT_defufunc_tmpd {##2}%
3490     \edef\XINT_defufunc_tmpd {\xint_zapspaces_o\XINT_defufunc_tmpd}%
3491     \expandafter#5\romannumerals\XINT_expr_fetch_to_semicolon
3492   }% end of \xint_defufunc_a
3493   \def#5##1{%
3494     \def\XINT_defufunc_tmpc{##1}%
3495     \ifnum\xintLength:f:csv{\XINT_defufunc_tmpd}=\xint_c_i
3496       \expandafter#6%

```

```

3497 \else
3498 \xintMessage {xintexpr}{ERROR}
3499     {Universal functions must be functions of one argument only,
3500     but the declaration of \XINT_defufunc_tmpa\space
3501     has \xintLength:f:csv{\XINT_defufunc_tmpd} of them. Cancelled.}%
3502 \xintexprRestoreCatcodes
3503 \fi
3504 }% end of \xint_defufunc_b
3505 \def #6{%
3506 \XINT_expr_makedummy{\XINT_defufunc_tmpd}%
3507 \edef\XINT_defufunc_tmpc {subs(\unexpanded\expandafter{\XINT_defufunc_tmpc},%
3508     \XINT_defufunc_tmpd=#####1)}%
3509 \expandafter\XINT_expr_defuserufunc
3510 \csname XINT_#2_func_\XINT_defufunc_tmpa\expandafter\endcsname
3511 \csname XINT_#2_userufunc_\XINT_defufunc_tmpa\expandafter\endcsname
3512 \expandafter{\XINT_defufunc_tmpa}{#2}%
3513 \expandafter#3\csname XINT_#2_userufunc_\XINT_defufunc_tmpa\endcsname
3514     [1]{\XINT_defufunc_tmpc}%
3515 \ifxintverbose\xintMessage {xintexpr}{Info}
3516     {Universal function \XINT_defufunc_tmpa\space for \string\xint #4 parser
3517     associated to \string\XINT_#2_userufunc_\XINT_defufunc_tmpa\space
3518     with \ifxintglobaldefs global \fi meaning \expandafter\meaning
3519     \csname XINT_#2_userufunc_\XINT_defufunc_tmpa\endcsname}%
3520 \fi
3521 }% end of \xint_defufunc_c
3522 }%
3523 \def\xintdefufunc     {\xintexprSafeCatcodes\xintdefufunc_a}%
3524 \def\xintdefiiufunc  {\xintexprSafeCatcodes\xintdefiiufunc_a}%
3525 \def\xintdeffloatufunc {\xintexprSafeCatcodes\xintdeffloatufunc_a}%
3526 \XINT_tmpa\xintdefufunc_a {expr} \XINT_NewFunc {expr}%
3527     \xintdefufunc_b\xintdefufunc_c
3528 \XINT_tmpa\xintdefiiufunc_a {iiexpr}\XINT_NewIIFunc {iiexpr}%
3529     \xintdefiiufunc_b\xintdefiiufunc_c
3530 \XINT_tmpa\xintdeffloatufunc_a{fexpr}\XINT_NewFloatFunc{floatexpr}%
3531     \xintdeffloatufunc_b\xintdeffloatufunc_c
3532 \def\XINT_expr_defuserufunc #1#2#3#4%
3533 {%
3534     \XINT_global
3535     \def #1##1##2%##3%
3536     {%
3537         \expandafter ##1\expandafter##2\expanded
3538         \XINT:NEhook:userufunc{XINT_#4_userufunc_#3}##2%##3%
3539     }%
3540 }%
3541 \def\XINT:NEhook:userufunc #1{\XINT:expr:mapwithin}%

```

11.29.3 `\xintunassignexprfunc`, `\xintunassigniiexprfunc`, `\xintunassignfloatexprfunc`

See the [\xintunassignvar](#) for the embarrassing explanations why I had not done that earlier. A bit lazy here, no warning if undefining something not defined, and attention no precaution respective built-in functions.

```

3542 \def\XINT_tmpa #1{\expandafter\def\csname xintunassign#1func\endcsname ##1{%
3543     \edef\XINT_unfunc_tmpa{##1}%

```

```

3544 \edef\XINT_unfunc_tmpa {\xint_zapspace_o\XINT_unfunc_tmpa}%
3545 \XINT_global\expandafter
3546 \let\csname XINT_#1_func_\XINT_unfunc_tmpa\endcsname\xint_undefined
3547 \XINT_global\expandafter
3548 \let\csname XINT_#1_userfunc_\XINT_unfunc_tmpa\endcsname\xint_undefined
3549 \XINT_global\expandafter
3550 \let\csname XINT_#1_userufunc_\XINT_unfunc_tmpa\endcsname\xint_undefined
3551 \ifxintverbose\xintMessage {xintexpr}{Info}
3552 {Function \XINT_unfunc_tmpa\space for \string\xint #1 parser now
3553 \ifxintglobaldefs globally \fi undefined.}%
3554 \fi}}%
3555 \XINT_tmpa{expr}\XINT_tmpa{iiexpr}\XINT_tmpa{floatexpr}%

```

11.29.4 \xintNewFunction

1.2h (2016/11/20). Syntax is `\xintNewFunction<name>[nb of arguments]{expression with #1, #2, ... as in \xintNewExpr}`. This defines a function for all three parsers but the expression parsing is delayed until function execution. Hence the expression admits all constructs, contrarily to `\xintNewExpr` or `\xintdeffunc`.

As the letters used for variables in `\xintdeffunc`, #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.

```

3556 \def\xintNewFunction #1#2[#3]#4%
3557 {%
3558 \edef\XINT_newfunc_tmpa {#1}%
3559 \edef\XINT_newfunc_tmpa {\xint_zapspace_o \XINT_newfunc_tmpa}%
3560 \def\XINT_newfunc_tmpb ##1##2##3##4##5##6##7##8##9{#4}%
3561 \begingroup
3562 \ifcase #3\relax
3563 \toks0{}%
3564 \or \toks0{##1}%
3565 \or \toks0{##1##2}%
3566 \or \toks0{##1##2##3}%
3567 \or \toks0{##1##2##3##4}%
3568 \or \toks0{##1##2##3##4##5}%
3569 \or \toks0{##1##2##3##4##5##6}%
3570 \or \toks0{##1##2##3##4##5##6##7}%
3571 \or \toks0{##1##2##3##4##5##6##7##8}%
3572 \else \toks0{##1##2##3##4##5##6##7##8##9}%
3573 \fi
3574 \expandafter
3575 \endgroup\expandafter
3576 \XINT_global\expandafter
3577 \def\csname XINT_expr_macrofunc_\XINT_newfunc_tmpa\expandafter\endcsname
3578 \the\toks0\expandafter{\XINT_newfunc_tmpb
3579 {\XINTfstop.{{##1}}}\XINTfstop.{{##2}}}\XINTfstop.{{##3}}}%
3580 {\XINTfstop.{{##4}}}\XINTfstop.{{##5}}}\XINTfstop.{{##6}}}%
3581 {\XINTfstop.{{##7}}}\XINTfstop.{{##8}}}\XINTfstop.{{##9}}}%
3582 \expandafter\XINT_expr_newfunction
3583 \csname XINT_expr_func_\XINT_newfunc_tmpa\expandafter\endcsname
3584 \expandafter{\XINT_newfunc_tmpa}\xintbareeval

```

```

3585 \expandafter\XINT_expr_newfunction
3586   \csname XINT_iiexpr_func_\XINT_newfunc_tmpa\expandafter\endcsname
3587   \expandafter{\XINT_newfunc_tmpa}\xintbareiieval
3588 \expandafter\XINT_expr_newfunction
3589   \csname XINT_flexpr_func_\XINT_newfunc_tmpa\expandafter\endcsname
3590   \expandafter{\XINT_newfunc_tmpa}\xintbarefloateval
3591 \ifxintverbose
3592   \xintMessage {xintexpr}{Info}
3593     {Function \XINT_newfunc_tmpa\space for the expression parsers is
3594     associated to \string\XINT_expr_macrofunc_\XINT_newfunc_tmpa\space
3595     with \ifxintglobaldefs global \fi meaning \expandafter\meaning
3596     \csname XINT_expr_macrofunc_\XINT_newfunc_tmpa\endcsname}%
3597 \fi
3598 }%
3599 \def\XINT_expr_newfunction #1#2#3%
3600 {%
3601   \XINT_global
3602   \def#1##1##2##3%
3603     {\expandafter ##1\expandafter ##2%
3604     \romannumeral0\XINT:NEhook:macrofunc
3605     #3{\csname XINT_expr_macrofunc_#2\endcsname##3}\relax
3606     }%
3607 }%
3608 \let\XINT:NEhook:macrofunc\empty

```

11.29.5 Mysterious stuff

There was an `\xintNewExpr` already in 1.07 from May 2013, which was modified in September 2013 to work with the `#` macro parameter character, and then refactored into a more powerful version in June 2014 for 1.1 release of 2014/10/28.

It is always too soon to try to comment and explain. In brief, this attempts to hack into the *purely numeric* `\xintexpr` parsers to transform them into *symbolic* parsers, allowing to do once and for all the parsing job and inherit a gigantic nested macro. Originally only f-expandable nesting. The initial motivation was that the `\csname` encapsulation impacted the string pool memory. Later this work proved to be the basis to provide support for implementing user-defined functions and it is now its main purpose.

Deep refactorings happened at 1.3 and 1.4.

At 1.3 the crucial idea of the «hook» macros was introduced, reducing considerably the preparatory work done by `\xintNewExpr`.

At 1.4 further considerable simplifications happened, and it is possible that the author currently does at long last understand the code!

The 1.3 code had serious complications with trying to identify would-be «list» arguments, distinguishing them from «single» arguments (things like parsing `#2+[[#1..[#3]..#4][#5:#6]]*#7` and convert it to a single nested f-expandable macro...)

The conversion at 1.4 is both more powerful and simpler, due in part to the new storage model which from `\csname` encapsulated comma separated values up to 1.3f became simply a braced list of braced values, and also crucially due to the possibilities opened up by usage of `\expanded` primitive.

```

3609 \catcode`~ 12
3610 \def\XINT:NE:hashtilde#1~#2#3\relax{\unless\if !#21\fi}%
3611 \def\XINT:NE:hashash#1{%
3612 \def\XINT:NE:hashash##1#1##2##3\relax{\unless\if !##21\fi}%

```

```

3613 }\expandafter\XINT:NE:hashash\string#%
3614 \def\XINT:NE:unpack #1{%
3615 \def\XINT:NE:unpack ##1%
3616 {%
3617   \if0\XINT:NE:haskilde ##1~!\relax
3618     \XINT:NE:hashash ##1#1!\relax 0\else
3619     \expandafter\XINT:NE:unpack:p\fi
3620   \xint_stop_atfirstofone{##1}%
3621 }%\expandafter\XINT:NE:unpack\string#%
3622 \def\XINT:NE:unpack:p#1#2%
3623   {\romannumeral0~expandafter~xint_stop_atfirstofone~expanded{#2}}}%
3624 \def\XINT:NE:f:one:from:one #1{%
3625 \def\XINT:NE:f:one:from:one ##1%
3626 {%
3627   \if0\XINT:NE:haskilde ##1~!\relax
3628     \XINT:NE:hashash ##1#1!\relax 0\else
3629     \xint_dothis\XINT:NE:f:one:from:one_a\fi
3630   \xint_orthat\XINT:NE:f:one:from:one_b
3631   ##1&&A%
3632 }%\expandafter\XINT:NE:f:one:from:one\string#%
3633 \def\XINT:NE:f:one:from:one_a\romannumeral`&&@#1#2&&A%
3634 {%
3635   \expandafter{\detokenize{\expandafter#1}#2}%
3636 }%
3637 \def\XINT:NE:f:one:from:one_b#1{%
3638 \def\XINT:NE:f:one:from:one_b\romannumeral`&&@##1##2&&A%
3639 {%
3640   \expandafter{\romannumeral`&&@%
3641     \if0\XINT:NE:haskilde ##2~!\relax
3642     \XINT:NE:hashash ##2#1!\relax 0\else
3643     \expandafter\string\fi
3644     ##1{##2}}}%
3645 }%\expandafter\XINT:NE:f:one:from:one_b\string#%
3646 \def\XINT:NE:f:one:from:one:direct #1#2{\XINT:NE:f:one:from:one:direct_a #2&&A{#1}}%
3647 \def\XINT:NE:f:one:from:one:direct_a #1#2&&A#3%
3648 {%
3649   \if ##1\xint_dothis {\detokenize{#3}}\fi
3650   \if ~#1\xint_dothis {\detokenize{#3}}\fi
3651   \xint_orthat {#3}{#1#2}%
3652 }%
3653 \def\XINT:NE:f:one:from:two #1{%
3654 \def\XINT:NE:f:one:from:two ##1%
3655 {%
3656   \if0\XINT:NE:haskilde ##1~!\relax
3657     \XINT:NE:hashash ##1#1!\relax 0\else
3658     \xint_dothis\XINT:NE:f:one:from:two_a\fi
3659   \xint_orthat\XINT:NE:f:one:from:two_b ##1&&A%
3660 }%\expandafter\XINT:NE:f:one:from:two\string#%
3661 \def\XINT:NE:f:one:from:two_a\romannumeral`&&@#1#2&&A%
3662 {%
3663   \expandafter{\detokenize{\expandafter#1\expanded}{#2}}}%
3664 }%

```

```

3665 \def\XINT:NE:f:one:from:two_b#1{%
3666 \def\XINT:NE:f:one:from:two_b\romannumeral`&&@##1##2##3&&A%
3667 {%
3668   \expandafter{\romannumeral`&&@%
3669     \if0\XINT:NE:hasilde ##2##3~!\relax
3670       \XINT:NE:hashash ##2##3#!\relax 0\else
3671     \expandafter\string\fi
3672     ##1{##2}{##3}}%
3673 }}\expandafter\XINT:NE:f:one:from:two_b\string#%
3674 \def\XINT:NE:f:one:from:two:direct #1#2#3{\XINT:NE:two_fork #2&&A#3&&A#1{#2}{#3}}%
3675 \def\XINT:NE:two_fork #1#2&&A#3#4&&A{\XINT:NE:two_fork_nn#1#3}%
3676 \def\XINT:NE:two_fork_nn #1#2%
3677 {%
3678   \if #1#\xint_dothis\string\fi
3679   \if #1~\xint_dothis\string\fi
3680   \if #2#\xint_dothis\string\fi
3681   \if #2~\xint_dothis\string\fi
3682   \xint_orthat{}}%
3683 }%
3684 \def\XINT:NE:f:one:and:opt:direct#1{%
3685 \def\XINT:NE:f:one:and:opt:direct##1!%
3686 {%
3687   \if0\XINT:NE:hasilde ##1~!\relax
3688     \XINT:NE:hashash ##1#!\relax 0\else
3689     \xint_dothis\XINT:NE:f:one:and:opt_a\fi
3690     \xint_orthat\XINT:NE:f:one:and:opt_b ##1&&A%
3691 }}\expandafter\XINT:NE:f:one:and:opt:direct\string#%
3692 \def\XINT:NE:f:one:and:opt_a #1#2&&A#3#4%
3693 {%
3694   \detokenize{\romannumeral-`0\expandafter#1\expanded{#2}$XINT_expr_exclam#3#4}%$
3695 }%
3696 \def\XINT:NE:f:one:and:opt_b\XINT:expr:f:one:and:opt #1#2#3&&A#4#5%
3697 {%
3698   \if\relax#3\relax\expandafter\xint_firstoftwo\else
3699     \expandafter\xint_secondoftwo\fi
3700   {\XINT:NE:f:one:from:one:direct#4}%
3701   {\expandafter\XINT:NE:f:one:withopttoone\expandafter#5%
3702     \expanded{{\XINT:NE:f:one:from:one:direct\xintNum{#2}}}}%
3703   {#1}%
3704 }%
3705 \def\XINT:NE:f:one:withopttoone#1#2#3{\XINT:NE:two_fork #2&&A#3&&A#1[#2]{#3}}%
3706 \def\XINT:NE:f:tacitzeroifone:direct#1{%
3707 \def\XINT:NE:f:tacitzeroifone:direct##1!%
3708 {%
3709   \if0\XINT:NE:hasilde ##1~!\relax
3710     \XINT:NE:hashash ##1#!\relax 0\else
3711     \xint_dothis\XINT:NE:f:one:and:opt_a\fi
3712     \xint_orthat\XINT:NE:f:tacitzeroifone_b ##1&&A%
3713 }}\expandafter\XINT:NE:f:tacitzeroifone:direct\string#%
3714 \def\XINT:NE:f:tacitzeroifone_b\XINT:expr:f:tacitzeroifone #1#2#3&&A#4#5%
3715 {%
3716   \if\relax#3\relax\expandafter\xint_firstoftwo\else

```

```

3717         \expandafter\xint_secondoftwo\fi
3718     {\XINT:NE:f:one:from:two:direct#4{0}}%
3719     {\expandafter\XINT:NE:f:one:from:two:direct\expandafter#5%
3720         \expanded{{\XINT:NE:f:one:from:one:direct\xintNum{#2}}}}%
3721     {#1}%
3722 }%
3723 \def\XINT:NE:f:iitacitzeroifone:direct#1{%
3724 \def\XINT:NE:f:iitacitzeroifone:direct##1!%
3725 {%
3726     \if0\XINT:NE:haskilde ##1~!\relax
3727         \XINT:NE:hashash ##1#1!\relax 0\else
3728         \xint_dothis\XINT:NE:f:iitacitzeroifone_a\fi
3729         \xint_orthat\XINT:NE:f:iitacitzeroifone_b ##1&&A%
3730 }}\expandafter\XINT:NE:f:iitacitzeroifone:direct\string#%
3731 \def\XINT:NE:f:iitacitzeroifone_a #1#2&&A#3%
3732 {%
3733     \detokenize{\romannumeral`-0\expandafter#1\expanded{#2}$XINT_expr_exclam#3}$%
3734 }%
3735 \def\XINT:NE:f:iitacitzeroifone_b\XINT:expr:f:iitacitzeroifone #1#2#3&&A#4%
3736 {%
3737     \if\relax#3\relax\expandafter\xint_firstoftwo\else
3738         \expandafter\xint_secondoftwo\fi
3739     {\XINT:NE:f:one:from:two:direct#4{0}}%
3740     {\XINT:NE:f:one:from:two:direct#4{#2}}%
3741     {#1}%
3742 }%
3743 \def\XINT:NE:x:one:from:two #1#2#3{\XINT:NE:x:one:from:two_fork #2&&A#3&&A#1{#2}{#3}}%
3744 \def\XINT:NE:x:one:from:two_fork #1{%
3745 \def\XINT:NE:x:one:from:two_fork ##1##2&&A##3##4&&A%
3746 {%
3747     \if0\XINT:NE:haskilde ##1##3~!\relax\XINT:NE:hashash ##1##3#1!\relax 0%
3748     \else
3749         \expandafter\XINT:NE:x:one:from:two:p
3750     \fi
3751 }}\expandafter\XINT:NE:x:one:from:two_fork\string#%
3752 \def\XINT:NE:x:one:from:two:p #1#2#3%
3753     {\~expanded{\detokenize{\expandafter#1}\~expanded{{#2}{#3}}}}%
3754 \def\XINT:NE:x:one:from:twoandone #1#2#3{\XINT:NE:x:one:from:twoandone_a #2#3&&A#1{#2}{#3}}%
3755 \def\XINT:NE:x:one:from:twoandone_a #1#2{\XINT:NE:x:one:from:twoandone_fork #1&&A#2&&A}%
3756 \def\XINT:NE:x:one:from:twoandone_fork #1{%
3757 \def\XINT:NE:x:one:from:twoandone_fork ##1##2&&A##3##4&&A##5##6&&A%
3758 {%
3759     \if0\XINT:NE:haskilde ##1##3##5~!\relax\XINT:NE:hashash ##1##3##5#1!\relax 0%
3760     \else
3761         \expandafter\XINT:NE:x:one:from:two:p
3762     \fi
3763 }}\expandafter\XINT:NE:x:one:from:twoandone_fork\string#%
3764 \def\XINT:NE:x:listselsel #1{%
3765 \def\XINT:NE:x:listselsel ##1##2&&%
3766 {%
3767     \if0\expandafter\XINT:NE:haskilde\detokenize{##2}~!\relax
3768         \expandafter\XINT:NE:hashash\detokenize{##2}#1!\relax 0%

```

```

3769 \else
3770 \expandafter\XINT:NE:x:listsels:p
3771 \fi
3772 ##1##2&%
3773 }}\expandafter\XINT:NE:x:listsels:string#%
3774 \def\XINT:NE:x:listsels:p #1#2&(#3%
3775 {%
3776 \detokenize
3777 {%
3778 \expanded{\expandafter#1\expanded{#2$XINT_expr_tab({#3}}\expandafter\empty\empty}%$
3779 }%
3780 }%
3781 \def\XINT:NE:f:reverse #1{%
3782 \def\XINT:NE:f:reverse ##1^%
3783 {%
3784 \if0\expandafter\XINT:NE:hashtilde\detokenize\expandafter{\xint_gobble_i##1}~!\relax
3785 \expandafter\XINT:NE:hashash\detokenize{##1}#1!\relax 0%
3786 \else
3787 \expandafter\XINT:NE:f:reverse:p
3788 \fi
3789 ##1^%
3790 }}\expandafter\XINT:NE:f:reverse:string#%
3791 \def\XINT:NE:f:reverse:p #1^#2\xint_bye
3792 {%
3793 \detokenize
3794 {%
3795 \romannumeral0\expandafter\XINT:expr:f:reverse
3796 \expandafter{\expanded\expandafter{\xint_gobble_i#1}}%
3797 }%
3798 }%
3799 \def\XINT:expr:f:reverse #1%
3800 {%
3801 \XINT_expr_reverse #1^^#1\xint:\xint:\xint:\xint:
3802 \xint:\xint:\xint:\xint:\xint_bye
3803 }%
3804 \def\XINT:NE:f:from:delim:u #1{%
3805 \def\XINT:NE:f:from:delim:u ##1##2^%
3806 {%
3807 \if0\expandafter\XINT:NE:hashtilde\detokenize{##2}~!\relax
3808 \expandafter\XINT:NE:hashash\detokenize{##2}#1!\relax 0%
3809 \expandafter##1%
3810 \else
3811 \xint_afterfi{\XINT:NE:f:from:delim:u:p##1\empty}%
3812 \fi
3813 ##2^%
3814 }}\expandafter\XINT:NE:f:from:delim:u:string#%
3815 \def\XINT:NE:f:from:delim:u:p #1#2^%
3816 {\detokenize{\expandafter#1}~expanded{#2}$XINT_expr_caret}%$
3817 \def\XINT:NE:f:noeval:from:braced:u #1{%
3818 \def\XINT:NE:f:noeval:from:braced:u ##1##2^%
3819 {%
3820 \if0\XINT:NE:hashtilde ##2~!\relax\XINT:NE:hashash ##2#1!\relax 0%

```



```

3821 \else
3822 \expandafter\XINT:NE:f:noeval:from:braced:u:p
3823 \fi
3824 ##1{##2}%
3825 }}\expandafter\XINT:NE:f:noeval:from:braced:u:string#%
3826 \def\XINT:NE:f:noeval:from:braced:u:p #1#2%
3827 {\detokenize{\expandafter#1}~expanded{##2}}}%
3828 \catcode`- 11
3829 \def\XINT:NE:exec_? #1#2%
3830 {%
3831 \XINT:NE:exec_?_b #2&&A#1{#2}%
3832 }%
3833 \def\XINT:NE:exec_?_b #1{%
3834 \def\XINT:NE:exec_?_b ##1&&A%
3835 {%
3836 \if0\XINT:NE:hastilde ##1~!\relax
3837 \XINT:NE:hashash ##1#1!\relax 0%
3838 \xint_dothis\XINT:NE:exec_?:x\fi
3839 \xint_orthat\XINT:NE:exec_?:p
3840 }}\expandafter\XINT:NE:exec_?_b\string#%
3841 \def\XINT:NE:exec_?:x #1#2#3%
3842 {%
3843 \expandafter\XINT_expr_check-_after?\expandafter#1%
3844 \romannumeral`&&@\expandafter\XINT_expr_getnext\romannumeral0\xintiiifnotzero#3%
3845 }%
3846 \def\XINT:NE:exec_?:p #1#2#3#4#5%
3847 {%
3848 \csname XINT_expr_func_*If\expandafter\endcsname
3849 \romannumeral`&&@#2\XINTfstop.{#3},[#4],[#5]}%
3850 }%
3851 \expandafter\def\csname XINT_expr_func_*If\endcsname #1#2#3%
3852 {%
3853 #1#2{~expanded{~xintiiifNotZero#3}}}%
3854 }%
3855 \def\XINT:NE:exec_?? #1#2#3%
3856 {%
3857 \XINT:NE:exec_??_b #2&&A#1{#2}%
3858 }%
3859 \def\XINT:NE:exec_??_b #1{%
3860 \def\XINT:NE:exec_??_b ##1&&A%
3861 {%
3862 \if0\XINT:NE:hastilde ##1~!\relax
3863 \XINT:NE:hashash ##1#1!\relax 0%
3864 \xint_dothis\XINT:NE:exec_??:x\fi
3865 \xint_orthat\XINT:NE:exec_??:p
3866 }}\expandafter\XINT:NE:exec_??_b\string#%
3867 \def\XINT:NE:exec_??:x #1#2#3%
3868 {%
3869 \expandafter\XINT_expr_check-_after?\expandafter#1%
3870 \romannumeral`&&@\expandafter\XINT_expr_getnext\romannumeral0\xintiiifsgn#3%
3871 }%
3872 \def\XINT:NE:exec_??:p #1#2#3#4#5#6%

```

```

3873 {%
3874   \csname XINT_expr_func_*IfSgn\expandafter\endcsname
3875   \romannumeral`&&@#2\XINTfstop.{#3},{#4},{#5},{#6})%
3876 }%
3877 \expandafter\def\csname XINT_expr_func_*IfSgn\endcsname #1#2#3%
3878 {%
3879   #1#2{~expanded{~xintiifSgn#3}}%
3880 }%
3881 \catcode`- 12
3882 \def\XINT:NE:branch #1%
3883 {%
3884   \if0\XINT:NE:hasilde #1~!\relax 0\else
3885     \xint_dothis\XINT:NE:branch_a\fi
3886     \xint_orthat\XINT:NE:branch_b #1&&A%
3887 }%
3888 \def\XINT:NE:branch_a\romannumeral`&&@#1#2&&A%
3889 {%
3890   \expandafter{\detokenize{\expandafter#1\expanded}{#2}}%
3891 }%
3892 \def\XINT:NE:branch_b#1{%
3893 \def\XINT:NE:branch_b\romannumeral`&&@##1##2##3&&A%
3894 {%
3895   \expandafter{\romannumeral`&&@%
3896     \if0\XINT:NE:hasilde ##2~!\relax
3897       \XINT:NE:hashash ##2#1!\relax 0\else
3898       \expandafter\string\fi
3899     ##1{##2}##3}%
3900 }}\expandafter\XINT:NE:branch_b\string#%
3901 \def\XINT:NE:seqx#1{%
3902 \def\XINT:NE:seqx\XINT_allexpr_seqx##1##2%
3903 {%
3904   \if 0\expandafter\XINT:NE:hasilde\detokenize{##2}~!\relax
3905     \expandafter\XINT:NE:hashash \detokenize{##2}#1!\relax 0%
3906   \else
3907     \expandafter\XINT:NE:seqx:p
3908   \fi \XINT_allexpr_seqx{##1}{##2}%
3909 }}\expandafter\XINT:NE:seqx\string#%
3910 \def\XINT:NE:seqx:p\XINT_allexpr_seqx #1#2#3#4%
3911 {%
3912   \expandafter\XINT_expr_put_op_first
3913   \expanded {%
3914     {%
3915       \detokenize
3916       {%
3917         \expanded\bgroup
3918         \expanded
3919         {\unexpanded{\XINT_expr_seq:_b{#1#4\relax $XINT_expr_exclam #3}}%
3920          #2$XINT_expr_caret}%
3921       }%
3922     }%
3923   \expandafter}\romannumeral`&&\XINT_expr_getop
3924 }%

```

```

3925 \def\XINT:NE:opx#1{%
3926 \def\XINT:NE:opx\XINT_allexpr_opx ##1##2##3##4##5##6##7##8%
3927 {%
3928   \if 0\expandafter\XINT:NE:hasilde\detokenize{##4}~!\relax
3929     \expandafter\XINT:NE:hashash \detokenize{##4}#1!\relax 0%
3930   \else
3931     \expandafter\XINT:NE:opx:p
3932     \fi \XINT_allexpr_opx ##1{##2}{##3}{##4}% en fait ##2 = \xint_c_, ##3 = \relax
3933 }}\expandafter\XINT:NE:opx\string#%
3934 \def\XINT:NE:opx:p\XINT_allexpr_opx #1#2#3#4#5#6#7#8%
3935 {%
3936   \expandafter\XINT_expr_put_op_first
3937   \expanded {%
3938     {%
3939       \detokenize
3940       {%
3941         \expanded\bgroup
3942         \expanded{\unexpanded{\XINT_expr_iter:_b
3943           {#1\expandafter\XINT_allexpr_opx_ifnotomitted
3944             \romannumeral0#1#6\relax#7@\relax $XINT_expr_exclam #5}}}%
3945             #4$XINT_expr_caret$XINT_expr_tilde{{#8}}}}$
3946       }%
3947     }%
3948   \expandafter}\romannumeral`&&\XINT_expr_getop
3949 }%
3950 \def\XINT:NE:iter{\expandafter\XINT:NE:itery\expandafter}%
3951 \def\XINT:NE:itery#1{%
3952 \def\XINT:NE:itery\XINT_expr_itery##1##2%
3953 {%
3954   \if 0\expandafter\XINT:NE:hasilde\detokenize{##1##2}~!\relax
3955     \expandafter\XINT:NE:hashash \detokenize{##1##2}#1!\relax 0%
3956   \else
3957     \expandafter\XINT:NE:itery:p
3958     \fi \XINT_expr_itery{##1}{##2}%
3959 }}\expandafter\XINT:NE:itery\string#%
3960 \def\XINT:NE:itery:p\XINT_expr_itery #1#2#3#4#5%
3961 {%
3962   \expandafter\XINT_expr_put_op_first
3963   \expanded {%
3964     {%
3965       \detokenize
3966       {%
3967         \expanded\bgroup
3968         \expanded{\unexpanded{\XINT_expr_iter:_b {#5#4\relax $XINT_expr_exclam #3}}}%
3969           #1$XINT_expr_caret$XINT_expr_tilde{#2}}}%
3970       }%
3971     }%
3972   \expandafter}\romannumeral`&&\XINT_expr_getop
3973 }%
3974 \def\XINT:NE:rseq{\expandafter\XINT:NE:rseqy\expandafter}%
3975 \def\XINT:NE:rseqy#1{%
3976 \def\XINT:NE:rseqy\XINT_expr_rseqy##1##2%

```

```

3977 {%
3978   \if 0\expandafter\XINT:NE:hasilde\detokenize{##1##2}~!\relax
3979     \expandafter\XINT:NE:hashash \detokenize{##1##2}#1!\relax 0%
3980   \else
3981     \expandafter\XINT:NE:rseqy:p
3982     \fi \XINT_expr_rseqy{##1}{##2}%
3983 }}\expandafter\XINT:NE:rseqy\string#%
3984 \def\XINT:NE:rseqy:p\XINT_expr_rseqy #1#2#3#4#5%
3985 {%
3986   \expandafter\XINT_expr_put_op_first
3987   \expanded {%
3988     {%
3989       \detokenize
3990       {%
3991         \expanded\bgroup
3992         \expanded{#2\unexpanded{\XINT_expr_rseq:_b {#5#4\relax $XINT_expr_exclam #3}}%
3993           #1$XINT_expr_caret$XINT_expr_tilde{#2}}%$
3994       }%
3995     }%
3996   \expandafter}\romannumeral`&&\XINT_expr_getop
3997 }%
3998 \def\XINT:NE:iterr{\expandafter\XINT:NE:iterr\expandafter}%
3999 \def\XINT:NE:iterr#1{%
4000 \def\XINT:NE:iterr\XINT_expr_iterry##1##2%
4001 {%
4002   \if 0\expandafter\XINT:NE:hasilde\detokenize{##1##2}~!\relax
4003     \expandafter\XINT:NE:hashash \detokenize{##1##2}#1!\relax 0%
4004   \else
4005     \expandafter\XINT:NE:iterr:p
4006     \fi \XINT_expr_iterry{##1}{##2}%
4007 }}\expandafter\XINT:NE:iterr\string#%
4008 \def\XINT:NE:iterr:p\XINT_expr_iterry #1#2#3#4#5%
4009 {%
4010   \expandafter\XINT_expr_put_op_first
4011   \expanded {%
4012     {%
4013       \detokenize
4014       {%
4015         \expanded\bgroup
4016         \expanded{\unexpanded{\XINT_expr_iterr:_b {#5#4\relax $XINT_expr_exclam #3}}%
4017           #1$XINT_expr_caret$XINT_expr_tilde #20$XINT_expr_qmark}%
4018       }%
4019     }%
4020   \expandafter}\romannumeral`&&\XINT_expr_getop
4021 }%
4022 \def\XINT:NE:rrseq{\expandafter\XINT:NE:rrseq\expandafter}%
4023 \def\XINT:NE:rrseq#1{%
4024 \def\XINT:NE:rrseq\XINT_expr_rrseq##1##2%
4025 {%
4026   \if 0\expandafter\XINT:NE:hasilde\detokenize{##1##2}~!\relax
4027     \expandafter\XINT:NE:hashash \detokenize{##1##2}#1!\relax 0%
4028   \else

```

```

4029     \expandafter\XINT:NE:rrseqy:p
4030     \fi \XINT_expr_rrseqy{##1}{##2}%
4031 }}\expandafter\XINT:NE:rrseqy\string#%
4032 \def\XINT:NE:rrseqy:p\XINT_expr_rrseqy #1#2#3#4#5#6%
4033 {%
4034     \expandafter\XINT_expr_put_op_first
4035     \expanded {%
4036     {%
4037         \detokenize
4038         {%
4039             \expanded\bgroup
4040             \expanded{#2\unexpanded{\XINT_expr_rrseq:_b {#6#5\relax $XINT_expr_exclam #4}}}%
4041             #1$XINT_expr_caret$XINT_expr_tilde #30$XINT_expr_qmark}%
4042         }%
4043     }%
4044     \expandafter}\romannumeral`&&\XINT_expr_getop
4045 }%
4046 \def\XINT:NE:x:toblist#1{%
4047 \def\XINT:NE:x:toblist\XINT:expr:toblistwith##1##2%
4048 {%
4049     \if 0\expandafter\XINT:NE:hashtilde\detokenize{##2}~!\relax
4050     \expandafter\XINT:NE:hashash \detokenize{##2}#1!\relax 0%
4051     \else
4052     \expandafter\XINT:NE:x:toblist:p
4053     \fi \XINT:expr:toblistwith{##1}{##2}%
4054 }}\expandafter\XINT:NE:x:toblist\string#%
4055 \def\XINT:NE:x:toblist:p\XINT:expr:toblistwith #1#2{\XINTfstop.#2}}%
4056 \def\XINT:NE:x:flatten#1{%
4057 \def\XINT:NE:x:flatten\XINT:expr:flatten##1%
4058 {%
4059     \if 0\expandafter\XINT:NE:hashtilde\detokenize{##1}~!\relax
4060     \expandafter\XINT:NE:hashash \detokenize{##1}#1!\relax 0%
4061     \else
4062     \expandafter\XINT:NE:x:flatten:p
4063     \fi \XINT:expr:flatten{##1}%
4064 }}\expandafter\XINT:NE:x:flatten\string#%
4065 \def\XINT:NE:x:flatten:p\XINT:expr:flatten #1%
4066 {%
4067     {{%
4068         \detokenize
4069         {%
4070             \expandafter\XINT:expr:flatten_checkempty
4071             \detokenize\expandafter{\expanded{#1}}$XINT_expr_caret%$
4072         }%
4073     }}%
4074 }%
4075 \def\XINT:NE:x:zip#1{%
4076 \def\XINT:NE:x:zip\XINT:expr:zip##1%
4077 {%
4078     \if 0\expandafter\XINT:NE:hashtilde\detokenize{##1}~!\relax
4079     \expandafter\XINT:NE:hashash \detokenize{##1}#1!\relax 0%
4080     \else

```

```

4081     \expandafter\XINT:NE:x:zip:p
4082     \fi \XINT:expr:zip{##1}%
4083 }}\expandafter\XINT:NE:x:zip:string#%
4084 \def\XINT:NE:x:zip:p\XINT:expr:zip #1%
4085 {%
4086     \expandafter{%
4087         \detokenize
4088         {%
4089             \expanded\expandafter\XINT_zip_A\expanded{#1}\xint_bye\xint_bye
4090         }%
4091     }%
4092 }%
4093 \def\XINT:NE:x:mapwithin#1{%
4094 \def\XINT:NE:x:mapwithin\XINT:expr:mapwithin ##1##2%
4095 {%
4096     \if 0\expandafter\XINT:NE:hasilde\detokenize{##2}~!\relax
4097         \expandafter\XINT:NE:hashash \detokenize{##2}#1!\relax 0%
4098     \else
4099         \expandafter\XINT:NE:x:mapwithin:p
4100     \fi \XINT:expr:mapwithin {##1}{##2}%
4101 }}\expandafter\XINT:NE:x:mapwithin:string#%
4102 \def\XINT:NE:x:mapwithin:p \XINT:expr:mapwithin #1#2%
4103 {%
4104     {{%
4105         \detokenize
4106         {%
4107             \expanded
4108             {%
4109                 \expandafter\XINT:expr:mapwithin_checkempty
4110                 \expanded{\noexpand#1$XINT_expr_exclam\expandafter}%$
4111                 \detokenize\expandafter{\expanded{#2}}$XINT_expr_caret%$
4112             }%
4113         }%
4114     }}%
4115 }%
4116 \def\XINT:NE:x:ndmapx#1{%
4117 \def\XINT:NE:x:ndmapx\XINT_allexpr_ndmapx_a ##1##2^%
4118 {%
4119     \if 0\expandafter\XINT:NE:hasilde\detokenize{##2}~!\relax
4120         \expandafter\XINT:NE:hashash \detokenize{##2}#1!\relax 0%
4121     \else
4122         \expandafter\XINT:NE:x:ndmapx:p
4123     \fi \XINT_allexpr_ndmapx_a ##1##2^%
4124 }}\expandafter\XINT:NE:x:ndmapx:string#%
4125 \def\XINT:NE:x:ndmapx:p #1#2#3^ \relax
4126 {%
4127     \detokenize
4128     {%
4129         \expanded{%
4130             \expandafter#1\expandafter#2\expanded{#3}$XINT_expr_caret\relax %$
4131         }%
4132     }%

```

4133 }%

Attention here that user function names may contain digits, so we don't use a `\detokenize` or `~` approach.

This syntax means that a function defined by `\xintdeffunc` never expands when used in another definition, so it can implement recursive definitions.

`\XINT:NE:userfunc` et al. added at 1.3e.

I added at `\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.

```

4134 \def\xINTusenoargfunc #1%
4135 {%
4136   0\csname #1\endcsname
4137 }%
4138 \def\xINT:NE:usernoargfunc\csname #1\endcsname
4139 {%
4140   ~romannumeral~XINTusenoargfunc{#1}%
4141 }%
4142 \def\xINTusefunc #1%
4143 {%
4144   0\csname #1\expandafter\endcsname\expanded
4145 }%
4146 \def\xINT:NE:usefunc #1#2#3%
4147 {%
4148   ~romannumeral~XINTusefunc{#1}{#3}\iffalse{\fi}%
4149 }%
4150 \def\xINTuseufunc #1%
4151 {%
4152   \expanded\expandafter\xINT:expr:mapwithin\csname #1\expandafter\endcsname\expanded
4153 }%
4154 \def\xINT:NE:useufunc #1#2#3%
4155 {%
4156   {{~expanded~XINTuseufunc{#1}{#3}}}%
4157 }%
4158 \def\xINT:NE:userfunc #1{%
4159 \def\xINT:NE:userfunc ##1##2##3%
4160 {%
4161   \if0\expandafter\xINT:NE:hasilde\detokenize{##3}~!\relax
4162     \expandafter\xINT:NE:hashash\detokenize{##3}#1!\relax 0%
4163     \expandafter\xINT:NE:userfunc_x
4164   \else
4165     \expandafter\xINT:NE:usefunc
4166     \fi {##1}{##2}{##3}%
4167 }}\expandafter\xINT:NE:userfunc\string#%
4168 \def\xINT:NE:userfunc_x #1#2#3{#2#3\iffalse{\fi}}%
4169 \def\xINT:NE:userufunc #1{%
4170 \def\xINT:NE:userufunc ##1##2##3%
4171 {%
4172   \if0\expandafter\xINT:NE:hasilde\detokenize{##3}~!\relax
4173     \expandafter\xINT:NE:hashash\detokenize{##3}#1!\relax 0%
4174     \expandafter\xINT:NE:userufunc_x

```

```

4175 \else
4176 \expandafter\XINT:NE:useufunc
4177 \fi {##1}{##2}{##3}%
4178 }}\expandafter\XINT:NE:userufunc\string#%
4179 \def\XINT:NE:userufunc_x #1{\XINT:expr:mapwithin}%
4180 \def\XINT:NE:macrofunc #1#2%
4181 {\expandafter\XINT:NE:macrofunc:a\string#1#2\empty&}%
4182 \def\XINT:NE:macrofunc:a#1\csname #2\endcsname#3&%
4183 {{~\XINTusemacrofunc{#1}{#2}{#3}}}%
4184 \def\XINTusemacrofunc #1#2#3%
4185 {%
4186 \romannumeral0\expandafter\xint_stop_atfirstofone
4187 \romannumeral0#1\csname #2\endcsname#3\relax
4188 }%

```

11.29.6 \XINT_expr_redefinmacros

Completely refactored at 1.3.

Again refactored at 1.4. The availability of \expanded allows more powerful mechanisms and more importantly I better thought out the root problems caused by the handling of list operations in this context and this helped simplify considerably the code.

```

4189 \catcode`- 11
4190 \def\XINT_expr_redefinmacros {%
4191 \let\XINT:NEhook:unpack \XINT:NE:unpack
4192 \let\XINT:NEhook:f:one:from:one \XINT:NE:f:one:from:one
4193 \let\XINT:NEhook:f:one:from:one:direct \XINT:NE:f:one:from:one:direct
4194 \let\XINT:NEhook:f:one:from:two \XINT:NE:f:one:from:two
4195 \let\XINT:NEhook:f:one:from:two:direct \XINT:NE:f:one:from:two:direct
4196 \let\XINT:NEhook:x:one:from:two \XINT:NE:x:one:from:two
4197 \let\XINT:NEhook:x:one:from:twoandone \XINT:NE:x:one:from:twoandone
4198 \let\XINT:NEhook:f:one:and:opt:direct \XINT:NE:f:one:and:opt:direct
4199 \let\XINT:NEhook:f:tacitzeroifone:direct \XINT:NE:f:tacitzeroifone:direct
4200 \let\XINT:NEhook:f:iitacitzeroifone:direct \XINT:NE:f:iitacitzeroifone:direct
4201 \let\XINT:NEhook:x:listscl \XINT:NE:x:listscl
4202 \let\XINT:NEhook:f:reverse \XINT:NE:f:reverse
4203 \let\XINT:NEhook:f:from:delim:u \XINT:NE:f:from:delim:u
4204 \let\XINT:NEhook:f:noeval:from:braced:u\XINT:NE:f:noeval:from:braced:u
4205 \let\XINT:NEhook:branch \XINT:NE:branch
4206 \let\XINT:NEhook:seqx \XINT:NE:seqx
4207 \let\XINT:NEhook:opx \XINT:NE:opx
4208 \let\XINT:NEhook:rseq \XINT:NE:rseq
4209 \let\XINT:NEhook:iter \XINT:NE:iter
4210 \let\XINT:NEhook:rrseq \XINT:NE:rrseq
4211 \let\XINT:NEhook:iterr \XINT:NE:iterr
4212 \let\XINT:NEhook:x:toblist \XINT:NE:x:toblist
4213 \let\XINT:NEhook:x:flatten \XINT:NE:x:flatten
4214 \let\XINT:NEhook:x:zip \XINT:NE:x:zip
4215 \let\XINT:NEhook:x:mapwithin \XINT:NE:x:mapwithin
4216 \let\XINT:NEhook:x:ndmapx \XINT:NE:x:ndmapx
4217 \let\XINT:NEhook:userfunc \XINT:NE:userfunc
4218 \let\XINT:NEhook:userufunc \XINT:NE:userufunc
4219 \let\XINT:NEhook:usernoargfunc \XINT:NE:usernoargfunc

```



```

4220 \let\XINT:NEhook:macrofunc \XINT:NE:macrofunc
4221 \def\XINTinRandomFloatSdigits{~XINTinRandomFloatSdigits }%
4222 \def\XINTinRandomFloatSixteen{~XINTinRandomFloatSixteen }%
4223 \def\xintiiRandRange{~xintiiRandRange }%
4224 \def\xintiiRandRangeAtoB{~xintiiRandRangeAtoB }%
4225 \def\xintRandBit{~xintRandBit }%
4226 \let\XINT_expr_exec_? \XINT:NE:exec_?
4227 \let\XINT_expr_exec_?? \XINT:NE:exec_??
4228 \def\XINT_expr_op_? {\XINT_expr_op__?{\XINT_expr_op_-xii\XINT_expr_oparen}}%
4229 \def\XINT_flexpr_op_?{\XINT_expr_op__?{\XINT_flexpr_op_-xii\XINT_flexpr_oparen}}%
4230 \def\XINT_iiexpr_op_?{\XINT_expr_op__?{\XINT_iiexpr_op_-xii\XINT_iiexpr_oparen}}%
4231 }%
4232 \catcode`- 12

```

11.29.7 \xintNewExpr, \xintNewIExpr, \xintNewFloatExpr, \xintNewIIExpr

1.2c modifications to accomodate \XINT_expr_deffunc_newexpr etc..

1.2f adds token \XINT_newexpr_clean to be able to have a different \XINT_newfunc_clean.

As \XINT_NewExpr always execute \XINT_expr_redefineprints since 1.3e whether with \xintNewExpr or \XINT_NewFunc, it has been moved from argument to hardcoded in replacement text.

NO MORE \XINT_expr_redefineprints at 1.4 ! This allows better support for \xinteval, \xinttheexpr as sub-entities inside an \xintNewExpr. And the «cleaning» will remove the new \XINTfstop (detokenized from \meaning output), to maintain backwards compatibility with former behaviour that created macros expand to explicit digits and not an encapsulated result.

The #2#3 in clean stands for \noexpand\XINTfstop (where the actual scantoken-ized input uses \$ originally with catcode letter as the escape character).

```

4233 \def\xintNewExpr {\XINT_NewExpr\xint_firstofone\xintexpr \XINT_newexpr_clean}%
4234 \def\xintNewFloatExpr{\XINT_NewExpr\xint_firstofone\xintfloatexpr\XINT_newexpr_clean}%
4235 \def\xintNewIExpr {\XINT_NewExpr\xint_firstofone\xintiexpr \XINT_newexpr_clean}%
4236 \def\xintNewIIExpr {\XINT_NewExpr\xint_firstofone\xintiexpr \XINT_newexpr_clean}%
4237 \def\xintNewBoolExpr {\XINT_NewExpr\xint_firstofone\xintboolexpr \XINT_newexpr_clean}%
4238 \def\XINT_newexpr_clean #1>#2#3{\noexpand\expanded\noexpand\xintNEprinthead}%
4239 \def\xintNEprinthead#1.#2{\expanded{\unexpanded{#1.}{#2}}}%

```

1.2c for \xintdeffunc, \xintdefiifunc, \xintdeffloatfunc.

At 1.3, NewFunc does not use anymore a comma delimited pattern for the arguments to the macro being defined.

At 1.4 we use \xintthebareeval, whose meaning now does not mean unlock from csname but firstofone to remove a level of braces This is involved in functioning of expr:userfunc and expr:userefunc

```

4240 \def\XINT_NewFunc {\XINT_NewExpr\xint_gobble_i\xintthebareeval\XINT_newfunc_clean}%
4241 \def\XINT_NewFloatFunc{\XINT_NewExpr\xint_gobble_i\xintthebarefloveval\XINT_newfunc_clean}%
4242 \def\XINT_NewIIFunc {\XINT_NewExpr\xint_gobble_i\xintthebareiieval\XINT_newfunc_clean}%
4243 \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.

```

4244 \def\XINT_NewExpr #1#2#3#4#5[#6]%
4245 { }%
4246 \begingroup

```

```

4247 \ifcase #6\relax
4248 \toks0 {\endgroup\XINT_global\def#4}%
4249 \or \toks0 {\endgroup\XINT_global\def#4##1}%
4250 \or \toks0 {\endgroup\XINT_global\def#4##1##2}%
4251 \or \toks0 {\endgroup\XINT_global\def#4##1##2##3}%
4252 \or \toks0 {\endgroup\XINT_global\def#4##1##2##3##4}%
4253 \or \toks0 {\endgroup\XINT_global\def#4##1##2##3##4##5}%
4254 \or \toks0 {\endgroup\XINT_global\def#4##1##2##3##4##5##6}%
4255 \or \toks0 {\endgroup\XINT_global\def#4##1##2##3##4##5##6##7}%
4256 \or \toks0 {\endgroup\XINT_global\def#4##1##2##3##4##5##6##7##8}%
4257 \or \toks0 {\endgroup\XINT_global\def#4##1##2##3##4##5##6##7##8##9}%
4258 \fi
4259 #1\xintexprSafeCatcodes
4260 \XINT_expr_redefinmacros
4261 \XINT_NewExpr_a #1#2#3#4%
4262 }%

```

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.

```

4263 \catcode`~ 3 \catcode`? 3
4264 \def\XINT_expr_tilde{~}\def\XINT_expr_qmark{?}% catcode 3
4265 \def\XINT_expr_caret{^}\def\XINT_expr_exclam{!}% catcode 11
4266 \def\XINT_expr_tab{&}% catcode 7
4267 \catcode`~ 13 \catcode`@ 14 \catcode`\% 6 \catcode`# 12 \catcode`$ 11 @ $
4268 \def\XINT_NewExpr_a %1%2%3%4%5@
4269 {@
4270 \def\XINT_tmpa %1%2%3%4%5%6%7%8%9{%5}@
4271 \def~{$noexpand$}@
4272 \catcode`: 11 \catcode`_ 11 \catcode`\@ 11

```

```

4273 \catcode`# 12 \catcode`~ 13 \escapechar 126
4274 \endlinechar -1 \everyeof {\noexpand }@
4275 \edef\XINT_tmpb
4276 {\scantokens\expandafter{\romannumeral`&&\expandafter
4277 %2\XINT_tmpa{#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9}\relax}@
4278 }@
4279 \escapechar 92 \catcode`# 6 \catcode`$ 0 @ $
4280 \edef\XINT_tmpa %%1%%2%%3%%4%%5%%6%%7%%8%%9@
4281 {\scantokens\expandafter{\expandafter%3\meaning\XINT_tmpb}}@
4282 \the\toks0\expandafter
4283 {\XINT_tmpa{%%1}{%%2}{%%3}{%%4}{%%5}{%%6}{%%7}{%%8}{%%9}}@
4284 %1{\ifxintverbose
4285 \xintMessage{xintexpr}{Info}@
4286 {\string%4\space now with @
4287 \ifxintglobaldefs global \fi meaning \meaning%4}@
4288 \fi}@
4289 }@
4290 \catcode`% 14
4291 \XINT_setcatcodes % clean up to avoid surprises if something changes

```

11.29.8 \ifxintexprsafecatcodes, \xintexprSafeCatcodes, \xintexprRestoreCatcodes

1.3c (2018/06/17).

Added `\ifxintexprsafecatcodes` to allow nesting

```

4292 \newif\ifxintexprsafecatcodes
4293 \let\xintexprRestoreCatcodes\empty
4294 \def\xintexprSafeCatcodes
4295 {%
4296 \unless\ifxintexprsafecatcodes
4297 \edef\xintexprRestoreCatcodes {%
4298 \catcode59=\the\catcode59 % ;
4299 \catcode34=\the\catcode34 % "
4300 \catcode63=\the\catcode63 % ?
4301 \catcode124=\the\catcode124 % |
4302 \catcode38=\the\catcode38 % &
4303 \catcode33=\the\catcode33 % !
4304 \catcode93=\the\catcode93 % ]
4305 \catcode91=\the\catcode91 % [
4306 \catcode94=\the\catcode94 % ^
4307 \catcode95=\the\catcode95 % _
4308 \catcode47=\the\catcode47 % /
4309 \catcode41=\the\catcode41 % )
4310 \catcode40=\the\catcode40 % (
4311 \catcode42=\the\catcode42 % *
4312 \catcode43=\the\catcode43 % +
4313 \catcode62=\the\catcode62 % >
4314 \catcode60=\the\catcode60 % <
4315 \catcode58=\the\catcode58 % :
4316 \catcode46=\the\catcode46 % .
4317 \catcode45=\the\catcode45 % -
4318 \catcode44=\the\catcode44 % ,
4319 \catcode61=\the\catcode61 % =

```

```

4320     \catcode96=\the\catcode96 % `
4321     \catcode32=\the\catcode32\relax % space
4322     \noexpand\xintexprsafecatcodesfalse
4323 }%
4324 \fi
4325 \xintexprsafecatcodestrue
4326     \catcode59=12 % ;
4327     \catcode34=12 % "
4328     \catcode63=12 % ?
4329     \catcode124=12 % |
4330     \catcode38=4 % &
4331     \catcode33=12 % !
4332     \catcode93=12 % ]
4333     \catcode91=12 % [
4334     \catcode94=7 % ^
4335     \catcode95=8 % _
4336     \catcode47=12 % /
4337     \catcode41=12 % )
4338     \catcode40=12 % (
4339     \catcode42=12 % *
4340     \catcode43=12 % +
4341     \catcode62=12 % >
4342     \catcode60=12 % <
4343     \catcode58=12 % :
4344     \catcode46=12 % .
4345     \catcode45=12 % -
4346     \catcode44=12 % ,
4347     \catcode61=12 % =
4348     \catcode96=12 % `
4349     \catcode32=10 % space
4350 }%
4351 \let\XINT_tmpa\undefined \let\XINT_tmpb\undefined \let\XINT_tmpc\undefined
4352 \let\XINT_tmpd\undefined \let\XINT_tmpe\undefined
4353 \ifdefined\RequirePackage\expandafter\xint_firstoftwo\else\expandafter\xint_secondoftwo\fi
4354 {\RequirePackage{xinttrig}}%
4355 \RequirePackage{xintlog}}%
4356 {\input xinttrig.sty
4357 \input xintlog.sty
4358 }%
4359 \XINT_restorecatcodes_endinput%

```

12 Package [xinttrig](#) implementation

Contents

12.1	Catcodes, ϵ -TeX and reload detection	422
12.2	Library identification	422
12.3	Ensure used letters are dummy letters	423
12.4	<code>\xintreloadxinttrig</code>	423
12.5	Auxiliary variables (only temporarily needed, but left free to re-use)	423
12.5.1	<code>twoPi</code> , <code>threePiover2</code> , <code>Pi</code> , <code>Piover2</code>	423
12.5.2	<code>oneDegree</code> , <code>oneRadian</code>	423
12.5.3	Inverse factorial coefficients: <code>invfact2</code> , ..., <code>invfact44</code>	423
12.6	The sine and cosine series	424
12.6.1	<code>sin_aux()</code> , <code>cos_aux()</code>	424
12.6.2	Make <code>sin_aux()</code> and <code>cos_aux()</code> known to <code>\xintexpr</code>	425
12.6.3	<code>sin_()</code> , <code>cos_()</code>	426
12.7	Range reduction for sine and cosine using degrees	426
12.7.1	Core level macro <code>\XINT_mod_ccclx_i</code>	426
12.7.2	<code>sind_()</code> , <code>cosd_()</code> , and support macros <code>\xintSind</code> , <code>\xintCosd</code>	427
12.8	<code>sind()</code> , <code>cosd()</code>	431
12.9	<code>sin()</code> , <code>cos()</code>	431
12.10	<code>sinc()</code>	431
12.11	<code>tan()</code> , <code>tand()</code> , <code>cot()</code> , <code>cotd()</code>	432
12.12	<code>sec()</code> , <code>secd()</code> , <code>csc()</code> , <code>cscd()</code>	432
12.13	Core routine for inverse trigonometry	432
12.14	<code>asin()</code> , <code>asind()</code>	434
12.15	<code>acos()</code> , <code>acosd()</code>	435
12.16	<code>atan()</code> , <code>atand()</code>	435
12.17	<code>Arg()</code> , <code>atan2()</code> , <code>Argd()</code> , <code>atan2d()</code> , <code>pArg()</code> , <code>pArgd()</code>	435
12.18	Synonyms: <code>tg()</code> , <code>cotg()</code>	437
12.19	Let the functions be known to the <code>\xintexpr</code> parser	437

Comments under reconstruction.

The original was done in January 15 and 16, 2019. It provided `asin()` and `acos()` based on a Newton algorithm approach. Then during March 25-31 I revisited the code, adding more inverse trigonometrical functions (with a modified algorithm, quintically convergent), extending the precision range (so that the package reacts to the `\xintDigits` value at time of load, or reload), and replaced high level range reduction by some optimized lower level coding.

This led me next to improve upon the innards of `\xintdeffunc` and `\xintNewExpr`, and to add to `xintexpr` the `\xintdefefunc` macro (see user documentation).

Finally on April 5, 2019 I pushed further the idea of the algorithm for the arcsine function. The cost is at least the one of a combined `sin()/cos()` evaluation, surely this is not best approach for low precision, but I like the principle and its suitability to go into hundreds of digits if desired.

Almost all of the code remains written at high level, and in particular it is not easily feasible from this interface to execute computations with guard digits. Expect the last one or two digits to be systematically off.

Also, small floating-point inputs are handled quite sub-optimally both for the direct and inverse functions; substantial gains are possible. I added the `ilog10()` function too late to consider using it here with the high level interface.

12.1 Catcodes, ϵ -TeX and reload detection

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2 \catcode13=5 % ^^M
3 \endlinechar=13 %
4 \catcode123=1 % {
5 \catcode125=2 % }
6 \catcode64=11 % @
7 \catcode35=6 % #
8 \catcode44=12 % ,
9 \catcode45=12 % -
10 \catcode46=12 % .
11 \catcode58=12 % :
12 \catcode94=7 % ^
13 \def\z{\endgroup}%
14 \def\empty{}\def\space{ }\newlinechar10
15 \expandafter\let\expandafter\w\csname ver@xintexpr.sty\endcsname
16 \expandafter
17 \ifx\csname PackageInfo\endcsname\relax
18 \def\y#1#2{\immediate\write-1{Package #1 Info:^^}%
19 \space\space\space\space#2.}}%
20 \else
21 \def\y#1#2{\PackageInfo{#1}{#2}}%
22 \fi
23 \expandafter
24 \ifx\csname numexpr\endcsname\relax
25 \y{xinttrig}{numexpr not available, aborting input}%
26 \aftergroup\endinput
27 \else
28 \ifx\w\relax % xintexpr.sty not yet loaded.
29 \y{xinttrig}%
30 {Loading should be via \ifx\x\empty\string\usepackage{xintexpr.sty}
31 \else\string\input\space xintexpr.sty \fi
32 rather, aborting}%
33 \aftergroup\endinput
34 \fi
35 \fi
36 \z%
37 \catcode`_ 11 \XINT_setcatcodes \catcode`? 12

```

12.2 Library identification

```

38 \ifcsname xintlibver@trig\endcsname
39 \expandafter\xint_firstoftwo
40 \else
41 \expandafter\xint_secondoftwo
42 \fi
43 {\immediate\write-1{Reloading xinttrig library using Digits=\xinttheDigits.}}%
44 {\expandafter\gdef\csname xintlibver@trig\endcsname{2020/02/25 v1.4b}}%
45 \XINT_providespackage
46 \ProvidesPackage{xinttrig}%
47 [2020/02/25 v1.4b Trigonometrical functions for xintexpr (JFB)]%
48 }%

```

12.3 Ensure used letters are dummy letters

```
49 \xintFor* #1 in {iDTvtuvwxyzX}\do{\xintensuredummy{#1}}%
```

12.4 \xintreloadxinttrig

```
50 \def\xintreloadxinttrig
51   {\edef\XINT_restorecatcodes_now{\XINT_restorecatcodes}%
52   \XINT_setcatcodes\catcode`? 12
53   \input xinttrig.sty
54   \XINT_restorecatcodes_now}%
```

12.5 Auxiliary variables (only temporarily needed, but left free to re-use)

These variables don't have really private names but this does not matter because only their actual values will be stored in the functions defined next. Nevertheless they are not unassigned, and are left free to use as is.

12.5.1 twoPi, threePiover2, Pi, Piover2

We take them with 60 digits and force conversion to \xintDigits setting via "0 + " syntax.

```
55 \xintdeffloatvar twoPi      := 0 +
56   6.28318530717958647692528676655900576839433879875021164194989;%
57 \xintdeffloatvar threePiover2 := 0 +
58   4.71238898038468985769396507491925432629575409906265873146242;%
59 \xintdeffloatvar Pi        := 0 +
60   3.14159265358979323846264338327950288419716939937510582097494;%
61 \xintdeffloatvar Piover2   := 0 +
62   1.57079632679489661923132169163975144209858469968755291048747;%
```

12.5.2 oneDegree, oneRadian

```
63 \xintdeffloatvar oneDegree := 0 +
64   0.0174532925199432957692369076848861271344287188854172545609719;% Pi/180
65 \xintdeffloatvar oneRadian := 0 +
66   57.2957795130823208767981548141051703324054724665643215491602;% 180/Pi
```

12.5.3 Inverse factorial coefficients: invfact2, ..., invfact44

Pre-compute $1/n!$ for $n = 2, \dots, 44$

We have to be careful that $1/i!$ in a float expression first evaluates $i!$ as a floating point number then computes the inverse. Even if $i!$ was computed exactly before being float-rounded, this process would not necessarily lead to the correct rounding of the exact fraction $1/i!$.

We could use `\xintexpr1/i!\relax` encapsulation but then the actual rounding is delayed to the time when functions are used... this is bad.

We need to get now the correct rounding of the exact $1/i!$.

1.4 update: use `\xintfloatexpr` with optional argument for the rounding rather than «0+x» method. And there is no need now to hide within braces the inner semi-colon.

```
67 \xintdeffloatvar invfact\xintListWithSep{, invfact}\xintSeq{2}{44}}%
68   := \xintfloatexpr [\XINTdigits] % force float rounding after exact evaluations
69   \xintexpr rseq(1/2; @/i, i=3..44)\relax % no need to hide this inner ;
70   \relax;%
```


12.6.3 `sin_()`, `cos_()`

Use this only between $-\pi/4$ and $\pi/4$

```
157 \xintdeffloatfunc sin_(x) := x * sin_aux(sqr(x));%
```

Use this only between $-\pi/4$ and $\pi/4$

```
158 \xintdeffloatfunc cos_(x) := cos_aux(sqr(x));%
```

12.7 Range reduction for sine and cosine using degrees

Notice that even when handling radians it is much better to convert to degrees and then do range reduction there, because this can be done in the fixed point sense. I lost 1h puzzled about some mismatch of my results with those of Maple (at 16 digits) near $-\pi$. Turns out that Maple probably adds π in the floating point sense causing catastrophic loss of digits when one is near $-\pi$. On the other hand my `sin(x)` function will first convert to degrees then add 180 without any loss of floating point precision, even for a result near zero, then convert back to radians and use the sine series.

12.7.1 Core level macro `\XINT_mod_ccclx_i`

input: `\the\numexpr\XINT_mod_ccclx_i k.N.` (delimited by dots)

output: `(N times 10^k) modulo 360.` (with a final dot)

Attention `N` must be non-negative (I could make it accept negative but the fact that `numexpr /` is not periodical in numerator adds overhead).

`360` divides `9000` hence `10^k` is `280` for `k` at least `3` and the additive group generated by it modulo `360` is the set of multiples of `40`.

```
159 \def\XINT_mod_ccclx_i #1.% input <k>.<N>. k is a non-negative exponent
160 {%
161   \expandafter\XINT_mod_ccclx_e\the\numexpr
162   \expandafter\XINT_mod_ccclx_j\the\numexpr1\ifcase#1 \or0\or00\else000\fi.%
163 }%
164 \def\XINT_mod_ccclx_j 1#1.#2.% #2=N is a non-negative mantissa
165 {%
166   (\XINT_mod_ccclx_ja {++}#2#1\XINT_mod_ccclx_jb 0000000\relax
167   1      2345678
168 \def\XINT_mod_ccclx_ja #1#2#3#4#5#6#7#8#9%
169 {%
170   #9+#8+#7+#6+#5+#4+#3+#2\xint_firstoftwo{+\XINT_mod_ccclx_ja{+#9+#8+#7}}{#1}%
171 }%
172 \def\XINT_mod_ccclx_jb #1\xint_firstoftwo#2#3{#1+0)*280\XINT_mod_ccclx_jc #1#3}%
```

Attention that `\XINT_cclcx_e` wants non negative input because `\numexpr` division is not periodical ...

```
173 \def\XINT_mod_ccclx_jc  +#1+#2+#3#4\relax{+80*(#3+#2+#1)+#3#2#1.}%
174 \def\XINT_mod_ccclx_e#1.{\expandafter\XINT_mod_ccclx_z\the\numexpr(#1+180)/360-1.#1.}%
175 \def\XINT_mod_ccclx_z#1.#2.{#2-360*#1.}%
```

12.7.2 sind_(), cosd_(), and support macros \xintSind, \xintCosd

sind_() coded directly at macro level with a macro \xintSind (ATTENTION! it requires a positive argument) which will suitably use \XINT_flexpr_func_sin_ defined from \xintdeffloatfunc

```
176 \def\XINT_flexpr_func_sind_ #1#2#3%
177 {%
178   \expandafter #1\expandafter #2\expandafter{%
179     \romannumeral`&&\XINT:NEhook:f:one:from:one{\romannumeral`&&\xintSind#3}}%
180 }%
```

Must be f-expandable for nesting macros from \xintNewExpr
ATTENTION ONLY FOR POSITIVE ARGUMENTS

```
181 \def\XINT_expr_unlock{\expandafter\xint_firstofone\romannumeral`&&@}%
182 \def\xintSind#1{\romannumeral`&&\expandafter\xintsind
183   \romannumeral0\XINTinfloatS[\XINTdigits]{#1}}%
184 \def\xintsind #1[#2#3]%
185 {%
186   \xint_UDsignfork
187     #2\XINT_sind
188     -\XINT_sind_int
189   \krof#2#3.#1..%<< attention extra dot
190 }%
191 \def\XINT_sind #1.#2.% NOT TO BE USED WITH VANISHING (OR NEGATIVE) #2.
192 {%
193   \expandafter\XINT_sind_a
194   \romannumeral0\xinttrunc{\XINTdigits}{#2[#1]}%
195 }%
196 \def\XINT_sind_a{\expandafter\XINT_sind_i\the\numexpr\XINT_mod_ccclx_i0.}%
197 \def\XINT_sind_int
198 {%
199   \expandafter\XINT_sind_i\the\numexpr\expandafter\XINT_mod_ccclx_i
200 }%
201 \def\XINT_sind_i #1.% range reduction inside [0, 360[
202 {%
203   \ifcase\numexpr#1/90\relax
204     \expandafter\XINT_sind_A
205   \or\expandafter\XINT_sind_B\the\numexpr-90+%
206   \or\expandafter\XINT_sind_C\the\numexpr-180+%
207   \or\expandafter\XINT_sind_D\the\numexpr-270+%
208   \else\expandafter\XINT_sind_E\the\numexpr-360+%
209   \fi#1.%
210 }%
```

#2 will be empty in the "integer branch". Notice that a single dot "." is valid as input to the xintfrac macros. During developing phase I did many silly mistakes due to wanting to use too low-level interface, e.g. I would use something like #2[-\XINTdigits] with #2 the fractional digits, but there maybe some leading zero and then xintfrac.sty will think the whole thing is zero due to the requirements of my own core format A[N]...

Multiplication is done exactly but anyway currently float multiplication goes via exact multiplication after rounding arguments ; as here integer part has at most three digits, doing exact multiplication will prove not only more accurate but probably faster.

```

211 \def\XINT_sind_A#1{%
212 \def\XINT_sind_A##1.##2.%
213 {%
214     \XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_sin\expandafter
215         {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{##1.##2}#1}}%
216 }%
217 }\expandafter
218 \XINT_sind_A\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%
219 \def\XINT_sind_B#1{\xint_UDsignfork#1\XINT_sind_B_n-\XINT_sind_B_p\krof #1}%
220 \def\XINT_tmpa#1{%
221 \def\XINT_sind_B_n-##1.##2.%
222 {%
223     \XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_cos\expandafter
224         {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{\xintSub{##1[0]}.{##2}}#1}}%
225 }%
226 \def\XINT_sind_B_p##1.##2.%
227 {%
228     \XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_cos\expandafter
229         {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{##1.##2}#1}}%
230 }%
231 }\expandafter
232 \XINT_tmpa\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%
233 \def\XINT_sind_C#1{\xint_UDsignfork#1\XINT_sind_C_n-\XINT_sind_C_p\krof #1}%
234 \def\XINT_tmpa#1{%
235 \def\XINT_sind_C_n-##1.##2.%
236 {%
237     \XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_sin\expandafter
238         {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{\xintSub{##1[0]}.{##2}}#1}}%
239 }%
240 \def\XINT_sind_C_p##1.##2.%
241 {%
242     \xintiiopp\XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_sin\expandafter
243         {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{##1.##2}#1}}%
244 }%
245 }\expandafter
246 \XINT_tmpa\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%
247 \def\XINT_sind_D#1{\xint_UDsignfork#1\XINT_sind_D_n-\XINT_sind_D_p\krof #1}%
248 \def\XINT_tmpa#1{%
249 \def\XINT_sind_D_n-##1.##2.%
250 {%
251     \xintiiopp\XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_cos\expandafter
252         {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{\xintSub{##1[0]}.{##2}}#1}}%
253 }%
254 \def\XINT_sind_D_p##1.##2.%
255 {%
256     \xintiiopp\XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_cos\expandafter
257         {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{##1.##2}#1}}%
258 }%
259 }\expandafter
260 \XINT_tmpa\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%
261 \def\XINT_sind_E#1{%
262 \def\XINT_sind_E-##1.##2.%

```

```

263 {%
264   \xintiopp\XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_sin_\expandafter
265     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{\xintSub{##1[0]}.{##2}}#1}}%
266 }%
267 }\expandafter
268 \XINT_sind_E\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%

```

The cosd_ auxiliary function

```

269 \def\XINT_flexpr_func_cosd_ #1#2#3%
270 {%
271   \expandafter #1\expandafter #2\expandafter{%
272     \romannumeral`&&\XINT:NEhook:f:one:from:one{\romannumeral`&&\xintCosd#3}}%
273 }%

```

ATTENTION ONLY FOR POSITIVE ARGUMENTS

```

274 \def\xintCosd#1{\romannumeral`&&\expandafter\xintcosd
275   \romannumeral0\XINTinfloatS[\XINTdigits]{#1}}%
276 \def\xintcosd #1[#2#3]%
277 {%
278   \xint_UDsignfork
279     #2\XINT_cosd
280     -\XINT_cosd_int
281   \krof#2#3.#1..%<< attention extra dot
282 }%
283 \def\XINT_cosd #1.#2.% NOT TO BE USED WITH VANISHING (OR NEGATIVE) #2.
284 {%
285   \expandafter\XINT_cosd_a
286   \romannumeral0\xinttrunc{\XINTdigits}{#2[#1]}%
287 }%
288 \def\XINT_cosd_a{\expandafter\XINT_cosd_i\the\numexpr\XINT_mod_ccclx_i0.}%
289 \def\XINT_cosd_int
290 {%
291   \expandafter\XINT_cosd_i\the\numexpr\expandafter\XINT_mod_ccclx_i
292 }%
293 \def\XINT_cosd_i #1.%
294 {%
295   \ifcase\numexpr#1/90\relax
296     \expandafter\XINT_cosd_A
297   \or\expandafter\XINT_cosd_B\the\numexpr-90+%
298   \or\expandafter\XINT_cosd_C\the\numexpr-180+%
299   \or\expandafter\XINT_cosd_D\the\numexpr-270+%
300   \else\expandafter\XINT_cosd_E\the\numexpr-360+%
301   \fi#1.%
302 }%

```

#2 will be empty in the "integer" branch, but attention in general branch to handling of negative integer part after the subtraction of 90, 180, 270, or 360, and avoid abusing A[N] notation which yes speeds up xintfrac parsing but has its pitfalls.

```

303 \def\XINT_cosd_A#1{%
304 \def\XINT_cosd_A##1.##2.%
305 {%

```

```

306 \XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_cos_\expandafter
307     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{##1.##2}#1}}%
308 }%
309 }\expandafter
310 \XINT_cosd_A\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%
311 \def\XINT_cosd_B#1{\xint_UDsignfork#1\XINT_cosd_B_n-\XINT_cosd_B_p\krof #1}%
312 \def\XINT_tmpa#1{%
313 \def\XINT_cosd_B_n-##1.##2.%
314 {%
315 \XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_sin_\expandafter
316     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{\xintSub{##1[0]}.{##2}}#1}}%
317 }%
318 \def\XINT_cosd_B_p##1.##2.%
319 {%
320 \xintiiopp\XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_sin_\expandafter
321     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{##1.##2}#1}}%
322 }%
323 }\expandafter
324 \XINT_tmpa\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%
325 \def\XINT_cosd_C#1{\xint_UDsignfork#1\XINT_cosd_C_n-\XINT_cosd_C_p\krof #1}%
326 \def\XINT_tmpa#1{%
327 \def\XINT_cosd_C_n-##1.##2.%
328 {%
329 \xintiiopp\XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_cos_\expandafter
330     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{\xintSub{##1[0]}.{##2}}#1}}%
331 }%
332 \def\XINT_cosd_C_p##1.##2.%
333 {%
334 \xintiiopp\XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_cos_\expandafter
335     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{##1.##2}#1}}%
336 }%
337 }\expandafter
338 \XINT_tmpa\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%
339 \def\XINT_cosd_D#1{\xint_UDsignfork#1\XINT_cosd_D_n-\XINT_cosd_D_p\krof #1}%
340 \def\XINT_tmpa#1{%
341 \def\XINT_cosd_D_n-##1.##2.%
342 {%
343 \xintiiopp\XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_sin_\expandafter
344     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{\xintSub{##1[0]}.{##2}}#1}}%
345 }%
346 \def\XINT_cosd_D_p##1.##2.%
347 {%
348 \XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_sin_\expandafter
349     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{##1.##2}#1}}%
350 }%
351 }\expandafter
352 \XINT_tmpa\expandafter{\romannumeral`&&\xintthebarefloateval oneDegree\relax}%
353 \def\XINT_cosd_E#1{%
354 \def\XINT_cosd_E-##1.##2.%
355 {%
356 \XINT_expr_unlock\expandafter\XINT_flexpr_userfunc_cos_\expandafter
357     {\romannumeral0\XINTinfloat[\XINTdigits]{\xintMul{\xintSub{##1[0]}.{##2}}#1}}%

```

```

358 }%
359 }\expandafter
360 \XINT_cosd_E\expandafter{\romannumeral`&&@\xintthebarefloateval oneDegree\relax}%

```

12.8 sind(), cosd()

```

361 \xintdeffloatfunc sind(x) := (x)??
362         {(x>=-45)?
363         {sin_(x*oneDegree)}
364         {-sind_(-x)}
365         }
366         {0}
367         {(x<=45)?
368         {sin_(x*oneDegree)}
369         {sind_(x)}
370         }
371         ;%
372 \xintdeffloatfunc cosd(x) := (x)??
373         {(x>=-45)?
374         {cos_(x*oneDegree)}
375         {cosd_(-x)}
376         }
377         {1}
378         {(x<=45)?
379         {cos_(x*oneDegree)}
380         {cosd_(x)}
381         }
382         ;%

```

12.9 sin(), cos()

For some reason I did not define sin() and cos() in January 2019 ??

```

383 \xintdeffloatfunc sin(x):= (abs(x)<0.79)?
384         {sin_(x)}
385         {(x)??
386         {-sind_(-x*oneRadian)}
387         {0}
388         {sind_(x*oneRadian)}
389         }
390         ;%
391 \xintdeffloatfunc cos(x):= (abs(x)<0.79)?
392         {cos_(x)}
393         {cosd_(abs(x*oneRadian))}
394         ;%

```

12.10 sinc()

Should I also consider adding $(1-\cos(x))/(x^2/2)$? it is $\text{sinc}^2(x/2)$ but avoids a square.

```

395 \xintdeffloatfunc sinc(x):= (abs(x)<0.79) ?
396         {sin_aux(sqr(x))}
397         {sind_(abs(x)*oneRadian)/abs(x)}
398         ;%

```

12.11 tan(), tand(), cot(), cotd()

The 0 in cot(x) is a dummy place holder, 1/0 would raise an error at time of definition...

```

399 \xintdeffloatfunc tand(x):= sind(x)/cosd(x);%
400 \xintdeffloatfunc cotd(x):= cosd(x)/sind(x);%
401 \xintdeffloatfunc tan(x) := (x)??
402             {(x>-0.79)?
403             {sin(x)/cos(x)}
404             {-cotd(90+x*oneRadian)
405             }
406             }
407             {0}
408             {(x<0.79)?
409             {sin(x)/cos(x)}
410             {cotd(90-x*oneRadian)}
411             }
412             ;%
413 \xintdeffloatfunc cot(x) := (abs(x)<0.79)?
414             {cos(x)/sin(x)}
415             {(x)??
416             {-tand(90+x*oneRadian)}
417             {0}
418             {tand(90-x*oneRadian)}
419             };%

```

12.12 sec(), secd(), csc(), cscd()

```

420 \xintdeffloatfunc sec(x) := inv(cos(x));%
421 \xintdeffloatfunc csc(x) := inv(sin(x));%
422 \xintdeffloatfunc secd(x):= inv(cosd(x));%
423 \xintdeffloatfunc cscd(x):= inv(sind(x));%

```

12.13 Core routine for inverse trigonometry

Compute asin(x)

The approach I shall first describe (which is only a first step towards our final approach) converges quintically but requires an initial square root computation. For atan(x), we do not have to do any such square root extraction. See code next.

The algorithm (for this first approach): we have $0 \leq t < 0.72$, let $t_1 = t*(1+t^2/6)$. We also have $u = \sqrt{1 - t^2}$. We seek $a = \text{Arcsin } t$ with $t = \sin(a)$.

Then $t_1 < \text{Arcsin } t$ and the difference (we don't know it!) δ_1 is < 0.02 . We compute $D = t*\cos(t_1) - u*\sin(t_1)$. This computation is done "exactly" via the `\xintexpr` encapsulation. In other terms we use doubled precision. Anyhow, currently (1.3e) the Float macros of `xintfrac.sty` for multiplication do go via such exact multiplication when the mantissas have the expected sizes. So we can't gain but only lose due to catastrophic subtraction in using float operations here.

Thus D is $\sin(a-t_1) = \sin(\delta_1)$. And $\delta_1 = \text{Arcsin } D$, but D is small! We then use again two terms of the Arcsin series and define $t_2 = t_1 + D * (1 + D^2/6)$. Let $\delta_2 = a - t_2$. Then δ_2 is of the order of the neglected term $3*(\delta_1)^5/40$.

©copyright J.F. Burnol, March 30, 2019. This surely has a name.

The algorithm is quintically convergent. One can do the same to go from exp to log. Basically the idea is that we can improve the Newton Method for any function f for which knowing target value of f implies one also knows target value of its derivative. In fact I obtained the quintic algorithm by

all cases. Using it for arcsine requires a preliminary square root extraction, but for arctangent one arranges things to avoid having to compute a square root.

©copyright J.F. Burnol, April 5, 2019. This surely has a name.

Certainly I can do similar things to compute logarithms.

```

424 \xintdeffloatfunc asin_aux(X) := 1
425 \ifnum\XINTdigits>3 % actually 4 would achieve 1ulp in place of <0.5ulp
426         + X(1/6
427 \ifnum\XINTdigits>9
428         + X(3/40
429 \ifnum\XINTdigits>16
430         + X(5/112
431 \ifnum\XINTdigits>25
432         + X(35/1152
433 \ifnum\XINTdigits>35
434         + X(63/2816
435 \ifnum\XINTdigits>46
436         + X(231/13312
437         )\fi)\fi)\fi)\fi)\fi)\fi;%
438 \xintdeffloatfunc asin_o(D, T) := T + D*asin_aux(sqrt(D));%
439 \xintdeffloatfunc asin_n(V, T, t, u) :=% V is square of T
440         asin_o (\xintexpr t*cos_aux(V) - u*T*sin_aux(V)\relax, T);%
441 \xintdeffloatfunc asin_m(T, t, u) := asin_n(sqrt(T), T, t, u);%
442 \xintdeffloatfunc asin_l(t, u) := asin_m(t*asin_aux(sqrt(t)), t, u);%

```

12.14 asin(), asind()

Only non-negative arguments t and u for $\text{asin}_a(t,u)$, and $\text{asind}_a(t,u)$.

At 1.4 usage of $\text{sqrt}_()$ which has only one argument, whereas currently $\text{sqrt}()$ admits a second optional argument hence sub-optimality here if we use $\text{sqrt}()$, especially since 1.4 handles more fully such functions with optional argument in xintdeffunc .

Actually thinking of making $\text{sqrt}()$ a one argument only function and $\text{sqrt}_()$ will be the one with two arguments. But I worked hard on the xintdeffunc hooks, thus some reticence, because why then not do that for all others?

```

443 \xintdeffloatfunc asin_a(t, u) := (t<u)?
444         {asin_l(t, u)}
445         {Piover2 - asin_l(u, t)}
446         ;%
447 \xintdeffloatfunc asind_a(t, u):= (t<u)?
448         {asin_l(t, u) * oneRadian}
449         {90 - asin_l(u, t) * oneRadian}
450         ;%
451 \xintdeffloatfunc asin(t) := (t)??
452         {-asin_a(-t, sqrt_(1-sqr(t)))}
453         {0}
454         {asin_a(t, sqrt_(1-sqr(t)))}
455         ;%
456 \xintdeffloatfunc asind(t) := (t)??
457         {-asind_a(-t, sqrt_(1-sqr(t)))}
458         {0}
459         {asind_a(t, sqrt_(1-sqr(t)))}
460         ;%

```

12.15 acos(), acosd()

```
461 \xintdeffloatfunc acos(t) := Piover2 - asin(t);%
462 \xintdeffloatfunc acosd(t) := 90 - asind(t);%
```

12.16 atan(), atand()

This involves no square root!

TeX hackers note 1:

The `subs(, x = ..)` mechanism has no utility in a function definition, there is no parallel mechanism at the underlying macros, so in fact the substituted things will remain unevaluated if they involve indeterminates, so this is exactly like not trying to make things more efficient at all.

Currently, the only way is thus to employ auxiliary functions like is done next. Contrarily to TeX macros, we must define the functions one after the other in the correct order, so the auxiliaries come first.

TeX hackers note 2:

At 1.4, the way to inject lazy conditionals in function definitions has changed. Prior one used `if(,,)` and `ifsgn(,,)` which was counter-intuitive because in pure numeric context they evaluate all branches. Now one must use `?` and `??` which are the lazy conditionals from the numeric context.

radians

```
463 \xintdeffloatfunc atan_b(t, w, z) := 0.5 * (w < 0)?
464           {Pi - asin_a(2z * t, -w*z)}
465           {asin_a(2z * t, w*z)}
466           ;%
467 \xintdeffloatfunc atan_a(t, T) := atan_b(t, 1-T, inv(1+T));%
468 \xintdeffloatfunc atan(t) := (t)??
469           {-atan_a(-t, sqr(t))}
470           {0}
471           {atan_a(t, sqr(t))}
472           ;%
```

degrees

```
473 \xintdeffloatfunc atand_b(t, w, z) := 0.5 * (w < 0)?
474           {180 - asind_a(2z * t, -w*z)}
475           {asind_a(2z * t, w*z)}
476           ;%
477 \xintdeffloatfunc atand_a(t, T) := atand_b(t, 1-T, inv(1+T));%
478 \xintdeffloatfunc atand(t) := (t)??
479           {-atand_a(-t, sqr(t))}
480           {0}
481           {atand_a(t, sqr(t))}
482           ;%
```

12.17 Arg(), atan2(), Argd(), atan2d(), pArg(), pArgd()

`Arg(x,y)` function from $-\pi$ (excluded) to $+\pi$ (included)

```
483 \xintdeffloatfunc Arg(x, y) := (y > x)?
484           {(y > -x)?
485           {Piover2 - atan(x/y)}
```

```

486             {(y<0)?
487             {-Pi + atan(y/x)}
488             {Pi + atan(y/x)}
489             }
490         }
491     {(y>-x)?
492     {atan(y/x)}
493     {-Piover2 + atan(x/-y)}
494     }
495     ;%

```

`atan2(y,x) = Arg(x,y) ... (some people have atan2 with arguments reversed but the convention here seems the most often encountered)`

```

496 \xintdeffloatfunc atan2(y,x) := Arg(x, y);%

```

`Argd(x,y) function from -180 (excluded) to +180 (included)`

```

497 \xintdeffloatfunc Argd(x, y):= (y>x)?
498             {(y>-x)?
499             {90 - atand(x/y)}
500             {(y<0)?
501             {-180 + atand(y/x)}
502             {180 + atand(y/x)}
503             }
504         }
505     {(y>-x)?
506     {atand(y/x)}
507     {-90 + atand(x/-y)}
508     }
509     ;%

```

`atan2d(y,x) = Argd(x,y)`

```

510 \xintdeffloatfunc atan2d(y,x) := Argd(x, y);%

```

`pArg(x,y) function from 0 (included) to 2π (excluded) I hesitated between pArg, Argpos, and Argplus. Opting for pArg in the end.`

```

511 \xintdeffloatfunc pArg(x, y):= (y>x)?
512             {(y>-x)?
513             {Piover2 - atan(x/y)}
514             {Pi + atan(y/x)}
515             }
516     {(y>-x)?
517     {(y<0)?
518     {twoPi + atan(y/x)}
519     {atan(y/x)}
520     }
521     {threePiover2 + atan(x/-y)}
522     }
523     ;%

```

`pArgd(x,y) function from 0 (included) to 360 (excluded)`

```

524 \xintdeffloatfunc pArgd(x, y):=(y>x)?
525           {(y>-x)?
526             {90 - atan(x/y)*oneRadian}
527             {180 + atan(y/x)*oneRadian}
528           }
529           {(y>-x)?
530           {(y<0)?
531             {360 + atan(y/x)*oneRadian}
532             {atan(y/x)*oneRadian}
533           }
534           {270 + atan(x/-y)*oneRadian}
535         }
536       ;%

```

12.18 Synonyms: tg(), cotg()

These are my childhood notations and I am attached to them. In radians only. We skip some overhead here by using a `\let` at core level.

```

537 \expandafter\let\csname XINT_flexpr_func_tg\expandafter\endcsname
538       \csname XINT_flexpr_func_tan\endcsname
539 \expandafter\let\csname XINT_flexpr_func_cotg\expandafter\endcsname
540       \csname XINT_flexpr_func_cot\endcsname

```

12.19 Let the functions be known to the `\xintexpr` parser

See [xint.pdf](#) for some explanations (as well as code comments in `xintexpr.sty`). In fact it is this context which led to my addition at 1.3e of `\xintdefefunc` to the `\xintexpr` syntax.

```

541 \xintFor #1 in {sin, cos, tan, sec, csc, cot,
542               asin, acos, atan}\do
543 {%
544   \xintdeffunc #1(x) := \xintfloatexpr #1(sfloat(x))\relax;%
545   \xintdeffunc #1d(x):= \xintfloatexpr #1d(sfloat(x))\relax;%
546 }%
547 \xintFor #1 in {Arg, pArg, atan2}\do
548 {%
549   \xintdeffunc #1(x, y) := \xintfloatexpr #1(sfloat(x), sfloat(y))\relax;%
550   \xintdeffunc #1d(x, y):= \xintfloatexpr #1d(sfloat(x), sfloat(y))\relax;%
551 }%
552 \xintdeffunc tg(x) := \xintfloatexpr tg(sfloat(x))\relax;%
553 \xintdeffunc cotg(x):= \xintfloatexpr cotg(sfloat(x))\relax;%
554 \xintdeffunc sinc(x):= \xintfloatexpr sinc(sfloat(x))\relax;%

```

Restore used dummy variables to their status prior to the package reloading. On first loading this is not needed naturally, because this is done immediately at end of `xintexpr.sty`.

```

555 \xintFor* #1 in {iDTVtuwxyzX}\do{\xintrestorevariable{#1}}%

```

13 Package *xintlog* implementation

Contents

13.1	Catcodes, ε -TeX and reload detection	438
13.2	Library identification	439
13.3	Loading of <i>poormanlog</i> package	439
13.4	The <code>log10()</code> and <code>pow10()</code> functions	439
13.5	The <code>log()</code> , <code>exp()</code> , and <code>pow()</code> functions	440
13.6	<code>\poormanloghack</code>	441

I almost included extended precision implementation for 1.3e but was a bit short on time; besides I hesitated between using *poormanlog* at starting point or not. For up to 50 digits, it would help reduce considerably the needed series for the logarithm. For more digits I should rather apply my copyrighted method of the arcsine (it must be in literature).

13.1 Catcodes, ε -TeX and reload detection

```

1 \begingroup\catcode61\catcode48\catcode32=10\relax%
2 \catcode13=5 % ^^M
3 \endlinechar=13 %
4 \catcode123=1 % {
5 \catcode125=2 % }
6 \catcode64=11 % @
7 \catcode35=6 % #
8 \catcode44=12 % ,
9 \catcode45=12 % -
10 \catcode46=12 % .
11 \catcode58=12 % :
12 \catcode94=7 % ^
13 \def\z{\endgroup}%
14 \def\empty{}\def\space{ }\newlinechar10
15 \expandafter\let\expandafter\w\csname ver@xintexpr.sty\endcsname
16 \expandafter\let\expandafter\x\csname ver@xintlog.sty\endcsname
17 \expandafter
18 \ifx\csname PackageInfo\endcsname\relax
19 \def\y#1#2{\immediate\write-1{Package #1 Info:^^J%
20 \space\space\space\space#2.}}%
21 \else
22 \def\y#1#2{\PackageInfo{#1}{#2}}%
23 \fi
24 \expandafter
25 \ifx\csname numexpr\endcsname\relax
26 \y{xintlog}{numexpr not available, aborting input}%
27 \aftergroup\endinput
28 \else
29 \ifx\w\relax % xintexpr.sty not yet loaded.
30 \y{xintlog}%
31 {Loading should be via \ifx\x\empty\string\usepackage{xintexpr.sty}
32 \else\string\input\space xintexpr.sty \fi
33 rather, aborting}%
34 \aftergroup\endinput
35 \else

```

```

36     \ifx\x\relax % first loading (initiated from xintexpr.sty)
37     \else
38     \ifx\x\empty % LaTeX first loading, \ProvidesPackage not yet seen
39     \else
40     \y{xintlog}{Already loaded, aborting}%
41     \aftergroup\endinput
42     \fi
43     \fi
44     \fi
45     \fi
46 \z%
```

Attention to catcode regime when loading below poormanlog. It (v0.04) uses ^ with its normal catcode but `\XINT_setcatcodes` would set it to letter.

This file can only be loaded from `xintexpr.sty` and it restores catcodes near its end. To play it safe and be hopefully immune to whatever is done in `poormanlog` or in `xinttrig.sty` which is loaded before, we will switch to standard catcode regime here.

As I learned the hard way (I never use my user macros), at the worst moment when wrapping up the final things for 1.3e release, `\xintexprSafeCatcodes` MUST be followed by some `\xintexprRestoreCatcodes` quickly, else next time it is used (for example by `\xintdefvar`) the `\xintexprRestoreCatcodes` will restore an obsolete catcode regime...

13.2 Library identification

```

47 \xintexprSafeCatcodes\catcode`_ 11
48 \XINT_providespackage
49 \ProvidesPackage{xintlog}%
50 [2020/02/25 v1.4b Logarithms and exponentials for xintexpr (JFB)]%
```

13.3 Loading of poormanlog package

Attention to catcode regime when loading `poormanlog`. It matters less now for 1.3f as those chunks of code from `poormanlog.tex` v0.04 which needed specific `xintexpr` like catcodes got transferred here anyway.

```

51 \ifdefined\RequirePackage
52 \RequirePackage{poormanlog}%
53 \else
54 \input poormanlog.tex
55 \fi
```

`\XINT_setcatcodes` switches to the standard catcode regime of `xint*.sty` files. And we need the `xintexpr` catcode for ! too (cf `\XINT_expr_func_pow`)

See the remark above about importance of doing `\xintexprRestoreCatcodes` if `\xintexprSafeCatcodes` has been used...

```

56 \xintexprRestoreCatcodes\csname XINT_setcatcodes\endcsname\catcode`\! 11
```

13.4 The `log10()` and `pow10()` functions

The support macros from `poormanlog` v0.04 `\PoorManLogBaseTen`, `\PoorManLogPowerOfTen`, `\PoorManPower` got transferred into `xintfrac.sty` at 1.3f.

```

57 \expandafter\def\csname XINT_expr_func_log10\endcsname#1#2#3%
58 {%
```

```

59 \expandafter #1\expandafter #2\expandafter{%
60 \romannumeral`&&\XINT:NEhook:f:one:from:one
61 {\romannumeral`&&\PoorManLogBaseTen#3}}%
62 }%
63 \expandafter\let\csname XINT_flexpr_func_log10\expandafter\endcsname
64 \csname XINT_expr_func_log10\endcsname
65 \expandafter\def\csname XINT_expr_func_pow10\endcsname#1#2#3%
66 {%
67 \expandafter #1\expandafter #2\expandafter{%
68 \romannumeral`&&\XINT:NEhook:f:one:from:one
69 {\romannumeral`&&\PoorManPowerOfTen#3}}%
70 }%
71 \expandafter\let\csname XINT_flexpr_func_pow10\expandafter\endcsname
72 \csname XINT_expr_func_pow10\endcsname

```

13.5 The log(), exp(), and pow() functions

The `log10()` and `pow10()` were defined by `poormanlog v0.04` but have been moved here at `xint 1.3f`. The support macros are defined in `xintfrac.sty`.

```

73 \def\XINT_expr_func_log #1#2#3%
74 {%
75 \expandafter #1\expandafter #2\expandafter{%
76 \romannumeral`&&\XINT:NEhook:f:one:from:one
77 {\romannumeral`&&\xintLog#3}}%
78 }%
79 \def\XINT_flexpr_func_log #1#2#3%
80 {%
81 \expandafter #1\expandafter #2\expandafter{%
82 \romannumeral`&&\XINT:NEhook:f:one:from:one
83 {\romannumeral`&&\XINTinFloatLog#3}}%
84 }%
85 \def\XINT_expr_func_exp #1#2#3%
86 {%
87 \expandafter #1\expandafter #2\expandafter{%
88 \romannumeral`&&\XINT:NEhook:f:one:from:one
89 {\romannumeral`&&\xintExp#3}}%
90 }%
91 \def\XINT_flexpr_func_exp #1#2#3%
92 {%
93 \expandafter #1\expandafter #2\expandafter{%
94 \romannumeral`&&\XINT:NEhook:f:one:from:one
95 {\romannumeral`&&\XINTinFloatExp#3}}%
96 }%

```

Attention that the ! is of catcode 11 here.

```

97 \def\XINT_expr_func_pow #1#2#3%
98 {%
99 \expandafter #1\expandafter #2\expandafter{%
100 \romannumeral`&&\XINT:NEhook:f:one:from:two
101 {\romannumeral`&&\PoorManPower#3}}%
102 }%
103 \let\XINT_flexpr_func_pow\XINT_expr_func_pow

```


13.6 \poormanloghack

With `\poormanloghack{**}`, the `**` operator will use `pow10(y*log10(x))`. Same for `^`. Sync'd with `xintexpr 1.4`.

```

104 \catcode\* 11
105 \def\poormanloghack**
106 {%
107 \def\XINT_tmpa ##1##2##3##4##5##6%
108 {%
109 \def ##3####1% \XINT_expr_op_<op>
110 {%
111 \expanded{\unexpanded{##4{####1}}\expandafter}%
112 \romannumeral`&&\expandafter##2\romannumeral`&&\XINT_expr_getnext
113 }%
114 \def##2####1% \XINT_expr_check_<op>
115 {%
116 \xint_UDsignfork
117 ####1{\expandafter##2\romannumeral`&&@##1}%
118 -{##5####1}%
119 \krof
120 }%
121 \def##5####1####2% \XINT_expr_checkp_<op>
122 {%
123 \ifnum ####1>\XINT_expr_precedence_**
124 \expandafter##5%
125 \romannumeral`&&\csname XINT_##6_op_####2\expandafter\endcsname
126 \else
127 \expandafter ##3##1\expandafter ##4##2%
128 \fi
129 }%
130 }%
131 \expandafter\XINT_tmpa
132 \csname XINT_expr_op_-ix\expandafter\endcsname
133 \csname XINT_expr_check_<op>\endcsname
134 \XINT_expr_op_**
135 \XINT_expr_exec_**
136 \XINT_expr_checkp_** {expr}%
137 \expandafter\XINT_tmpa
138 \csname XINT_flexpr_op_-ix\expandafter\endcsname
139 \csname XINT_flexpr_check_<op>\endcsname
140 \XINT_flexpr_op_**
141 \XINT_flexpr_exec_**
142 \XINT_flexpr_checkp_** {flexpr}%
143 \def\XINT_expr_exec_** ##1##2##3##4% \XINT_expr_exec_<op>
144 {%
145 \expandafter##2\expandafter##3\expandafter{%
146 \romannumeral`&&\XINT:NEhook:f:one:from:two
147 {\romannumeral`&&\PoorManPower##1##4}}%
148 }%
149 \let\XINT_flexpr_exec_**\XINT_expr_exec_**
150 }%
151 \def\poormanloghack^

```

```

152 {%
153 \def\XINT_tmpa ##1##2##3##4##5##6%
154 {%
155 \def ##3####1% \XINT_expr_op_<op>
156 {%
157 \expanded{\unexpanded{##4{####1}}\expandafter}%
158 \romannumeral`&&\expandafter##2\romannumeral`&&\XINT_expr_getnext
159 }%
160 \def##2####1% \XINT_expr_check_-_<op>
161 {%
162 \xint_UDsignfork
163 ####1{\expandafter##2\romannumeral`&&@##1}%
164 -{##5####1}%
165 \krof
166 }%
167 \def##5####1####2% \XINT_expr_checkp_<op>
168 {%
169 \ifnum ####1>\XINT_expr_precedence_^
170 \expandafter##5%
171 \romannumeral`&&\csname XINT_##6_op_####2\expandafter\endcsname
172 \else
173 \expandafter ##3#1\expandafter ##3#2%
174 \fi
175 }%
176 }%
177 \expandafter\XINT_tmpa
178 \csname XINT_expr_op_-ix\expandafter\endcsname
179 \csname XINT_expr_check_-_^\endcsname
180 \XINT_expr_op_^
181 \XINT_expr_exec_^
182 \XINT_expr_checkp_^ {expr}%
183 \expandafter\XINT_tmpa
184 \csname XINT_flexpr_op_-ix\expandafter\endcsname
185 \csname XINT_flexpr_check_-_^\endcsname
186 \XINT_flexpr_op_^
187 \XINT_flexpr_exec_^
188 \XINT_flexpr_checkp_^ {flexpr}%
189 \def\XINT_expr_exec_^ ##1##2##3##4% \XINT_expr_exec_<op>
190 {%
191 \expandafter##2\expandafter##3\expandafter{%
192 \romannumeral`&&\XINT:NEhook:f:one:from:two
193 {\romannumeral`&&\PoorManPower##1##4}}%
194 }%
195 \let\XINT_flexpr_exec_^\XINT_expr_exec_^
196 }%
197 \def\poormanloghack#1{\csname poormanloghack#1\endcsname}%

```

IMPORTANT: We don't worry about resetting catcodes now as this file is theoretically only loadable from xintexpr.sty itself which will take care of the needed restore.

14 Cumulative line count

`xintkernel`: 597. Total number of code lines: 16885. (but 3866 lines among them
`xinttools`:1629. start either with `{%` or with `}%`.)
`xintcore`:2165. Each package starts with circa 50 lines dealing with cat-
`xint`:1623. codes, package identification and reloading management,
`xintbinhex`: 472. also for Plain TeX. Version 1.4b of 2020/02/25.
`xintgcd`: 368.
`xintfrac`:3505.
`xintseries`: 386.
`xintcfrac`:1029.
`xintexpr`:4359.
`xinttrig`: 555.
`xintlog`: 197.